Compute (Bridgend) Ltd

REPORT Utility

Release 3.60

8 Merthyr Mawr Road, Bridgend, Wales UK CF31 3NH

Tel: +44 (1656) 65 2222
Eml: support@cbl.com

CBL Web Site - www.cbl.com

Contents

Contents

Contents

Documentation Notes

Fourth Edition, March 2025

Information in this document details use of the SMF record processing utilities provided by the **CBL Product Suite** component, **FileKit**.

CBL Product Suite for z/OS, z/VM (CMS) and z/VSE operating systems, which includes SELCOPY, SLC, FileKit and CBLVCAT, is available for download and install from **www.cbl.com/selcdl.php**.

The following publications for CBL Product Suite and its component products are available in Adobe Acrobat PDF format at CBL web page **www.cbl.com/documentation.php**:

- CBL Product Suite Customisation Guide
- SELCOPY User Manual
- SELCOPY C++ (SLC) Language Reference
- CBLVCAT User Manual
- FileKit Reference and User Guide
- FileKit Text Editor
- FileKit Data Editor (SDE)
- FileKit Quick Reference
- FileKit REPORT Utility
- FileKit SMF Utilities
- FileKit Training Manual

At the time of publication, this document is believed to be correct. Where the program product differs from that stated herein, Compute (Bridgend) Ltd reserve the right to revise either the program or its documentation at their discretion. CBL do not warrant that upward compatibility will be maintained for any use made of this program product to perform any operation in a manner not documented within the user manual.

The following generic terms are used throughout this document to indicate all available versions and releases of IBM mainframe operating systems:

z/OS	-	z/OS, OS/390, MVS/ESA, MVS/XA, MVS/SP, OS.
z/VSE	-	z/VSE, VSE/ESA, VSE/SP, DOS.
z/VM CMS	-	z/VM, VM/ESA, VM/XA, VM/SP.
All	-	All z/OS, z/VSE and z/VM CMS operating systems.

Introduction

About this Book

This book is a user guide and reference for the REPORT Utility included with the FileKit element of CBL Product Suite for z/OS.

It provides all the relevant information and guidance required to produce a printable report or generate CSV, XML or JSON format output from formatted data.

Report Utility Overview

The REPORT Utility is provided as part of FileKit, the interactive tools and utilities element of the CBL Product Suite for z/OS. It allows users to create attractive reports from information contained in data set records or DB2 tables.

The utility takes advantage of FileKit's data structuring and mapping features to expand and format records into a number of data fields. The structure used to map the input data may be a COBOL or PL/1 copybook, an Assembler DSECT or a FileKit structure definition object (SDO).

Special processing is reserved for SMF records which each have a complex, yet well-defined format which is published in IBM documentation. The REPORT utility uses the SMF SDO structures provided in FileKit to map and report on SMF input records. See also publication "*FileKit SMF Utilities*".

Field values obtained from each of the input records or DB2 table rows processed, may be displayed in a printable report. Alternatively, field values may be written to Comma Separated Variable (CSV) format records or as eXtensible Markup Language (XML) or JavaScript Object Notation (JSON) formatted output.

The appearance of headers, footers, detail lines and other elements in any printable report generated by the REPORT utility is very flexible. Simple, easy-to-learn REPORT utility control statement syntax is used to define the report layout.

FileKit REPORT generation panel windows and the REPORT primary command provide the interfaces by which users can execute the REPORT utility in the FileKit foreground (TSO/ISPF) or generate JCL for execution in batch.

Notation Conventions

The following list defines notations used in this publication.

- Text in syntax diagrams and examples of SELCOPY syntax are presented in a `monospace` font.
- REPORT Utlity keyword operands and REPORT definition keywords are shown in upper case (e.g. COLUMNS, BREAK, RPTDEF, OLIM) although may be entered in upper or lower case, or a mixture of both cases.
- Keywords may be shown with trailing, lower case characters. The upper cased portion of the keyword identifies the minimum abbreviation for the keyword. (e.g. Left indicates that L, LE, LEF and LEFT are all acceptable alternatives)
- Variables appear in lowercase *italics* (e.g. *input-field*) and represent programmer defined parameters or keyword parameter values.
- Syntax diagram footnote references are represented by a number in parentheses (e.g. (1)).
- A single blank may be represented by character b.

Syntax diagrams adhere to the following standards:

Arrow Symbols

Diagrams should be read from left to right, top to bottom and follow the path of the line. Junctions in the line are represented by a plus (+) symbol.

- ◊ `>>-` indicates the beginning of a statement.
- ◊ `->` indicates the statement syntax continues on the next line.
- ◊ `>-` indicates the statement syntax has been continued from the previous line.
- ◊ `-><` indicates the end of a statement.

The horizontal path line, delimited by arrow symbols, denotes the main path of the syntax diagram.

Required Items

Required items appear on the main path.

```
>>-- REQUIRED_ITEM --------------------------------><
```

Optional Items

Optional items appear below the main path.

```
>>-- REQUIRED_ITEM ---+----------------------+-----><
                      |                      |
                      +-- optional_item -----+
```

If an optional item appears above the main path, then that item has no effect on the execution of the statement and is used only for readability.

```
                      +-- optional_item -----+
                      |                      |
>>-- REQUIRED_ITEM ---+----------------------+-----><
```

Multiple Required or Optional Items

If one or more alternative optional items exist, they appear vertically on separate paths. If selection of one of the items is optional, the items appear in paths below the main path.

```
>>-- REQUIRED_ITEM ---+----------------------+-----><
                      |                      |
                      +-- optional_choice1 --+
                      |                      |
                      +-- optional_choice2 --+
```

If selection of one of the items is mandatory, one of the items appears on the main path and all other items appear on paths below the main path.

```
>>-- REQUIRED_ITEM ---+-- required_choice1 --+-----><
                      |                      |
                      +-- required_choice2 --+
```

Repeatable Items

An arrow occurring above a path line returning to a junction to the left of another junction indicates the item between the junctions may be repeated.

```
                      +----------------------+
                      v                      |
>>-- REQUIRED_ITEM ---+-- repeatable_item ---+-----><
```

If the arrow occurs above a number of multiple item paths, this indicates that more than one of the items may be specified and each of the items are repeatable.

Default Items

If one of a number of optional items is default, the path containing that item appears above the main path.

```
                      +-- default_choice ----+
                      |                      |
>>-- REQUIRED_ITEM ---+----------------------+-----><
                      |                      |
                      +-- optional_choice ---+
                      |                      |
                      +-- optional_choice ---+
```

Fragments

Syntax diagrams may be split into fragments so that related syntax items are removed from the diagram and displayed in a syntax diagram fragment below.

Fragments are named and their location within the parent diagram represented by the fragment name in **bold** print, enclosed in vertical bars (or symbols). The same vertical bars are used to indicate the beginning and end of the syntax diagram fragment below the parent diagram.

```
>>-- REQUIRED_ITEM --- item1 --------| fragment |-->< 
```

fragment:

```
|---+-- item2 --+--- KEYWORD --+-----------+-------|
    |           |               |           |
    +-- item3 --+               +-- item4 --+
                                |           |
                                +-- item5 --+
```

Summary of Changes

This section describes changes made to the publication and includes a summary of the new features added to the REPORT Utility since its first release in FileKit 3.50.

Third Edition (April 2024)

This section is a summary of significant new REPORT Utility features provided in FileKit Release 3.60 (and FileKit Release 3.50 with PTF SYSMODs RS35001 and RS35002 applied).

Report Definition INPUT/OUTPUT Sections

Sections INPUT: and OUTPUT: introduced to define default input source and output data set destination respectively. Entries specified in these sections may be overridden using REPORT command parameters or values entered in REPORT utility panel input fields

For details, see:

- INPUT
- OUTPUT

Report Definition MAP Section

The MAP: section has been updated to support field definitions specified using FileKit CREATE STRUCTURE command syntax. This provides a more comprehensive alternative to the SYMNAMES format currently supported.

For details, see:

- MAP

Built-In Functions

The following built-in REXX functions have been introduced for use in the COMPUTE: section:

- **BYPASS()**
  Used to skip reporting on the record or record segment currently being processed.
- **DATEINC()**
  Used to increment (or decrement) a date value by a number of days, months or years.
- **EOF()**
  Used to force end of input to skip reporting on the current record or record segment, and all records that follow.
- **MONTHBEG()**
  Used to return the ISO format date for the first day in the month of the current or specified date.
- **MONTHEND()**
  Used to return the ISO format date for the last day in the month of the current or specified date.
- **TIMEINC()**
  Increment (or decrement) a time or timestamp (date & time) value by a number of hours, minutes or seconds.

For details, see:

- Appendix B. Built-in Functions

Report Definition OPTIONS Section

Support has been included for the following OPTIONS: section options:

- **DETAIL(*nlines*[,ALL|DISPLAY])**
  ALL or DISPLAY options indicate whether generated statistics values are derived from all detail lines in control group or only those displayed. The DETAIL option is now also obeyed for CSV, JSON and XML report output.

- **FIELDNAME([SHORT] | [LONG])**
  For input fields, forces REPORT to assign field values to the unqualified (SHORT) format of the field name variable, the qualified (LONG) format of the field name variable, or both.

- **FIND(*string* [, ...])**
  For non-DB2 type input, specifies search strings to be used for record selection. For SMF type input, FIND is one of the SMF record content match criteria options.

- **HEADWIDTH(*int*)**
  Specifies the width of header and footer lines within a printed report.

- **ILIM(*int*)**
  Specifies the input limit, maximum number of records or DB2 rows to be read from the data source.

- **NUMTRUNC(YES|NO[,*char*])**
  Specifies whether numeric value truncation is tolerated (and so partial numeric values displayed) or if the overflowing field value is to be substituted with repeating truncation filler character (*char*) filling the display field width.

- **OLIM(*int*)**
  Specifies the output limit, maximum number of detail report lines that may be written to the output report.

- **PAGEDEPTH(*int*)**
  Specifies the number of lines written to each page of the printed report output.

- **REPORT(DB2|SDE|SMF)**
  Specifies the type of data in the input source.

- **SHORTSTATS(YES|NO)**
  Specifies whether statistics values that overflow the display area width will be shortened to a value with a multiplier suffix and possible "greater than" (">") prefix, or will be substituted with repeating truncation filler characters as defined by option NUMTRUNC.

- **SMFDATEHI(*timestamp* | *-days*)**
  For SMF type input only, specifies the latest SMF record timestamp to be processed.

- **SMFDATELO(*timestamp* | *-days*)**
  For SMF type input only, specifies the earliest SMF record timestamp to be processed.

- **SMFJOBNAME(*jobname* [, ...])**
  For SMF type input only, specifies jobname masks for SMF record content match criteria processing.

- **SMFLOGIC(OR|AND)**
  For SMF type input only, specifies the logical operation performed between SMF record content match criteria

- **SMFONLINE(YES|NO)**
  For SMF type input only, specifies whether SMF records are processed directly from an online SMF log data set or from SMF archive data sets.

- **SMFSID(*sid* [, ...])**
  For SMF type input only, specifies system identification masks for SMF record content match criteria processing.

- **SMFTYPES(*rectype* | *rectype*:*rectype* | {*rectype*-*subtype* | *rectype*#*subtype*} [, ...])**:
  For SMF type input only, specifies record type/subtype masks for SMF record content match criteria processing.

- **SMFUSER(*username* [, ...])**
  For SMF type input only, specifies user name masks for SMF record content match criteria processing.

For details, see:

- OPTIONS

Input Field Value RESET

The RESET: section will reset all input field values to null after processing a record or record segment of the specified record type mapping. RESET of individual input field values can now be further controlled using the following field definition operands in COLUMNS: and REQUIRED: sections:

- **NORESET**
  Suppresses reset of the field value that would occur as a result of a RESET: section specification.

- **NORESETBREAK**
  Suppresses reset of the field value that would occur as a result of a RESET: section specification unless the field value is to be displayed in the first line of a control group. Only applicable to COLUMNS: section.

- **NORESETPAGE**
  Suppresses reset of the field value that would occur as a result of a RESET: section specification unless the field value is to be displayed in the first line of a new page. Only applicable to COLUMNS: section.

For details, see:

- COLUMNS
- REQUIRED

External Field Values

In previous releases, fields identified as a SORT key, BREAK key or a variable value in a *print-expression* would inherit an external formatted value defined by the matching field name entry in the COLUMNS: or REQUIRED: section.

To overcome this restriction, the external field value format may now be controlled independently on entries in the BREAK: and SORT: sections, as well as on individual elements of a *print-expression*.

Any of the following optional operands may be specified on an entry in the COLUMNS: or REQUIRED: sections, on a BREAK: or SORT: section key field definition, or on a *print-expression* element:

- **SUBSTR**
  Specifies the start character number and optionally the length of the field data to be used as the field value. This occurs before any STRIP operation takes place.

- **STRIP**
  Specifies that leading and trailing blanks are to be stripped from the field value. This occurs before field alignment takes place.

- *width*:
  The width of the field value text. The field value will be aligned within this field width and blank padded or truncated accordingly. If not specified, the defined maximum width of the built-in or input field is used. Otherwise, for compute fields a default width of 9 is used.

- **CENTER** | **CENTRE** | **LEFT** | **RIGHT**
  The alignment of the field value within the field width text. Alignment will occur after any STRIP keyword operation has taken place.

For details, see:

- Print Expressions
- BREAK
- COLUMNS
- REQUIRED
- SORT

Statistical Values

The following enhancements have been introduced relating to the display of statistical values:

- Entries in the COLUMNS: and REQUIRED: sections now support operand **NBTOTAL** which will generate total numbers of Non-Blank field values for each control group. By default, this value is reported in the control break totals line.

- Break line *print-expression* field elements may now specify one of the statistics based operands: **AVERAGE**, **MAXIMUM**, **MINIMUM**, **NBTOTAL**, **NZAVERAGE**, **NZMINIMUM** or **TOTAL**. Instead of inserting the prevailing field value in the break line, the relevant statistics value will be inserted for the current control group.

- The default statistics columns now include columns defined with compute fields if these fields are recognised as containing numeric data. Previously, compute fields were always treated as being non-numeric unless a datatype specification was provided on the COLUMNS:/REQUIRED: section field reference.

For details, see:

- Print Expressions
- COLUMNS
- REQUIRED

Restrictions Removed

The following restrictions no longer apply:

◊ The REPORT command syntax now supports specification of a report definition source file only with no other operands, i.e. `REPORT report_ctl`. This is equivalent to specifying `REPORT RUN RPTDEF(report_ctl)`.

Supporting this syntax, enables foreground execution of a report from a FileKit list of report definition source library members. Simply type REPORT in the list's command area. INPUT: and OUTPUT: sections in the report definition are mandatory when using this method to execute REPORT.

◊ Input fields and compute fields used as a key field in the SORT: or BREAK: sections, or referenced within a *print-expression*, no longer need to be explicitly defined in the COLUMNS: or REQUIRED: sections.

However, an input field must still be defined in the COLUMNS: or REQUIRED: section if it is to be referenced as a REXX variable within the COMPUTE: section routine.

Repeating Segments

Obtaining input field values from repeating, secondary segments is only possible if the secondary segment record-type mapping is named in the REPEAT: section.

However, output of a report detail line is triggered whenever the current input segment matches a record-type mapping named in the REPEAT: section. This may not be desirable if the next report output line requires further field values to be obtained from a subsequent segment.

To allow for this situation, operand "INPUT" may be entered following a record-type name specification in the REPEAT: section, to identfy the record segment as being for input only. Segments with this record-type mapping will *not* trigger output of a report detail line.

Reporting on segmented records, and thus use of a REPEAT: section, is most common for SMF record processing.

For details, see:

◊ REPEAT

Various Fixes

The following fixes have been implemented:

◊ No longer print only the last line of column headers when no input field entries are specified in the COLUMNS: section and column headers for compute field columns are split over multiple lines using the header break symbol ("|").

◊ Use an input field's value, prior to it being reset to null, when that field is referenced in a *print-expression* or used as a key field in the SORT: or BREAK: section.

Execution Speed

Significant improvement has been made to the performance of REPORT utility execution.

Conversion of the REPORT utility source code to High Level Assembler has meant a large reduction in REPORT execution times and CPU usage with many simple jobs running up to 10 times faster than previously achieved.

Unless a "BROWSE-EXIT:" section exists in the REPORT definition or input is via the focus Data-Edit view of formatted data, then REPORT will now read records (or DB2 table rows) using the FILEIO utility. The FILEIO utility has also undergone significant enhancements and, for sequential I/O, is more efficient than using Data-Edit BROWSE processing previously used by REPORT.

The original (REXX based) version of the REPORT utility command has been renamed as "REPORX" and may still be used if necessary. Note, however, that this version will no longer receive maintenance or include new product enhancements.

INIT-EXIT: Section

The INIT-EXIT: section has been introduced to allow initialisation of *compute-field* values when input is via FILEIO (default).

A *compute-field* is one that corresponds to a REXX variable of the same name, and which may be updated in the REXX statements of the COMPUTE: section.

The sections INIT-EXIT: and BROWSE-EXIT: are similar in that they are both executed prior to processing the first input record. However, presence of a BROWSE-EXIT section will trigger use of REPORT's Data-Edit BROWSE input processing.

◊ INIT-EXIT

BLANKIFEQUAL

Option BLANKIFEQUAL (synonym BIEQual, BLANKWHENEQUAL, BWEQual) may be specified on individual "COLUMNS:" section entries to force a blank column value when the column value matches that in the previous report detail line. This option is applicable only to PRINT output.

BLANKIFEQUAL(YES), or one of its synonyms, may be specified in the OPTIONS: section to imply BLANKIFEQUAL on all column entries.

For details, see:

◊ COLUMNS
◊ OPTIONS

CSV Output Options

CSVLITERALS, CSVQUOTED and CSVSTRIPALL options introduced to manage the appearance of values in CSV output.

◊ **CSVLITERALS(YES|NO)**
Determines whether or not *literal* values specified in the COLUMNS: section are included as values in the CSV output.

◊ **CSVQUOTED(YES|NO)**
Determines whether values are **always** enclosed in quotation marks ("), or are enclosed in quotation marks only when necessary. e.g. values containing commas (",").

◊ **CSVSTRIPALL(YES|NO)**
Determines whether or not leading and trailing blanks are stripped from the values so that the comma separator immediately follows the last non-blank character on all but the last value in the output line. If blanks are not stripped, then the value will be of a fixed length equal to the specified (or default) field width.

For details, see:

♦ OPTIONS

JSON Output Options

JSONARRAY, JSONINDENT, JSONLITERALS, JSONQUOTED and JSONSTRIPALL options introduced to manage the appearance of JSON output.

◊ **JSONARRAY(YES|NO)**
Determines whether field values for each report line are part of a single JSON object, or one object within an array of objects.

◊ **JSONINDENT(YES|NO)**
Determines whether the key/value pairs for each report field are to occur on the same report output line, or are to be written to a new line of the report output and indented beneath the opening and closing JSON object string braces ("{}").

◊ **JSONLITERALS(YES|NO)**
Determines whether or not *literal* values specified in the COLUMNS: section are included as the "string" value in a key/value pair of the JSON output.

◊ **JSONQUOTED(YES|NO)**
Determines whether or not values are **always** treated as JSON strings and enclosed in quotation marks ("), or are treated as strings for non-numeric field values only.

◊ **JSONSTRIPALL(YES|NO)**
Determines whether or not leading and trailing blanks are stripped from the values. This is particularly relevant to quoted JSON string values where leading trailing blanks would be treated as part of the string value. If blanks are not stripped, then the value will be of a fixed length equal to the specified (or default)

field width.

For details, see:

- OPTIONS

XML Output Options

XMLINDENT, XMLLITERALS and XMLSTRIPALL options introduced to manage the appearance of XML output.

- **XMLINDENT(YES|NO)**
  Determines whether the XML tagged report field values are to occur on the same report output line, or are to be each written to a new line of the report output and indented within the report line tags.

- **XMLLITERALS(YES|NO)**
  Determines whether or not *literal* values specified in the COLUMNS: section are included as values in the XML output.

- **XMLSTRIPALL(YES|NO)**
  Determines whether or not leading and trailing blanks are stripped from the XML values. If blanks are not stripped, then the value will be of a fixed length equal to the specified (or default) field width.

For details, see:

- OPTIONS

Other Report Definition OPTIONS

Support has also been included for the following OPTIONS: section options:

- **DB2NULL(YES|NO)**
  Determines whether or not the default Data-Edit NULL value output indicator character is displayed for a null value in a DB2 column defined with NULL. (See the NULLCHAR Data-Edit SET/QUERY/EXTRACT option). If DB2NULL(NO), the output value for a DB2 NULL value is blank.

- **LINESTRIP(YES|NO)**
  Determines whether or not trailing blank characters are to be stripped from the lines of text written to the REPORT output.

- **NUMBLANK(INCLUDE|EXCLUDE)**
  Determines whether numeric field values, displayed as blanks in the report PRINT output as a result of a BLANKIFEQUAL specification, are to be included in or excluded from column statistics calculations.

- **NUMDUP(INCLUDE|EXCLUDE)**
  Determines whether numeric field values, displayed as duplicates of the same column values on the previous detail line, are to be included in or excluded from column statistics calculations. Duplicate column values may occur when column values are not reset following output of a report detail line.

- **NUMTRUNC(...,INCLUDE|EXCLUDE)**
  NUMTRUNC option extended to include INCLUDE/EXCLUDE operands which determine whether truncated numeric field values that have been overwritten in the report output with truncation filler characters, are to be included in or excluded from column statistics calculations. Numeric field truncation occurs when the specified field width is shorter than the width required to display the value without loosing numeric precision.

- **PAGEPAD(YES|NO|AUTO)**
  Determines whether or not blank lines are to be written to the last page of a PRINT output report to pad the page to the specified (or defaulted) PAGEDEPTH. The AUTO operand will pad the last page with blank lines only if it is not the first (and therefore only) page of the report. No page padding occurs if a page footing (report definition FOOT: section) exists.

- **REXXCOMPOUND(YES|NO)**
  Determines whether or not the REXX variable names, defined for input-fields identified using a qualified field name, inherit the dot/period (".") field name qualifier separator character and so define a REXX compound symbol variable name. REXXCOMPOUND(NO) will instead use an underscore ("_") in place of a dot/period character in the variable name.

For details, see:

- OPTIONS

#RECNUM Built-in Field

The #RECNUM *built-in-field* name has a value equal to the **actual** record number of the input file as opposed to the input record sequence number after record filtering has occurred.

For details, see:

◊ Record Filtering
◊ Appendix A. Built-in Fields

Statistical Values

An *input-field* that has a floating point data type, is displayed as a numeric value comprising a signed mantissa and signed exponent values (e.g. "1.23760E-12"). Statistical values calculated on these types of field will now display the result in the same format unless the value can be accurately represented in the available display area width as a decimal value without an exponent.

For details, see:

◊ Statistical Values

Fourth Edition (March 2025)

This section is a summary of significant new REPORT Utility features provided in FileKit Release 3.60 with PTF SYSMOD RS36001 applied.

SMF Extended Headers

Support introduced for SMF record types 256-2047 that have an extended header. IBM published SMF record types 1153 and 1154 have this type of header.

WHERE: Section

WHERE section supported to define where-clauses which filter input records or segments.

The function of the **WHERE** section is similar to that of the **FILTER** section but with the following differences:

◊ WHERE section filtering may be used in addition to content match criteria filtering that are specified using options FIND, SMFTYPES, SMFSID, SMFUSERID and SMFJOBNAME (or their REPORT primary command override equivalents).

◊ For segmented record input (e.g. SMF records), the WHERE section will filter record segments of a specific record type mapping within the record, whereas the FILTER section will filter the entire record.

◊ For segmented record input, WHERE section specifications may reference any primary or secondary segment record-type on which the associated expression will be applied. FILTER section INCLUDE/EXCLUDE specifications must reference a primary segment record-type definition (i.e. one that maps data that occurs first in the record).

Note that FILTER expressions may still test field values in the secondary segment record-types that follow the primary record-type using fully qualified field names (i.e. <rectype>.<fieldname>).

◊ Unlike the WHERE section, FILTER will honour multiple INCLUDE (or EXCLUDE) specifications that reference the same record-type name. Doing so provides alternative criteria for selecting the record. This may however be easily achieved in a single expression in the WHERE section simply by enclosing each of the alternate selection criteria expressions in parentheses and then joining them with a separating logical OR ("|") symbol.

For details, see:

◊ REPORT Definition section WHERE

SORT Temporary Data Sets

Support introduced for **SORT** temporary data set dynamic allocation SPACE attributes saved in a User's FileKit INI file. The following options may be specified with a default value in the SYSTEM section of the User INI file:

- ◊ **SORTSPACEUNIT** (Blocks, Cylinders or Tracks)
- ◊ **SORTSPACEPRI** (Primary allocation)
- ◊ **SORTSPACESEC** (Secondary allocation)

Values specified by these INI options will be used when allocating SORTIN and SORTOUT instead of the product default of "TRACKS(300,300)" For example,

```
SYSTEM:
 SORTSPACEUNIT=CYLINDERS
 SORTSPACEPRI=5
 SORTSPACESEC=1
```

To set values for these options in the user INI file, use the SET INIVAR primary command. For example,

```
SET INIVAR  SYSTEM.SORTSPACEUNIT=CYLINDERS
SET INIVAR  SYSTEM.SORTSPACEPRI=5
SET INIVAR  SYSTEM.SORTSPACESEC=1
```

Basic Reporting

This chapter provides a step-by-step guide to generating reports using simple report definition control statements.

Select Report Columns

The very simplest report only has column field selections in the report definition input. To do this, you only specify the following report section in the report definition member:

- **COLUMNS:**

The REPORT utility parameter input identifies the input data source and, unless the input source is a DB2 result table or contains SMF log records, a mapping structure to be used to format the input record data. For DB2 table and SMF record input, the input data is formatted dynamically using a derived mapping structure.

The formatted input data record is split into a number of fields each having a unique name. These field names may be used to identify a report column.

The following will demonstrate how to specify column field names to generate simple reports from formatted input records.

The COLUMNS: Section

The **COLUMNS:** section identifies the names of mapped input record fields to be included in the report output and also the order in which they are presented.

A formatted input data field name may be specified on a new statement in the COLUMNS: section to define the next column to be displayed in the output report.

Note that the first column definition statement may appear on the same line as the **COLUMNS:** section header. Furthermore, statement separation character (default ";") may be used to join the column statements onto a single line.

For each input record, the contents of the field and the field data type combine to assign a specific value to the field name. When a report output detail line is written, the value currently assigned to the field name will appear as the next entry in the defined column.

For Example, the following COLUMNS: section defines 4 columns which will contain values from mapped fields **CODE**, **COUNTRY**, **TRACK** and **LAP-LENGTH-KM**. The order in which the columns are defined identifies the specific sequence in which they occur in the output report.

```
COLUMNS:   CODE;  COUNTRY;  TRACK;  LAP-LENGTH-KM
```

Because each of the column definitions include only the field name and no other parameters, default values will be used for the column header, column width and column data alignment. These defaults are determined by the characteristics of the corresponding input field.

Examples

The following examples demonstrate generating appealing reports using just a basic selection of columns. The reports each contain:

- A standard page header which includes a timestamp and printed report page number.
- Columns of data in a sequence corresponding to the order in which the columns are defined.
- Default, underlined column headers corresponding to the column's field name or field comment text (if included in the copybook or FileKit SDO structure field definition). Note that a column header may span a number of report lines.
- Report detail lines containing formatted field values. In particular values corresponding to date, time and numeric field definitions.
- A **Grand Totals** line displaying the accumulated total for all values in numeric data type columns.
- The Grand Total line also contains the total number of items (report detail lines) created by the report.

Select Report Columns - Example 1.

This example uses the sample Formula 1 Drivers COBOL copy book (ZZSCF1DR) to format records from the input data set which contains details of Formula 1 drivers who competed in the 2019 championship.

COBOL Copy Book - **ZZS.ZZSSAM1(ZZSCF1DR)**:

```
01 F1-Driver.
   05 NUMBER                    PIC  99  COMP-4.
   05 NAME                      PIC  X(20).
   05 COUNTRY                   PIC  X(20).
   05 BIRTH-PLACE               PIC  X(20).
   05 DATE-OF-BIRTH             PIC  9999/99/99.
   05 FIRST-RACE                PIC  9999/99/99.
   05 FIRST-RACE-CIRCUIT        PIC  X(3).
```

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF0D1)**:

The definition includes only a COLUMNS: section which simply selects the columns to report in the sequence in which they are to appear in the printed report. Default column headers, column widths and column alignments will be used.

```
COLUMNS:
  NAME
  COUNTRY
  NUMBER
  BIRTH-PLACE
  DATE-OF-BIRTH
  FIRST-RACE
  FIRST-RACE-CIRCUIT
```

REPORT Utility Execution:

Using the FileKit Formatted Record Report panel, we enter the names of the report definition, input data file and record mapping library member (type COBOL). A run type "**F**" is selected to execute the REPORT utility in the foreground, but "**B**" or "**C**" could be used to generate the equivalent REPORT utility batch job or primary command respectively.

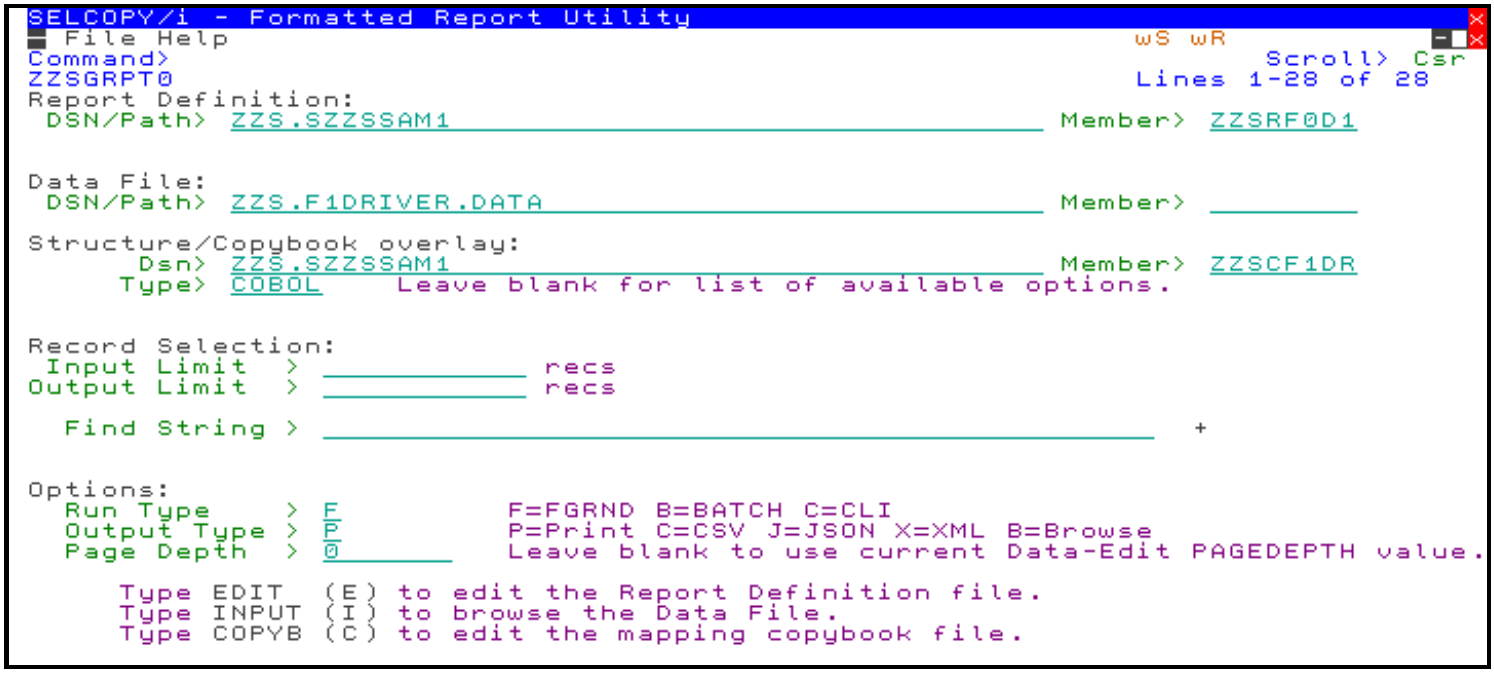


Figure 1. Generate Formula 1 Drivers printed report.

Report Output:

Column headings default to be the corresponding field names. A hyphon or minus symbol ("-") in the field name forces a break in the column header text.

```
 12020/04/23 16:45                                                                                           PAGE    1

                                                                          DATE                 FIRST
                                                     BIRTH                OF         FIRST     RACE
  NAME                 COUNTRY              NUMBER PLACE                BIRTH      RACE      CIRCUIT
  -------------------- -------------------- ------ -------------------- ---------- ---------- -------
  Daniel Ricciardo     Australia                 3 Perth                1989/07/01 2011/07/10 BRI
  Lando Norris         United Kingdom            4 Bristol              1999/10/13 2019/03/17 AUS
  Sebastian Vettel     Germany                   5 Heppenheim           1987/07/03 2007/06/17 USA
  Kimi Raikkonen       Finland                   7 Espoo                1979/10/17 2001/03/04 AUS
  Romain Grosjean      France                    8 Geneva               1986/04/17 2009/08/23 BEL
  Pierre Gasly         France                   10 Rouen                1996/02/07 2017/10/01 RUS
  Sergio Perez         Mexico                   11 Guadalajara          1990/01/26 2011/03/27 AUS
  Charles Leclerc      Monaco                   16 Monte Carlo          1997/10/16 2018/03/25 AUS
  Lance Stroll         Canada                   18 Montreal             1998/10/29 2017/03/26 AUS
  Kevin Magnussen      Denmark                  20 Roskilde             1992/10/05 2014/03/16 AUS
  Alexander Albon      Thailand                 23 London               1996/03/23 2019/03/17 AUS
  Daniil Kvyat         Russian Federation       26 Oefa                 1994/04/26 2014/03/16 AUS
  Nico Hulkenberg      Germany                  27 Emmerich am Rhein    1987/08/19 2010/03/14 BAH
  Max Verstappen       Netherlands              33 Hasselt              1997/10/30 2015/03/15 AUS
  Lewis Hamilton       United Kingdom           44 Stevenage            1985/01/07 2007/03/18 AUS
  Carlos Sainz Jr.     Spain                    55 Madrid               1994/09/01 2015/03/15 AUS
  George Russell       United Kingdom           63 Kings Lynn           1998/02/15 2019/03/17 AUS
  Valtteri Bottas      Finland                  77 Nastola              1989/08/28 2013/03/17 AUS
  Robert Kubica        Poland                   88 Krakau               1984/12/07 2006/08/06 HUN
  Antonio Giovinazzi   Italy                    99 Martina Franca       1993/12/14 2017/03/26 AUS
                                            ======
  == Grand Totals (20 Items)                   637
                                            ======
```

Select Report Columns - Example 2.

This example generates a report from SMF log records. Only SMF record type 119 (TCP/IP Statistics) sub-type 2 (TCP Connection Termination) records are processed. All other SMF record types/sub-types are bypassed.

The required SMF type/sub-types are determined by the REPORT utility based upon the record-type name qualifiers specified before each field name in the column definition section. In this example, each of the required field names are found in the **SMF119#02_TCP_Connection_Termination** record-type definition which is found in the **T119ST02** (SMF record type 119, sub-type 2) SDO structure.

The REPORT utility will use the standard FileKit SMF SDO structures to format the input SMF records. (See the "*FileKit SMF Utilities*" publication for details on SMF record segment mappings and field names.)

Report Definition Input - **ZZS.SZZSSAM1(ZZSRS001)**:

The definition includes only a COLUMNS: section which selects the columns to report in the sequence in which they are to appear in the printed report. Each field name specification must be prefixed by the record-type structure in which the field is defined. This is so the REPORT utility can I identify those SMF record types that it needs to process.

```
COLUMNS:
  SMF119#02_TCP_Connection_Termination.zRName
  SMF119#02_TCP_Connection_Termination.zConnectStart
  SMF119#02_TCP_Connection_Termination.zConnectEnd
  SMF119#02_TCP_Connection_Termination.zInBytes
  SMF119#02_TCP_Connection_Termination.zOutBytes
  SMF119#02_TCP_Connection_Termination.zTermCode
```

REPORT Utility Execution:

Using the FileKit SMF Report panel, we enter the names of the report definition library member and input data file (GDG relative generation 0). An Output Limit of **20** is specified to restrict the size of the printed report. The format of the input is "**OFFLINE**" to indicate that the SMF records are **not** being read directly from an SMF log data set. A run type "**F**" is selected to execute the REPORT utility in the foreground.

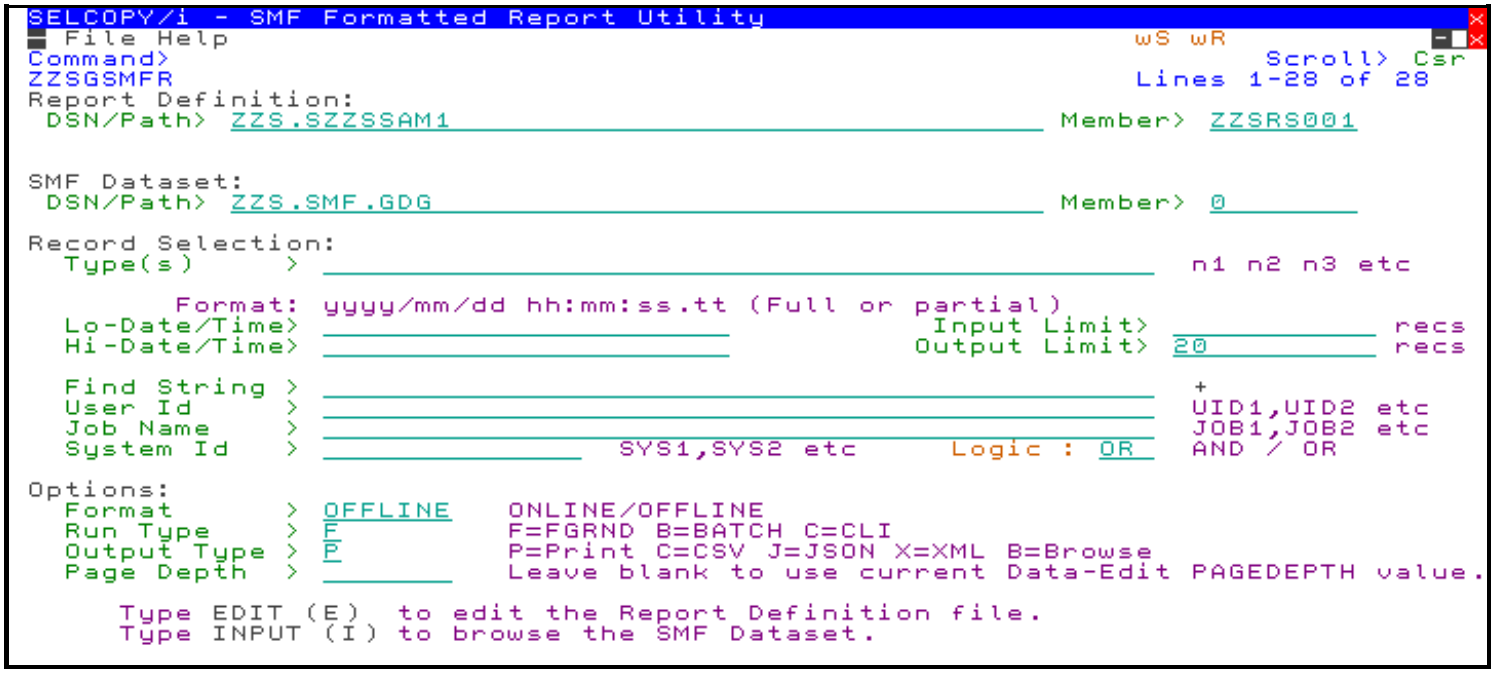


Figure 2. Generate simple SMF TCP/IP Statistics - TCP Terminations printed report.

Report Output:

Default column headings are derived from field comment text defined in the FileKit SMF structure definition.

12020/04/21 17:22 PAGE 1

TCP socket resource name :zRNAME	Connection start date & time :zCONNECTSTART	Connection end date & time :zCONNECTEND	Inbound byte count :zINBYTES	Outbound byte count :zOUTBYTES	Connection Termination reason :zTERMCODE
TN3270	2019/05/07 08:40:37.60	2019/05/07 08:40:39.99	31	1439	RESET_Received
JGE	2019/05/07 09:02:35.91	2019/05/07 09:02:36.14	21565	0	App_Close
FTPD1	2019/05/07 09:02:35.11	2019/05/07 09:02:36.37	125	485	App_Close
RXSERVE	2019/05/07 09:02:36.56	2019/05/07 09:02:41.86	145	42	App_Close
RXSERVE	2019/05/07 09:02:37.26	2019/05/07 09:02:41.97	0	0	App_Close
JGE	2019/05/07 09:04:07.80	2019/05/07 09:04:07.98	21594	0	App_Close
FTPD1	2019/05/07 09:04:07.26	2019/05/07 09:04:08.18	123	483	App_Close
RXSERVE	2019/05/07 09:04:08.40	2019/05/07 09:04:11.16	143	40	App_Close
RXSERVE	2019/05/07 09:04:08.53	2019/05/07 09:04:11.28	0	0	App_Close
TN3270	2019/05/07 10:19:04.72	2019/05/07 10:32:39.26	57	1439	RESET_Received
TN3270	2019/05/07 08:40:39.99	2019/05/07 13:28:14.42	7444	952807	No_FIN
TN3270	2019/05/07 10:32:39.27	2019/05/07 13:54:56.90	25214	4368142	RESET_Received
JGE	2019/05/07 15:23:29.06	2019/05/07 15:23:29.34	38614	0	App_Close
FTPD1	2019/05/07 15:23:28.16	2019/05/07 15:23:29.54	122	483	App_Close
RXSERVE	2019/05/07 15:23:29.77	2019/05/07 15:23:56.37	143	40	App_Close
RXSERVE	2019/05/07 15:23:30.36	2019/05/07 15:23:56.49	0	0	App_Close
JGE	2019/05/07 16:22:16.25	2019/05/07 16:22:16.47	0	2598	App_Close
FTPD1	2019/05/07 16:22:15.49	2019/05/07 16:22:16.49	120	465	App_Close
JGE	2019/05/07 16:22:17.98	2019/05/07 16:22:18.17	0	13368	App_Close
FTPD1	2019/05/07 16:22:17.55	2019/05/07 16:22:18.18	117	474	App_Close
== Grand Totals (20 Items)			115557	5342305	

Change Column Display

Although specifying only field names in your column definitions will generate an attractive report using default headers and data alignment, you may wish to tailor the appearence of the column output.

The COLUMNS: section definition statements provide the ability to control column data width and alignment as well as header text and header text alignment. Spacing between columns may be tailored and, if desired, use of character literal constants and REPORT utility built-in field values is also supported.

The following demonstrates how COLUMNS: section column definition statements may be enhanced to change the default column display in a printed report output.

Column Data

Every column defined in the report output is assigned a data **width** and data **alignment** which applies to all values reported in that column. The data width defines the maximum display width of the value and the data alignment is the positioning of the values within the column area. (Values may be centralised or left or right adjusted.)

By default, the column data **width** is the number of characters required to display **any** value that may be represented by the column's source field data type.

For character data type fields, this is the defined (fixed or variable maximum) field length. Fields of data type DATE, TIME or TIMESTAMP have fixed, default column data widths that are determined by the the sub-type value. For example, TIME(DECIMAL) has column data width 8 (e.g. 14:12:26) whereas TIME(STCK) contains an additional elapsed hour digit plus a number of microseconds and has a column data width of 16 (e.g. 014:12:26.17629).

For fields of numeric data type, the width depends on the type of numeric data, whether the values are signed or unsigned and whether or not the value contains a decimal point. For example, a signed numeric field of data type DECIMAL with precision 5 and scale 2 may represent a value that occupies a maximum of 7 characters in the display (e.g. -999.99). Therefore, the column data width would be 7 by default.

To override the default column data width, simply specify the number of characters width following the column definition field name (or column header value, if specified). The column field value will be truncated or padded to this length. For example, suppose field name ARTIST has a CHAR data type of length 70 and we want to display only the first 31 characters, then we would use the following:

```
COLUMNS:
  ARTIST    31             /* Artist name truncated to 31 characters. */
```

Note that the actual width of the printed report column text is the larger of the column data width value and the column header width. Therefore, if the ARTIST field was assigned a header of width larger than the data width (31), then all the column values would be padded with blanks to the width of the column header. (See Column Headers below.)

Column data **alignment** defaults to right adjusted for all numeric data types and data type TIME. Otherwise, for all other data types, column values are left adjusted.

To override the default column data alignment, specify the required alignment (LEFT, RIGHT, CENTER or CENTRE) following the column definition field name (or column header value, if specified).

In the following example, the NUMBER field has an unsigned numeric data type of INTEGER(2) and so values are right adjusted in the column area with a data width of 5 (the widest output value being 65535). Supposing we know that the field contains only values 0-99 (i.e. at most 2-digits), then we could restrict the data width to 2 and centralise the values within the column area.

```
COLUMNS:
  NUMBER  2  CENTRE        /* 2-digit numbers centalised under header "NUMBER".  */
```

Column Constants and Gaps

A column may be defined as having a value that remains constant for each output report detail line. This may be useful when producing a form where report detail lines are split over several lines of the printed output or if a character other than blank is to be used to separate the columns.

For example, the following inserts a vertical bar ("|") before and after each of the three columns (CODE, COUNTRY and TRACK) in the report detail line.

```
COLUMNS:  '|';  CODE;  '|';  COUNTRY;  '|';  TRACK;  '|'
```

Similarly, the amount of spacing between two columns may be controlled by specifying a gap value (a number of spaces) between column definitions in the COLUMNS: section. The default gap value is 1, indicating that a single space will be used to separate report columns. The gap value may be set to 0 (zero) if desired, so that the contents of two columns are juxtuposed.

Expanding on the last example, the gap values of "0" remove spacing between the vertical bar and the start of the column value that follows and gap values of "2" add an additional space following the column value and the next vertical bar.

```
COLUMNS:  '|';  0;  CODE;  2;  '|';  0;  COUNTRY;  2;  '|';  0;  TRACK;  2;  '|'
```

Note that, like column definitions based on field names, constant value column definitions and gap values must be specified on a separate control statement. In the above examples, the statement separator symbol semi-colon (";") is used to specify multiple report control statements on the same input record.

Column Headings

Unless **COLHEAD(NO)** has been specified in the OPTIONS: section of the report definition, the printed output will include column headers before the first line of column data on each page of the report. These headers may occupy one or more lines of the page and are underlined by a single line containing hyphon/minus symbols ("-").

The default column header text is the field name or constant literal specified in the column definition which is **left** adjusted in the column display area. An exception to this occurs when report option SHORTHEADERS(NO) is set (the default setting) **and** remark/comment text exists for the field definition in the FileKit SDO structure used to map the input record data. In this case, the default column header is generated from the comment text. Note that default column header generated from comment text is usual for SMF record reports.

To override the default header, simply specify the preferred header text as a character string literal following the column definition field name and before any data width and alignment values. For example, the following will override the default column header "ARTIST" with "Performing Vocalist or Group Name". The column data has a display width of 31 and will be right adjusted:

```
COLUMNS:
  ARTIST   "Performing Vocalist or Group Name"    31  RIGHT
```

To override the default alignment of left adjusted, the header text character string literal and header alignment specification must be placed in parentheses "( )". This is to distinguish the header alignment from the data alignment that may follow. Using the previous example, the header text is centralised and the column data values are right adjusted:

```
COLUMNS:
  ARTIST  ("Performing Vocalist or Group Name"   CENTRE)     31  RIGHT
```

The header text will appear on a single line of the report and the column header width will be the length of the header text. The width of the column display will be the larger of the column data width and the header width. In our example, the header width is 33 and the data width is 31, so the column will have an overall width of 33.

To reduce the header width, the header text may be split into a number of header text elements using the column header break symbol, vertical bar ("|"). Each header text element will appear on a new line of the column area within the report and aligned using the specified or default header alignment. The header width will then become the width of the longest header text element. Updating our example:

```
COLUMNS:
  ARTIST  ("Performing Vocalist|or Group Name"   CENTRE)     31  RIGHT
```

The header text is split into 2 header text elements of length 19 and 13, so the new header width is 19 and the overall column width becomes 31 (the column data width). See Example 2. below for sample output using this column definition.

Note that header text may be suppressed altogether by specifying a null header string literal enclosed within parentheses, i.e. a header specification of `("")`. No header or header underline will be generated.

If the header break symbol is to appear as text within the header and not treated as a header break, then it must be escaped by entering 2 adjacent vertical bar symbols (e.g. "Header || Text"). However, if the header is just a single vertical bar, then it will automatically be treated as header text and does not need to be escaped.

Examples

The following examples demonstrate how the display of column in the standard report format may be changed. The reports demonstrate:

- Use of column data width to pad or truncate column values.
- Specification of a column data alignment to override the default alignment based on the column source field data type.
- Use of gap values to provide extra spacing between columns and also to remove spacing between columns.
- Constant column values that are repeated for each detail line of the report output.
- Specification of column headers and header alignment to override the defaults. This includes use of a null header specification to suppress the underlined header.

Change Column Data Display - Example 1.

This example uses the same sample Formula 1 Drivers COBOL copy book (ZZSCF1DR) described in Select Report Columns - Example 1.

Use the same FileKit **Formatted Record Report** panel input as specified in **Select Report Columns - Example 1** but change the report definition member name to be **ZZSRF0D2**.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF0D2)**:

The definition includes the COLUMNS: section with a reduced selection of column fields. The selected columns are each assigned a column data width which is greater than the original COBOL copy book field width and so field values will be padded with blank characters to occupy this width. Column data alignment values are specified to determine how the field values are to be aligned in the column display.

```
COLUMNS:
  NAME              "Full Name"               25
 "|"                ("")                          /* Literal constant with null header. */
  BIRTH-PLACE       ("Birth Place"   RIGHT)     RIGHT 22
  0;   ",";  0                                    /* Null column spacing.  */
  COUNTRY            "Country"
 "|"                ("")
  DATE-OF-BIRTH     ("Date of Birth" CENTRE)  16   CENTRE
```

Report Output:

```
12020/05/01 12:16                                                                PAGE    1

 Full Name                          Birth Place,Country                    Date of Birth
 -------------------------    ------------------------------------------   ----------------
 Daniel Ricciardo           |                Perth,Australia             |    1989/07/01
 Lando Norris               |              Bristol,United Kingdom        |    1999/10/13
 Sebastian Vettel           |           Heppenheim,Germany               |    1987/07/03
 Kimi Raikkonen             |                Espoo,Finland               |    1979/10/17
 Romain Grosjean            |               Geneva,France                |    1986/04/17
 Pierre Gasly               |                Rouen,France                |    1996/02/07
 Sergio Perez               |            Guadalajara,Mexico              |    1990/01/26
 Charles Leclerc            |            Monte Carlo,Monaco              |    1997/10/16
 Lance Stroll               |              Montreal,Canada               |    1998/10/29
 Kevin Magnussen            |              Roskilde,Denmark              |    1992/10/05
 Alexander Albon            |                London,Thailand             |    1996/03/23
 Daniil Kvyat               |                  Oefa,Russian Federation   |    1994/04/26
 Nico Hulkenberg            |     Emmerich am Rhein,Germany              |    1987/08/19
 Max Verstappen             |              Hasselt,Netherlands           |    1997/10/30
 Lewis Hamilton             |           Stevenage,United Kingdom         |    1985/01/07
 Carlos Sainz Jr.           |              Madrid,Spain                  |    1994/09/01
 George Russell             |           Kings Lynn,United Kingdom        |    1998/02/15
 Valtteri Bottas            |              Nastola,Finland               |    1989/08/28
 Robert Kubica              |               Krakau,Poland                |    1984/12/07
 Antonio Giovinazzi         |         Martina Franca,Italy               |    1993/12/14

 == Grand Totals (20 Items)
```

The 0 (zero) gap value joins the BIRTH-PLACE, constant literal "," and COUNTRY columns so they appear as one column. The input field columns' values are right and left adjusted respectively, as are the cpecifeid column headers.

The NAME and DATE-OF-BIRTH columns are asigned new column headers. Thet are also assigned data widths which exceed the original defined field widths and so values are padded with blank characters.

The vertical bar symbol is specified as a constant literal twice, each occurrence with a null (suppressed) column header. This results in a more tabular style report.

Change Column Data Display - Example 2.

This example uses the sample Album Tracks COBOL copy book (ZZST1CPC) to format records from the input music collection data set.

COBOL Copy Book - **ZZS.ZZSSAM1(ZZST1CPC)**:

```
01 TRACK                                .
   05 PERSISTENT-ID             PIC  X(016).
   05 TRACK-NUM                 PIC  9(003).
   05 TRACK-ID                  PIC  9(004).
   05 NAME                      PIC  X(120).
   05 ARTIST                    PIC  X(070).
   05 ALBUM                     PIC  X(070).
   05 TOTAL-TIME                PIC  9(007) BINARY.
   05 FILE-SIZE                 PIC  9(009) BINARY.
   05 BIT-RATE                  PIC  9(004) BINARY.
   05 SAMPLE-RATE               PIC  9(005) PACKED-DECIMAL.
   05 YEAR                      PIC  9(004).
   05 NORMALIZATION             PIC S9(005) PACKED-DECIMAL.
   05 DISC-NUMBER               PIC  9(003).
   05 ALBUM-ARTIST              PIC  X(041).
   05 RELEASE-DATE              PIC  X(020).
   05 DATE-ADDED                PIC  X(020).
   05 DATE-MODIFIED             PIC  X(020).
```

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF001)**:

The definition includes only a COLUMNS: section. Column data widths and alignments are specified to truncate the input field values and to adjust the values in the column display areas.

```
COLUMNS:
  RELEASE-DATE  "Release|Date"                                10
  3
  ARTIST       ("Performing Vocalist|or Group Name" CENTRE)  31  RIGHT
  TRACK-NUM     "Track#"                                      2  LEFT
  NAME         ("Album|Track Name"                  CENTRE)  34
  3
  TOTAL-TIME   ("Track|Duration|(1/1000 sec)"       RIGHT )   6
```

REPORT Utility Execution:

Using the FileKit Formatted Record Report panel, we enter the names of the report definition, input data file and record mapping library member (type COBOL).

The number of output report detail lines is restricted to 20 and a FIND string of `C'I '` is used to give us just a sample selection of album tracks. The FIND search string will disregard any input record that does **not** contain an upper case "`I`" followed by a blank **anywhere** within the input record data.

```
SELCOPY/i - Formatted Report Utility
 File Help                                                    wS wR
Command>                                                          Scroll> Csr
ZZSGRPT0                                                    Lines 1-28 of 28
Report Definition:
 DSN/Path> ZZS.SZZSSAM1                                  Member> ZZSRF001

Data File:
 DSN/Path> NBJ.SELCTRN.ZZST1DAT                          Member>

Structure/Copybook overlay:
      Dsn> ZZS.SZZSSAM1                                  Member> ZZST1CPC
     Type> COBOL     Leave blank for list of available options.

Record Selection:
 Input Limit   >              recs
Output Limit   > 20           recs

  Find String > C'I '                                                  +

Options:
  Run Type     > F          F=FGRND B=BATCH C=CLI
  Output Type  > P          P=Print C=CSV J=JSON X=XML B=Browse
  Page Depth   > 0          Leave blank to use current Data-Edit PAGEDEPTH value.

     Type EDIT   (E) to edit the Report Definition file.
     Type INPUT  (I) to browse the Data File.
     Type COPYB  (C) to edit the mapping copybook file.
```

Figure 4. Generate Album Track printed report.

Report Output:

An additional 2 spaces are inserted between the RELEASE-DATE and ARTIST columns and also between the NAME and TOTAL-TIME columns.

Column data widths are provided which are smaller than the field definition width values and so that non-significant blanks in the field values get truncated. Data alignments are specified for ARTIST (RIGHT) and TRACK-NUM (LEFT) to override the fields' data type defaults of LEFT and RIGHT respectively.

New column headers with header break symbols are assigned to all but one of the columns. These column headers appear on more than one line of the column header area.

```
12020/05/01 12:54                                                                PAGE    1

                                                                                    Track
Release            Performing Vocalist                          Album               Duration
Date                  or Group Name          Track#            Track Name          (1/1000 sec)
----------   ------------------------------ ------ ---------------------------------- ------------
2011-01-21                          Adele 12     I Found a Boy (Bonus Track)              217338
2012-04-06                 Alabama Shakes 2      I Found You                              179653
2012-04-06                 Alabama Shakes 10     I Ain't the Same                         175800
1995-06-13              Alanis Morissette 1      All I Really Want                        284640
                              Alex Harvey 5      The Poet and I                           260360
                              Alex Harvey 9      The Poet and I (Reprise) (Reprise)        88613
                                    AC/DC 6      Can I Sit Next To You Girl               252000
2000-10-31                    Bob Dylan 11     I Want You                               183680
2000-10-31                    Bob Dylan 18     I Shall Be Released                      181826
1993-04-07             Bruce Springsteen 8      I Wish I Were Blind                      312973
1993-04-07             Bruce Springsteen 11     If I Should Fall Behind                  284533
2006-09-29             Bruce Springsteen 15     How Can I Keep from Singing              138933
2007-05-31   Springsteen & The Sessions Band 8      If I Should Fall Behind                  313266
1998-09-29   Burt Bacharach & Elvis Costello 3      I Still Have That Other Girl             166133
1993-01-01           Crash Test Dummies 6      Here I Stand Before Me                   186640
1993-01-01           Crash Test Dummies 7      I Think I'll Disappear Now               292133
1993-01-01           Crash Test Dummies 9      When I Go Out With Artists               221306
1989-08-09                   Del Amitri 6      When I Want You                          275066
1998-07-28                      Embrace 8      I Want The World                         344605
2010-01-18                Gretchen Wilson 4      When I Think About Cheatin'              248868
                                          ======                                     ============
== Grand Totals (20 Items)                159                                        4608366
                                          ======                                     ============
```

Create New Fields

Sometimes it may be necessary to report (and/or SORT) on field values that don't exist in the input data but may be derived from the input field values.

For example, the input data may contain a start and end time value but you want to report the elapsed time, or the data may have fields containing an employee's first name and last name and you want to report a single value comprising the first name initial and last name.

To do this, you will need to create your own field names for subsequent reference in the COLUMNS: section (or any section where the new field value is required). These types of fields are called computed fields. The names of these fields and the logic used to update the field's value are established in the following report section within the report definition member:

- **`COMPUTE:`**

The following will demonstrate how to use the COMPUTE: section to create and maintain computed fields and display the field values in the generated report.

The COMPUTE: Section

The **COMPUTE:** section contains a fragment of REXX logic which is terminated by the next section header or end of report definition input. The REXX includes one or more expressions and/or conditional logic that assigns a value to one or more REXX variables.

A computed field name is simply the name of a variable specified in the REXX logic and the value assigned to the variable at the end of the REXX logic execution will be the value of the computed field. Since the COMPUTE section executes REXX code, any command of function supported by TSO/E REXX may be used. (See IBM publication *"z/OS TSO/E REXX Reference"* for details.)

For Example, the following COMPUTE: section defines a computed field **SHORTNAME** whose value is derived from two input field names **FIRSTNAME** and **LASTNAME**.

```
COMPUTE:
  SHORTNAME = left(FIRSTNAME,1) || "." || LASTNAME  /* e.g. "JOHN" and "DOE" becomes "J.DOE" */
```

The COMPUTE: section REXX routine may establish the name and values of any number of computed field names and is executed every time a report detail line is about to be written to the report output. Therefore, the values of computed fields will alway reflect the latest values of the input fields on which they are based.

When referenced in other sections of the report definition, a computed field name is always prefixed by a colon symbol (":") to distinguish it from other types of report field.

Notes:

- If the input field on which a computed field is based is **not** included in the report detail line output (i.e. identified in the COLUMNS: section as a column definition field), then the input field must be defined in the **REQUIRED:** section.
- If an input field name includes a hyphon/minus symbol ("-") (valid for COBOL data description names), then, in order to conform with REXX standards, the REPORT utility re-assigns the field values to fields of the same name but with the "-" symbols translated to underscore symbols ("_"). Therefore, reference to any of these fields in the the COMPUTE: section must use the underscore version of the field name. For example, input field name "LAP-TIME" must be referred to as "LAP_TIME".
- Computed field names have a default display width of 9 characters. Therefore, it is likely necessary to have to specify a column data display width if the computed field is referenced as a column definition field in the COLUMNS: section.

Examples

The following examples demonstrate creation of new (computed) fields whose values are based on other input data fields. The report output is in the same default format as shown for Select Report Columns examples, but includes columns for the computed field values.

Create New Fields - Example 1.

This example uses the same sample Formula 1 Drivers COBOL copy book (F1DRIVER) described in Select Report Columns - Example 1.

Use the same FileKit **Formatted Record Report** panel input as specified in **Select Report Columns - Example 1** but change the report definition member name to be **ZZSRF0D5**.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF0D5)**:

The definition includes a REQUIRED: section to obtain values for input data fields that are not included as columns in the output report but are required for computed field values defined in the COMPUTE: section.

The COMPUTE: section establishes the values for computed fields "**AGE**" and "**HOME**" which are used to define output report columns in the COLUMNS: section. These computed field column definitions each have a data width specification to override the default of width 9 characters for computed field values. Computed field "**AGE**" is the driver's age on his first Formula 1 race, computed field "**HOME**" is the driver's home town and country combined into a single value.

```
COMPUTE:
 /* ### AGE - Computed Field - Age of driver at first race. ### */
    parse var DATE_OF_BIRTH    DYEAR '/' DMM '/' DDD
    parse var FIRST_RACE       FYEAR '/' FMM '/' FDD

    AGE = FYEAR-DYEAR                /* Age if birthday on or before race. */

  /* Adjust AGE if first race day-of-year is after driver's birth day-of-year. */
    if DMM > FMM  |  (DMM = FMM  &  DDD > FDD)
      then AGE = AGE-1

 /* ### HOME - Computed Field - Merge BIRTH-PLACE and COUNTRY. ### */
    HOME =  strip(BIRTH_PLACE) || ',' COUNTRY

REQUIRED:      /* Input fields not in report but used by COMPUTE: */
  COUNTRY
  BIRTH-PLACE

COLUMNS:
  NAME               "Full Name"
 :HOME               "Place of Birth"    30    /* Override default data width of 9. */
  NUMBER             "Number"
  DATE-OF-BIRTH      "DOB"
  FIRST-RACE         "First|Race|Date"
 :AGE                "First|Race|Age"     2    /* Use data width 2. */
  FIRST-RACE-CIRCUIT "First|Race|Circuit"
```

Report Output:

```
12020/05/01 14:59                                                                PAGE    1

                                                                    First      First First
                                                                    Race       Race  Race
Full Name           Place of Birth                  Number DOB      Date       Age   Circuit
------------------- ------------------------------ ------ ---------- ---------- ----- -------
Daniel Ricciardo    Perth, Australia                    3 1989/07/01 2011/07/10 22    BRI
Lando Norris        Bristol, United Kingdom             4 1999/10/13 2019/03/17 19    AUS
Sebastian Vettel    Heppenheim, Germany                 5 1987/07/03 2007/06/17 19    USA
Kimi Raikkonen      Espoo, Finland                      7 1979/10/17 2001/03/04 21    AUS
Romain Grosjean     Geneva, France                      8 1986/04/17 2009/08/23 23    BEL
Pierre Gasly        Rouen, France                      10 1996/02/07 2017/10/01 21    RUS
Sergio Perez        Guadalajara, Mexico                11 1990/01/26 2011/03/27 21    AUS
Charles Leclerc     Monte Carlo, Monaco                16 1997/10/16 2018/03/25 20    AUS
Lance Stroll        Montreal, Canada                   18 1998/10/29 2017/03/26 18    AUS
Kevin Magnussen     Roskilde, Denmark                  20 1992/10/05 2014/03/16 21    AUS
Alexander Albon     London, Thailand                   23 1996/03/23 2019/03/17 22    AUS
Daniil Kvyat        Oefa, Russian Federation           26 1994/04/26 2014/03/16 19    AUS
Nico Hulkenberg     Emmerich am Rhein, Germany         27 1987/08/19 2010/03/14 22    BAH
Max Verstappen      Hasselt, Netherlands               33 1997/10/30 2015/03/15 17    AUS
Lewis Hamilton      Stevenage, United Kingdom          44 1985/01/07 2007/03/18 22    AUS
Carlos Sainz Jr.    Madrid, Spain                      55 1994/09/01 2015/03/15 20    AUS
George Russell      Kings Lynn, United Kingdom         63 1998/02/15 2019/03/17 21    AUS
Valtteri Bottas     Nastola, Finland                   77 1989/08/28 2013/03/17 23    AUS
Robert Kubica       Krakau, Poland                     88 1984/12/07 2006/08/06 21    HUN
Antonio Giovinazzi  Martina Franca, Italy              99 1993/12/14 2017/03/26 23    AUS
                                                    ======
== Grand Totals (20 Items)                             637
                                                    ======
```

Create New Fields - Example 2.

This example is a variation of the report produced for SMF 119 TCP/IP Statistics, Connection Termination log records described in Select Report Columns - Example 2.

Use the same FileKit **SMF Report** panel input as specified in **Select Report Columns - Example 2** but change the report definition member name to be **SMF119A2**.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRS002)**:

The COMPUTE: section establishes the value for computed field "**DURATION**" which is used to define output report columns in the COLUMNS: section.

The computation uses REPORT utility built-in REXX functions **Time2Secs** and **Secs2Time** to convert between timestamp values and an integer number of seconds. Converting the end and start timestamps to seconds allows us to subtract one from the other to get a number of seconds elapsed. Conversion back to a timestamp format gives us the elapsed hours, minutes and seconds.

```
COMPUTE:
 /* ### DURATION - Computed Field - Time elapsed between connection start &amp end. */
  DURATION = Secs2Time( Time2Secs(zConnectEnd) - Time2Secs(zConnectStart) )

COLUMNS:
  SMF119#02_TCP_Connection_Termination.zRName          'Resource'
  SMF119#02_TCP_Connection_Termination.zConnectStart   'Connection Start'
  SMF119#02_TCP_Connection_Termination.zConnectEnd     'Connection End'

 :DURATION          ('Connection|Duration|HHH:MM:SS.SS' RIGHT) 12 RIGHT

  SMF119#02_TCP_Connection_Termination.zInBytes    8   'Inbound|Bytes'
  SMF119#02_TCP_Connection_Termination.zOutBytes   8   'Outbound|Bytes'
  SMF119#02_TCP_Connection_Termination.zTermCode       'Termination|Description'
```

Report Output:

```
12020/05/01 15:39                                                                PAGE    1

                                                                 Duration Inbound  Outbound Termination
Resource Connection Start       Connection End          HHH:MM:SS.SS Bytes    Bytes    Description
-------- ---------------------- ---------------------- ------------ -------- -------- --------------
TN3270   2019/05/07 08:40:37.60 2019/05/07 08:40:39.99  00:00:02.39       31     1439 RESET_Received
JGE      2019/05/07 09:02:35.91 2019/05/07 09:02:36.14  00:00:00.23    21565        0 App_Close
FTPD1    2019/05/07 09:02:35.11 2019/05/07 09:02:36.37  00:00:01.26      125      485 App_Close
RXSERVE  2019/05/07 09:02:36.56 2019/05/07 09:02:41.86  00:00:05.30      145       42 App_Close
RXSERVE  2019/05/07 09:02:37.26 2019/05/07 09:02:41.97  00:00:04.71        0        0 App_Close
JGE      2019/05/07 09:04:07.80 2019/05/07 09:04:07.98  00:00:00.18    21594        0 App_Close
FTPD1    2019/05/07 09:04:07.26 2019/05/07 09:04:08.18  00:00:00.92      123      483 App_Close
RXSERVE  2019/05/07 09:04:08.40 2019/05/07 09:04:11.16  00:00:02.76      143       40 App_Close
RXSERVE  2019/05/07 09:04:08.53 2019/05/07 09:04:11.28  00:00:02.75        0        0 App_Close
TN3270   2019/05/07 10:19:04.72 2019/05/07 10:32:39.26  00:13:34.54       57     1439 RESET_Received
TN3270   2019/05/07 08:40:39.99 2019/05/07 13:28:14.42  04:47:34.43     7444   952807 No_FIN
TN3270   2019/05/07 10:32:39.27 2019/05/07 13:54:56.90  03:22:17.63    25214  4368142 RESET_Received
JGE      2019/05/07 15:23:29.06 2019/05/07 15:23:29.34  00:00:00.28    38614        0 App_Close
FTPD1    2019/05/07 15:23:28.16 2019/05/07 15:23:29.54  00:00:01.38      122      483 App_Close
RXSERVE  2019/05/07 15:23:29.77 2019/05/07 15:23:56.37  00:00:26.60      143       40 App_Close
RXSERVE  2019/05/07 15:23:30.36 2019/05/07 15:23:56.49  00:00:26.13        0        0 App_Close
JGE      2019/05/07 16:22:16.25 2019/05/07 16:22:16.47  00:00:00.22        0     2598 App_Close
FTPD1    2019/05/07 16:22:15.49 2019/05/07 16:22:16.49  00:00:01.00      120      465 App_Close
JGE      2019/05/07 16:22:17.98 2019/05/07 16:22:18.17  00:00:00.19        0    13368 App_Close
FTPD1    2019/05/07 16:22:17.55 2019/05/07 16:22:18.18  00:00:00.63      117      474 App_Close
                                                                      ======== ========
== Grand Totals (20 Items)                                             115557  5342305
                                                                      ======== ========
```

Change Page Display

Page header and footer lines may be defined to provide a template for each page of the printed report output.

The standard printed page report template has only a single header line and no footer lines. The header line contains only the current timestamp (the date and time at which the report is generated) aligned at the left margin and the current report page number aligned at the right margin.

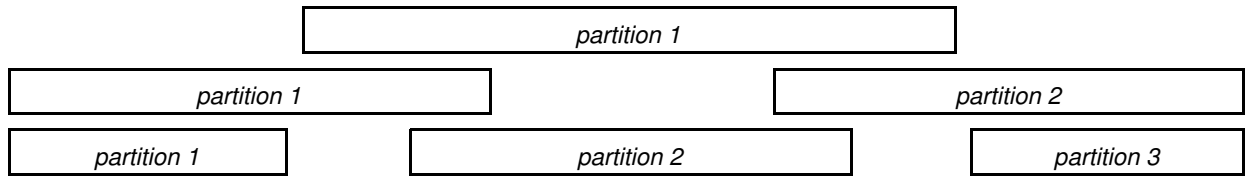
To create your own page template, you add entries to one or both of the following report sections within the report definition member:

- **HEAD:**
- **FOOT:**

The following will demonstrate how to specify entries in these sections to tailor the printed report page header and footer lines.

The HEAD: and FOOT: Sections

Each control statement belonging to a HEAD: or FOOT: section defines a single page header or footer line. Each of these lines may comprise 1, 2 or 3 **partitions** where each partition specification is separated from the next by a slash symbol ("/"). The number of partitions determine how each partition is aligned within the page width as follows:



A single partition will be centralised within the established page width. If 2 partitions are defined, the first will be left andjusted and the second right adjusted within the page width. If 3 partitions are defined, the first will be left andjusted, the second centralised and the third right adjusted. For example, the following page header definition contains 2 partitions:

```
HEAD:
  'Left Adjusted Text' / 'Right Adjusted Text'
```

Note that, if necessary, the REPORT utility may dynamically increase the page width in order to fit all specified header/footer line partitions with at least 2 intervening blanks onto the same line of the report output. For example, the following header defintion contains 3 partitions:

```
HEAD:
  'Left Adjusted Text (Length 30)' / 'Centralised Text (Length 28)'  /  'Right (Length 17)'
```

If the page width required to print the column detail lines is 80, then the page width will increase from 80 to 92 in order to display the centralised text plus 2 intervening blanks without overwriting the left adjusted text. The header output is as follows. Note the ASA character in column 1 denoting the start of a new page. A counting guide is included here to detail the report page width but is not part of the printed output.

```
    ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9..
   1Left Adjusted Text (Length 30)  Centralised Text (Length 28)              Right (Length 17)
```

Each partition is a string of text represented by a print expression. A print expression is comprised of one or more text fragment definitions and optional gap values. Each text fragment may be a character string literal or a value obtained from a named field as described next under Variable Substitution.

Variable Substitution

In addition to constant literals, a field name may be specified within the partition print expression to represent a fragment of text which has a variable value.

The field name may be the name of a field in a mapped input record, a computed variable name or the name of a REPORT utility built-in field. If a field name is used in a header or footer line print expression, then the value assigned to the field will substitute the field name as the text fragment.

Header lines are written to the report output after establishing values for the next column detail line to be written. Therefore, fields in header lines always reflect values obtained from the first detail line of the new page and not in the last detail line of the previous page.

For example, the following header line definition will substitute the input field value **ALBUM** with the album name obtained from the input record representing the first report detail line on the new page. Similarly, the built-in field **#PAGE** will be substituted with the page number of the new page.

```
HEAD:
  'Track List for Album:' ALBUM  /  'PAGE:' #PAGE
```

In contrast, footer lines always reflect values obtained from the last detail of the current page and not in the first detail line of the next page. For example, the following will output a footer line with the album name obtained from the input record from which the last detail line was constructed.

```
FOOT:
  'End of Album:'  ALBUM
```

Text Fragment Width, Alignment & Gaps

By default, text fragments occupy an area in the header or footer line equivalent to the **maximum** display width of the text it represents.

For example, a text fragment represented by an input field value of display length 60 will occupy 60 characters in the header text. Similarly, the field value will be positioned within the text fragment display using the default alignment for the field data type (i.e. right adjusted for numeric and time values, left adjusted for all other data types).

These defaults may be overridden by placing the required width and/or alignment in parentheses following the fragment specification. A width specification means that the field value will be truncated or padded with blanks on the left or right based on the value alignment. Whether or not a width is specified, blank padding may be suppressed by including the keyword "STRIP" in the parentheses.

In the following example, input fields ALBUM and ARTIST are used which both map values of character length 70. Both the ALBUM and ARTIST field values are truncated on the right to length 30 but the ARTIST value then has leading and trailing blanks stripped. Because of this, the length of the ARTIST value may vary for different artist names, as will the position of the text fragment ("###") that follows the artist name within the same partition. Also, since we know there will be no more than 999 report pages, the #PAGE numeric, right adjusted value is truncated on the left to be displayed as a 3 digit number.

```
HEAD:
  'Album:' ALBUM (LEFT,30)  'Artist: ###' ARTIST (STRIP,30) '###'   /   'PAGE:' #PAGE (RIGHT,3)
```

The header output is as follows. A counting guide is included but is not part of the printed output.

```
     ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,
    1Album: High Voltage                  Artist: ### AC/DC ###                            PAGE:   1
```

By default, a single space is inserted between each text fragment in a header or footer line partition. To override this, simply insert the required gap value (number of spaces) between the two fragment specifications in the print expression. A gap value may be 0 (zero) if two fragments are to be joined. In our example, we could insert a gap value of 4 before the ALBUM field name and a gap value of 0 either side of the ARTIST field name.

```
HEAD:
  'Album:' 4 ALBUM (LEFT,30)  'Artist: ###' 0 ARTIST (STRIP,30) 0  '###'   /   'PAGE:' #PAGE (RIGHT,3)
```

This would produce the following header output:

```
     ....,....1....,....2....,....3....,....4....,....5....,....6....,....7....,....8....,....9....,.
    1Album:    High Voltage                  Artist: ###AC/DC###                            PAGE:   1
```

Examples

The following examples show how the page header in the standard report format may be changed and how a footer may be added. The reports demonstrate:

- Use of the HEAD: and FOOT: report sections.
- Creating multiple page header and footer lines.
- Left adjusted, centralised and right adjusted page header and footer line partitions.
- Substitution of variable, input record field values for both header and footer lines.
- Use of gap values to provide extra spacing and also to remove spacing between partition text fragments.

Change Page Display - Example 1.

This example uses the same sample Formula 1 Drivers COBOL copy book (ZZSCF1DR) described in Select Report Columns - Example 1.

Use the same FileKit **Formatted Record Report** panel input as specified in **Select Report Columns - Example 1** but change the report definition member name to be **ZZSRF0D3**, the Output Limit to be **10** and the Page Depth value to be **18**.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF0D3)**:

The definition includes an OPTIONS: section and sets TOTALS(NO) to suppress automatic accumulation of a grand total for values in numeric columns. It includes a HEAD: section that defines 2 page header lines where the first line has 2 partitions and the second line only 1. It also includes a FOOT: section to define a single page footer line comprising 2 partitions.

Built-in fields #TODAY, #DAYNAME and #PAGE are used in the header and footer line partition print expressions to represent the current values for date, day-of-week and page number respectively.

A gap value of "0" (zero) is used in the first header line partition print expression between built-in field #DAYNAME and the character string literal comma (","). This ensures that the comma will appear immediately following the day-of-week with no intervening blank character.

```
OPTIONS:   TOTALS(NO)

HEAD:
  #DAYNAME 0 "," #TODAY     /    "Formula 1 Statistics"
                "2019 Season Drivers"

FOOT:
  "FIA Database"            /    "Page" #PAGE (RIGHT,3)

COLUMNS:
  1                      /* Insert 1 space before columns. */
  NAME                "Driver Name"
  BIRTH-PLACE         "Place of Birth"
  COUNTRY             "Country"
  NUMBER              "Driver#"
  DATE-OF-BIRTH       "DOB"
  FIRST-RACE          "First Race"
  FIRST-RACE-CIRCUIT "Circuit"
```

Report Output:

```
12020/05/05, Tuesday                                                Formula 1 Statistics
                                      2019 Season Drivers

  Driver Name          Place of Birth       Country              Driver# DOB        First Race Circuit
  -------------------- -------------------- -------------------- ------- ---------- ---------- -------
  Daniel Ricciardo     Perth                Australia                  3 1989/07/01 2011/07/10 BRI
  Lando Norris         Bristol              United Kingdom             4 1999/10/13 2019/03/17 AUS
  Sebastian Vettel     Heppenheim           Germany                    5 1987/07/03 2007/06/17 USA
  Kimi Raikkonen       Espoo                Finland                    7 1979/10/17 2001/03/04 AUS
  Romain Grosjean      Geneva               France                     8 1986/04/17 2009/08/23 BEL
  Pierre Gasly         Rouen                France                    10 1996/02/07 2017/10/01 RUS
  Sergio Perez         Guadalajara          Mexico                    11 1990/01/26 2011/03/27 AUS
  Charles Leclerc      Monte Carlo          Monaco                    16 1997/10/16 2018/03/25 AUS
  Lance Stroll         Montreal             Canada                    18 1998/10/29 2017/03/26 AUS
  Kevin Magnussen      Roskilde             Denmark                   20 1992/10/05 2014/03/16 AUS


 FIA Database                                                                        Page   1
```

Change Page Display - Example 2.

This example generates a report from SMF log records. Only SMF record type 30 (Common Address Space Work) records are processed to report job step totals. All other SMF record types are bypassed.

The required SMF type/sub-types are determined by the REPORT utility based upon the record-type name qualifiers specified before each field name in the column definition section. In this example, the required field names are found in the **SMF030_Identification**, **SMF030_Completion**, **SMF030_Processor_Accounting** and **SMF030_IO_Activity** record-type definitions which are found in the **T030** (SMF record type 30) SDO structure.

The REPORT utility will use the standard FileKit SMF SDO structures to format the input SMF records. (See the "*FileKit SMF Utilities*" publication for details on SMF record segment mappings and field names.)

Report Definition Input - **ZZS.SZZSSAM1(ZZSRS002)**:

The definition includes an OPTION section and sets TOTALS(NO) to suppress automatic accumulation of a grand total for values in numeric columns. It includes a HEAD: section that defines 3 page header lines where the first line has 2 partitions and the second and third lines only 1 partition. It also includes a FOOT: section to define a single page footer line of one partition.

Input field names zJOBNAME and zRST are used in both header and footer line partition print expressions and a gap value of "5" is used between the zJOBNAME field specification and the character string literal that follows.

```
OPTIONS: TOTALS(NO)

HEAD:
  #TIMESTMP / 'PAGE' #PAGE (RIGHT,3)
  "SMF Record Type 30-4 (Step Termination) Statistics"
  "First Jobname:" zJOBNAME  5 'Terminating at:' zRST

FOOT:
   "Last Jobname:" zJOBNAME  5 'Terminating at:' zRST

COLUMNS:
  SMF030_Identification.zRST         'Reader Timestamp'
  SMF030_Identification.zJOBNAME     'Job Name'
  SMF030_Identification.zSTN         'Step#'                    5 RIGHT
  SMF030_Identification.zPGM         'Program|Name'
  SMF030_Completion.zSCC            ('CC'                RIGHT) 3 RIGHT
  SMF030_Processor_Accounting.zCPT  ('CPU|Time'          RIGHT) 8 RIGHT
  SMF030_IO_Activity.zTEP           ('EXCPs'             RIGHT) 6 RIGHT
  SMF030_IO_Activity.zAIC           ('Connect|Time'      RIGHT) 9 RIGHT
  SMF030_IO_Activity.zAIW           ('Control|Unit Time' RIGHT) 9 RIGHT
  SMF030_IO_Activity.zAIS           ('I/O|Starts'        RIGHT) 6 RIGHT
```

REPORT Utility Execution:

```
SELCOPY/i - SMF Formatted Report Utility
 File Help                                                    wS wR
Command>                                                        Scroll> Csr
ZZSGSMFR                                                     Lines 1-28 of 28
Report Definition:
 DSN/Path> ZZS.SZZSSAM1                                  Member> ZZSRS002

SMF Dataset:
 DSN/Path> ZZS.SMF.GDG                                   Member> -3

Record Selection:
  Type(s)      > ____________________________________________________  n1 n2 n3 etc

       Format: yyyy/mm/dd hh:mm:ss.tt (Full or partial)
  Lo-Date/Time> ____________________  Input Limit> _____________ recs
  Hi-Date/Time> ____________________ Output Limit> 16____________ recs

  Find String > ____________________________________________________  +
  User Id     > ____________________________________________________  UID1,UID2 etc
  Job Name    > ____________________________________________________  JOB1,JOB2 etc
  System Id   > _____________ SYS1,SYS2 etc     Logic : OR          AND / OR

Options:
  Format      > OFFLINE    ONLINE/OFFLINE
  Run Type    > F          F=FGRND B=BATCH C=CLI
  Output Type > P          P=Print C=CSV J=JSON X=XML B=Browse
  Page Depth  > 25         Leave blank to use current Data-Edit PAGEDEPTH value.

    Type EDIT (E)  to edit the Report Definition file.
    Type INPUT (I) to browse the SMF Dataset.
```

Figure 6. Generate SMF Job Step Report - Page Headers and Footers.

In the FileKit SMF Report panel, we enter the names of the report definition library member and input data file (GDG relative generation -3).

An Output Limit of **16** and a Page Depth of **25** is used to restrict the size of the printed report. The format of the input is "**OFFLINE**" to indicate that the SMF records are **not** being read directly from an SMF log data set. A run type "**F**" is selected to execute the REPORT utility in the foreground.

Report Output:

The values substituted for the zJOBNAME and zRST field names have been obtained from the **first** report column detail line on the page for **header** lines and from the **last** report column detail line on the page for **footer** lines.

The gap value provides 5 spaces between the job name (length 8) and job termination time specifications in the third page header line and in the page footer line.

```
12020/05/06 11:50                                                                PAGE   1
                    SMF Record Type 30-4 (Step Termination) Statistics
               First Jobname: SMFCLEAR     Terminating at: 2019/02/11 23:25:00.89

                                       Program           CPU          Connect   Control     I/O
 Reader Timestamp       Job Name Step# Name      CC     Time  EXCPs       Time Unit Time Starts
 ---------------------- -------- ----- -------- --- -------- ------ ---------- ---------- ------
 2019/02/11 23:25:00.89 SMFCLEAR     1 IFASMFDP   0 00:02.78  15209 00.189952 00.057472  14612
 2019/02/12 01:07:43.36 SMFCLEAR     1 IFASMFDP   0 00:02.86  15209 00.175488 00.053888  14614
 2019/02/12 02:49:59.50 SMFCLEAR     1 IFASMFDP   0 00:02.87  15204 00.202112 00.055296  14610
 2019/02/12 04:31:35.90 SMFCLEAR     1 IFASMFDP   0 00:02.84  15206 00.209536 00.062976  14611
 2019/02/12 06:13:41.43 SMFCLEAR     1 IFASMFDP   0 00:02.45  15204 00.192512 00.054400  14611
 2019/02/12 07:55:07.04 SMFCLEAR     1 IFASMFDP   0 00:03.03  15201 00.191872 00.050944  14610
 2019/02/12 09:37:03.20 SMFCLEAR     1 IFASMFDP   0 00:02.61  15215 00.160256 00.053632  14614
 2019/02/12 10:08:01.62 JGESI        1 SDEAMAIN   0 00:01.28   1460 00.014592 00.002304    518
 2019/02/12 10:33:22.47 JGESI        1 SDEAMAIN   0 00:01.17   1449 00.012288 00.002432    513
 2019/02/12 10:34:15.24 JGESI        1 SDEAMAIN   0 00:01.16   1435 00.017536 00.004352    513
 2019/02/12 10:37:14.73 JGESI        1 SDEAMAIN   0 00:01.15   1436 00.014720 00.003072    514
 2019/02/12 10:39:07.86 JGESI        1 SDEAMAIN   0 00:01.14   1434 00.013824 00.001920    513
 2019/02/12 10:43:12.39 JGESI        1 SDEAMAIN   0 00:01.13   1435 00.014720 00.002816    513
 2019/02/12 10:44:52.76 JGESI        1 SDEAMAIN   0 00:22.31   2556 00.106240 00.015488   1633
 2019/02/12 11:11:37.68 SMFCLEAR     1 IFASMFDP   0 00:02.83  15240 00.283776 00.058496  14619
 2019/02/12 10:52:44.92 JGESI        1 SDEAMAIN   0 20:55.01  34268 02.467712 00.365568  33346

               Last Jobname: JGESI        Terminating at: 2019/02/12 10:52:44.92
```

Filter Input Records

By default, all records belonging to the input data set will be read sequentially and values in each record included in the report output.

There may be occasions when you want to exclude records because they contain values that are not to be included in the report. For example, you may want to report only on records that contain a particular value in a specific field.

The REPORT utility itself supports record filtering options that may be provided by the user when utility is started. See example 2. under "*Change Column Data Display*" where a search string value is supplied in the report panel Find> input field so that only records that contain the search string are included in the report.

Note that, the REPORT utility supports additional record filtering options for SMF record report generation based on values at fixed locations within the SMF record data. Specifically, the SMF record type, sub-type, timestamp, sub-system id, job name and user id.

Although these options provide an perfectly adequate set of possible conditions for report record selection, the following report section within the report definition member provides a potentially more flexible alternative:

- **`FILTER:`**

The following will demonstrate how to specify a filter clause to select and deselect input records for further processing.

The FILTER: Section

The FILTER: section header is followed by a filter clause which may span several records of the report definition member and is terminated by another report definition header or the end-of-file.

The filter clause supports one or more alternative expressions by which a record may be selected for report processing.

Unlike the FIND search string REPORT utility input option which will select the record if the string is found at **any** location within the record, a filter clause expression may test specific fields within the record. Furthermore, the tests need not be for equality as a number of operators other than "=" (equals) are supported.

For example, the following filter clause will include only records mapped by the "TRACK" record-type mapping where the track name contains the search string "Love" **and** artist name is either "Journey" or "U2".

```
FILTER:
  INCLUDE TRACK   WHERE (NAME << 'Love'  &  ARTIST IN ('Journey' 'U2') )
```

Adding this FILTER: section to the report definition in Change Column Display Example 2. would produce the following output.

```
12020/05/07 11:17                                                                PAGE    1

                                                                                    Track
 Release            Performing Vocalist                        Album              Duration
 Date                  or Group Name          Track#          Track Name          (1/1000 sec)
 ----------   ------------------------------ ------ ---------------------------- ------------
 2006-01-08                          Journey 2      Stone In Love                      265483
 1983-01-01                          Journey 2      Send Her My Love                   234546
 1998-04-21                          Journey 11     Send Her My Love (Live)            217640
 1998-04-21                          Journey 13     Stone In Love (Live)               278533
 1986-01-01                          Journey 5      Once You Love Somebody             280346
 1991-11-19                               U2 12     Love Is Blindness                  263272
 1988-11-04                               U2 9      Pride (In the Name of Love) [Live] 267000
 1988-11-04                               U2 11     Love Rescue Me                     384666
                                             ======                                ============
 == Grand Totals (8 Items)                   65                                      2191486
                                             ======                                ============
```

Important Note:

If a report definition includes a FILTER: section, then the filter clause will be used in place of any Find search string and, for SMF reports, any Record type, System Id, Job name and User id record selection criteria specified on the REPORT utility execution. Therefore, in the above example, the FIND string `C'I '` specified in the FileKit Formatted Record Report panel will be ignored.

Examples

The following examples demonstrate how a FILTER: section may be used to report only on input records that match the filter clause criteria.

Filter Input Records - Example 1.

This example generates a report from SMF log records. Only SMF record type 14 (INPUT/RDBACK Dataset) records are processed to report input dataset usage by job name. All other SMF record types are bypassed.

In this example, the required field names are found in the **SMF014_INPUT_or_RDBACK_Dataset**, **SMF014#2_SMS_Class** and **SMF014#3_Step_Info** record-type definitions which are found in the **T014** (SMF record type 14) SDO structure. (See the "*FileKit SMF Utilities*" publication for details on SMF record segment mappings and field names.)

Report Definition Input - **ZZS.ZZSSAM1(ZZSRS003)**:

The filter clause in section "FILTER:" will select an input record for processing if the DDname field (SMFTIOE5) does **not** begin with "`SYS0004`" **and** the allocated DSN begins with either "`CBL.CBLI`", "`JGE`" or "`NBJ`". Alternativley, it will select the input record if the management class (zMCN) field is "`CBLHSM`".

```
HEAD:
  #TIMESTMP / 'PAGE:' #PAGE (RIGHT,3)
  'Dataset Usage by Job Name'

COLUMNS:
  SMF014_INPUT_or_RDBACK_Dataset.zJOBNAME       'Job Name'
  SMF014_INPUT_or_RDBACK_Dataset.zRST           'Reader Timestamp'
  SMF014#3_Step_Info.zSPN                       'StepName'
  SMF014#3_Step_Info.zPGN                       'PGMName'
  SMF014_INPUT_or_RDBACK_Dataset.JFCB.DSN       'Dataset Name'  24
  SMF014_INPUT_or_RDBACK_Dataset.UCB.SMFEXCP(1) 'EXCPs'          6
  SMF014_INPUT_or_RDBACK_Dataset.JFCB.CRDT      'Created'
  SMF014#2_SMS_Class.zMCN                       'MGMTCLAS'
  SMF014#2_SMS_Class.zSCN                       'STORCLAS'

FILTER:
  (   SMFTIOE5 \>> 'SYS004'
   and   (  DSN >> 'CBL.CBLI'  or  DSN >> 'JGE'  or  DSN >> 'NBJ'  )
  )
 or
  ( SMF014#2_SMS_Class.zMCN = 'CBLHSM' )
```

REPORT Utility Execution:

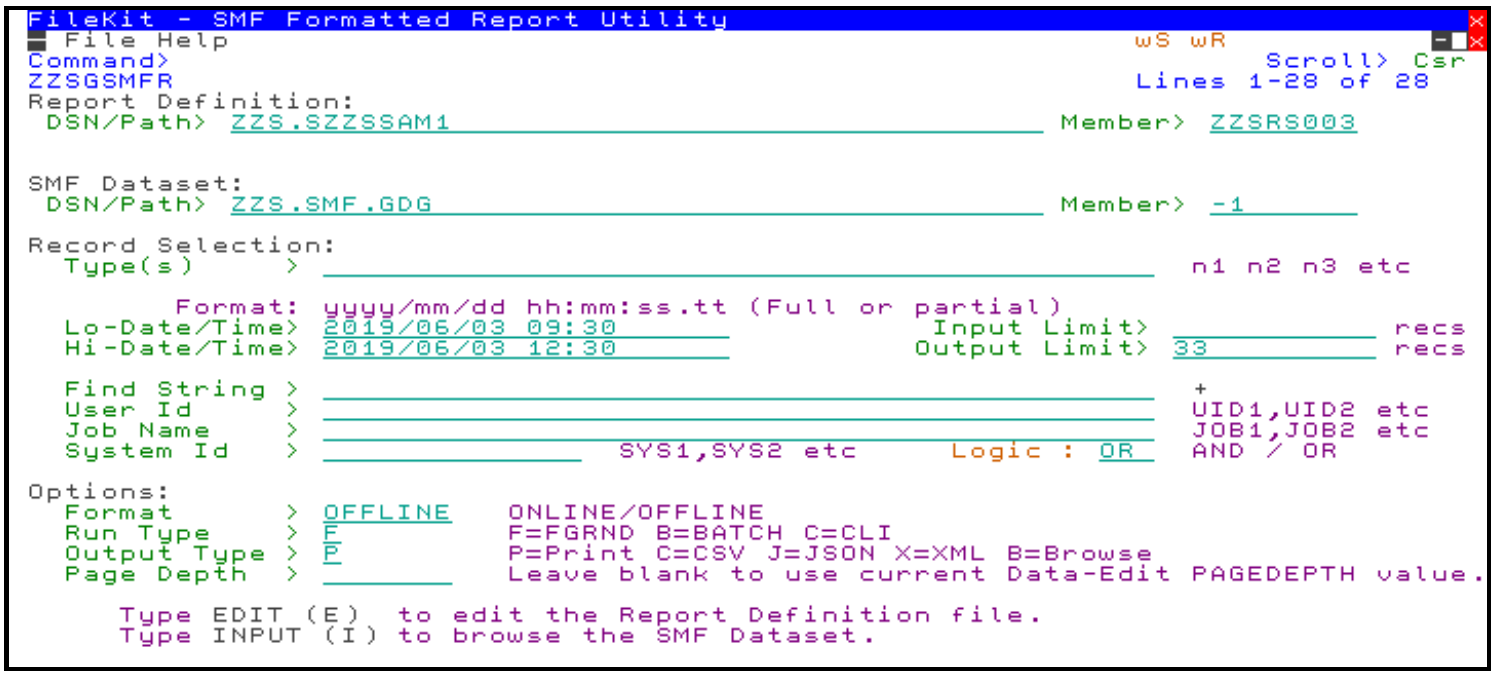


Figure 7. Generate SMF Dataset Usage Report - Filter Input Records.

Only SMF records written between 09:30 and 12:30 on 7th May 2020 will be passed for filter clause testing.

Unlike values entered in Find String>, User Id>, Job Name> and System Id> input fields, values entered in the Lo-Date/Time> and HiDate/Time> fields are **not** overridden by the FILTER clause specification in the report definition.

Report Output:

The report shows values obtained from those input SMF records that satisfy the FILTER clause. Note that the Reader Timestamp field is the date/time that the reader recognised the JOB card for this job and should not be confused with the SMF record timestamp used by HiDate/LoDate record selection testing.

```
12024/03/11 15:53                                                                                              PAGE:    1
                                           Dataset Usage by Job Name

 Job Name Reader Timestamp       StepName PGMName  Dataset Name              EXCPs  Created    MGMTCLAS STORCLAS
 -------- ---------------------- -------- -------- ------------------------- ------ ---------- -------- --------
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.DB.INI                     2 2009/08/04 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI350.INI                2 2017/10/26 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.CBLI.INI                   7 2015/01/12 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI350.TLIB              32 2017/10/26 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI.SITE.TLIB            33 2013/07/02 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.TLIB              14 2013/10/06 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.WINX               4 2005/11/01 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.WINX               4 2005/11/01 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.TLIB              18 2013/10/06 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI350.SMFMAP            21 2019/01/17 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          366 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO          270 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.FILEKIT.SMF.SDO           37 2019/05/09 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI.SITE.SDO            648 2007/10/25 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI350.SDO              395 2018/12/10 CBLDFLT  CBLDFLT
 JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI.SITE.SDO            648 2007/10/25 CBLDFLT  CBLDFLT
                                                                            ======
 == Grand Totals (33 Items)                                                   5227
                                                                            ======
```

Order Report Output

By default, the REPORT utility processes records in their original input file order.

For example, records in the Formula 1 Drivers data set detailed in Select Report Columns, example 1. are naturally sequenced in ascending order of unique driver number. Similarly, records in the Album Tracks input file used in Change Column Data Display, example 2. are sequenced in ascending order of ARTIST name, then ALBUM name and then Track number. These constitute primary, secondary and tertiary sort key fields respectively.

You may wish to process records in a different order, in which case the following report section may be specified in the report definition member:

- **SORT:**

The following will demonstrate use of a SORT: section to re-order input file records for processing.

The SORT: Section

The SORT: section identifies one or more key field names by which input records will be sorted.

Each SORT field name must match the name of an input record field or computed field specified in either the COLUMNS: or REQUIRED: section. The order in which the field names occur in the SORT: section indicates the hierarchy of the sort fields so that the first field is the primary key, the second the secondary key, etc.

For example, to sort the Formula 1 Drivers data set so that the driver names are reported in alphabetical order, simply use the following. (Note that the default sort order for a given sort key field name is ASCENDING.)

```
SORT:   NAME
```

Similarly, to change the order of Album Track records so that they are reported in ascending order of ALBUM name (primary key) and then in descending order of track number (secondary key), use the following:

```
SORT:   ALBUM;    TRACK-NUM (DESCENDING)
```

See examples that follow for use and sample ouput from these SORT: specifications.

The REPORT utility calls the locally installed SORT utility (e.g. DFSORT or SYNCSORT) to execute the sort process.

Examples

The following examples demonstrate use of the SORT: section to re-organise the order in which records are reported.

Order Report Output - Example 1.

This example uses the same sample Formula 1 Drivers COBOL copy book (ZZSCF1DR) described in Select Report Columns, Example 1.

Use the same FileKit **Formatted Record Report** panel input as specified in **Select Report Columns, Example 1.** but change the report definition member name to be **ZZSRF0D4** and the Page Depth value to be **28**.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF0D4)**:

The definition is identical to that used in Change Page Display, Example 1. but includes a SORT: section to sort the report records by driver name (NAME).

```
OPTIONS:   TOTALS(NO)

HEAD:
  #DAYNAME 0 "," #TODAY     /    "Formula 1 Statistics"
                "2019 Season Drivers"

FOOT:
  "FIA Database"            /    "Page" #PAGE (RIGHT,3)

COLUMNS:
  1                        /* Insert 1 space before columns. */
  NAME                "Driver Name"
  BIRTH-PLACE         "Place of Birth"
  COUNTRY             "Country"
  NUMBER              "Driver#"
  DATE-OF-BIRTH       "DOB"
  FIRST-RACE          "First Race"
  FIRST-RACE-CIRCUIT "Circuit"

SORT:      NAME           /* Sort records by NAME field. */
```

Report Output:

```
12020/05/12, Tuesday                                          Formula 1 Statistics
                                   2019 Season Drivers

  Driver Name          Place of Birth       Country              Driver# DOB        First Race Circuit
  -------------------- -------------------- -------------------- ------- ---------- ---------- -------
  Alexander Albon      London               Thailand                  23 1996/03/23 2019/03/17 AUS
  Antonio Giovinazzi   Martina Franca       Italy                     99 1993/12/14 2017/03/26 AUS
  Carlos Sainz Jr.     Madrid               Spain                     55 1994/09/01 2015/03/15 AUS
  Charles Leclerc      Monte Carlo          Monaco                    16 1997/10/16 2018/03/25 AUS
  Daniel Ricciardo     Perth                Australia                  3 1989/07/01 2011/07/10 BRI
  Daniil Kvyat         Oefa                 Russian Federation        26 1994/04/26 2014/03/16 AUS
  George Russell       Kings Lynn           United Kingdom            63 1998/02/15 2019/03/17 AUS
  Kevin Magnussen      Roskilde             Denmark                   20 1992/10/05 2014/03/16 AUS
  Kimi Raikkonen       Espoo                Finland                    7 1979/10/17 2001/03/04 AUS
  Lance Stroll         Montreal             Canada                    18 1998/10/29 2017/03/26 AUS
  Lando Norris         Bristol              United Kingdom             4 1999/10/13 2019/03/17 AUS
  Lewis Hamilton       Stevenage            United Kingdom            44 1985/01/07 2007/03/18 AUS
  Max Verstappen       Hasselt              Netherlands               33 1997/10/30 2015/03/15 AUS
  Nico Hulkenberg      Emmerich am Rhein    Germany                   27 1987/08/19 2010/03/14 BAH
  Pierre Gasly         Rouen                France                    10 1996/02/07 2017/10/01 RUS
  Robert Kubica        Krakau               Poland                    88 1984/12/07 2006/08/06 HUN
  Romain Grosjean      Geneva               France                     8 1986/04/17 2009/08/23 BEL
  Sebastian Vettel     Heppenheim           Germany                    5 1987/07/03 2007/06/17 USA
  Sergio Perez         Guadalajara          Mexico                    11 1990/01/26 2011/03/27 AUS
  Valtteri Bottas      Nastola              Finland                   77 1989/08/28 2013/03/17 AUS

 FIA Database                                                                    Page   1
```

Order Report Output - Example 2.

This example uses the same sample Album Tracks COBOL copy book (ZZST1CPC) described in Change Column Data Display, Example 2.

Use the same FileKit **Formatted Record Report** panel input as specified in **Change Column Data Display, Example 2.** but change the report definition member name to be **ZZSRF002** and remove the Output Limit value.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF002)**:

The definition includes a SORT section to sort records by album title (ALBUM) and then, for each album, sort the tracks in reverse order of track number (TRACK-NUM). A FILTER: section is included to restrict the report output to only those album titles that start with the letter "F".

The STATS(NO) option is used to suppress display of totals for numeric data type field columns, but to still output the grand total number of report items.

```
OPTIONS:  STATS(NO)

HEAD:
 #TIMESTMP  /  'PAGE:' #PAGE (RIGHT,3)
 "Album Contents - Sorted in Reverse Track Order"

COLUMNS:
 ALBUM          "Album"       20
 TRACK-NUM      "Track#"       4
 NAME           "Track Name"  35
 ARTIST         "Artist"      20

SORT:  ALBUM;  TRACK-NUM (DESCENDING)

FILTER:
 (ALBUM >> 'F')
```

Report Output:

```
12020/05/12 15:35                                                    PAGE:   1
                    Album Contents - Sorted in Reverse Track Order

 Album                Track# Track Name                           Artist
 -------------------- ------ ------------------------------------ --------------------
 Feel                     11 Flow                                 Roachford
 Feel                     10 Time                                 Roachford
 Feel                      9 Testify                              Roachford
 Feel                      8 Down                                 Roachford
 Feel                      7 Move On                              Roachford
 Feel                      6 Nothing Free                         Roachford
 Feel                      5 Naked Without You                    Roachford
 Feel                      4 Someday                              Roachford
 Feel                      3 Don't Make Me Love You               Roachford
 Feel                      2 How Could I? (Insecurity)            Roachford
 Feel                      1 Way I Feel                           Roachford
 Feels Like Today         12 Skin (Sarabeth)                      Rascal Flatts
 Feels Like Today         11 Oklahoma-Texas Line                  Rascal Flatts
 Feels Like Today         10 Holes                                Rascal Flatts
 Feels Like Today          9 Break Away                           Rascal Flatts
 Feels Like Today          8 The Day Before You                   Rascal Flatts
 Feels Like Today          7 Here's to You                        Rascal Flatts
 Feels Like Today          6 When the Sand Runs Out               Rascal Flatts
 Feels Like Today          5 Fast Cars and Freedom                Rascal Flatts
 Feels Like Today          4 Feels Like Today                     Rascal Flatts
 Feels Like Today          3 Then I Did                           Rascal Flatts
 Feels Like Today          2 Bless the Broken Road                Rascal Flatts
 Feels Like Today          1 Where You Are                        Rascal Flatts
 Freedom Fields           13 Send Yourself Away (Bonus Track)     Seth Lakeman
 Freedom Fields           12 Band Of Gold                         Seth Lakeman
 Freedom Fields           11 Final Lot                            Seth Lakeman
 Freedom Fields           10 The Charmer                          Seth Lakeman
 Freedom Fields            9 Riflemen Of War                      Seth Lakeman
 Freedom Fields            8 1643                                 Seth Lakeman
 Freedom Fields            7 Take No Roufes                       Seth Lakeman
 Freedom Fields            6 Childe The Hunter                    Seth Lakeman
 Freedom Fields            5 King And Country                     Seth Lakeman
 Freedom Fields            4 The Colliers                         Seth Lakeman
 Freedom Fields            3 The White Hare                       Seth Lakeman
 Freedom Fields            2 Setting Of The Sun                   Seth Lakeman
 Freedom Fields            1 Lady Of The Sea                      Seth Lakeman
 Frontiers                14 Only Solutions                       Journey
 Frontiers                13 Liberty                              Journey
 Frontiers                12 Ask the Lonely                       Journey
 Frontiers                11 Only the Young                       Journey
 Frontiers                10 Rubicon                              Journey
 Frontiers                 9 Frontiers                            Journey
 Frontiers                 8 Back Talk                            Journey
 Frontiers                 7 Troubled Child                       Journey
 Frontiers                 6 Edge of the Blade                    Journey
 Frontiers                 5 Faithfully                           Journey
 Frontiers                 4 After the Fall                       Journey
 Frontiers                 3 Chain Reaction                       Journey
 Frontiers                 2 Send Her My Love                     Journey
 Frontiers                 1 Separate Ways (Worlds Apart)         Journey

 == Grand Totals (50 Items)
```

Order Report Output - Example 3.

This example expands on Change Page Display, example 2. which generates a report from SMF log records. Only SMF record type 30 (Common Address Space Work) records are processed to report job step totals. All other SMF record types are bypassed.

Use the same FileKit **SMF Report** panel input as specified in **Change Page Display, example 2.** but change the report definition member name to be **ZZSRS004**, set the Page Depth to be **47** and remove the Output Limit value.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRS004)**:

A SORT:, FILTER: and STATISTICS: section is added to the report definition specified by member ZZSRS002, so that only job steps that finish with a non-zero condition code are reported and the report records are first sorted by job name and then by job reader timestamp.

The STATISTICS: section specifies the columns of numeric or TIME data type for which statistics will be gathered (default is totals).

```
HEAD:
  #TIMESTMP / 'PAGE' #PAGE (RIGHT,3)
  "SMF Record Type 30-4 (Step Termination) Statistics"
  "First Jobname:" zJOBNAME  5 'Terminating at:' zRST

FOOT:
   "Last Jobname:" zJOBNAME  5 'Terminating at:' zRST

COLUMNS:
  SMF030_Identification.zRST         'Reader Timestamp'
  SMF030_Identification.zJOBNAME     'Job Name'
  SMF030_Identification.zSTN         'Step#'                     5 RIGHT
  SMF030_Identification.zPGM         'Program|name'
  SMF030_Completion.zSCC            ('CC'                 RIGHT) 3 RIGHT
  SMF030_Processor_Accounting.zCPT  ('CPU|Time'           RIGHT) 8 RIGHT
  SMF030_IO_Activity.zTEP           ('EXCPs'              RIGHT) 6 RIGHT
  SMF030_IO_Activity.zAIC           ('Connect|Time'       RIGHT) 9 RIGHT
  SMF030_IO_Activity.zAIW           ('Control|Unit Time'  RIGHT) 9 RIGHT
  SMF030_IO_Activity.zAIS           ('I/O|Starts'         RIGHT) 6 RIGHT

FILTER:
  SMF030_Completion.zSCC > 0

SORT:
  SMF030_Identification.zJOBNAME
  SMF030_Identification.zRST

STATISTICS:
  SMF030_IO_Activity.zAIC
  SMF030_IO_Activity.zTEP
  SMF030_Processor_Accounting.zCPT
  SMF030_IO_Activity.zAIW
  SMF030_IO_Activity.zAIS
```

Report Output:

Only job steps with a non-zero condition code are displayed (CC<>0).

The job step report records are in alphabetical order of job name and, where the job names are the same, ascending chronological order of reader timestamp.

```
12020/05/14 16:43                                                                PAGE   1
                    SMF Record Type 30-4 (Step Termination) Statistics
              First Jobname: JGE          Terminating at: 2019/02/12 09:34:47.30

                                    Program          CPU          Connect   Control    I/O
Reader Timestamp       Job Name Step# name      CC     Time  EXCPs      Time Unit Time Starts
---------------------- -------- ----- -------- --- -------- ------ --------- --------- ------
2019/02/12 09:34:47.30 JGE          1 ADFMDF03 314 06:00.00 192641 03.113728 00.286976 100028
2019/02/13 09:47:00.38 JGE          1 ADFMDF03 314 08:49.80 432414 12.649088 02.514816 336998
2019/02/14 09:36:17.86 JGE          1 ADFMDF03 570 02:07.19  63145 01.526784 00.150784  22593
2019/02/14 12:37:10.84 JGE          1 ADFMDF03 570 06:29.31 259307 07.144192 01.639424 214213
2019/02/14 15:15:25.94 JGE          1 ADFMDF03 314 02:58.72 208674 04.070016 00.325376  97680
2019/02/12 12:38:57.08 JGESI        1 SDEAMAIN 546 00:42.23   3628 00.137856 00.019456   2697
2019/02/12 12:50:39.59 JGESI        1 SDEAMAIN 546 01:41.70   5993 00.300416 00.045312   5071
2019/02/12 14:08:32.20 JGESI        1 SDEAMAIN 082 00:00.00      0 00.001408 00.000256     38
2019/02/12 14:08:56.70 JGESI        1 SDEAMAIN 082 00:00.00      0 00.000896 00.000384     38
2019/02/12 14:09:07.73 JGESI        1 SDEAMAIN 546 00:02.67   1564 00.026496 00.004992    620
2019/02/12 14:09:22.87 JGESI        1 SDEAMAIN 546 00:10.48   1928 00.033152 00.007424   1005
2019/02/12 14:09:55.04 JGESI        1 SDEAMAIN 058 27:12.31  40042 02.913792 00.387072  39120
2019/02/12 14:49:54.73 JGESI        1 SDEAMAIN  22 00:00.74   1190 00.009344 00.001792    356
2019/02/12 14:50:11.42 JGESI        1 SDEAMAIN   4 00:00.83   1172 00.010624 00.002176    356
2019/02/12 09:45:13.10 JGE2         1 ADFMDF03 314 01:53.17  63685 02.413824 00.464768  58846
2019/02/14 15:17:57.40 JGE2         1 ADFMDF03 314 01:16.29  53900 02.133760 00.375424  50097
2019/02/12 15:30:45.39 SDEFFOB1     1 ASMA90     8 00:02.42   3072 00.350720 00.002560    301
2019/02/13 10:59:05.63 SDEFFOB1     1 ASMA90     4 00:00.24    307 00.004864 00.001536    157
2019/02/13 10:59:05.63 SDEFFOB1     2 IEWL      12 00:00.13    149 00.003200 00.001152     65
2019/02/13 11:02:38.53 SDEFFOB1     1 ASMA90     4 00:00.23    297 00.005376 00.001280    157
2019/02/13 11:02:38.53 SDEFFOB1     2 IEWL      12 00:00.13    149 00.002304 00.000256     66
2019/02/14 10:57:38.46 SDEFFSU0     1 ASMA90     8 00:02.19   2917 00.025216 00.003072    283
2019/02/14 12:10:50.55 SDEFFSU2     1 ASMA90     8 00:02.47   3459 00.062336 00.002304    314
2019/02/14 10:31:37.83 SDEFFSU3     1 ASMA90     8 00:01.94   2909 00.024448 00.002816    279
2019/02/14 10:41:24.06 SDEFFSU3     1 ASMA90     8 00:02.30   2882 00.118656 00.001920    265
2019/02/14 10:44:28.21 SDEFFSU3     1 ASMA90     8 00:02.40   2884 00.023936 00.002048    278
2019/02/13 10:22:58.62 SDEFRPO1     1 ASMA90     8 00:02.79   3911 00.044160 00.004608    404
2019/02/12 14:44:49.48 SDEFSEO2     1 ASMA90     4 00:00.54    802 00.004736 00.001408    202
2019/02/12 14:44:49.48 SDEFSEO2     2 IEWL      12 00:00.16    161 00.002816 00.000384     73
2019/02/12 14:45:41.22 SDEFSEO2     1 ASMA90     8 00:02.85   3362 00.212736 00.003456    312
2019/02/12 15:59:17.64 SDEFSEO2     1 ASMA90     8 00:02.36   3373 00.043264 00.002944    316
2019/02/12 16:07:29.96 SDEFSEO2     1 ASMA90     8 00:02.38   3536 00.142080 00.002432    300
2019/02/12 16:16:13.34 SDEFSEO2     1 ASMA90    12 00:02.40   3546 00.031616 00.002304    318
2019/02/12 16:17:42.67 SDEFSEO2     1 ASMA90     8 00:02.43   3552 00.042368 00.002560    304
2019/02/13 15:33:03.50 SDELMAIN     2 IEWL       4 00:00.30    259 00.019840 00.000384     71
                                                   ======== ====== ========= ========= ======
== Grand Totals (35 Items)                         59:56.10 >1370K 37.650048 06.265856 934221
                                                   ======== ====== ========= ========= ======

              Last Jobname: SDELMAIN     Terminating at: 2019/02/13 15:33:03.50
```

Insert Breaks

Inserting *control breaks* in the printed report provides a method by which groups of report lines that share a common column value may be distinguished from each other.

For example, the report generated by Order Report Ouput, example 2. has been sorted first by ALBUM column values and so all the report detail lines are grouped together by a common album title. The first group of detail lines are for the album entitled "Feel", the second group for album entitled "Feel Like Today" and so on. To make the report more readable and potentially insert statistical information and lines of text between the group detail lines, a control break may be defined which is triggered whenever there is a change in the ALBUM column value.

To define a control break and optionally identify columns for which statistical data will be reported (e.g. maximum or average values), then the following report sections may be specified in the report definition member:

- **BREAK:**
- **STATISTICS:**

The following will demonstrate use of the BREAK: section to define a single control break for printed report output. Use of BREAK: to define multiple control breaks discussed later.

The BREAK: Section

The BREAK: section defines one or more control breaks where each control break definition is specified on a single control statement.

A control break definition identifies the field name for which a change in value will trigger the control break. This field name must match the name of an input record field or computed field specified in either the COLUMNS: or REQUIRED: section.

For example, to trigger a control break in the printed report when there is a change in album title (input record field name ALBUM), simply use the following.

```
COLUMNS:   ALBUM;   TRACK-NUM;  NAME;  ARTIST
SORT:      ALBUM                  /* Sort by Album Title. */
BREAK:     ALBUM                  /* Control break on change in Album Title. */
```

A control break denotes the end of one group of report detail lines and the start of the next group. Each group of report detail lines produced by a single control break definition is referred to as a *control group*. In our example, there is only one control break definition (for field name ALBUM) and so the groups may be referenced as **ALBUM** control groups.

The report lines printed between control groups are break lines defined by the control break. These may be customised so that blank lines, lines of text and/or lines containing statistical values (totals, averages, etc.) are displayed. If no customisation is specified on a control break definition, then 3 or 4 default break lines are printed after the control group as follows:

1. A line containing blanks and underline symbols "-" (hyphon) or, for the #GRAND (end-of-report) break, "=" (equals). The underline symbols underline column values for which statistical values are generated and extend for the full width of each column. If no statistics values are to be generated for any of the report display columns, then no underlining exists and this line will contain only blanks.
2. A line containing the following text:
   - Either "**== Totals for** *fieldname*" (where *fieldname* is the break trigger field) or "**== Grand Totals**" if output is for the **#GRAND** (end-of-report) break.
   - "**(***items* **ITEMS)**" where *items* is the number of items (record detail lines) in the last control group.
   - A total of the values in each column for which statistics are generated. These totals are obtained for record detail lines belonging to the last control group only. (Note that the #GRAND control group comprises **all** detail lines in the report.)
3. For the #GRAND break only, a line which is identical to the first break line containing statistics column underlines.
4. A blank line.

Example 1. that follows demonstrates the default break line output.

Control break definitions support operands that customise the break lines produced when the control break is triggered. These include operands that display a report line for each of the **average**, **total**, **minimum**, **maximum**, **non-zero average** and **non-zero minimum** values generated for the statistics columns in addition to specific control group **header** and **footer** text. See Example 2. and Example 3. that follow.

BREAK Line Text

Break heading, footing and statistics line contain text that may be costomised as part of the control break definition.

The control break operands **AVERAGE**, **MAXIMUM**, **MINIMUM**, **NZAVERAGE**, **NZMINIMUM**, **TOTAL**, **FOOTING** and **HEADING** each support specification of a parenthesised *print-expression*. A print-expression is comprised of one or more text fragment specifications which together produce a single line of text. (The text line itself may be split over several report output lines using the "<NEWLINE>" keyword in the print-expression.)

A text fragment may be represented as a character literal or as an input, computed or built-in field name. If specified as a field name, then the current value of the field will be substitued when the text is written to the report. For break HEADING lines, this will be a value obtained from the **next** output report detail line. For all other break lines, theis will be a value obtained from the **previous** output report detail line.

Each text fragment has a default width and alignment which may be changed using parenthesised overrides that immediately follow the fragment definition. Additionally, the default gap of 1 blank between 2 fragments of text may be overridden simply by specifying a number of blanks (gap) value between the 2 fragment definitions.

For example, the following is a control break definition for field name "ALBUM":

```
BREAK:
  ALBUM  FOOTING('Number of entries for' ALBUM (20,STRIP) 0 ':' #ITEMS (LEFT) )
```

The break footing line text is defined by a print-expression comprised of 4 text fragments:

1. A character string literal, `'Number of entries for'`.
2. An input field name, `ALBUM`.
3. A character string literal, `':'`.
4. The built-in field name: `#ITEMS`.

The value inserted by the "ALBUM" (album name) text fragment is truncated to a width of 20 characters and then leading and trailing blanks stripped due to the parenthesised override "`(20,STRIP)`". A gap value of "0" (zero) immediately follows so that the next text fragment (character string ":") is placed immediately following the album title. The value substitued for the "#ITEMS" field name represents the number of items (report detail lines) contained in the ALBUM control group just printed. The #ITEMS value is left adjusted because of the "`(LEFT)`" override, so that only 1 blank will occur between the ":" and the first digit of the numeric value.

The STATISTICS: Section

The STATISTICS: section identifies field names assigned to columns definitions in the COLUMNS: section. These columns are then identified as being statistics columns for which statistical data will be accumulated and reported.

Field names specified in the STATISTICS: section must identify fields that are of numeric data type or data type TIME, which will be treated as an elapsed time value. Fields of character data type may be specified if the values are numeric (i.e. contain only decimal digits and possibly a decimal point).

By default, **every** report column defined by an input field of numeric data type is treated as being a statistics column.

For example, the following report contains 3 columns of numeric data type ("LAPS", "LAP-LENGTH-KM" and "TURNS") and 1 of data type TIME ("RACE-LAP-RECORD"). By default, statistical information (totals values) would be maintained for the 3 columns of numeric data type, however, the STATISTICS: section overrides this and identifies columns "LAP-LENGTH-KM" and "RACE-LAP-RECORD" as the staistics columns instead.

```
COLUMNS:
  TRACK                "Track Name"
  LAPS                 "#Laps"
  LAP-LENGTH-KM        "Lap Distance (KM)"
  TURNS                "#Turns/Lap"
  RACE-LAP-RECORD      "Lap Record (HH:MM:SS.MMM)"

STATISTICS:
  LAP-LENGTH-KM;    RACE-LAP-RECORD
```

By default, totals values are generated for the statistics columns and are reported at each control break. This includes the implied #GRAND control break which is triggered at end-of-report. (Options BRKTOTALS, GRANDTOTAL and TOTALS may suppress this default behaviour.)

The BREAK: section defines control breaks for which output of the totals values may be suppressed (**NOTOTAL**) and/or for which additional statistical information may be reported.

For example, a control break definition may contain parameter keyword **NZAVERAGE** which will ensure that average values are generated for non-zero values in the statistics columns. When the control break is triggered, a break line containing the average values for the control group is reported.

Examples

The following examples demonstrate use of the BREAK: section to trigger control breaks and to customise the printed break report lines.

The following examples show how a control break may be triggered for groups of report lines that contain the same values in one of the fields either displayed as column data or associated with the report lines. In particular, the examples demonstrate:

- Use of the BREAK: report section to define a single control break.
- Use of the STATISTICS: report section to eplicitly identify report columns for which statistical information will be collated and reported at each control break.
- Use of the REQUIRED: report section in order to sort and create a control break on a field not displayed as column data.
- Specification of the types of statistical data to be reported for a particular control break definition.
- Use of a *print-expression* to customise break line text.
- Substitution of input record and built-in field values in print-expressions.
- Use of gap values to override spacing between print-expression text fragments.

Insert Breaks - Example 1.

This example uses the same sample Album Tracks COBOL copy book (ZZST1CPC) described in Change Column Data Display, Example 2.

Use the same FileKit **Formatted Record Report** panel input as specified in **Change Column Data Display, Example 2.** but change the report definition member name to be **ZZSRF003** and remove the Output Limit value.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF003)**:

The definition is almost identical to that used in Order Report Output, Example 2. but includes a BREAK: section to trigger a control break when the album title (ALBUM) changes and output default break line information. It also reports the track time in milli-seconds (TOTAL-TIME) field, for which statistical values are generated. and filters records so that only album titles beginning with the string "Feel" are selected.

```
HEAD:
 #TIMESTMP  /  'PAGE:' #PAGE (RIGHT,3)
 "Album Contents - Sorted in Reverse Track Order"

COLUMNS:
 ALBUM          "Album"       20
 TRACK-NUM      "Track#"       4
 NAME           "Track Name"  35
 ARTIST         "Artist"      20
 TOTAL-TIME     "Time (1/1000 sec)"

SORT:  ALBUM;  TRACK-NUM (DESCENDING)
BREAK: ALBUM

STATISTICS:    TOTAL-TIME

FILTER:
 (ALBUM >> 'Feel')
```

Report Output:

```
12020/05/18 16:10                                                                PAGE:   1
                          Album Contents - Sorted in Reverse Track Order

 Album                Track# Track Name                          Artist               Time (1/1000 sec)
 -------------------- ------ ----------------------------------- -------------------- -----------------
 Feel                     11 Flow                                Roachford                       281054
 Feel                     10 Time                                Roachford                       452417
 Feel                      9 Testify                             Roachford                       217292
 Feel                      8 Down                                Roachford                       244645
 Feel                      7 Move On                             Roachford                       267818
 Feel                      6 Nothing Free                        Roachford                       269026
 Feel                      5 Naked Without You                   Roachford                       209536
 Feel                      4 Someday                             Roachford                       210326
 Feel                      3 Don't Make Me Love You              Roachford                       232710
 Feel                      2 How Could I? (Insecurity)           Roachford                       224769
 Feel                      1 Way I Feel                          Roachford                       226023
                                                                                          -----------------
   == Totals for Feel                (11 Items)                                                2835616

 Feels Like Today         12 Skin (Sarabeth)                     Rascal Flatts                   261280
 Feels Like Today         11 Oklahoma-Texas Line                 Rascal Flatts                   175013
 Feels Like Today         10 Holes                               Rascal Flatts                   258933
 Feels Like Today          9 Break Away                          Rascal Flatts                   191680
 Feels Like Today          8 The Day Before You                  Rascal Flatts                   246440
 Feels Like Today          7 Here's to You                       Rascal Flatts                   217840
 Feels Like Today          6 When the Sand Runs Out              Rascal Flatts                   226506
 Feels Like Today          5 Fast Cars and Freedom               Rascal Flatts                   263053
 Feels Like Today          4 Feels Like Today                    Rascal Flatts                   201640
 Feels Like Today          3 Then I Did                          Rascal Flatts                   192586
 Feels Like Today          2 Bless the Broken Road               Rascal Flatts                   227186
 Feels Like Today          1 Where You Are                       Rascal Flatts                   232933
                                                                                          -----------------
   == Totals for Feels Like Today     (12 Items)                                                2695090

                                                                                          =================
 ==== Grand Totals (23 Items)                                                                   5530706
                                                                                          =================
```

Insert Breaks - Example 2.

This example uses the sample Formula 1 Race Results FileKit SDO structure (ZZSSF1RE) to format records from the input data set which contains details of the results of Formula 1 races staged in the 2019 championship.

See Appendix C. Sample Data for description of the sample ZZSSF1RE FileKit SDO structure layout.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRF0R1)**:

Input field name "DRIVER" is used to SORT the report output detail lines so that report detail lines are grouped together by driver name. It is also used as the trigger for a control BREAK and referenced in the break line HEADING and FOOTING print-expressions in order to display the next/previous driver name value in the break's heading and footing text.

"DRIVER" is not used as a column definition in the COLUMNS: section and so its values are not reported in the detail lines. Therefore, field name "DRIVER" must be specified in the REQUIRED: section. Similarly for "DRIVER-TEAM" which is used in the FILTER: section to select only drivers that race for team Mercedes.

Statistical data will be generated for column "FINISH-TIME" only, as specified by the STATISTICS: section. By default, the statistical information includes the sum total of the elapsed values reported in the "FINISH-TIME" column.

In addition to the default #GRAND break which will report the grand total of "FINISH-TIME" values for all detail lines in the report, the control break definition for field name "DRIVER" will mean that a separate "FINISH-TIME" totals values will be generated for detail lines in each DRIVER control group. Furthermore, NZAVERAGE and MAXIMUM in the control break definition means that an average value (which excludes zero values) and the maximum value will also be reported for "FINISH-TIME" values in each DRIVER control group.

The single control break definition includes a **HEADING** and **FOOTING** specification to output heading text before the first detail line in the DRIVER control group and footing text after the last line of control group statistics values. **<NEWLINE>** is used in the print-expressions to force a line break. **SPACEAFTER(1)** will output a blank line following the last break line.

```
REQUIRED:   DRIVER;  DRIVER-TEAM

COLUMNS:    EVENT           "Race#"
            TRACK           "Track"
            GRID-POSITION   "Grid"    2
            POSITION        "Finish"  2
            LAPS-COMPLETED  "Laps"    2
            POINTS          "Points"  2
            NOTES           "Notes"
            FINISH-TIME     "Time"

SORT:       DRIVER
BREAK:      DRIVER                                                    \
             HEADING(                                                 \
               <NEWLINE> 'Race Finish Times for' DRIVER               \
               <NEWLINE> '--------------------------------------'     \
                    )                                                 \
             NZAVERAGE                                                \
             MAXIMUM                                                  \
             FOOTING(                                                 \
               <NEWLINE> 10 '-- End of Results for' DRIVER (STRIP) '---'  \
                    )                                                 \
             SPACEAFTER(1)

STATISTICS: FINISH-TIME

FILTER:     (DRIVER-TEAM = "Mercedes")
```

REPORT Utility Execution:

Using the FileKit Formatted Record Report panel, we enter the names of the report definition, input data file and record mapping library member (type SDO). A run type "**F**" is selected to execute the REPORT utility in the foreground and to ensure report output fits on one page, the Page Depth is set to 100.

```
SELCOPY/i - Formatted Report Utility
 File Help                                                  wS wR
Command>                                                      Scroll> Csr
ZZSGRPT0                                                   Lines 1-28 of 28
Report Definition:
 DSN/Path> ZZS.SZZSSAM1                                  Member> ZZSRF0R1

Data File:
 DSN/Path> ZZS.F1#2019.DATA                              Member>

Structure/Copybook overlay:
      Dsn> ZZS.SZZSDIST.SDO                              Member> ZZSCF1RE
     Type> COBOL     Leave blank for list of available options.

Record Selection:
 Input Limit  >              recs
Output Limit  >              recs

  Find String >                                                  +

Options:
  Run Type     > F          F=FGRND B=BATCH C=CLI
  Output Type  > P          P=Print C=CSV J=JSON X=XML B=Browse
  Page Depth   > 100        Leave blank to use current Data-Edit PAGEDEPTH value.

     Type EDIT  (E) to edit the Report Definition file.
     Type INPUT (I) to browse the Data File.
     Type COPYB (C) to edit the mapping copybook file.
```

Figure 8. Generate Formula 1 Results by Driver printed report.

Report Output:

```
12020/05/21 12:10                                                PAGE    1

 Race#   Track Grid Finish Laps Points Notes                Time
 ------- ----- ---- ------ ---- ------ -------------------- ----------------

 Race Finish Times for Lewis Hamilton
 ----------------------------------------
 2019#01 AUS      1      2   58     18                       01:25:48.211200
 2019#02 BAH      3      1   57     25                       01:34:21.294592
 2019#03 CHI      2      1   56     25                       01:32:06.349824
 2019#04 AZE      2      2   51     18                       01:31:54.466304
 2019#05 SPA      2      1   66     26 Fastest Lap           01:35:50.443008
 2019#06 MON      1      1   78     25                       01:43:28.437248
 2019#07 CAN      2      1   70     25                       01:29:07.084288
 2019#08 FRE      1      1   53     25                       01:24:31.198208
 2019#09 OST      4      5   71     10                       01:22:24.626688
 2019#10 BRI      2      1   52     26 Fastest Lap           01:21:08.452352
 2019#11 GER      1      9   64      2                       01:44:50.941952
 2019#12 HUN      3      1   70     25                       01:35:03.795712
 2019#13 BEL      3      2   44     18                       01:23:46.691072
 2019#14 ITA      2      3   53     16 Fastest Lap           01:16:01.863680
 2019#15 SIN      2      4   61     12                       01:58:38.274560
 2019#16 RUS      2      1   53     26 Fastest Lap           01:33:38.992128
 2019#17 JAP      4      3   52     16 Fastest Lap           01:22:00.612864
 2019#18 MEX      3      1   71     25                       01:36:48.904192
 2019#19 USA      5      2   56     18                       01:33:59.800832
 2019#20 BRA      3      7   71      6                       01:33:25.817344
 2019#21 ABU      1      1   55     26 Fastest Lap           01:34:05.715456
                                                            ----------------
   == Totals for Lewis Hamilton       (21 Items)             32:13:01.973504
      Maximum Value                                         01:58:38.274560
      Average of NON-ZERO Values                            01:32:02.951119

           -- End of Results for Lewis Hamilton ---


 Race Finish Times for Valtteri Bottas
 ----------------------------------------
 2019#01 AUS      2      1   58     26 Fastest Lap           01:25:27.324672
 2019#02 BAH      4      2   57     18                       01:34:24.275456
 2019#03 CHI      1      2   56     18                       01:32:12.902400
 2019#04 AZE      1      1   51     25                       01:31:52.941568
 2019#05 SPA      1      2   66     18                       01:35:54.517504
 2019#06 MON      2      3   78     15                       01:43:31.599360
 2019#07 CAN      6      4   70     13 Fastest Lap           01:29:58.126592
 2019#08 FRE      2      2   53     18                       01:24:49.254400
 2019#09 OST      3      3   71     15                       01:22:20.781568
 2019#10 BRI      1      2   52     18                       01:21:33.379584
 2019#11 GER      3      0   56      0 Spun Off              00:00:00.000000
 2019#12 HUN      2      8   69      4                       01:35:35.923712
 2019#13 BEL      4      3   44     15                       01:23:58.295040
 2019#14 ITA      3      2   53     18                       01:15:27.500288
 2019#15 SIN      5      5   61     10                       01:58:39.785984
 2019#16 RUS      4      2   53     18                       01:33:42.820864
 2019#17 JAP      3      1   52     25                       01:21:46.755072
 2019#18 MEX      5      3   71     15                       01:36:52.457472
 2019#19 USA      1      1   56     25                       01:33:55.652608
 2019#20 BRA      4      0   51      0 Power Unit            00:00:00.000000
 2019#21 ABU     20      4   55     12                       01:34:50.093568
                                                            ----------------
   == Totals for Valtteri Bottas      (21 Items)             28:56:54.387712
      Maximum Value                                         01:58:39.785984
      Average of NON-ZERO Values                            01:31:24.967774

           -- End of Results for Valtteri Bottas ---

                                                            ================
 ==== Grand Totals (42 Items)                                61:09:56.361216
                                                            ================
```

Insert Breaks - Example 3.

This example expands on Order Report Output, example 3. which generates a report from SMF log records. Only SMF record type 30 (Common Address Space Work) records are processed to report job step totals. All other SMF record types are bypassed.

Use the same FileKit **SMF Report** panel input as specified in **Change Page Display, example 2.** but change the report definition member name to be **ZZSRS005**, set the Page Depth to be **85** and remove the Output Limit value.

Report Definition Input - **ZZS.ZZSSAM1(ZZSRS005)**:

A BREAK: section is added to the report definition specified by member ZZSRS004, so that a control break is triggered when there is a change in Job Name (field **zJOBNAME**). To limit the size of the report, the FILTER: section filter clause is also updated to select only records where the reader timestamp is "2019/02/13" or later.

When the zJOBNAME coontrol break is triggered, report break lines are written that display the totals, non-zero average, non-zero minimum and maximum values of entries in the statistics columns (identified by the STATISTICS: section). Also, for each zJOBNAME control break, 2 blank lines are written after the last break line.

```
HEAD:
  #TIMESTMP / 'PAGE' #PAGE (RIGHT,3)
  "SMF Record Type 30-4 (Step Termination) Statistics"
  "First Jobname:" zJOBNAME  5 'Terminating at:' zRST

FOOT:
   "Last Jobname:" zJOBNAME  5 'Terminating at:' zRST

COLUMNS:
  SMF030_Identification.zRST        'Reader Timestamp'
  SMF030_Identification.zJOBNAME    'Job Name'
  SMF030_Identification.zSTN        'Step#'                    5 RIGHT
  SMF030_Identification.zPGM        'Program|name'
  SMF030_Completion.zSCC           ('CC'                RIGHT) 3 RIGHT
  SMF030_Processor_Accounting.zCPT ('CPU|Time'          RIGHT) 8 RIGHT
  SMF030_IO_Activity.zTEP          ('EXCPs'             RIGHT) 6 RIGHT
  SMF030_IO_Activity.zAIC          ('Connect|Time'      RIGHT) 9 RIGHT
  SMF030_IO_Activity.zAIW          ('Control|Unit Time' RIGHT) 9 RIGHT
  SMF030_IO_Activity.zAIS          ('I/O|Starts'        RIGHT) 6 RIGHT

FILTER:
 (SMF030_Completion.zSCC > x'0'  &  SMF030_Identification.zRST >= '2019/02/13')

SORT:
  SMF030_Identification.zJOBNAME
  SMF030_Identification.zRST

BREAK:
  SMF030_Identification.zJOBNAME       \
    NZAVERAGE  MAXIMUM  NZMINIMUM  SPACEAFTER(2)

STATISTICS:
  SMF030_IO_Activity.zAIC
  SMF030_IO_Activity.zTEP
  SMF030_Processor_Accounting.zCPT
  SMF030_IO_Activity.zAIW
  SMF030_IO_Activity.zAIS
```

Report Output:

```
12020/05/21 17:37                                                                PAGE   1
                    SMF Record Type 30-4 (Step Termination) Statistics
              First Jobname: JGE          Terminating at: 2019/02/13 09:47:00.38

                                    Program           CPU          Connect   Control    I/O
 Reader Timestamp       Job Name Step# name      CC     Time  EXCPs      Time Unit Time Starts
 ---------------------- -------- ----- -------- --- -------- ------ --------- --------- ------
 2019/02/13 09:47:00.38 JGE          1 ADFMDF03 314 08:49.80 432414 12.649088 02.514816 336998
 2019/02/14 09:36:17.86 JGE          1 ADFMDF03 570 02:07.19  63145 01.526784 00.150784  22593
 2019/02/14 12:37:10.84 JGE          1 ADFMDF03 570 06:29.31 259307 07.144192 01.639424 214213
 2019/02/14 15:15:25.94 JGE          1 ADFMDF03 314 02:58.72 208674 04.070016 00.325376  97680
                                                    -------- ------ --------- --------- ------
   == Totals for JGE      (4 Items)                  20:25.02 963540 25.390080 04.630400 671484
      Maximum Value                                  08:49.80 432414 12.649088 02.514816 336998
      Average of NON-ZERO Values                     05:06.26 240885 06.347520 01.157600 167871
      Minimum of NON-ZERO Values                     02:07.19  63145 01.526784 00.150784  22593


 2019/02/14 15:17:57.40 JGE2         1 ADFMDF03 314 01:16.29  53900 02.133760 00.375424  50097
                                                    -------- ------ --------- --------- ------
   == Totals for JGE2     (1 Items)                  01:16.29  53900 02.133760 00.375424  50097
      Maximum Value                                  01:16.29  53900 02.133760 00.375424  50097
      Average of NON-ZERO Values                     01:16.29  53900 02.133760 00.375424  50097
      Minimum of NON-ZERO Values                     01:16.29  53900 02.133760 00.375424  50097


 2019/02/13 10:59:05.63 SDEFFOB1     1 ASMA90     4 00:00.24    307 00.004864 00.001536    157
 2019/02/13 10:59:05.63 SDEFFOB1     2 IEWL      12 00:00.13    149 00.003200 00.001152     65
 2019/02/13 11:02:38.53 SDEFFOB1     1 ASMA90     4 00:00.23    297 00.005376 00.001280    157
 2019/02/13 11:02:38.53 SDEFFOB1     2 IEWL      12 00:00.13    149 00.002304 00.000256     66
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFFOB1 (4 Items)                  00:00.73    902 00.015744 00.004224    445
      Maximum Value                                  00:00.24    307 00.005376 00.001536    157
      Average of NON-ZERO Values                     00:00.18    226 00.003936 00.001056    111
      Minimum of NON-ZERO Values                     00:00.13    149 00.002304 00.000256     65


 2019/02/14 10:57:38.46 SDEFFSU0     1 ASMA90     8 00:02.19   2917 00.025216 00.003072    283
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFFSU0 (1 Items)                  00:02.19   2917 00.025216 00.003072    283
      Maximum Value                                  00:02.19   2917 00.025216 00.003072    283
      Average of NON-ZERO Values                     00:02.19   2917 00.025216 00.003072    283
      Minimum of NON-ZERO Values                     00:02.19   2917 00.025216 00.003072    283


 2019/02/14 12:10:50.55 SDEFFSU2     1 ASMA90     8 00:02.47   3459 00.062336 00.002304    314
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFFSU2 (1 Items)                  00:02.47   3459 00.062336 00.002304    314
      Maximum Value                                  00:02.47   3459 00.062336 00.002304    314
      Average of NON-ZERO Values                     00:02.47   3459 00.062336 00.002304    314
      Minimum of NON-ZERO Values                     00:02.47   3459 00.062336 00.002304    314


 2019/02/14 10:31:37.83 SDEFFSU3     1 ASMA90     8 00:01.94   2909 00.024448 00.002816    279
 2019/02/14 10:41:24.06 SDEFFSU3     1 ASMA90     8 00:02.30   2882 00.118656 00.001920    265
 2019/02/14 10:44:28.21 SDEFFSU3     1 ASMA90     8 00:02.40   2884 00.023936 00.002048    278
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFFSU3 (3 Items)                  00:06.64   8675 00.167040 00.006784    822
      Maximum Value                                  00:02.40   2909 00.118656 00.002816    279
      Average of NON-ZERO Values                     00:02.21   2892 00.055680 00.002261    274
      Minimum of NON-ZERO Values                     00:01.94   2882 00.023936 00.001920    265


 2019/02/13 10:22:58.62 SDEFRPO1     1 ASMA90     8 00:02.79   3911 00.044160 00.004608    404
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFRPO1 (1 Items)                  00:02.79   3911 00.044160 00.004608    404
      Maximum Value                                  00:02.79   3911 00.044160 00.004608    404
      Average of NON-ZERO Values                     00:02.79   3911 00.044160 00.004608    404
      Minimum of NON-ZERO Values                     00:02.79   3911 00.044160 00.004608    404


 2019/02/13 15:33:03.50 SDELMAIN     2 IEWL       4 00:00.30    259 00.019840 00.000384     71
                                                    -------- ------ --------- --------- ------
   == Totals for SDELMAIN (1 Items)                  00:00.30    259 00.019840 00.000384     71
      Maximum Value                                  00:00.30    259 00.019840 00.000384     71
      Average of NON-ZERO Values                     00:00.30    259 00.019840 00.000384     71
      Minimum of NON-ZERO Values                     00:00.30    259 00.019840 00.000384     71


                                                    ======== ====== ========= ========= ======
 ==== Grand Totals (16 Items)                       21:56.43 >1037K 27.858176 05.027200 723920
                                                    ======== ====== ========= ========= ======


              Last Jobname: SDELMAIN     Terminating at: 2019/02/13 15:33:03.50
```

Summary Reports

A summary report suppresses output of the report detail lines but includes page headers, footers and control break lines.

Generating a summary report is particularly useful if you only wish to report statistical values for groups of records that contain the same value in a particular column.

To suppress all report detail lines and generate a summary report, the following must be specified in the report definition member:

- **OPTIONS: SUMMARY**

Adding this to each of the report definitions described in examples for Insert Breaks produces summary reports as follows:

Examples

Summary Report - Example 1.

Report Output:

See Insert Breaks, example 1.

```
12020/05/22 12:47                                                                PAGE:    1
                          Album Contents - Sorted in Reverse Track Order

 Album               Track# Track Name                          Artist              Time (1/1000 sec)
 ------------------- ------ ----------------------------------- ------------------- -----------------
                                                                                    -----------------
   == Totals for Feel                (11 Items)                                            2835616
                                                                                    -----------------
   == Totals for Feels Like Today    (12 Items)                                            2695090
                                                                                    =================
 ==== Grand Totals (23 Items)                                                             5530706
                                                                                    =================
```

Summary Report - Example 2.

Report Output:

See Insert Breaks, example 2.

```
12020/05/22 12:48                                                    PAGE    1

 Race#   Track Grid Finish Laps Points Notes                Time
 ------- ----- ---- ------ ---- ------ -------------------- -----------------

 Race Finish Times for Lewis Hamilton
 ----------------------------------------
                                                            -----------------
   == Totals for Lewis Hamilton       (21 Items)             32:13:01.973504
      Maximum Value                                         01:58:38.274560
      Average of NON-ZERO Values                            01:32:02.951119

           -- End of Results for Lewis Hamilton ---

 Race Finish Times for Valtteri Bottas
 ----------------------------------------
                                                            -----------------
   == Totals for Valtteri Bottas      (21 Items)             28:56:54.387712
      Maximum Value                                         01:58:39.785984
      Average of NON-ZERO Values                            01:31:24.967774

           -- End of Results for Valtteri Bottas ---
                                                            =================
 ==== Grand Totals (42 Items)                               61:09:56.361216
                                                            =================
```

Summary Report - Example 3.

Report Output:

See Insert Breaks, example 3.

```
12020/05/22 12:49                                                                PAGE   1
                    SMF Record Type 30-4 (Step Termination) Statistics
              First Jobname: JGE          Terminating at: 2019/02/13 09:47:00.38

                                    Program          CPU          Connect   Control    I/O
 Reader Timestamp       Job Name Step# name      CC     Time  EXCPs       Time Unit Time Starts
 ---------------------- -------- ----- -------- --- -------- ------ --------- --------- ------
                                                    -------- ------ --------- --------- ------
   == Totals for JGE      (4 Items)                 20:25.02 963540 25.390080 04.630400 671484
      Maximum Value                                 08:49.80 432414 12.649088 02.514816 336998
      Average of NON-ZERO Values                    05:06.26 240885 06.347520 01.157600 167871
      Minimum of NON-ZERO Values                    02:07.19  63145 01.526784 00.150784  22593
                                                    -------- ------ --------- --------- ------
   == Totals for JGE2     (1 Items)                 01:16.29  53900 02.133760 00.375424  50097
      Maximum Value                                 01:16.29  53900 02.133760 00.375424  50097
      Average of NON-ZERO Values                    01:16.29  53900 02.133760 00.375424  50097
      Minimum of NON-ZERO Values                    01:16.29  53900 02.133760 00.375424  50097
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFFOB1 (4 Items)                 00:00.73    902 00.015744 00.004224    445
      Maximum Value                                 00:00.24    307 00.005376 00.001536    157
      Average of NON-ZERO Values                    00:00.18    226 00.003936 00.001056    111
      Minimum of NON-ZERO Values                    00:00.13    149 00.002304 00.000256     65
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFFSU0 (1 Items)                 00:02.19   2917 00.025216 00.003072    283
      Maximum Value                                 00:02.19   2917 00.025216 00.003072    283
      Average of NON-ZERO Values                    00:02.19   2917 00.025216 00.003072    283
      Minimum of NON-ZERO Values                    00:02.19   2917 00.025216 00.003072    283
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFFSU2 (1 Items)                 00:02.47   3459 00.062336 00.002304    314
      Maximum Value                                 00:02.47   3459 00.062336 00.002304    314
      Average of NON-ZERO Values                    00:02.47   3459 00.062336 00.002304    314
      Minimum of NON-ZERO Values                    00:02.47   3459 00.062336 00.002304    314
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFFSU3 (3 Items)                 00:06.64   8675 00.167040 00.006784    822
      Maximum Value                                 00:02.40   2909 00.118656 00.002816    279
      Average of NON-ZERO Values                    00:02.21   2892 00.055680 00.002261    274
      Minimum of NON-ZERO Values                    00:01.94   2882 00.023936 00.001920    265
                                                    -------- ------ --------- --------- ------
   == Totals for SDEFRPO1 (1 Items)                 00:02.79   3911 00.044160 00.004608    404
      Maximum Value                                 00:02.79   3911 00.044160 00.004608    404
      Average of NON-ZERO Values                    00:02.79   3911 00.044160 00.004608    404
      Minimum of NON-ZERO Values                    00:02.79   3911 00.044160 00.004608    404
                                                    -------- ------ --------- --------- ------
   == Totals for SDELMAIN (1 Items)                 00:00.30    259 00.019840 00.000384     71
      Maximum Value                                 00:00.30    259 00.019840 00.000384     71
      Average of NON-ZERO Values                    00:00.30    259 00.019840 00.000384     71
      Minimum of NON-ZERO Values                    00:00.30    259 00.019840 00.000384     71
                                                    ======== ====== ========= ========= ======
==== Grand Totals (16 Items)                        21:56.43 >1037K 27.858176 05.027200 723920
                                                    ======== ====== ========= ========= ======

              Last Jobname: SDELMAIN     Terminating at: 2019/02/13 15:33:03.50
```

CSV Output

In addition to generating a printable report, the REPORT utility can generate Comma Separated Variable (CSV) output from the COLUMNS: section field values, in a format suitable for import to databases and spreadsheets.

The first CSV output line will be the specified (or default) column header names for each column. All subsequent output lines are the report detail lines containing the prevailing value for each column field.

Page and report break header and footer line definitions are applicable only to printable report output and so are ignored for CSV output.

CSV output is triggered when "CSV" is selected as the output type in one of the REPORT Utility panels, or is specified as an operand on the REPORT command.

Examples

CSV Report - Example 1.

Selecting output type CSV for the first report definition described in example 1. for Insert Breaks produces CSV report output as follows:

```
"Album","Track#","Track Name","Artist","Time (1/1000 sec)"
"Feel                ","  11","Flow                          ","Roachford          ","     281054"
"Feel                ","  10","Time                          ","Roachford          ","     452417"
"Feel                ","   9","Testify                       ","Roachford          ","     217292"
"Feel                ","   8","Down                          ","Roachford          ","     244645"
"Feel                ","   7","Move On                       ","Roachford          ","     267818"
"Feel                ","   6","Nothing Free                  ","Roachford          ","     269026"
"Feel                ","   5","Naked Without You             ","Roachford          ","     209536"
"Feel                ","   4","Someday                       ","Roachford          ","     210326"
"Feel                ","   3","Don't Make Me Love You        ","Roachford          ","     232710"
"Feel                ","   2","How Could I? (Insecurity)     ","Roachford          ","     224769"
"Feel                ","   1","Way I Feel                    ","Roachford          ","     226023"
"Feels Like Today    ","  12","Skin (Sarabeth)               ","Rascal Flatts      ","     261280"
"Feels Like Today    ","  11","Oklahoma-Texas Line           ","Rascal Flatts      ","     175013"
"Feels Like Today    ","  10","Holes                         ","Rascal Flatts      ","     258933"
"Feels Like Today    ","   9","Break Away                    ","Rascal Flatts      ","     191680"
"Feels Like Today    ","   8","The Day Before You            ","Rascal Flatts      ","     246440"
"Feels Like Today    ","   7","Here's to You                 ","Rascal Flatts      ","     217840"
"Feels Like Today    ","   6","When the Sand Runs Out        ","Rascal Flatts      ","     226506"
"Feels Like Today    ","   5","Fast Cars and Freedom         ","Rascal Flatts      ","     263053"
"Feels Like Today    ","   4","Feels Like Today              ","Rascal Flatts      ","     201640"
"Feels Like Today    ","   3","Then I Did                    ","Rascal Flatts      ","     192586"
"Feels Like Today    ","   2","Bless the Broken Road         ","Rascal Flatts      ","     227186"
"Feels Like Today    ","   1","Where You Are                 ","Rascal Flatts      ","     232933"
```

CSV Report - Example 2.

By default, literal strings (constants) specified in the COLUMNS section are omitted from CSV output. Furthermore, all values are placed in quotation marks (") and, to align the values within output lines, all values are padded with blanks to the defined width of the display field.

The format of this CSV output can be tweaked by specifying one or more of the following Boolean options in the report definition.

- **`OPTIONS: CSVLITERALS(YES) CSVQUOTED(NO) CSVSTRIPALL(YES)`**

CSVLITERALS(YES) will include string literals in the output, CSVQUOTED(NO) will suppress inserting values in quotation marks unless it is necessary to do so, and CSVSTRIPALL(YES) will prevent padding field values with blanks.

Adding these 3 options to the report definition in example 1. will produce the following CSV Output:

```
Album,Track#,Track Name,Artist,Time (1/1000 sec)
Feel,11,Flow,Roachford,281054
Feel,10,Time,Roachford,452417
Feel,9,Testify,Roachford,217292
Feel,8,Down,Roachford,244645
Feel,7,Move On,Roachford,267818
Feel,6,Nothing Free,Roachford,269026
Feel,5,Naked Without You,Roachford,209536
Feel,4,Someday,Roachford,210326
Feel,3,Don't Make Me Love You,Roachford,232710
Feel,2,How Could I? (Insecurity),Roachford,224769
Feel,1,Way I Feel,Roachford,226023
Feels Like Today,12,Skin (Sarabeth),Rascal Flatts,261280
Feels Like Today,11,Oklahoma-Texas Line,Rascal Flatts,175013
Feels Like Today,10,Holes,Rascal Flatts,258933
Feels Like Today,9,Break Away,Rascal Flatts,191680
Feels Like Today,8,The Day Before You,Rascal Flatts,246440
Feels Like Today,7,Here's to You,Rascal Flatts,217840
Feels Like Today,6,When the Sand Runs Out,Rascal Flatts,226506
Feels Like Today,5,Fast Cars and Freedom,Rascal Flatts,263053
Feels Like Today,4,Feels Like Today,Rascal Flatts,201640
Feels Like Today,3,Then I Did,Rascal Flatts,192586
Feels Like Today,2,Bless the Broken Road,Rascal Flatts,227186
Feels Like Today,1,Where You Are,Rascal Flatts,232933
```

JSON Output

The REPORT utility can also generate JavaScript Object Notation (JSON) from the COLUMNS: section field values.

The JSON output will be a single JSON Object literal (enclosed in braces "{}").

Each output report detail line generates a JSON object literal containing a number of comma separated key:value pairs where each "key" corresponds to the column field column header string, and "value" is the prevailing column field value.

If option JSONARRAY(YES) is set, the JSON output object literal will comprise a single key:value pair where "key" is "FileKit_Report" and "value" is a JSON array of the report detail line JSON objects.

If option JSONARRAY(NO) is set, the JSON output object literal will comprise multiple key:value pairs, one for each output report detail line. In this case, the "key" is the output detail line sequence number (as described by built-in variable #SEQUENCE) and "value" is the report detail line JSON object.

Page and report break header and footer line definitions are applicable only to printable report output and so are ignored for JSON output.

JSON output is triggered when "JSON" is selected as the output type in one of the REPORT Utility panels, or is specified as an operand on the REPORT command.

Examples

JSON Report - Example 1.

Selecting output type JSON and reducing the COLUMNS: field entries to "ALBUM", "TRACK-NUM" and "NAME" for the first report definition described in example 1. for Insert Breaks, produces the following JSON report output. Note that option JSONARRAY(NO) is default.

```
{
   "000000001" : {"Album" : "Feel                ",     "Track#" : "  11",     "Track Name" : "Flow                          " }
  ,"000000002" : {"Album" : "Feel                ",     "Track#" : "  10",     "Track Name" : "Time                          " }
  ,"000000003" : {"Album" : "Feel                ",     "Track#" : "   9",     "Track Name" : "Testify                       " }
  ,"000000004" : {"Album" : "Feel                ",     "Track#" : "   8",     "Track Name" : "Down                          " }
  ,"000000005" : {"Album" : "Feel                ",     "Track#" : "   7",     "Track Name" : "Move On                       " }
  ,"000000006" : {"Album" : "Feel                ",     "Track#" : "   6",     "Track Name" : "Nothing Free                  " }
  ,"000000007" : {"Album" : "Feel                ",     "Track#" : "   5",     "Track Name" : "Naked Without You             " }
  ,"000000008" : {"Album" : "Feel                ",     "Track#" : "   4",     "Track Name" : "Someday                       " }
  ,"000000009" : {"Album" : "Feel                ",     "Track#" : "   3",     "Track Name" : "Don't Make Me Love You        " }
  ,"000000010" : {"Album" : "Feel                ",     "Track#" : "   2",     "Track Name" : "How Could I? (Insecurity)     " }
  ,"000000011" : {"Album" : "Feel                ",     "Track#" : "   1",     "Track Name" : "Way I Feel                    " }
  ,"000000012" : {"Album" : "Feels Like Today    ",     "Track#" : "  12",     "Track Name" : "Skin (Sarabeth)               " }
  ,"000000013" : {"Album" : "Feels Like Today    ",     "Track#" : "  11",     "Track Name" : "Oklahoma-Texas Line           " }
  ,"000000014" : {"Album" : "Feels Like Today    ",     "Track#" : "  10",     "Track Name" : "Holes                         " }
  ,"000000015" : {"Album" : "Feels Like Today    ",     "Track#" : "   9",     "Track Name" : "Break Away                    " }
  ,"000000016" : {"Album" : "Feels Like Today    ",     "Track#" : "   8",     "Track Name" : "The Day Before You            " }
  ,"000000017" : {"Album" : "Feels Like Today    ",     "Track#" : "   7",     "Track Name" : "Here's to You                 " }
  ,"000000018" : {"Album" : "Feels Like Today    ",     "Track#" : "   6",     "Track Name" : "When the Sand Runs Out        " }
  ,"000000019" : {"Album" : "Feels Like Today    ",     "Track#" : "   5",     "Track Name" : "Fast Cars and Freedom         " }
  ,"000000020" : {"Album" : "Feels Like Today    ",     "Track#" : "   4",     "Track Name" : "Feels Like Today              " }
  ,"000000021" : {"Album" : "Feels Like Today    ",     "Track#" : "   3",     "Track Name" : "Then I Did                    " }
  ,"000000022" : {"Album" : "Feels Like Today    ",     "Track#" : "   2",     "Track Name" : "Bless the Broken Road         " }
  ,"000000023" : {"Album" : "Feels Like Today    ",     "Track#" : "   1",     "Track Name" : "Where You Are                 " }

}
```

JSON Report - Example 2.

By default, JSON output is an object string comprising a key:value pair for each report detail line. This may be changed to a single key:value pair where "value" is an array of object strings (one for each report detail line) by specifying the following option in the report definition.

- **`OPTIONS:  JSONARRAY(YES)`**

Adding this option to the report definition in example 1. will produce the following JSON Output:

```
{"FileKit_Report" :
 [
   "000000001" : {"Album" : "Feel                ",     "Track#" : "  11",     "Track Name" : "Flow                          " }
  ,"000000002" : {"Album" : "Feel                ",     "Track#" : "  10",     "Track Name" : "Time                          " }
  ,"000000003" : {"Album" : "Feel                ",     "Track#" : "   9",     "Track Name" : "Testify                       " }
  ,"000000004" : {"Album" : "Feel                ",     "Track#" : "   8",     "Track Name" : "Down                          " }
  ,"000000005" : {"Album" : "Feel                ",     "Track#" : "   7",     "Track Name" : "Move On                       " }
  ,"000000006" : {"Album" : "Feel                ",     "Track#" : "   6",     "Track Name" : "Nothing Free                  " }
  ,"000000007" : {"Album" : "Feel                ",     "Track#" : "   5",     "Track Name" : "Naked Without You             " }
  ,"000000008" : {"Album" : "Feel                ",     "Track#" : "   4",     "Track Name" : "Someday                       " }
  ,"000000009" : {"Album" : "Feel                ",     "Track#" : "   3",     "Track Name" : "Don't Make Me Love You        " }
  ,"000000010" : {"Album" : "Feel                ",     "Track#" : "   2",     "Track Name" : "How Could I? (Insecurity)     " }
  ,"000000011" : {"Album" : "Feel                ",     "Track#" : "   1",     "Track Name" : "Way I Feel                    " }
  ,"000000012" : {"Album" : "Feels Like Today    ",     "Track#" : "  12",     "Track Name" : "Skin (Sarabeth)               " }
  ,"000000013" : {"Album" : "Feels Like Today    ",     "Track#" : "  11",     "Track Name" : "Oklahoma-Texas Line           " }
  ,"000000014" : {"Album" : "Feels Like Today    ",     "Track#" : "  10",     "Track Name" : "Holes                         " }
  ,"000000015" : {"Album" : "Feels Like Today    ",     "Track#" : "   9",     "Track Name" : "Break Away                    " }
  ,"000000016" : {"Album" : "Feels Like Today    ",     "Track#" : "   8",     "Track Name" : "The Day Before You            " }
  ,"000000017" : {"Album" : "Feels Like Today    ",     "Track#" : "   7",     "Track Name" : "Here's to You                 " }
  ,"000000018" : {"Album" : "Feels Like Today    ",     "Track#" : "   6",     "Track Name" : "When the Sand Runs Out        " }
  ,"000000019" : {"Album" : "Feels Like Today    ",     "Track#" : "   5",     "Track Name" : "Fast Cars and Freedom         " }
  ,"000000020" : {"Album" : "Feels Like Today    ",     "Track#" : "   4",     "Track Name" : "Feels Like Today              " }
  ,"000000021" : {"Album" : "Feels Like Today    ",     "Track#" : "   3",     "Track Name" : "Then I Did                    " }
  ,"000000022" : {"Album" : "Feels Like Today    ",     "Track#" : "   2",     "Track Name" : "Bless the Broken Road         " }
  ,"000000023" : {"Album" : "Feels Like Today    ",     "Track#" : "   1",     "Track Name" : "Where You Are                 " }
 ]
}
```


JSON Report - Example 3.

By default, literal strings (constants) specified in the COLUMNS section are omitted from JSON output. Furthermore, all values are treated as strings and so placed in quotation marks (") and, to align the values within output lines, all values are padded with blanks to the defined width of the display field.

The format of this JSON output can be tweaked by specifying one or more of the following Boolean options in the report definition.

- **OPTIONS: JSONLITERALS(YES) JSONQUOTED(NO) JSONSTRIPALL(YES)**

JSONLITERALS(YES) will include string literals as key:value pairs in the detail line output, JSONQUOTED(NO) will insert values in quotation marks only if the field data type is non-numeric, and JSONSTRIPALL(YES) will prevent padding field values with blanks.

Adding these 3 options to the report definition in example 2. will produce the following JSON Output:


```
{"FileKit_Report" :
 [
   "000000001" : {"Album" : "Feel",      "Track#" : 11,     "Track Name" : "Flow" }
  ,"000000002" : {"Album" : "Feel",      "Track#" : 10,     "Track Name" : "Time" }
  ,"000000003" : {"Album" : "Feel",      "Track#" : 9,     "Track Name" : "Testify" }
  ,"000000004" : {"Album" : "Feel",      "Track#" : 8,     "Track Name" : "Down" }
  ,"000000005" : {"Album" : "Feel",      "Track#" : 7,     "Track Name" : "Move On" }
  ,"000000006" : {"Album" : "Feel",      "Track#" : 6,     "Track Name" : "Nothing Free" }
  ,"000000007" : {"Album" : "Feel",      "Track#" : 5,     "Track Name" : "Naked Without You" }
  ,"000000008" : {"Album" : "Feel",      "Track#" : 4,     "Track Name" : "Someday" }
  ,"000000009" : {"Album" : "Feel",      "Track#" : 3,     "Track Name" : "Don't Make Me Love You" }
  ,"000000010" : {"Album" : "Feel",      "Track#" : 2,     "Track Name" : "How Could I? (Insecurity)" }
  ,"000000011" : {"Album" : "Feel",      "Track#" : 1,     "Track Name" : "Way I Feel" }
  ,"000000012" : {"Album" : "Feels Like Today",     "Track#" : 12,     "Track Name" : "Skin (Sarabeth)" }
  ,"000000013" : {"Album" : "Feels Like Today",     "Track#" : 11,     "Track Name" : "Oklahoma-Texas Line" }
  ,"000000014" : {"Album" : "Feels Like Today",     "Track#" : 10,     "Track Name" : "Holes" }
  ,"000000015" : {"Album" : "Feels Like Today",     "Track#" : 9,     "Track Name" : "Break Away" }
  ,"000000016" : {"Album" : "Feels Like Today",     "Track#" : 8,     "Track Name" : "The Day Before You" }
  ,"000000017" : {"Album" : "Feels Like Today",     "Track#" : 7,     "Track Name" : "Here's to You" }
  ,"000000018" : {"Album" : "Feels Like Today",     "Track#" : 6,     "Track Name" : "When the Sand Runs Out" }
  ,"000000019" : {"Album" : "Feels Like Today",     "Track#" : 5,     "Track Name" : "Fast Cars and Freedom" }
  ,"000000020" : {"Album" : "Feels Like Today",     "Track#" : 4,     "Track Name" : "Feels Like Today" }
  ,"000000021" : {"Album" : "Feels Like Today",     "Track#" : 3,     "Track Name" : "Then I Did" }
  ,"000000022" : {"Album" : "Feels Like Today",     "Track#" : 2,     "Track Name" : "Bless the Broken Road" }
  ,"000000023" : {"Album" : "Feels Like Today",     "Track#" : 1,     "Track Name" : "Where You Are" }
 ]
}
```


JSON Report - Example 4.

By default, all key:value pairs belonging to the output report detail line are arranged on the same line of the JSON output, resulting in long records. To break the output line so that all key:value pairs are arranged beneath each other on separate output records, specify the following option in the report definition.

- **OPTIONS: JSONINDENT(YES)**

Adding this option to the report definition in example 3. will produce the following JSON Output:

```
{"FileKit_Report" :
 [
    {"Album" : "Feel",
     "Track#" : 11,
     "Track Name" : "Flow"
    }
   ,{"Album" : "Feel",
     "Track#" : 10,
     "Track Name" : "Time"
    }
   ,{"Album" : "Feel",
     "Track#" : 9,
     "Track Name" : "Testify"
    }
   ,{"Album" : "Feel",
     "Track#" : 8,
     "Track Name" : "Down"
    }
   ,{"Album" : "Feel",
     "Track#" : 7,
     "Track Name" : "Move On"
    }
   ,{"Album" : "Feel",
     "Track#" : 6,
     "Track Name" : "Nothing Free"
    }
   ,{"Album" : "Feel",
     "Track#" : 5,
     "Track Name" : "Naked Without You"
    }
   ,{"Album" : "Feel",
     "Track#" : 4,
     "Track Name" : "Someday"
    }
   ,{"Album" : "Feel",
     "Track#" : 3,
     "Track Name" : "Don't Make Me Love You"
    }
   ,{"Album" : "Feel",
     "Track#" : 2,
     "Track Name" : "How Could I? (Insecurity)"
    }
   ,{"Album" : "Feel",
     "Track#" : 1,
     "Track Name" : "Way I Feel"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 12,
     "Track Name" : "Skin (Sarabeth)"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 11,
     "Track Name" : "Oklahoma-Texas Line"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 10,
     "Track Name" : "Holes"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 9,
     "Track Name" : "Break Away"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 8,
     "Track Name" : "The Day Before You"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 7,
     "Track Name" : "Here's to You"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 6,
     "Track Name" : "When the Sand Runs Out"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 5,
     "Track Name" : "Fast Cars and Freedom"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 4,
     "Track Name" : "Feels Like Today"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 3,
     "Track Name" : "Then I Did"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 2,
     "Track Name" : "Bless the Broken Road"
    }
   ,{"Album" : "Feels Like Today",
     "Track#" : 1,
     "Track Name" : "Where You Are"
    }
 ]
}
```

XML Output

The REPORT utility can also generate Extensible Markup Language (XML) from the COLUMNS: section field values.

The XML output will contain a "FileKit_Report" element with open/close tags for each output report detail line. These report detail line elements are child elements of a single element, "INPUT".

An XML open and close tag is generated for each field value in the output report detail line. These field tags have a name equal to the defined column field header, and are child elements of the "FileKit_Report" element.

Page and report break header and footer line definitions are applicable only to printable report output and so are ignored for XML output.

XML output is triggered when "XML" is selected as the output type in one of the REPORT Utility panels, or is specified as an operand on the REPORT command.

Examples

XML Report - Example 1.

Selecting output type XML and reducing the COLUMNS: field entries to "TRACK-NUM" and "NAME" for the first report definition described in example 1. for Insert Breaks, produces the following XML report output.

Note that special characters are represented by their "&" code name in the XML output. In the following output, occurrences of apostrophe (') within the field values are replaced with "'".

```
<?xml version="1.0"?>


<INPUT>
 <FileKit_Report>    <Track_>  11</Track_> <Track_Name>Flow                          </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>  10</Track_> <Track_Name>Time                          </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   9</Track_> <Track_Name>Testify                       </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   8</Track_> <Track_Name>Down                          </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   7</Track_> <Track_Name>Move On                       </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   6</Track_> <Track_Name>Nothing Free                  </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   5</Track_> <Track_Name>Naked Without You             </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   4</Track_> <Track_Name>Someday                       </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   3</Track_> <Track_Name>Don&apos;t Make Me Love You       </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   2</Track_> <Track_Name>How Could I? (Insecurity)     </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   1</Track_> <Track_Name>Way I Feel                    </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>  12</Track_> <Track_Name>Skin (Sarabeth)               </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>  11</Track_> <Track_Name>Oklahoma-Texas Line           </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>  10</Track_> <Track_Name>Holes                         </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   9</Track_> <Track_Name>Break Away                    </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   8</Track_> <Track_Name>The Day Before You            </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   7</Track_> <Track_Name>Here&apos;s to You                </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   6</Track_> <Track_Name>When the Sand Runs Out        </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   5</Track_> <Track_Name>Fast Cars and Freedom         </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   4</Track_> <Track_Name>Feels Like Today              </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   3</Track_> <Track_Name>Then I Did                    </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   2</Track_> <Track_Name>Bless the Broken Road         </Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>   1</Track_> <Track_Name>Where You Are                 </Track_Name> </FileKit_Report>


</INPUT>
```

XML Report - Example 2.

By default, literal strings (constants) specified in the COLUMNS section are omitted from XML output and, in order to align the values within output lines, all values are padded with blanks to the defined width of the display field.

The format of this XML output can be tweaked by specifying one or more of the following Boolean options in the report definition.

- **OPTIONS: XMLLITERALS(YES) XMLSTRIPALL(YES)**

XMLLITERALS(YES) will include string literals in the report detail line output, and XMLSTRIPALL(YES) will prevent padding field values with blanks.

Adding these 2 options to the report definition in example 1. will produce the following XML Output:

```
<?xml version="1.0"?>


<INPUT>
 <FileKit_Report>    <Track_>11</Track_> <Track_Name>Flow</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>10</Track_> <Track_Name>Time</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>9</Track_> <Track_Name>Testify</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>8</Track_> <Track_Name>Down</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>7</Track_> <Track_Name>Move On</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>6</Track_> <Track_Name>Nothing Free</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>5</Track_> <Track_Name>Naked Without You</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>4</Track_> <Track_Name>Someday</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>3</Track_> <Track_Name>Don&apos;t Make Me Love You</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>2</Track_> <Track_Name>How Could I? (Insecurity)</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>1</Track_> <Track_Name>Way I Feel</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>12</Track_> <Track_Name>Skin (Sarabeth)</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>11</Track_> <Track_Name>Oklahoma-Texas Line</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>10</Track_> <Track_Name>Holes</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>9</Track_> <Track_Name>Break Away</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>8</Track_> <Track_Name>The Day Before You</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>7</Track_> <Track_Name>Here&apos;s to You</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>6</Track_> <Track_Name>When the Sand Runs Out</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>5</Track_> <Track_Name>Fast Cars and Freedom</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>4</Track_> <Track_Name>Feels Like Today</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>3</Track_> <Track_Name>Then I Did</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>2</Track_> <Track_Name>Bless the Broken Road</Track_Name> </FileKit_Report>
 <FileKit_Report>    <Track_>1</Track_> <Track_Name>Where You Are</Track_Name> </FileKit_Report>


</INPUT>
```

XML Report - Example 3.

By default, all XML elements belonging to the output report detail line are arranged on the same line of the XML output, resulting in long records. To break the output line so that all elements are arranged beneath each other on separate output records, specify the following option in the report definition.

- **`OPTIONS: XMLINDENT(YES)`**

Adding this option to the report definition in example 2. will produce the following XML Output:

```
<?xml version="1.0"?>
<INPUT>
 <FileKit_Report>
   <Track_>11</Track_>
   <Track_Name>Flow</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>10</Track_>
   <Track_Name>Time</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>9</Track_>
   <Track_Name>Testify</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>8</Track_>
   <Track_Name>Down</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>7</Track_>
   <Track_Name>Move On</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>6</Track_>
   <Track_Name>Nothing Free</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>5</Track_>
   <Track_Name>Naked Without You</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>4</Track_>
   <Track_Name>Someday</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>3</Track_>
   <Track_Name>Don&amp;apos;t Make Me Love You</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>2</Track_>
   <Track_Name>How Could I? (Insecurity)</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>1</Track_>
   <Track_Name>Way I Feel</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>12</Track_>
   <Track_Name>Skin (Sarabeth)</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>11</Track_>
   <Track_Name>Oklahoma-Texas Line</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>10</Track_>
   <Track_Name>Holes</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>9</Track_>
   <Track_Name>Break Away</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>8</Track_>
   <Track_Name>The Day Before You</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>7</Track_>
   <Track_Name>Here&amp;apos;s to You</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>6</Track_>
   <Track_Name>When the Sand Runs Out</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>5</Track_>
   <Track_Name>Fast Cars and Freedom</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>4</Track_>
   <Track_Name>Feels Like Today</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>3</Track_>
   <Track_Name>Then I Did</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>2</Track_>
   <Track_Name>Bless the Broken Road</Track_Name>
 </FileKit_Report>
 <FileKit_Report>
   <Track_>1</Track_>
   <Track_Name>Where You Are</Track_Name>
 </FileKit_Report>
</INPUT>
```

BROWSE Output

When executed within the FileKit on-line environment (i.e. not in batch via FILEKITB), the REPORT utility can be used to open a Data Editor BROWSE view to display the formatted input records.

Only records or record segments with record-types matching those specified in the report definition, will be displayed. Furthermore, only *input-fields* whose names are referenced in the record definition will be selected for display.

For SMF or SDE segmented record input, only primary segment record-types and secondary segment record-types specified in a REPEAT: section will be displayed. Fields that are referenced in the report definition but belong to secondary segment record-types that are **not** specified in REPEAT, will be selected on the primary segment display. Option SELECTJOIN(NO) may be used to include segmants of all record-types specified in the report definition and so display selected fields in the segment to which they belong.

All record filtering criteria specified via the report definition or as parameters on the execution, will be applied to the input before records are displayed.

Page and report break header and footer line definitions are applicable only to printable report output and so are ignored for BROWSE output.

BROWSE output is triggered when "BROWSE" is selected as the output type in one of the REPORT Utility panels, or is specified as an operand on the REPORT command.

Examples

BROWSE Output - Example 1.

Selecting output type BROWSE for the first report definition described in example 1. for Filter Input Records, produces the following BROWSE display.

```
FileKit - Browse CBL.SMF.T014 using REPORT.D2024071.T155819      32764 V SEQ
 File Edit Actions Options Utilities Window SwapList Help  wS wR
Command> _                                                      Scroll> Csr
000000 *** Top of Data ***
Base:     SMF014_INPUT_or_RDBACK_Dataset    Variable(269,3390) Offset=0 Data elem
        zJOBNAME zRST                          zSPN     zPGN     DSN
        #7       #8                            #5       #6       #42
        <---+--> <---+----1----+----2-> <---+--> <---+--> <---+----1----+----2--
000001  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.DB.INI
000002  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI350.INI
000003  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.CBLI.INI
000004  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI350.TLIB
000005  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI.SITE.TLIB
000006  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.TLIB
000007  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.WINX
000008  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.WINX
000009  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.TLIB
000010  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 CBL.CBLI350.SMFMAP
000011  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000012  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000013  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000014  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000015  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000016  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000017  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000018  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000019  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000020  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000021  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000022  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
000023  JGE      2019/06/03 09:32:31.61 LGNPRC1  ADFMDF03 JGE.SELCOPYI.SMF.SDO
Se | Line=0 | Col=1 | Alt=0,0;0 | Size>28(P) | Recl=32764 | Fmt=V | Files=1 | V
```

Figure 9. Browse SMF Type 14 records - Dataset Usage by Job Name

BROWSE Output - Example 2.

Add the following option to the report definition in example 1. to prevent joining fields from secondary segments to the primary segment display.

- **`OPTIONS:  SELECTJOIN(NO)`**

Repeating the execution with output type BROWSE produces the following BROWSE display.

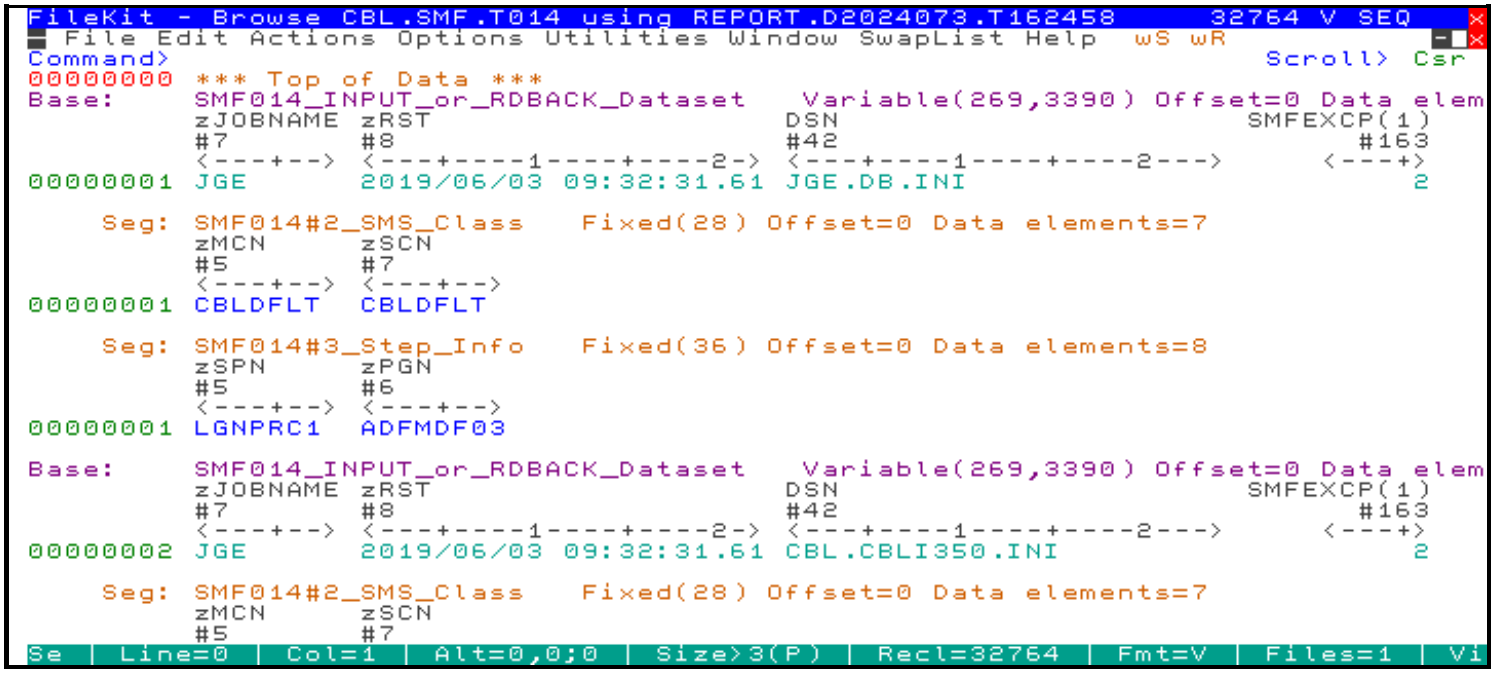


Figure 10. Browse SMF Type 14 records without SELECTJOIN - Dataset Usage by Job Name

REPORT Execution

This chapter details the methods that may be used to start the REPORT utility.

Record Input

The REPORT utility may process dataset record or DB2 table input using either of the following FileKit I/O methods:

- FILEIO
- Data Editor Browse

The FILEIO method is slightly more efficient, and uses the appropriate access method to input records sequentially. This is the default method adopted by the REPORT utility.

Data Editor Browse also inputs records sequentially but performs additional processing to allow scrolling forwards and backwards through the input records. This has the benefit of supporting a level of record pre-processing and exclusion before the main REPORT processing is performed. This method is triggered when input is the display of formatted records in the current FileKit Data Editor view, or if the BROWSE-EXIT section is present in the report definition.

The input method is of no real consequence to the user, but is mentioned here in order to explain the difference between use of the INIT-EXIT and BROWSE-EXIT sections in the report definition.

The following screen shot demonstrates execution of the REPORT utility to process records in the current Data Edit view using the report definition member, **ZZSRF0R1**. Input records will be processed using the Data Editor Browse method.

The Data Edit view displays browsed records from the sample "SZZSDIST.SAM2(ZZSDF1RE)" Formula 1 2019 results member, mapped using the "SZZSDIST.SDO(ZZSSF1RE)" structure. All records showing race positions outside the top 5 have been excluded and so will also be excluded from REPORT input processing.

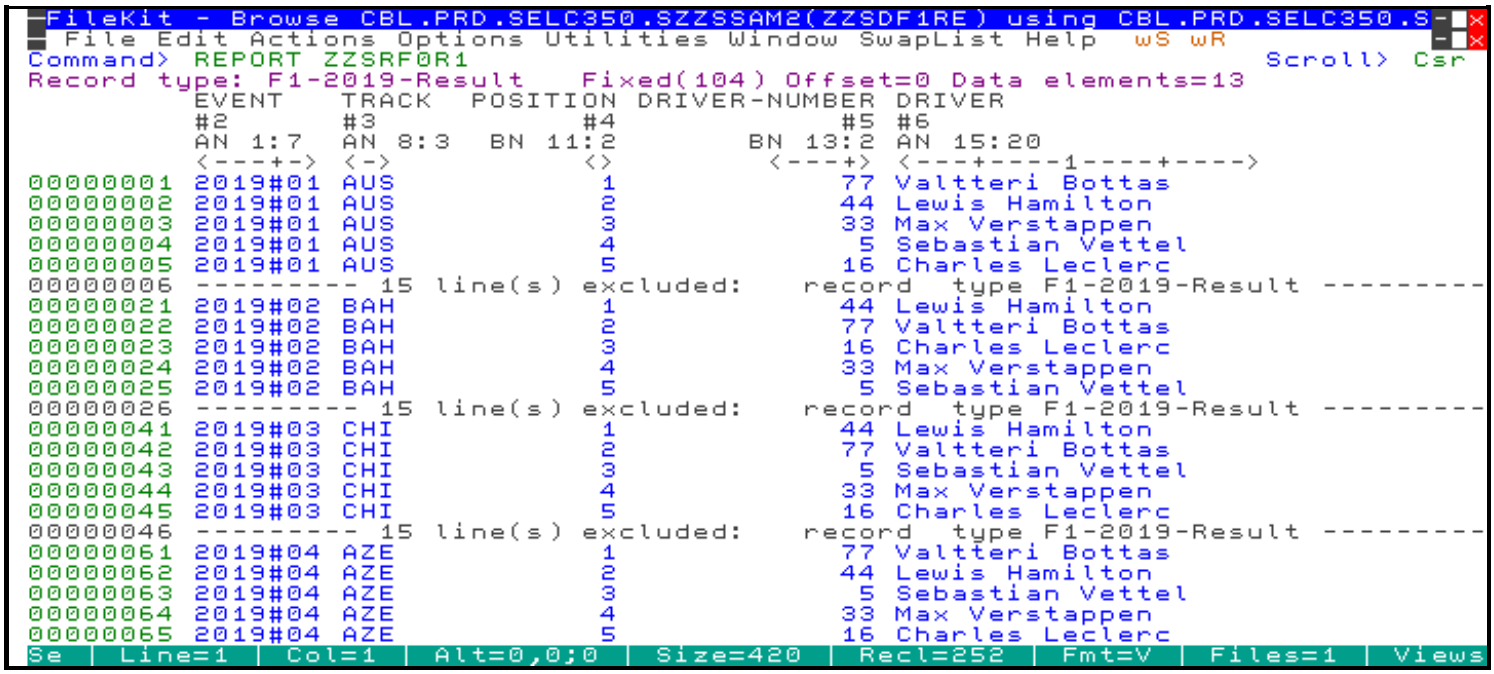


Figure 11. REPORT Data Editor Browse Processing for data in the current view.

Record Filtering

In some cases, it may not be desirable to include **all** input records or DB2 table rows in the output report. For this reason, the REPORT utility supports methods by which input records may be filtered before they are processed and included in the generated report output.

Record Filtering for SDE Record Input

Structure Data Edit (SDE) record input applies to any input from a data set source which does **not** contain SMF generated records. SDE record input relies on an accompanying structure to map fields within the record data. The structure may be a specific copybook or one generated by field mappings specified in the MAP section of the report definition.

SDE data set records may be filtered using the following:

Input Record Limit
An input limit may be specified using the ILIM report option, REPORT command ILIM operand or the REPORT Utility panels "*Input Limit*" field.

If specified, an input limit is applied **before** all other SDE data set record filtering occurs. Therefore, no record selection will occur for input records beyond the input limit record number.

FILTER Clause
A FILTER clause may be specified via the FILTER section of the report definition.

A FILTER clause includes one or more expressions, each comprising formatted record field names, operators, functions and/or constants. When applied to the (formatted or unformatted) input record data, the expressions return either a true (1) or false (0) result. This Boolean result determines whether the record is rejected or accepted and passed for further REPORT processing.

Unformatted Record Find String Matching
Find String(s) may be specified using the FIND report option, the REPORT command FIND operand or the REPORT Utility panels "*Find String*" field.

One or more comma separated Find Strings may be specified in a format described by search values under *"Record Filtering"*. If a match on **any** of the Find Strings is located at **any** position within the **unformatted** input record, then a true result (1) is returned and the record is passed for further REPORT processing.

Note that FILTER Clause and Unformatted Record Find String Matching are mutually exclusive record filtering techniques. If an attempt is made to use both techniques simultaneously, then error ZZSR065E is returned and execution of the REPORT utility is halted.

See "*SDE Dataset Processing*" in "*Appendix D. REPORT Logic Flow*" for an illustrated view of SDE record selection logic.

Record Filtering for SMF Record Input

IBM z/OS System Management Facilities (SMF) record input applies to input from a data set source which contains SMF generated records. The REPORT utility uses FileKit SMF record structures to map fields in the SMF records.

SMF data set records may be filtered using the following:

Input Record Limit
An input limit may be specified using the ILIM report option, REPORT command ILIM operand or the REPORT Utility panels "*Input Limit*" field.

If specified, an input limit is applied **before** all other SMF data set record filtering occurs. Therefore, no record selection will occur for input records beyond the input limit record number.

Low Date/High Date Timestamp
Every SMF generated record conatins a header that includes the date and time at which the record was written to the output buffer. With the exception of SMF type 2 (Dump Header) records, SMF dump data set records occur in ascending order of SMF record timestamp. Note that an SMF type 2 and type 3 record are usually the first and last records in an SMF dump data set and have a timestamp equal to the time and date the data set was created.

A low and/or high timestamp value may be specified to select only SMF records that were written on or later than a lower limit date and time, or written on or earlier than a higher limit date and time.

A **low date** limit may be specified using the SMFDATELO report option, the REPORT command DATELO operand or the SMF REPORT Utility panel "*Lo-Date/Time*" field. A **high date** limit may be specified using the SMFDATEHI report option, the REPORT command DATEHI operand or the SMF REPORT Utility panel "*Hi-Date/Time*" field.

A low date and/or high date may be specified as an absolute date and time or as a date relative to the current date as described by Timestamp Values under *"Record Filtering"*.

If the SMF record timestamp is before the low date threshold, then the record is rejected. Once an input record has been processed which has a timestamp **not** before the low date threshold and is **not** of SMF type 2 or type 3, then all subsequent input records are assumed to have a timestamp later than the low date threshold, so automatically pass low date testing.

Once low date processing is passed, record timestamps are tested against the high date threshold (if specified). If the timestamp value is later than the high date threshold, then no further input record processing will occur.

Low Date/High Date timestamp record filtering will occur before **Filter Clause** or **Content Match Criteria** processing.

FILTER Clause

A FILTER clause may be specified via the FILTER section of the report definition.

A FILTER clause includes one or more expressions, each comprising formatted record field names, operators, functions and/or constants. When applied to the (formatted or unformatted) SMF input record data, the expressions return either a true (1) or false (0) result. This Boolean result determines whether the record is rejected or accepted and passed for further REPORT processing.

Content Match Criteria

Content Match Criteria is comprised of one or more of the following field matching elements which each return either a true (1) or false (0) Boolean result:

- ◊ Unformatted Record Find String matching (FIND)
- ◊ SMF Record Job Name matching (JOBNAME)
- ◊ SMF Record System Id matching (SID)
- ◊ SMF Record Type matching (TYPES)
- ◊ SMF Record User Name matching (USERID)

A logical ("AND" or "OR") operation is performed between each of the Boolean results to return a final true or false result for all the Content Match Criteria.

The logical operation used to determine the Content Match Criteria result is specified by the SMFLOGIC report option, the REPORT command LOGIC operand or the SMF REPORT Utility panel "*Logic*" field. If no specified, the default operation is "OR".

Therefore, the result of Content Match Criteria is true (1) if either:

1. Logical operation **AND** is used and the result returned by **all** of the specified field matching elements is true (1).
2. If logical operation **OR** is used and at least **one** of the results returned by the specified field matching elements is true (1).

If a true result is returned by the Content Match Criteria, then the SMF record is passed for further REPORT generation processing.

Unformatted Record Find String Matching

Find String(s) may be specified using the FIND report option, the REPORT command FIND operand or the REPORT Utility panels "*Find String*" field.

One or more comma separated Find Strings may be specified in a format described by search values under *"Record Filtering"*. If a match on **any** of the Find Strings is located at **any** position within the **unformatted** input record, then a true result (1) is returned.

SMF Record Job Name Matching

A number of SMF record types contain a job name field **zJobName** at a fixed location within the record data. This fixed position may be different for each of the SMF record types. SMF records containing this field may be tested for a matching job name masks.

One or more comma separated job name masks may be specified using the SMFJOBNAME report option, the REPORT command JOBNAME operand or the SMF REPORT Utility panel "*Job Name*" field. Each job name mask is in a format described by search values under *"Record Filtering"*.

If an SMF record zJobName field contains a match on any of the supplied job name masks, then SMF Record Job Name matching will return a true result (1). Otherwise, if no match is found for any of the supplied job name masks or the SMF record does not contain a zJobName field, then a false result (0) is returned.

The following SMF records types contain a zJobName field:

004	010	017	025	034	040	061	064	067	080
005	014	018	026	035	042	062	065	068	110
006	015	020	030	036	060	063	066	069	118

SMF Record System Id Matching

All SMF record types contain a system identifier field **zSID** in the record header. This field may be used to test all SMF records for a matching system identifier mask.

One or more comma separated system identifier masks may be specified using the SMFSID report option, the REPORT command SID operand or the SMF REPORT Utility panel "*System Id*" field. Each system identifier mask is in a format described by search values under *"Record Filtering"*.

If an SMF record zSID field contains a match on any of the supplied system identifier masks, then SMF Record System Id matching will return a true result (1). Otherwise, if no match is found for any of the supplied system identifier masks, then a false result (0) is returned.

SMF Record Type Matching

All SMF record types contain an SMF record type field **zRTY** and some also contain a sub-type field **zSTY** in the record header. These fields may be used to test all SMF records for a matching type and optionally, a matching sub-type.

One or more comma separated record type, record sub-type and/or record type range may be specified using the SMFTYPES report option, the REPORT command TYPES operand or the SMF REPORT Utility panel "*Types*" field. Each record type, record sub-type and/or record type range is in a format described by SMF Type Values under *"Record Filtering"*.

If an SMF record contains a match on any of the supplied SMF record types, sub-types or record type ranges, then SMF Record Type matching will return a true result (1). Otherwise, a false result (0) is returned.

SMF Record User Name Matching

A number of SMF record types contain a user name field **zUserId** at a fixed location within the record data. This fixed position may be different for each of the SMF record types. SMF records containing this field may be tested for a matching user name masks.

One or more comma separated job name masks may be specified using the SMFUSERID report option, the REPORT command USERID operand or the SMF REPORT Utility panel "*User Name*" field. Each user name mask is in a format described by search values under *"Record Filtering"*.

If an SMF record zUserId field contains a match on any of the supplied user name masks, then SMF Record User Name matching will return a true result (1). Otherwise, if no match is found for any of the supplied user name masks or the SMF record does not contain a zUserId field, then a false result (0) is returned.

The following SMF records types contain a zUserId field:

004	014	020	030	035	042	062	065	068	110
005	015	025	032	036	060	063	066	069	118
006	017	026	034	040	061	064	067	080	119
010	018								

Note that FILTER Clause and Content Match Criteria are mutually exclusive record filtering techniques. If an attempt is made to use both techniques simiultaneously, then error ZZSR066E is returned and execution of the REPORT utility is halted.

See "*SMF Records Dataset Processing*" in "*Appendix D. REPORT Logic Flow*" for an illustrated view of SMF record selection logic.

Record Filtering for DB2 Table Input

DB2 table input applies to input from an IBM DB2 database table source. This is achieved by submitting a generated, or explicitly specified, DB2 SQL Query to the target DB2 sub-system in order to obtain a result table together with a description of its columns.

The column descriptions are used to generate a temporary structure with which the table row data is mapped.

DB2 table rows may be filtered using the following:

Input Row Limit

An input limit may be specified using the ILIM report option, REPORT command ILIM operand or the DB2 REPORT Utility panels "*Max*" field.

Alternatively, the DB2 result table definition operands specified via the report definition INPUT section or DB2-INPUT operands of the REPORT command, may include a FOR ROWS specification. Note that, if both an ILIM input limit and a FOR ROWS value is specified, then the FOR ROWS value is silently ignored.

If specified, an input limit value is included as a FETCH FIRST *nrows* clause in the SQL Query used to generate the DB2 result table. A FETCH FIRST clause will return the first *nrows* number of rows of the result table **after** any WHERE *search-clause* has been applied. This is different to data set record reporting where the input limit is

applied before all other record filtering.

DB2 result table operands also support a FROM ROW specification in the the report definition INPUT section or REPORT command DB2-INPUT operands. If an input limit is used, then FROM ROW *rownum* will add a value *rownum*-1 to the FETCH FIRST *nrows* value generated for the input limit specification. Therefater, the first *rownum*-1 rows of the DB2 result table will be rejected. For example, FROM ROW 101 FOR 10 ROWS will fetch 110 rows into the result table and select only 10 starting at row number 101.

FILTER Clause
A FILTER clause may be specified via the FILTER section of the report definition.

For DB2 format DB2 table or view row input, the FILTER section specifies a DB2 SQL query WHERE *search-condition*.

Alternatively, the DB2 result table definition operands specified via the report definition INPUT section, DB2-INPUT operands of the REPORT command or the DB2 REPORT Utility WHERE Clause panel, may provide a WHERE clause specification. Note that, if both a FILTER and WHERE clause specification exists, then processing continues but the FILTER is ignored and warning ZZSR064W (RC=4) is returned.

See "*DB2 Result Table Processing*" in "*Appendix D. REPORT Logic Flow*" for an illustrated view of DB2 row selection logic.

Search Values

REPORT utility content match filtering parameter **FIND** and, for SMF record input, parameters **SID**, **JOBNAME** and **USERID**, each support one or more alternate search string values.

A search string value entered for any of these parameters may be of one of the following formats:

Unquoted String
A string of characters without enclosing quotation mark (") or apostrophe (') symbols.

The string may contain any character other than a blank or a comma (","). Furthermore, it must not be one of the keyword operands supported by the Data Editor primary command "FIND" (e.g. PREV, NEXT, CHAR, WORD, etc.)

Since text in the SMF System ID, Job Name and User ID fields is in upper-case, the alpha characters in unquoted SID, JOBNAME and USERID search strings are always upper-cased before testing occurs.

When testing an unquoted search string against the record data, matching of alpha characters is case-insensitive. The only exception to this rule is when the search string includes a wildcard symbol. In this case, matching of alpha characters in the search string will be case sensitive. Because alpha characters in SID, JOBNAME and USERID parameter search strings are upper-cased, this only really effects FIND search strings.

For example, search strings `ABC` and `abc` are equivalent. However, for the **FIND** parameter only, the search strings `AB*` and `ab*` are **not** equivalent.

Quoted String
A string of characters enclosed within either quotation mark (") symbols or apostrophe (') symbols.

The string may contain any character but if it includes the same symbol that is used to enclose the string, then each occurrence of that symbol within the string must be escaped. The escaping symbol is identical to the escaped symbol itself. Therfore, each occurrence of the symbol within the string is represented by 2 occurrences of that symbol. For example, using apostrophes to enclose the string, `'It''s Peter O''Toole''s car.'` is equivalent to `"It's Peter O'Toole's car."` .

Like Unquoted strings, alpha characters in SID, JOBNAME and USERID quoted strings are automatically upper-cased before testing occurs and testing on alpha characters is case-insensitive unless the string contains a wildcard symbol.

For example, search strings `'A,B C'` and `"a,b C"` are equivalent. However, for the **FIND** parameter only, the search strings `'A,%B*'` and `'a,%B*'` are **not** equivalent.

Character Literal
A string of characters enclosed within either quotation mark (") symbols or apostrophe (') symbols and prefixed with the letter "C" or "c".

Unlike Unquoted and Quoted strings, no upper-casing of alpha characters is performed for SID, JOBNAME and USERID search strings and alpha character matching is always case-sensitive. However, character literals may include wildcard symbols.

For example, `C'A,B C'` and `C'a,b C'` are **not** equivalent.

Hexadecimal Literal
A string containing an even number of hexadecimal digits (0-F) enclosed within either quotation mark (") symbols or apostrophe (') symbols and prefixed with the letter "X" or "x".

A hexadecimal string is tested byte for byte against the hexadecimal representation of data in the input record. Wildcard symbols are not supported in hexadecimal literals.

For example, `X'81C2C340'`.

Picture String

A string of characters enclosed within either quotation mark (") symbols or apostrophe (') symbols and prefixed with the letter "P" or "p".

Picture strings are identical to character literals except that certain special character symbols within the string represent a generic group of characters as described below:

Symbol	Generic Group
P'='	Any character.
P'¬'	Any non-blank character.
P'.'	Any non-displayable character.
P'#'	Any numeric character, 0-9.
P'-'	Any non-numeric character.
P'@'	Any uppercase or lowercase alpha character.
P'<'	Any lowercase alpha character.
P'>'	Any uppercase alpha character.
P'$'	Any non-alphanumeric special character.

Like Character literals, no upper-casing of alpha characters is performed for SID, JOBNAME and USERID search strings and alpha character matching is always case-sensitive. However, wildcard symbols are **not** supported in picture strings.

For example, `P'A##-BC'` would match A21XBC, A01vBC and A99*BC but not A2SXBC, A012BC or Ap9*BC.

Regular Expression

A string of characters enclosed within either quotation mark (") symbols or apostrophe (') symbols and prefixed with the letter "R" or "r".

A regular expression provides powerful string pattern matching at the cost of rather complex syntax and potentially extended command processing time. Regular expressions are discussed in detail in the "*FileKit Text Editor*" publication.

Regular expression string pattern matching is precise and therefore case sensitivity and use of wildcard symbols is not applicable.

For example, `R'A:d+x'` would match the upper case character "A" followed by 1 or more numeric digits followed by character "x".

Wildcard Symbols

One or more wildcard symbols may be used in Unquoted, Quoted and Character Literal search strings.

Two wildcard symbols are supported as follows:

Symbol	Description
% (percent)	Represents exactly **one** character. For SID, JOBNAME and USERID search strings, this character may be any non-blank character. For FIND search strings, it may be any character (x'00'-x'FF').
* (asterisk)	Represents **zero or more** characters. For SID, JOBNAME and USERID search strings, these character may be any non-blank character. For FIND search strings, they may be any character (x'00'-x'FF').

Beware that use of wildcard symbols in **unquoted** and **quoted** strings forces alpha character matching to become **case-sensitive**.

Use of the asterisk ("*") wildcard symbol in a FIND search string may result in unintentional matches. Once a match has been found for characters preceding the "*" symbol, the characters that follow may be matched at **any** subsequent location within the record. For example, if the FIND search string is 'ABC*DEF', 'ABC' may be matched in the first three characters of the record and 'DEF' in the last three characters of the record.

This is not an issue with SID, JOBNAME and USERID search strings where the search is restricted to the 4 or 8 character length of the specific SMF record fields.

SMF Type Values

Applicable only to SMF record input, the REPORT utility content match filtering parameter, **TYPES** supports one or more alternate SMF record type values.

All SMF records have a record header which contains the SMF record type field **SMFRTY** and potentially a sub-type field **SMFSTY**. These are both integer fields and so SMF records are identified by their numeric type and, if applicable, numeric sub-type.

An SMF record type value may be of one of the following formats:

Record Type (*rectype*)
An integer value that identifies an individual SMF record type. If the record type has sub-types, then all sub-types will be selected.

For example, "30" identifies SMF type 30 ("Common Address Space Work") records and "119" identifies all sub-types of SMF type 119 ("TCP/IP Statistics") records.

Record Sub-Type (*rectype-subtype* or *rectype#subtype*)
Two integer values separated by a single hyphon ("-") or hash ("#") symbol and no intervening blanks identifies an individual SMF record sub-type.

For example, "110-2" identifies SMF type 110 ("CICS Transaction Server") sub-type 2 ("CICS statistics") records.

Record Type Range (*rectype:rectype*)
Two integer values separated by a single colon (":") symbol and no intervening blanks identifies a range of SMF record types.

The two integer values specify the first and last SMF record type in a range of SMF record types. Note that an SMF record sub-type may not be specified as the first or last value in the range. All sub-types belonging to SMF records included in the range will be selected.

For example, "60:69" identifies all SMF records of type 60 through 69 (VSAM related SMF records).

Timestamp Values

Applicable only to SMF record input, the REPORT utility content match filtering parameters, **DATELO** and **DATEHI**, each support one of two types of timestamp value specification.

Absolute Timestamp:
An absolute timestamp value is a date and time specification of length between 5 and 22 characters. The exact format of the full 22 characters is:

```
yyyy/mm/dd hh:MM:ss.nn
```

where:

yyyy	The 4-byte year
mm	The 2-byte month of year number
dd	The 2-byte day of month number
hh	The 2-byte hour of day number
MM	The 2-byte minute of hour number
ss	The 2-byte second of minute number
nn	A 2-byte hundredths of a second number

The date portion of the timestamp must be punctuated with slash ("/") symbols between the years, months and days values and contain no blank characters. Similarly, the time portion of the timestamp must contain no blank characters and must be punctuated with colon (":") symbols between the hours, minutes and seconds values and a dot/period (".") symbol between the seconds and hundredths of a second values. The date and time portions must be separated by a single blank character.

The timestamp may be truncated on the right to a minimum of 5 bytes ("*yyyy/*"). For a truncated absolute timestamp of length less than 22, all the truncated numeric digits will be set to "0" for a low threshold limit timestamp, or "9" for a high threshold limit timestamp.

For example, a low timestamp (DATELO) specification of "**2018/09**" is treated as "**2018/09/00 00:00:00.00**". A high timestamp (DATEHI) specification of "**2019/09/22 18**" is treated as "**2019/09/22 18:99:99.99**".

Relative Timestamp:

A relative timestamp value is specified as a signed integer number of days relative to the current date. This value must be zero (0) or negative since specifying a timestamp in the future is pointless for both low and high threshold limit timestamp filtering.

The derived absolute timestamp will contain only the date (i.e. timestamp is truncated to a length of 10 in the format "*yyyy/mm/dd*") and so the truncated time digits are set to "0" for a low threshold limit timestamp or "9" for a high threshold limit timestamp.

For example, if the current date is `2019/11/13` then a relative value of "`-28`" would be equivalent to absolute value "`2019/10/16 00:00:00.00`" for a DATELO timestamp or "`2019/10/16 99:99:99.99`" for a DATEHI timestamp.

Statistical Values

When generating printed report output, statistical values may be generated for one or more *input-field*, *compute-field* or *built-in-field* named in the report definition and defined as having a numeric or elapsed time data type. Additionally, a count of non-blank field values may be generated for fields of character data type.

Statistics values are generated at each control break level for which they have been requested. Breaks are defined in the BREAK section of the report definition.

When a break in the output of report detail lines is triggered, either as a result of a specific BREAK section definition or at the end of report output (the #GRAND break), then the REPORT utility will output the break footer lines for each break level affected by the change in break key field value. Note that, for the #GRAND break, all break levels are affected.

The control break footer lines displayed for a particular break level will contain the requested statistics values applicable to that break level. These values may be displayed in one or both of the following break line footers:

1. The break footer line output for the particular statistics type. The statistics values will appear aligned directly beneath the column of values to which they apply. These values are **column statistics**.
2. The text of the break footer line defined specifically by the break FOOTING operand. These values are **field statistics**.

Statistics Types

The types of statistical values that can be maintained by the REPORT utility are as follows:

TOTAL	Sum of all column/field values in the control group of detail lines.
NBTOTAL	Count of all non-blank column/field values in the control group of detail lines.
AVERAGE	Mean of all column/field values in the control group of detail lines.
NZAVERAGE	Mean of all **non-zero** column/field values in the control group of detail lines.
MINIMUM	Minimum of all column/field values in the control group of detail lines.
NZMINIMUM	Minimum of all **non-zero** column/field values in the control group of detail lines.
MAXIMUM	Maximum of all column/field values in the control group of detail lines.

For **column statistics**, the required statistics type is generated for the particular break level by specifying the corresponding keyword in the BREAK definition. Note that totals values are maintained and displayed by default for each break level and must be specifically suppressed if not required. This can be done for individual break levels using the keyword NOTOTAL, or for all break levels using option NOTOTALS.

For **field statistics**, the required statistics type is generated for the particular break level footing line output, by specifying the corresponding keyword as an option on the field name in the print expression. For example, :DURATION (TOTAL) will output the sum of all values in the break control group belonging to the *compute-field*, DURATION, as opposed to the last value of DURATION in the control group.

Statistics Example

This example generates a report from SMF log records of type 119 (TCP/IP statistics), sub-type 2(TCP Connection Termination).

Report Definition Input - **ZZS.ZZSSAM1(ZZSRS008)**:

The report definition has column *input-fields* and *compute-field* (DURATION) as described in in examples 2. for Create New Fields.

The BREAK and STATISTICS sections, displayed below, define 3 break levels, the types of statistics values, and the columns for which statistic values will be generated.

The 3 break levels correspond to changes in the TCP resource name (level 2), changes in the connection start date for the named resource (level 3) and finally, the #GRAND break for **all** resource connections (level 1). When one of these break levels are triggered, breaks with a higher break level number are also triggered. Breaks are processed in order of descending break level number. Thus, when the #GRAND break is triggered at end-of-report, break level 3 is processed first, then break level 2 and then finally break level 1.

```
BREAK:
 /* Break level 1 - The end-of-report break */
   #GRAND      SPACEBEFORE(2)

 /* Break level 2 */
   SMF119#02_TCP_Connection_Termination.zRName                        \
       HEADING(                                                       \
               <newline> 'TCP/IP Resource:' zRNAME (STRIP,RIGHT,10) \
               <newline> '----------------------------'             \
              )    REPEAT   /* Repeat heading on each new page. */   \
       TOTAL                /* TOTAL is default. */                  \
       SPACEAFTER( PAGE )   /* New page following break lines. */

 /* Break level 3 */
  SMF119#02_TCP_Connection_Termination.zConnectStart  10              \
       COLHEAD              /* Repeat column headings. */             \
       AVERAGE              /* Averages. */                           \
       NZAVERAGE            /* Averages of non-zero values. */        \
       MAXIMUM              /* Maximums. */                           \
       MINIMUM              /* Minimums. */                           \
       NZMINIMUM            /* Minimums of non-zero values. */        \
       SPACEBEFORE(1) SPACEAFTER(1)                                   \
       FOOTING(                                                       \
               <newline> ' -- End of' zRNAME (STRIP)                 \
                         ' statistics for'  zConnectStart (10) '--' \
                         2  'Total Duration:'   :Duration (TOTAL)    \
               <newline>                                              \
              )

STATISTICS:
  SMF119#02_TCP_Connection_Termination.zInBytes
  SMF119#02_TCP_Connection_Termination.zOutBytes
  :DURATION
```

Report Output:

The following is the last page of the printed report generated for this report definition.
Break lines containing column statistics values are highlighted in **blue**, break FOOTING lines containing field statistics values are highlighted in **green**.

```
12024/03/15 11:02                                                                PAGE:    9
              TCP/IP Connection Durations by Resource Name on: 2019/05/07 16:23:56.37

                                                    Connection
                                                      Duration Inbound  Outbound Termination
Resource Connection Start       Connection End      HHH:MM:SS.SS Bytes    Bytes    Description
-------- ---------------------- ------------------- ------------ -------- -------- --------------

TCP/IP Resource:    RXSERVE
----------------------------
RXSERVE  2019/05/07 15:23:29.77 2019/05/07 15:23:56.37  00:00:26.60      143       40 App_Close
RXSERVE  2019/05/07 15:23:30.36 2019/05/07 15:23:56.49  00:00:26.13        0        0 App_Close
RXSERVE  2019/05/07 09:02:36.56 2019/05/07 09:02:41.86  00:00:05.30      145       42 App_Close
RXSERVE  2019/05/07 09:02:37.26 2019/05/07 09:02:41.97  00:00:04.71        0        0 App_Close
RXSERVE  2019/05/07 09:04:08.40 2019/05/07 09:04:11.16  00:00:02.76      143       40 App_Close
RXSERVE  2019/05/07 09:04:08.53 2019/05/07 09:04:11.28  00:00:02.75        0        0 App_Close

                                                    ------------ -------- --------
    == Totals for 2019/05/07 (6 Items)                00:01:08.25      431      122
       Average Value                                  00:00:11.38       72       20
       Maximum Value                                  00:00:26.60      145       42
       Minimum Value                                  00:00:02.75        0        0
       Average of NON-ZERO Values                     00:00:11.38      144       41
       Minimum of NON-ZERO Values                     00:00:02.75      143       40

 -- End of RXSERVE  statistics for 2019/05/07 --  Total Duration:  00:01:08.25


                                                    Connection
                                                      Duration Inbound  Outbound Termination
Resource Connection Start       Connection End      HHH:MM:SS.SS Bytes    Bytes    Description
-------- ---------------------- ------------------- ------------ -------- -------- --------------
RXSERVE  2019/05/09 08:36:50.66 2019/05/09 08:36:55.73  00:00:05.07      166       55 App_Close
RXSERVE  2019/05/09 08:36:50.80 2019/05/09 08:36:55.84  00:00:05.04        0        0 App_Close
RXSERVE  2019/05/09 08:36:29.22 2019/05/09 08:36:31.65  00:00:02.43      166       55 App_Close
RXSERVE  2019/05/09 08:36:29.36 2019/05/09 08:36:31.76  00:00:02.40        0        0 App_Close

                                                    ------------ -------- --------
    == Totals for 2019/05/09 (4 Items)                00:00:14.94      332      110
       Average Value                                  00:00:03.74       83       28
       Maximum Value                                  00:00:05.07      166       55
       Minimum Value                                  00:00:02.40        0        0
       Average of NON-ZERO Values                     00:00:03.74      166       55
       Minimum of NON-ZERO Values                     00:00:02.40      166       55

 -- End of RXSERVE  statistics for 2019/05/09 --  Total Duration:  00:00:14.94


                                                    ------------ -------- --------
  ==== Totals for RXSERVE  (10 Items)                 00:01:23.19      763      232


                                                    ============ ======== ========
====== Grand Totals (159 Items)                      27:21:55.78   339892 11878642
                                                    ============ ======== ========
```

Break Lines

Operands specified on the break definitions in the BREAK section and options specified in the OPTIONS section determine which break lines appear in the display for each break level.

The order in which break footer lines appear is fixed as illustrated in the following table. The table also shows the REPORT definition keyword(s) that govern the presence and contents of a break line, and the break line decription.

Break Line Sequence	BREAK Keyword	OPTIONS Keyword	Description
Blank Lines	SPACEBEFORE	-	A number of blank lines written afer the last column detail line. (Default 0)
Underline	-	BRKULINE	Underline of the column values for which column statistics values are generated. The underline is comprised of hyphon ("-") symbols or, for the #GRAND break, equals ("=") symbols, and occupies the display width of the column.
Totals	TOTAL() NOTOTAL	BRKTOTALS GRANDTOTAL TOTALS NOTOTALS	Totals line containing text defined by a *print-expression*, followed by the sum column statistics values. If NBTOT has been specified for a COLUMN section entry, then the count of non-blank values for the column will also be displayed as a column statistics value in the totals line.
Averages	AVERAGE()	-	Averages line containing text defined by a *print-expression*, followed by the mean column statistics values.
Maximums	MAXIMUM()	-	Maximums line containing text defined by a *print-expression*, followed by the maximum column statistics values.
Minimums	MINIMUM()	-	Minimums line containing text defined by a *print-expression*, followed by the minimum column statistics values.
Non-Zero Averages	NZAVERAGE()	-	Non-zero Averages line containing text defined by a *print-expression*, followed by the mean of non-zero values column statistics values.
Non-Zero Minimums	NZMINIMUM()	-	Non-zero Minimums line containing text defined by a *print-expression*, followed by the minimum of non-zero values column statistics values.
#GRAND Underline	-	BRKULINE	Underline of the column statistics values. Applicable only to the #GRAND break at the end of the printed report, the underline is comprised of equals ("=") symbols, and occupies the display width of each column statistics column.
Footing	FOOTING()	-	General purpose break footer line of text defined by a *print-expression*. This text may contain any number of field statistics values, of any statistics type. For example, `COST (TOTAL)` in *print-expression* will include the totals value for field name "COST" in the generated text.
Blank Lines	SPACEAFTER	-	A number of blank lines written afer the last break footing lines. One or more of these lines may be suppressed if a new page is started. (Default 1)

Notes:

1. The format of *print-expression* is described in Print Expression.
2. See BREAK section for the default *print-expression* used for TOTALS, AVERAGE, MAXIMUM, MINIMUM, NZAVERAGE and NZMINIMUM break lines.
3. Break lines containing column statistics values will be displayed on a separate line immediately following the statistics line text if the text would otherwise be overlayed by the first statistics column value.
4. **<NEWLINE>** may be used in the *print-expression* to split text onto multiple break lines.

Column Statistics

By default, statistics values are generated for all fields specified in the COLUMNS section that have a numeric data type.

For an *input-field*, the data type is obtained from the mapping structure and, for a *compute-field*, it is the data type of the value assigned to the field following the first execution of the COMPUTE REXX statements. All REPORT *built-in-fields* have an established data type.

The data type assigned to field may be overridden using the "**CHAR**", "**NUM**" or "**TIME**" operand on the field entry specification in the COLUMNS or REQUIRED section. For example, you may want the REPORT utility to generate statistics values for character fields which contain character representation numeric data. Similarly, if a *compute-field* is not to be treated as numeric despite it being assigned a numeric value on the first call to the COMPUTE routine.

To prevent REPORT from producing unwanted column statistics, include statistics for specific columns, or include column fields of non-numeric data type, a STATISTICS section should be included in the report definition. This section is used to name the column fields for which column statistics will be generated and displayed.

Column Value State

Individual column values on which the statistical values are generated will be in one of the following states:

VALID

The column value is displayed in full and is of the correct data type (NUM or TIME).

A **numeric** (NUM) field value may have one of the numeric source data types described in the "*Data Editor (SDE) Manual*" for the "CREATE STRUCTURE" primary command. For numeric character fields (including *compute-fields*), the value must be comprised of numeric digits only, and any of the following:

- ◊ A single decimal point symbol (".")
- ◊ A single leading numeric sign symbol, plus ("+") or minus ("-").
- ◊ A single exponent symbol ("E" or "e"), potentially followed by a numeric sign symbol, plus ("+") or minus ("-"), to express a positive or negative exponent.

An **elapsed time** (TIME) field value may have one of the TIME data types described in the "*Data Editor (SDE) Manual*" for the "CREATE STRUCTURE" primary command. For character fields (including *compute-fields*), elapsed time may be a number of seconds only, a number of minutes and seconds, or a number of hours, minutes and seconds, with or without a fraction of a second specification. The value is in the following format:

`[[`*hours*`:]`*minutes*`:]`*seconds*`[.`*fraction*`]`

The *hours*, *minutes* and *seconds* value may be non-normalised (i.e. any number of digits and any integer value). The colon (":") symbol must be used to separate *hours*, *minutes* and *seconds* values, and the period/dot (".") symbol used to specify fraction of a second (*fraction*).

VALID column values are included in column statistics and field statistics calculations.

INVALID

For an *input-field* of numeric or TIME data type, the input data is not in a format which is consistant with the source field data type. For character fields (including *compute-fields*), the value is not in the required format as described for VALID values.

INVALID column values are excluded from column statistics and field statistics calculations.

HIDDEN

HIDDEN column values are those which belong to column detail lines that have been suppressed using the DETAIL(*nlines*) option. DETAIL specifies the maximum number of detail lines (*nlines*) to be reported in each control break group. Output of detail lines that would exceed this maximum are suppressed. The DETAIL option also supports operand **ALL** or **DISPLAY**.

If DETAIL(ALL) is specified (the default), then HIDDEN column values will be included in column statistics and field statistics calculations.

If DETAIL(DISPLAY) is specified, then HIDDEN column values will be excluded from column statistics and field statistics calculations.

TRUNCATED

TRUNCATED column values are VALID column values that cannot be displayed in full due to an insufficient default or specified column width.

The NUMTRUNC option determines whether the display area containing a TRUNCATED value is filled with the number truncation filler character (the default), or the value is abbreviated to fit within the display area. If an abbreviated value is displayed, then the original, unabbreviated value will be used in column statistics and field statistics calculations. The NUMTRUNC option also supports operand **INCLUDE** or **EXCLUDE**.

If NUMTRUNC(INCLUDE) is specified (the default), then the original value for a TRUNCATED column value displayed as filler characters, will be included in column statistics and field statistics calculations.

If NUMTRUNC(EXCLUDE) is specified, then TRUNCATED column values displayed as filler characters will be excluded from column statistics and field statistics calculations.

DUPLICATE

A DUPLICATE column value is a VALID *input-field* column value that appears in consecutive report detail lines due to it not having been reset. This occurs when a REPEAT section is used to specify a record-type that will trigger output of a report detail line, but no RESET section entry exists for that record-type. When output is triggered for input data of this record-type, the detail line *input-field* values are not reset. If these values are not then updated on input of a subsequent record or record segment, then they will be duplicated in the next output detail line.

The NUMDUP option supports operand **INCLUDE** or **EXCLUDE**.

If NUMDUP(INCLUDE) is specified (the default), then DUPLICATE column values will be included in column statistics and field statistics calculations.

If NUMDUP(EXCLUDE) is specified, then DUPLICATE column values will be excluded from column statistics and field statistics calculations.

EQUAL

An EQUAL column value is a VALID column value that has the same value as that in the previous detail line. Unlike DUPLICATE column values, which must be *input-field* values, EQUAL column values may also be *compute-field* or *built-in-field* values. Also, an EQUAL column value may be one where the value has been reset following detail line output, but then set to exactly the same value by input data processed before output of the next detail line.

By default, EQUAL column values will be included in in column statistics and field statistics calculations.

If BLANKIFEQUAL(YES) has been specified, then EQUAL column values will be displayed as blanks and option NUMBLANK determines whether EQUAL column values are included in statistics calculations.

If NUMBLANK(INCLUDE) is specified, then EQUAL column values that have been replaced with blanks will be included in column statistics and field statistics calculations.

If NUMBLANK(EXCLUDE) is specified (the default), then EQUAL column values that have been replaced with blanks will be excluded from column statistics and field statistics calculations.

Statistics Value Abbreviation

Statistics values are displayed in a designated display area.

For column statistics, this is an area below the column of values to which they apply. The display area width is the larger of that defined for the column values and that required to display the column header. For field statistics, the display area is that defined for the print expression element output. The default is the field value width.

Option SHORTSTATS determines whether a statistics value will be abbreviated if the display area width is not large enough to display the statistics value.

If SHORTSTATS(YES) is supplied (the default), then the value will be shortened to fit within the display area width. For values that contain an exponent, the mantissa is shortened. For values without an exponent, non-significant and then least significant numeric digits are removed and a multiplier suffix added if necessary. If this means loss of significant digits, then an inequality symbol is prefixed to the value. For example, for a display area of with 6, the value "-1.234567E16" would display as "-1.2E16", and the value "123456780" would display as ">123M".

If SHORTSTATS(NO) is supplied or abbreviation of the value is not possible in the available display width, then the display area is filled with the number truncation filler character defined by the NUMTRUNC option (default "*").

Report Panels

The REPORT utility may be run in the foreground using FileKit panels. The same panels may be used to generate a batch job template or an equivalent REPORT primary command.

The REPORT utility panels are accessed via the "Print/Report Features Menu" on option 11. of the FileKit primary options menu. Alternatively, this menu may be opened by executing primary command REPORT with no operands.

A user should select the REPORT utility panel option based on the the type of source data input. The supported input types are:

1. Data set input of any organisation
2. DB2 table input
3. Data set input containing SMF output records

All Report Utility panels are interactive panel window. (See "*Interactive Panel Windows*" in the "*FileKit Reference and User Guide*" for features that are common to all windows of this type.)

Primary Commands:

The following primary commands are common to each of the REPORT utility panels:

CLI | CMX
Generate a REPORT primary command with operand values that correspond to values entered in the panel fields. The command string is displayed as editable text in a new Text Editor window view and in a format suitable for execution using the ACTION key (shift-F4). (See "Command File Execution".)

Edit
Open a Text Editor window view to edit the Report Definition data set, library member or HFS/ZFS file specified in the Report Definition DSN/Path/Member panel input fields. The report definition text may be updated and saved if necessary. If the report definition file does not exist, then it will be created when changes are saved.

JCL
Generate a batch procedure containing skeleton JCL and REPORT primary command SDEIN input. The JCL DD statements and REPORT operand values correspond to values entered in the panel fields. The job procedure is displayed as editable text in a new Text Editor window view. (See "Batch Execution".)

Panel Input Fields:

The following input fields are common to each of the REPORT utility panels:

Report Definition:
Input fields which together identify a single, sequential or VSAM data set, GDG relative generation, HFS/ZFS file or PDS/PDSE library member from which the report definition control statements will be obtained.

Primary command **EDIT** (or **E**) may be run to edit this report definition file.

DSN/Path>
Identifies the fully qualified data set name of a sequential, VSAM, GDG or library data set or the file path of an HFS/ZFS file. An HFS/ZFS file path may be specified in full from the root directory or as path relative to the user's OMVS present working directory (displayed using primary command USS PWD).

Data set names beginning with "." (dot) will be treated as having the user's DSN prefix as defined by the User INI variable System.UserDSNPrefix.

A selectable list of data sets or HFS/ZFS files will be presented if the value entered contains wildcards characters "*" (asterisk) or "%" (percent).

Member>
If the DSN/Path> field contains the DSN of a PDS/PDSE library, then this field specifies the name of a library member. Otherwise, if DSN/Path> contains the DSN of a GDG, then this field specifies the relative generation number of a GDS (e.g. 0, -1, -10).

For a library data set, a selectable list of members will be presented if no member name is specified or a member name mask is entered. i.e. a member name containing wildcards characters "*" (asterisk) and/or "%" (percent). For a GDG, the relative generation 0 is used if no value is entered.

Options:

Run Type> B | C | F

Specifies "B", "C" or "F" for the action to be performed by the panel when the <Enter> key is pressed.

B	Generate a batch job containing JCL which executes FILEKITB with REPORT command input and write the report output to DD SDEOUT. See "Batch Execution". On completion, the generated job text is displayed in a Text Editor edit window. If necessary, the job may be amended, submitted to batch (SUBMIT) and optionally saved to DASD.
C	Generate a REPORT primary command. On completion, the generated REPORT command is displayed in a Text Editor edit window view in a format suitable for execution using the ACTION key (default shift-F4). See "Command File Execution". The REPORT command may also be copied to the user's HOME command file for execution at a later date.
F	Immediately execute the REPORT utility in the FileKit foreground and generate the report output. The report output will be created in storage and displayed in a Text Editor window view. It may then be saved to DASD, e.g. using edit primary commands CREATE, REPLACE or SAVE *newdsn*. Note that REPORT processing may require a large amount of storage and, if many records are to be processed, may take some time to complete. If the expected report output is potentially larger than the available TSO region size or more than a few thousand input records are to be processed then consider executing the REPORT utility in batch (run type "B").

Output Type> B | C | J | P | X

Specifies "B", "C", "J", "P" or "X" to identify the format of the report output generated when the REPORT utility is executed.

B	Opens a FileKit online **Browse** session for the formatted record data. Only record-types and fields identified in the COLUMNS or REQUIRED sections of the report definition are displayed. The fields will be displayed grouped together by their source record-type mapping and so not strictly in accordance with the order specified in the COLUMNS section of the report definition. Furthermore, if records are segmented, then unless a secondary segment mapping is repeated within the record, all the secondary segment fields will appear on the same line as the primary segment fields. Browse will also define a permanent user default display format for the record-types involved, meaning any future online browse will display only the selected fields. To revert to the default just type "SEL *" on the command line of the browse session. Report definitions sections other than BLANKWHENZERO, COLUMNS, FILTER, MAP, REPEAT and REQUIRED have no effect on **BROWSE** output.
C	**Comma Separated Variable (CSV)** output suitable for loading into various external formats such as a database table or spreadsheet. The first row will contain the column headings as defined by your report definition file.
J	**JavaScript Object Notation (JSON)** output.
P	Standard **Printed** report output with page formatting (headings, footings and control breaks). Printed output may also generate column totals and other statistical values based on input field data.
X	**Extensible Markup Language (XML)** output.

Page Depth>

For printed report output only, this value specifies the number of lines to be printed per page. This value will override a value specified by the PAGEDEPTH option in the report definition.

If left blank or specified as "0" (zero), and if the PAGEDEPTH option does not exist in the report definition, then the value assigned by the PAGEDEPTH Data Editor option will be used.

Type "**SD QUERY PAGEDEPTH**" to query your current Data Editor page depth value and "**SD SET PAGEDEPTH** *n*" to set it. (See "*PAGEDEPTH - SET/QUERY/EXTRACT Option*" in the "*FileKit Data Editor*" publication.)

Formatted Record Report

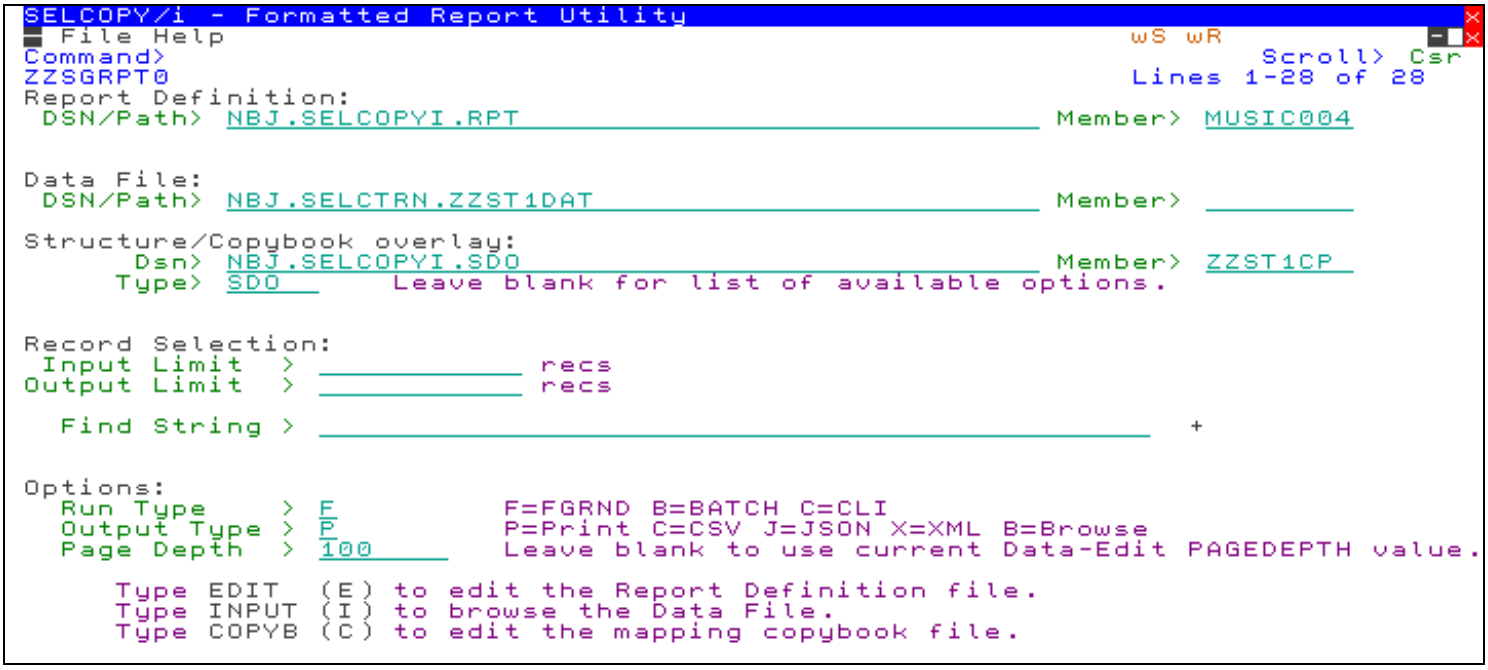


Figure 12. Formatted Records REPORT Utility Panel (=11.2).

Overview:

The Formatted Record Report Utility panel (ZZSGRPT0) is the FileKit display interface to the REPORT Utility where input is to be sourced from a data set, library member or HFS/ZFS file. For input file sources that contain records written by IBM System Management Facilities (SMF), the SMF Formatted Record Report Utility panel should be used instead.

To open the "Formatted Record Report Utility" panel, first select option 11. "Print/Report" from the FileKit primary options menu to open the "Print/Report Features" menu and then select option 2. "Report". Alternatively, simply enter the fast path "=11.2" from any FileKit command prompt.

Primary Commands:

In addition to commands CLI (or CMX), Edit and JCL which are common to each REPORT utility panel, the Formatted Record Report Utility panel supports the following primary commands:

COPYB
Open a Text Editor window view to edit the copy book dat set or library member specified in the Structure/Copybook overlay DSN/Member panel input fields. The copy book text may be updated and saved if necessary. If the copy book does not exist, then it will be created when changes are saved.

Input
Open a Data Editor window view to browse the input data records data set, library member or HFS/ZFS file specified in the Data File DSN/Path/Member panel input fields. The record data in the browse display will be formatted using the copy book specified in the Structure/Copybook overlay DSN/Member panel input fields. If no copy book structure is specified in these fields, then an error is returned.

Panel Input Fields:

By default, field entries are populated with arguments and options that were entered the last time the panel was used.

Report Definition:
Report definition input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Report Definition fields.

Data File:
Input fields which together identify an existing sequential or VSAM data set, GDG relative generation, HFS/ZFS file or PDS/PDSE library member containing the report source data records. This data source will override the input data source provided in the INPUT section of the report definition.

Primary command **INPUT** (or **I**) may be run to browse the formatted input data as mapped by the specified structure/copy book.

DSN/Path>
Identifies the fully qualified data set name of a sequential, VSAM, GDG or library data set or the file path of an HFS/ZFS file. An HFS/ZFS file path may be specified in full from the root directory or as path relative to the user's OMVS present working directory (displayed using primary command USS PWD).

Data set names beginning with "." (dot) will be treated as having the user's DSN prefix as defined by the User INI variable System.UserDSNPrefix.

A selectable list of data sets or HFS/ZFS files will be presented if the value entered contains wildcards characters "*" (asterisk) or "%" (percent).

Member>
If the DSN/Path> field contains the DSN of a PDS/PDSE library, then this field specifies the name of a library member. Otherwise, if DSN/Path> contains the DSN of a GDG, then this field specifies the relative generation number of a GDS (e.g. 0, -1, -10).

For a library data set, a selectable list of members will be presented if no member name is specified or a member name mask is entered. i.e. a member name containing wildcards characters "*" (asterisk) and/or "%" (percent). For a GDG, the relative generation 0 is used if no value is entered.

Structure/Copybook overlay:
Input fields which together identify an input record formatting structure.

This structure will override a structure definition provided via a USING operand in the INPUT section of the report definition. Providing a record formatting structure will also override use of record field mappings defined by a MAP section in the report definition.

The fields identify the name of an existing data set or PDS/PDSE library member containing one or more record mapping structures that will be used to map the layout of input data records. The structure input may be of any one of the following types:

- ◊ A FileKit SDO structure. (May contain a number of different record-type mappings.)
- ◊ A COBOL copy book containing data description source.
- ◊ An Assembler source module containing DSECT definitions.
- ◊ A PL1 %INCLUDE directive source member containing data declaration structures.
- ◊ SYSADATA output generated by the assembley of an assembler source using the HLASM (High Level Assembler) program, or generated by the compilation of a COBOL or PL1 source using the Enterprise COBOL or Enterprise PL1 compiler.

DSN>
Identifies the fully qualified data set name of a sequential data set or PDS/PDSE library containing the structure or copybook source.

Data set names beginning with "." (dot) will be treated as having the user's DSN prefix as defined by the User INI variable System.UserDSNPrefix.

A selectable list of data sets will be presented if the value entered contains wildcards characters "*" (asterisk) or "%" (percent).

Member>
If the DSN> field contains the DSN of a PDS/PDSE library, then this field specifies the name of a library member.

A selectable list of members will be presented if no member name is specified or a member name mask is entered. i.e. a member name containing wildcards characters "*" (asterisk) and/or "%" (percent).

Type>
Specifies the type (ADATA, ASM, COBOL, PL1 or SDO) of structure/copybook identified by the DSN> and Member> fields.

Record Selection:
Fields which specify selection criteria by which input data records may be filtered. Only records that satisfy the specified selection criteria are selected for reporting.

See Record Filtering for full details on the relationship between input limit, output limit and content match criteria.

Input Limit>
The **Input Limit>** input field value specifies the **maximum** number of records that may be read from the input file. This value will override a value specified by the ILIM option in the report definition.

Each input record is processed sequentially until this input record threshold is reached.

The input limit includes records which may subsequently be excluded from REPORT processing by a filter clause specified via a FILTER section in the report definition or, alternatively, via specification of find search strings.

An input limit of "0" (zero) implies no input record limit and is set by default when no input limit is supplied and no ILIM option is set in the report definition.

Output Limit>
The **Output Limit>** input field value specifies the **maximum** number of detail line records that may be written to the output report. This value will override a value specified by the OLIM option in the report definition.

Once the number of output report detail lines reaches this limit, no further input records will be processed.

An output limit of "0" (zero) implies no output record limit and is set by default when no OLIM operand value is supplied and no OLIM option is set in the report definition,

Find String>
Find String> field input specifies one or more comma separated search strings. These search string values will override values specified by the FIND option in the report definition for **Unformatted Record Find String matching**.

The format of a search string is described by search values under *"Record Filtering"*.

If a match on **any** of the search strings is located at **any** position within an unformatted input record, then Unformatted Record Find String matching will return a true result (1) and the record will be passed for REPORT processing. Otherwise a false result (0) is returned.

For example, the following input will set a true condition if the unformatted record contains any one of the strings "Belfast", "Cardiff", "Edinburgh" or "London". at any position within the record data.

```
Find String> c'Belfast', c'Cardiff', c'Edinburgh', c'London'
```

Specifying a search string is invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a find search string exist, then the error message ERR065E is returned.

The panel entry field for **Field String>** displays only the first 45 characters of any input. To enter or display search values that extend beyond this display length, position the cursor in the input field area and press shift-F2 (**EXPAND**). A Text Editor window will be displayed allowing entry of FIND search values that may stream multiple text edit lines.

Options:
Option input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Option fields.

DB2 Report

For reports generated from DB2 table data, input may be an existing named TABLE created in the local (or a remote) DN2 sub-system, or a result table generated by DB2 from an existing named VIEW or user supplied SQL query statement.

FileKit has three separate DB2 report panels, each providing an alternative method by which the DB2 table source may be specified. These are:

1. A named TABLE or VIEW.
2. An SQL Query entered directly via a panel input field.
3. An SQL Query supplied via an input file.

To access the DB2 Report panels, first select option 11. "Print/Report" from the FileKit primary options menu to open the "Print/Report Features" menu and then select option 3. "DB2 Report" to open the "Create a Report using DB2 Table Data" menu. The required DB2 report panel may be launched from this menu.

DB2 Report - Table/View

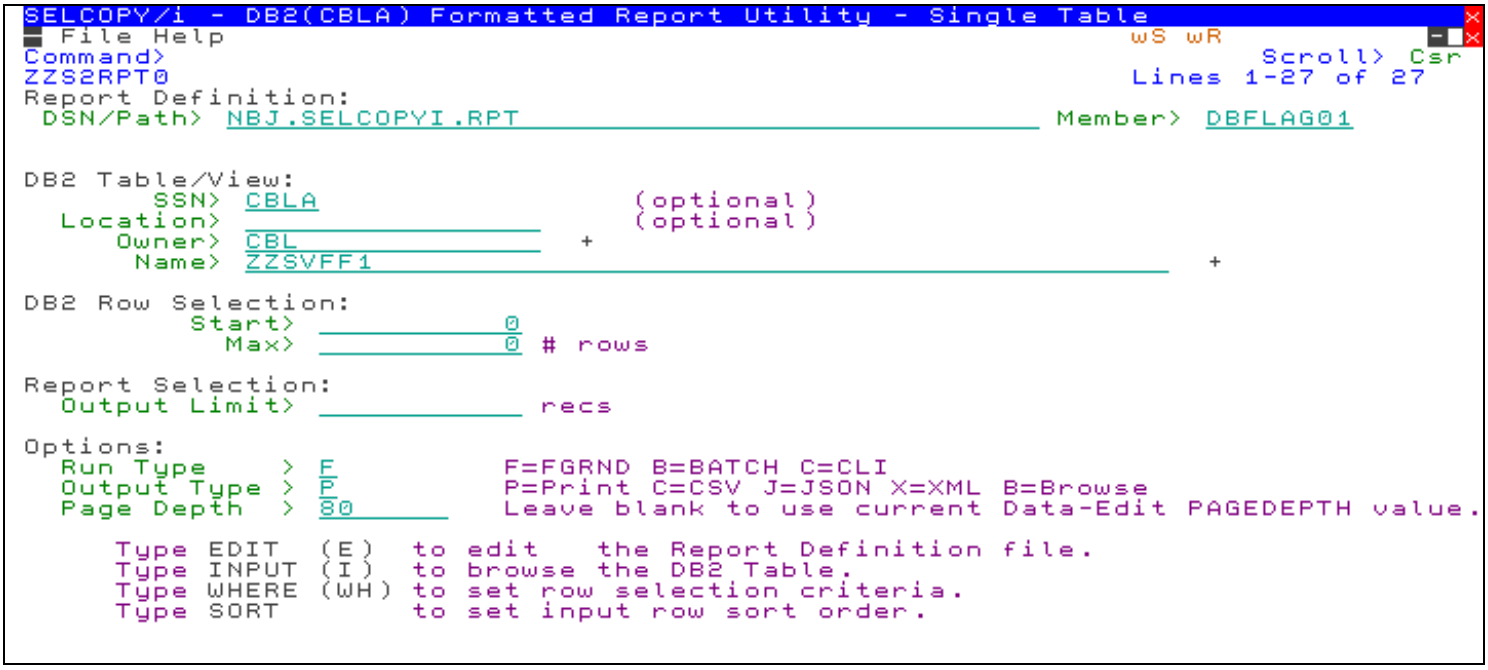


Figure 13. DB2 Table/View REPORT Utility Panel (=11.3.1).

Overview:

The DB2 Single Table Report Utility panel (ZZS2RPT0) is the FileKit display interface to the REPORT Utility where input is to be sourced from a named DB2 TABLE, ALIAS or VIEW that exists either in the local DB2 server or a remotely connected DB2 server.

To open the "DB2 Single Table Report Utility" panel, select option 1. "Single Table" from the "Create a Report using DB2 Table Data" menu. Alternatively, simply enter the fast path "=11.3.1" from any FileKit command prompt.

On opening the panel, a connection is made to the user's default DB2 sub-system (as defined by the "DB2 primary option menu"). Furthermore, if the **DB2 Table/View** input fields are empty, then the "DB2 Table Selection panel" will be displayed. This panel contains a list of existing table, alias and view names which have an Owner/Creator name matching that of the current user's SQLID and which may be further filtered using the panel input fields. Select an entry from the list to populate the **DB2 Table/View** input fields of the DB2 Single Table Report panel.

Primary Commands:

In addition to commands CLI (or CMX), Edit and JCL which are common to each REPORT utility panel, the Formatted Record Report Utility panel supports the following primary commands:

`Input`
Open a Data Editor window view to browse the input DB2 table rows identified by the table, alias or view name specified in the DB2 Table/View SSN/Location/Owner/Name panel input fields. The table row data in the browse display will be formatted using a temporary SDO structure generated by FileKit from the DB2 table column information returned in the SQLDA.

`SORT`
Open the Create DB2 ORDER BY Clause panel to select columns by which the DB2 rows will be sorted.

`WHERE`
Open the DB2 Select table rows by column value panel to build an SQL Query WHERE clause. The WHERE clause will be included in the SQL Query constructed by FileKit to retrieve DB2 table rows.

Use of a WHERE clause is the means by which DB2 table rows are filtered and is broadly equivalent to the use of content matching criteria elements (FIND, etc.) for data file and SMF record input.

Panel Input Fields:

By default, field entries are populated with arguments and options that were entered the last time the panel was used.

Report Definition:
Report definition input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Report Definition fields.

DB2 Table/View:
Input fields which together identify the source TABLE, VIEW or ALIAS from which a DB2 result table is created. The DB2 result table rows contain the values to be reported. This DB2 result table definition will override any result table definition provided in the INPUT section of the report definition.

A DB2 TABLE, VIEW and ALIAS may be referenced by a 3, 2 or 1 qualifier identfier representing *location.schema.name*, *schema.name* or *name* respectively. The *location* value corresponds to the **Location>** field value, *schema* to the **Owner>** field value and *name* to the **Name>** field value.

Wildcard symbols "%" (percent), "*" (asterisk) and/or "_" (underscore) may be entered in all but the **SSN>** to specify a generic mask value from which a matching entry may be selected.

Primary command **INPUT** (or **I**) may be run to browse the DB2 result table report input data.

SSN>
If specified, this field identifies the local DB2 sub-system (DB2 server) to which a connection will be made in order to locate the required TABLE, ALIAS or VIEW object. This may be a different sub-system to the default sub-system to which a connection has already been made and will override an *ssn* value specified by DB2(*ssn*) in the INPUT section of the report definition.

If no DB2 sub-system name is specified in this field or in the report definition, then the default sub-system (set by the "DB2 Primary Option menu") is used. Note that, the default sub-system to which FileKit is connected is displayed in parentheses following "DB2" in the panel's title bar.

Location>
If specified, this field identifies the location of a remote DB2 server at which the required DB2 TABLE, ALIAS or VIEW object is defined. Note that a BIND for the FileKit DB2 PLAN (CBLPLAN1) must have occurred for the remote DB2 server for successful connection.

Enter a wildcard symbol in this field to select from a list of available remote server locations.

If this field is empty then the source object identifier will not include a *location* qualifier and so the object must exist on the local DB2 server.

Owner>
If specified, this field identifies the owner (schema) name of the DB2 source TABLE,VIEW or ALIAS object.

If this field is empty or contains a mask value, then if the value entered in the Name> field does not uniquely identify a DB2 TABLE, VIEW or ALIAS object, the "DB2 Table Selection panel" is opened displaying all object identifiers that match the Location/Owner/Name mask.

Name>
This field identifies the name of the DB2 TABLE, VIEW or ALIAS object.

If this field is empty or contains a mask value, and the object name cannot be uniquely determined using the mask and the value in the Object> field, the "DB2 Table Selection panel" is opened displaying all object identifiers that match the Location/Owner/Name mask.

DB2 Row Selection:
Fields which together identify a window of rows to be fetched from the DB2 result table.

Start>
Specifies the number of the input DB2 result table row from which REPORT processing will start. Rows will be fetched sequentially from this row number. This will override a row number value specified by FROM ROW in the INPUT section of the report definition.

Rows that occur before the specified row number will be bypassed and not included in the number of rows count identified by an input limit (Max>) specification.

By default, REPORT processing starts from the first row of the result table.

Max>
Specifies the maximum number of rows that may be fetched from the DB2 result table. This will override a value specified by FOR ROWS in the INPUT section of the report definition.

Note that, if an ILIM input limit value is specified as an option in the report definition, then this will override the value specified in the Max> field.

Output Limit>
The **Output Limit>** input field value specifies the **maximum** number of detail line records that may be written to the output report. This value will override a value specified by the OLIM option in the report definition.

Once the number of output report detail lines reaches this limit, no further input records will be processed.

An output limit of "0" (zero) implies no output record limit and is set by default when no value is supplied and no OLIM option is set in the report definition.

Options:

Option input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Option fields.

DB2 Table Selection

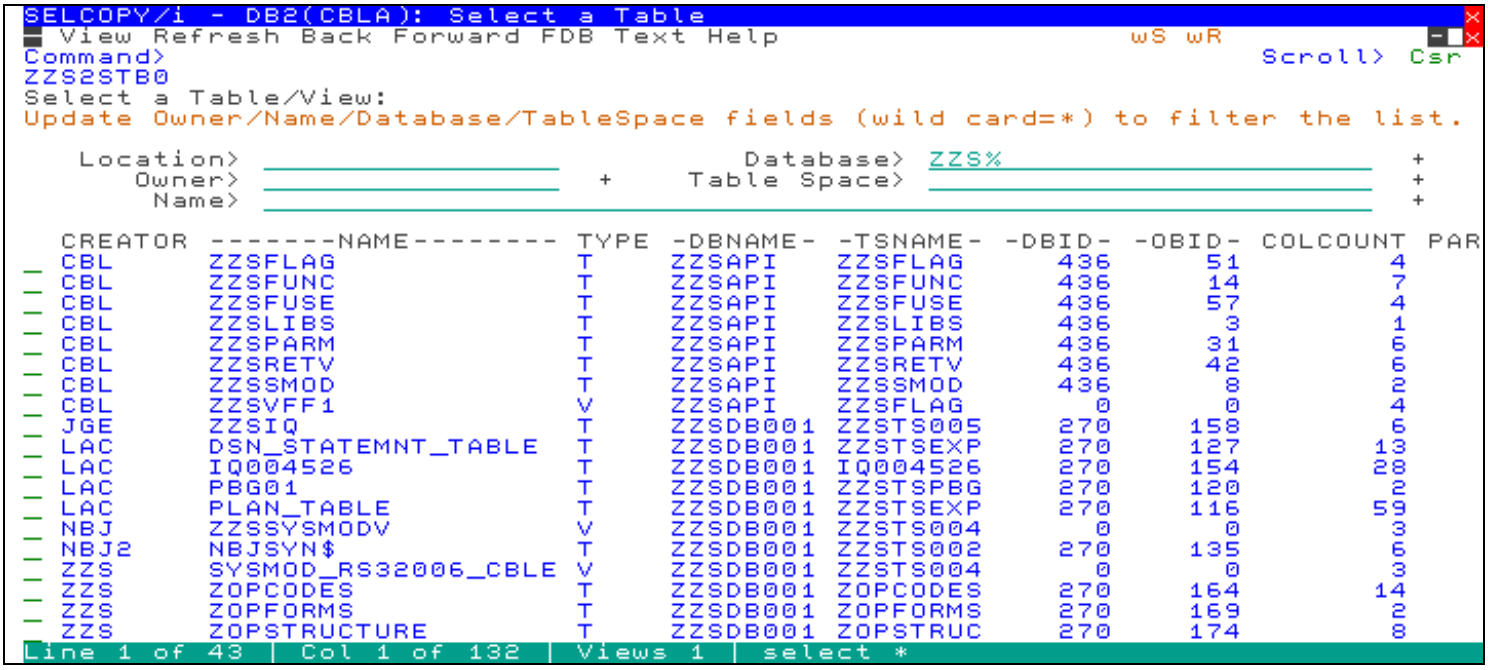


Figure 14. DB2 Table/View Selection Panel.

Overview:

This panel is used to select a DB2 TABLE, VIEW or ALIAS object which is defined at the local or remote DB2 server.

This panel supports filtering of entries based on location, owner, database and/or tablespace name. It also allows filter values to be masks containing one or more of the DB2 pattern-expression wild card symbols "%" (percent) and/or "_" (underscore).

The selection panel input fields may be ammended to re-apply the filter and so refresh the display of table entries fetched from the DB2 catalog table SYSIBM.SYSTABLES. See IBM publication "*DB2 SQL Reference*", "*Appendix - DB2 Catalog Tables*" for details on the column values displayed in this table.

Panel Input Fields:

`Location>`
Specifies the remote server location of the table name.
A server location has a maximum length of 16 characters.

`Owner>`
Specifies a filter on table schema (owner) ID.
A table schema has a maximum length of 128 characters.

`Name>`
Specifies a filter on table name.
A table name has a maximum length of 128 characters.

`DBName>`
Specifies a filter on the database name to which the table belongs.
The database name has a maximum length of 8 characters.

`TSName>`
Specifies a filter on the table space name in which the table is defined.
The table space name has a maximum length of 8 characters.

DB2 WHERE Clause - Select Table Rows by Column Value

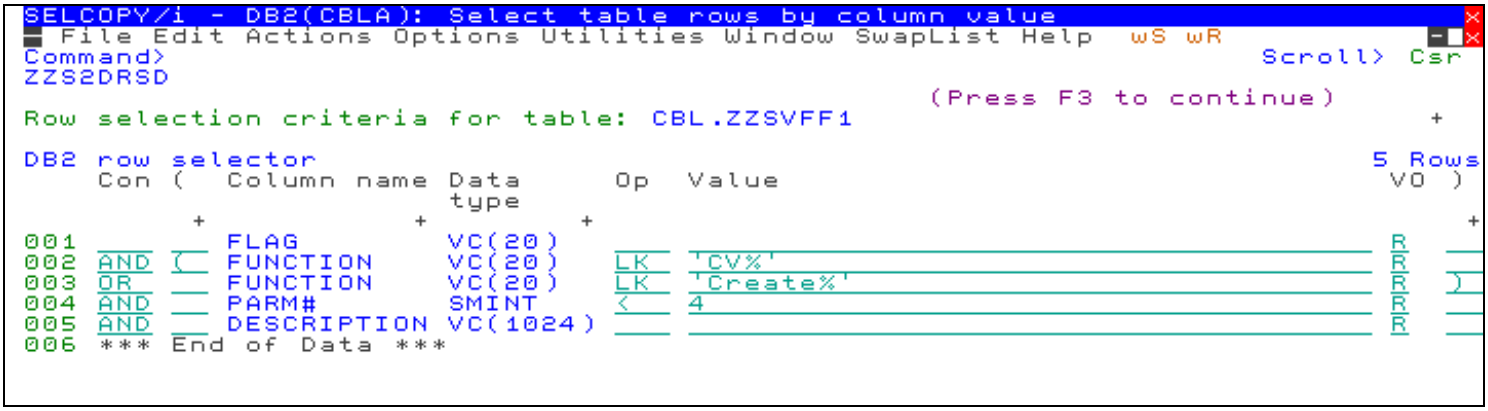


Figure 15. DB2 Table/View REPORT Utility Panel - WHERE Clause.

Overview:

This panel is opened when primary command **WHERE** (or **WH**) is executed and is used to generate a basic SQL WHERE clause to be included in the dynamically generated SQL Query passed to DB2. This WHERE clause will override a WHERE clause specification provided in the INPUT section of the report definition.

The panel contains an embedded table of rows. Each row represents a DB2 SQL WHERE clause predicate where the form of expression to be tested is a column name. Supported DB2 predicate types are Basic, BETWEEN, IN, LIKE and NULL as indicated by the Operator (Op) field. The embedded table is initialised so that an unselected predicate exists for each named column of valid data type defined by the DB2 TABLE, VIEW or ALIAS to which the generated WHERE clause will apply.

Standard table edit primary and line commands may be used to INSERT, DELETE, REPLICATE, COPY or MOVE panel table rows as appropriate, to scroll the table display UP, DOWN, LEFT and RIGHT and also to ZOOM the display of an individual table row.

A predicate is selected for inclusion in the final WHERE search condition by the presence of a valid operator value. A blank or null value in the operator column will deselect the predicate. The order in which selected predicate entries occur within the table dictates their location within the generated search condition and so table rows should be moved as required. Similarly, if a column name is to be referenced in more than one predicate, table rows should be replicated or copied as appropriate.

The WHERE search condition is built by concatenating the selected table rows so that search conditions started by a left parenthesis in one table row may be ended by a right parenthesis in a subsequent table row.

The embedded table includes left and right parenthesis columns in order to allow specification of predicate precedence. For selected predicates, The left parenthesis column may contain one or more "(" (left parenthesis) symbol, the right parenthesis column may contain one or more ")" (right parenthesis) symbol and the combination of the specified left and right parentheses must be balanced for selected predicates.

If the width of any of the input fields in the embedded table view is not sufficient to enter the required input value, then place the cursor on the table row and press the ZOOM key (default Shift-F5) to display the table row in single view. If necessary, the required input field in the zoomed table row panel may be expanded (default Shift-F2) in order to enter the long input field value.

Having selected and modified the table row entries, closing the panel (default F3) will first validate the field entries and then, if no errors are flagged, generate the WHERE search condition.

Panel Input Fields:

Field names that follow are as appear in the zoomed view of a table row. Names in parentheses correspond to the equivalent column name in table view.

`Row selection criteria for table:`
A non-enterable field displaying the qualified DB2 table or view for which WHERE clause row selection criteria is to be defined.

`Connector> (Con)`
Specifies the logical operator connector (AND or OR) that is to be applied to the result of the predicate specified in the table row when deriving the result of a search condition.

Since these logical operators are dyadic, the result of the predicate specified in the table row in which the operator is entered is applied to the result of the predicate or search condition entered in the selected table row (or rows) that occur immediately before it.

If one of these operators is entered in the first selected table row (predicate), it will be excluded from the generated WHERE search condition. If a blank value is entered for any selected table row other than the first, then error ZZSD633E is returned.

By default, all table rows except the first are primed with connector AND.

Parentheses> ("(")
Specifies up to eight "(" (left parenthesis) symbols which each denote the start of a search condition that exists within the final WHERE clause search condition.

A search condition is enclosed by a "(" left parenthesis and ")" right parenthesis symbol and comprises multiple predicates and/or search conditions each connected by a logical operator (AND or OR). Therefore, for each search condition started by a left parentheis, there must exist a subsequent ending right parenthesis specified in the right parentheses) column of a subsequent, selected table row. If not, the left and right parentheses are unbalanced and error ZZSD635E is returned.

In all selected table rows, a left parenthesis entered within a search condition that has not yet been ended by a right parenthesis, indicates the start of a new, nested search condition. Therefore, care should be taken when inserting parentheses or moving/copying/deleting/deselecting table rows so that the logical interpretation of the final WHERE clause search condition is as required.

Column Name: (Column name)
A non-enterable field displaying the name of a column in the DB2 table or view against which the predicate value(s) are tested. Column name is the form of expression specified on the left of the predicate operator when the WHERE clause search condition is built.

Column Type: (Data type)
A non-enterable field indicating the data type of the column and, if appropriate, its length, precision and scale in parentheses.

Operator> (Op)
Specifies the operator used to evaluate the predicate.

Entering a value in this field also selects the table row (logical connector, predicate and parentheses specification) for inclusion in the generated WHERE clause. A blank in this field will exclude (deselect) the table row.

Enter an invalid operator in this field to display the list of valid, selectable operator entries which are as follow:

<blank>	No operator - entry deselected.
= EQ	Equal to.
<> ¬= \= NE	Not equal to.
> GT	Greater than.
< LT	Less than.
¬< \< >= GE	Not less than / Greater than or equal to.
¬> \> <= LE	Not greater than / Less than or equal to.
LK	LIKE *pattern-expression*
¬LK \LK NLK	NOT LIKE *pattern-expression*
BT	BETWEEN *value* AND *value*
¬BT \BT NBT	NOT BETWEEN *value* AND *value*
IN	IN *value-list*
¬IN \IN NIN	NOT IN *value-list*
NL	NULL
¬NL \NL NNL	NOT NULL

Value> (Value)
Specifies the constant value(s) or alternatively, for columns of numeric data type only, the arithmetic expression value(s) used to test the named column value.

Constant values may be of type integer, floating-point, decimal, decimal floating-point, character string, binary or datetime. The type of constant specified will be validated against the data type of the named column.

Specification of multiple values may be supported or required by the predicate type as determined by the selected predicate operator. Multiple values are specified in this input field using unquoted comma (,) delimiter characters. e.g. 'A','B'

Predicate type value requirements are as follow:

Predicate Type	Operators	Value(s)
Basic	= EQ <> ¬= \= NE > GT < LT ¬< \< >= GE ¬> \> <= LE	A single constant or arithmetic expression value.
BETWEEN	BT ¬BT \BT NBT	Exactly two constant or arithmetic expression values. The second value must be greater than the first value so that they define the limits of an ascending range of values.
IN	IN ¬IN \IN NIN	One or more constant or arithmetic expression values which together define a list of possible values against which the column value will be tested.
NULL	NL ¬NL \NL NNL	No value must be entered. Any entry in the value field will return error ZZSD627E.

Each value entered for predicates that test columns of character data types must begin and end with the SQL string delimiter character. If these characters are missing, then the value will automatically be enclosed by apostrophe characters (') during vetting processing as the WHERE clause is generated.

The case of alpha characters entered for each field value is respected or ignored as indicated by the selected value option (VO). This is true, regardless of whether the value has been entered with enclosing SQL delimiter characters.

`Option> (VO)`
A single character option code which determines interpretation of character string constants in the Value input field.

`Respect Case`
Following successful value vetting, alpha characters within each specified value will be inserted, unchanged into the generated WHERE clause syntax.
This is the default option.

`Ignore Case`
Following successful value vetting, alpha characters within each specified value will be upper cased before being inserted into the generated WHERE clause syntax.

`Any Case`
Following successful value vetting, alpha characters within each specified value will be upper cased before being inserted into the generated WHERE clause syntax. Additionally, the scalar function UPPER is applied to the column name with default locale and defined column length. This makes the predicate a test on character strings where upper and lower case alpha character equivalents will test equal.

`Parentheses> (")")`
Specifies up to eight ")" (right parenthesis) symbols which each denote the end of a search condition that exists within the final WHERE clause search condition.
See also left parentheses "(" .

DB2 ORDER BY Clause

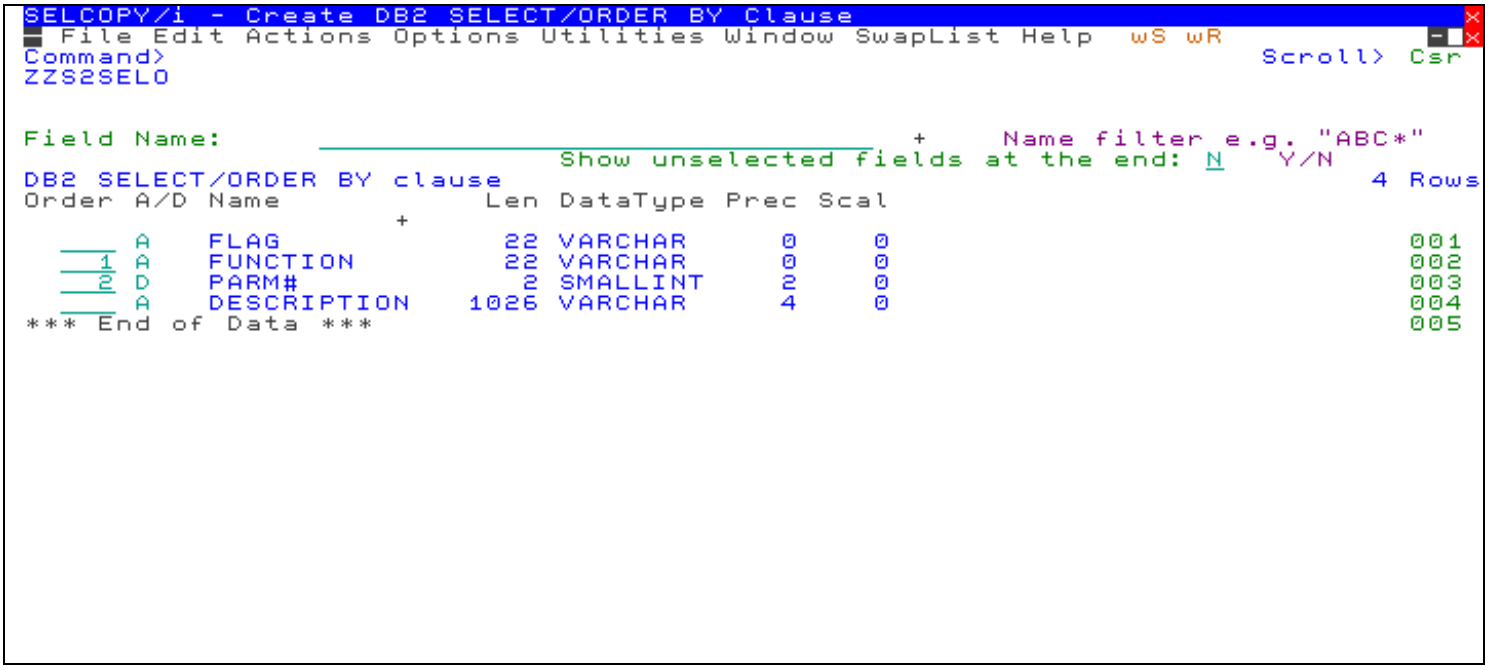


Figure 16. DB2 Table/View REPORT Utility Panel - ORDER BY Clause.

Overview:

This panel is opened when primary command **SORT** is executed. It is used to construct an SQL ORDER BY clause which will be included as part of the dynamically generated SQL Query passed to DB2. This ORDER BY clause will override a SORTINDEX, ORDER BY or SORT specification provided in the INPUT section of the report definition.

The panel contains an embedded table of rows, one for each column belonging to the the specified DB2 TABLE, VIEW or ALIAS object. A row may be selected to include the column name in a DB2 SQL ORDER BY clause.

A column name is selected for inclusion in the final ORDER BY clause by the presence of a sort hierarchy sequence number in the Order field for that column. A blank or null value in this field will deselect the entry as an ORDER BY column. The sort hierarchy sequence number defines the order in which the columns occur in the ORDER BY clause.

Having selected and modified the table row entries, closing the panel (default F3) will first validate the field entries and then, if no errors are flagged, generate the ORDER BY clause.

Panel Input Fields:

`Field Name>`
This input field may be used to supply a DB2 column name mask. The embedded table will refresh displaying only rows where the DB2 column name matches the mask.

A column name mask may contain any number of "*" (asterisk) or "%" (percent) wildcard symbols where "*" represents zero or more of any character and "%" represents exactly one of any character. If no value is entered in this filed, then all DB2 result table column names are displayed.

`Order`
An entry in this field will flag the DB2 column to be included in the generated ORDER BY clause. The input value must be numeric and defines the hierarchical position of the column within the ORDER BY clause.

Columns will be positioned in the generated ORDER BY clause in order of ascending hierarchical value. i.e. The column assigned the lowest value (usually, but not necessarily 1) will occur first in the ORDER BY clause and will be the primary column on which DB2 table rows are sorted.

If column names are assigned the same hierarchical number, then these columns will be positioned in the ORDER BY clause in the order in which they occur in the embedded panel table.

`A/D`
Specifies either "A" or "D" to indicate that the DB2 column is to be sorted in ascending or descending order respectively.

Name
A non-enterable field displaying the name of a DB2 result table column.

Len
A non-enterable field displaying the source field length of a DB2 result table column value. For DECIMAL, FLOAT and DECFLOAT data types, this value is the defined precision value.

DataType
A non-enterable field displaying the data type of a DB2 result table column value.

Prec
A non-enterable field displaying the precision of a DB2 result table column value. For all data types other than DECIMAL, FLOAT and DECFLOAT, this value is the column source field length.

Scal
A non-enterable field displaying the scale of a DB2 result table column value of DECIMAL data type. For all other data types, this field will display as zero (0).

DB2 Report - SQL Query Control File

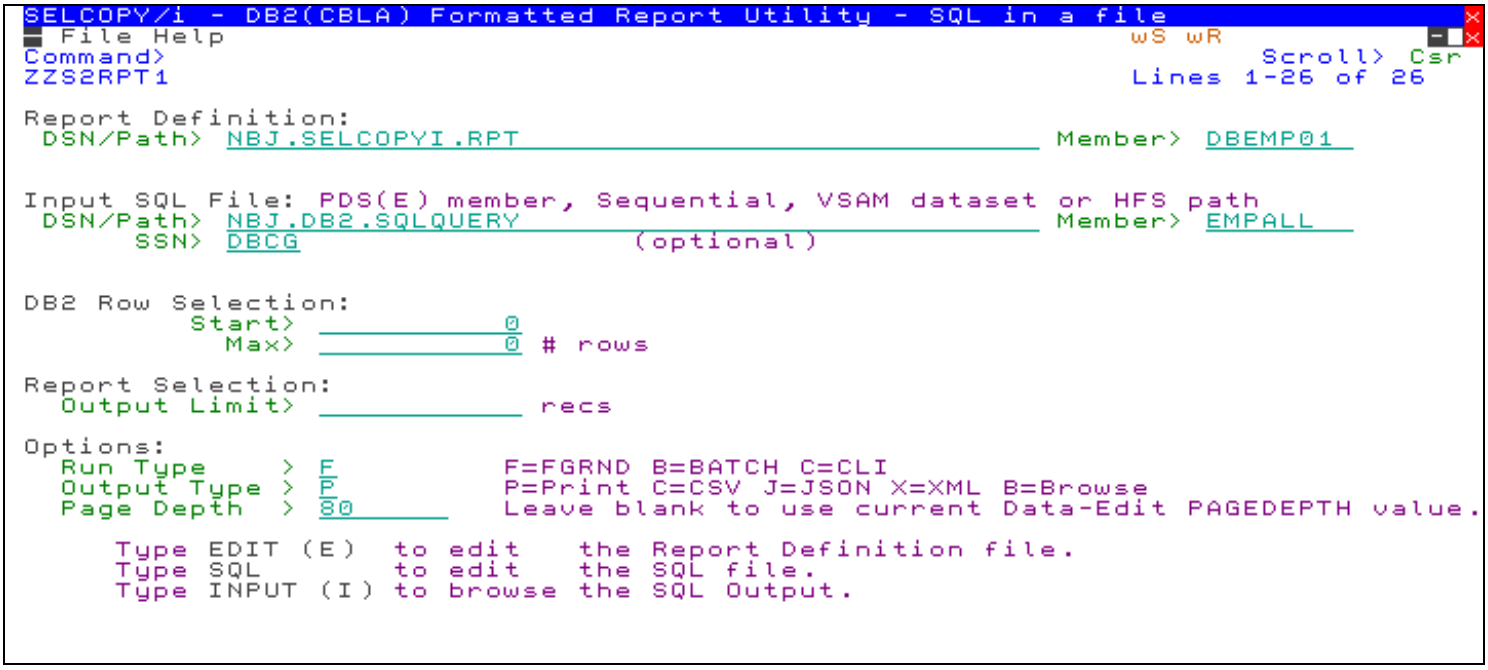


Figure 17. DB2 SQL Query Control File REPORT Utility Panel (=11.3.2).

Overview:

The DB2 SQL Query Control File Report Utility panel (ZZS2RPT1) is the FileKit display interface to the REPORT Utility where input is to be sourced from a DB2 result table. The result table is generated from an SQL Query statement provided as text in a data set or library member (e.g. used as input to IBM SPUFI or the FileKit EXECSQL utility).

To open the "DB2 SQL in a File Report Utility" panel, select option 2. "SQL File" from the "Create a Report using DB2 Table Data" menu. Alternatively, simply enter the fast path "=11.3.2" from any FileKit command prompt.

On opening the panel, a connection is made to the user's default DB2 sub-system (as defined by the "DB2 primary option menu").

Note that, if the report definition input includes a FILTER section, then the filter clause will be ignored and warning message ZZSR064W returned. This is because the FILTER section will attempt to generate a WHERE clause to add to the SQL Query. However, the SQL Query passed via the SQL input file is already fully formed.

Primary Commands:

In addition to commands CLI (or CMX), Edit and JCL which are common to each REPORT utility panel, the Formatted Record Report Utility panel supports the following primary commands:

`Input`
Open a Data Editor window view to browse the input DB2 result table rows generated by the SQL Query found in the SQL input file. The table row data in the browse display will be formatted using a temporary SDO structure generated by FileKit from the DB2 table column information returned in the SQLDA. (See "DB2 Result Table View")

`SQL`
Open a Text Editor window view to edit the SQL input data set, library member or HFS/ZFS file specified in the Input SQL File DSN/Path/Member panel input fields. The SQL Query statement text may be updated and saved if necessary. If the SQL file does not exist, then it will be created when changes are saved.

Panel Input Fields:

By default, field entries are populated with arguments and options that were entered the last time the panel was used.

`Report Definition:`
Report definition input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Report Definition fields.

Input SQL File:

Input fields which together identify a single, sequential or VSAM data set, GDG relative generation, HFS/ZFS file or PDS/PDSE library member from which the SQL Query statement will be obtained. The SQL source must be in a format suitable for input to the IBM SPUFI or FileKit EXECSQL utility.

Primary command **SQL** may be run to edit this SQL Query file and primary command **INPUT** (or **I**) may be run to display the contents of the result table generated by the SQL Query statement. (See "DB2 Result Table View")

The DB2 result table definition specified by the SQL Query will override any result table definition provided in the INPUT section of the report definition.

DSN/Path>

Identifies the fully qualified data set name of a sequential, VSAM, GDG or library data set or the file path of an HFS/ZFS file. An HFS/ZFS file path may be specified in full from the root directory or as path relative to the user's OMVS present working directory (displayed using primary command USS PWD).

Data set names beginning with "." (dot) will be treated as having the user's DSN prefix as defined by the User INI variable System.UserDSNPrefix.

A selectable list of data sets or HFS/ZFS files will be presented if the value entered contains wildcards characters "*" (asterisk) or "%" (percent).

Member>

If the DSN/Path> field contains the DSN of a PDS/PDSE library, then this field specifies the name of a library member. Otherwise, if DSN/Path> contains the DSN of a GDG, then this field specifies the relative generation number of a GDS (e.g. 0, -1, -10).

For a library data set, a selectable list of members will be presented if no member name is specified or a member name mask is entered. i.e. a member name containing wildcards characters "*" (asterisk) and/or "%" (percent). For a GDG, the relative generation 0 is used if no value is entered.

SSN>

If specified, this field identifies the local DB2 sub-system (DB2 server) to which a connection will be made before the SQL Query statement is executed. This may be a different sub-system to the default sub-system to which a connection has already been made and will override an *ssn* value specified by DB2(*ssn*) in the INPUT section of the report definition.

If no DB2 sub-system name is specified in this field or in the report definition, then the default sub-system (set by the "DB2 Primary Option menu") is used. Note that, the default sub-system to which FileKit is connected is displayed in parentheses following "DB2" in the panel's title bar.

DB2 Row Selection:

Fields which together identify a window of rows to be fetched from the DB2 result table.

Start>

Specifies the number of the input DB2 result table row from which REPORT processing will start. Rows will be fetched sequentially from this row number. This will override a row number value specified by FROM ROW in the INPUT section of the report definition.

Rows that occur before the specified row number will be bypassed and not included in the number of rows count identified by an input limit (Max>) specification.

By default, REPORT processing starts from the first row of the result table.

Max>

Specifies the maximum number of rows that may be fetched from the DB2 result table. This will override a value specified by FOR ROWS in the INPUT section of the report definition.

Note that, if an ILIM input limit value is specified as an option in the report definition, then this will override the value specified in the Max> field.

Output Limit>

The **Output Limit>** input field value specifies the **maximum** number of detail line records that may be written to the output report. This value will override a value specified by the OLIM option in the report definition.

Once the number of output report detail lines reaches this limit, no further input records will be processed.

An output limit of "0" (zero) implies no output record limit and is set by default when no value is supplied and no OLIM option is set in the report definition.

Options:

Option input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Option fields.

DB2 Report - SQL Query Statement

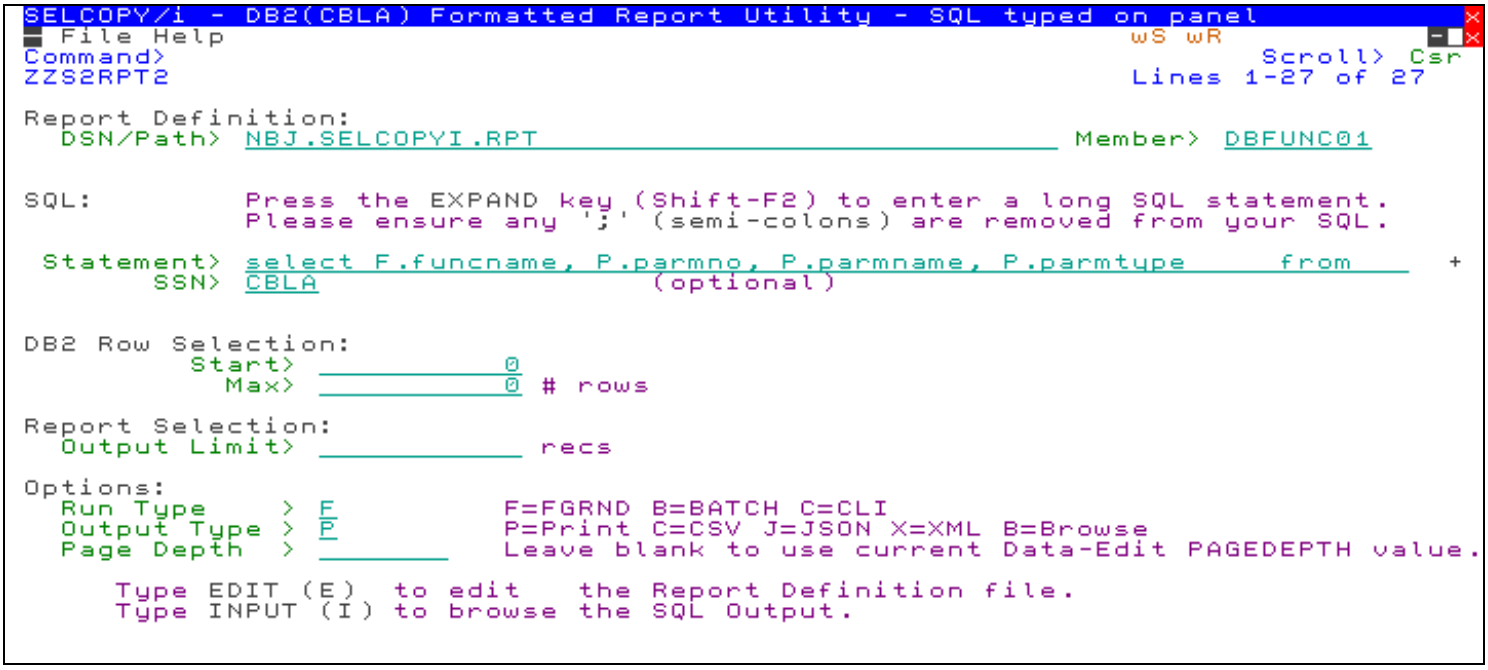


Figure 18. DB2 SQL Query Statement REPORT Utility Panel (=11.3.2).

Overview:

The DB2 SQL Query Staement Report Utility panel (ZZS2RPT2) is the FileKit display interface to the REPORT Utility where input is a DB2 result table generated from an SQL Query statement provided via a panel input field.

To open the "DB2 SQL Input Field" panel, select option 3. "SQL" from the "Create a Report using DB2 Table Data" menu. Alternatively, simply enter the fast path "=11.3.3" from any FileKit command prompt.

On opening the panel, a connection is made to the user's default DB2 sub-system (as defined by the "DB2 primary option menu").

Note that, if the report definition input includes a FILTER section, then the filter clause will be ignored and warning message ZZSR064W returned. This is because the FILTER section will attempt to generate a WHERE clause to add to the SQL Query. However, the SQL Query passed to the REPORT utility is already fully formed.

Primary Commands:

In addition to commands CLI (or CMX), Edit and JCL which are common to each REPORT utility panel, the Formatted Record Report Utility panel supports the following primary commands:

`Input`
Open a Data Editor window view to browse the input DB2 result table rows generated by the supplied SQL Query. The table row data in the browse display will be formatted using a temporary SDO structure generated by FileKit from the DB2 table column information returned in the SQLDA. (See "DB2 Result Table View")

Panel Input Fields:

By default, field entries are populated with arguments and options that were entered the last time the panel was used.

`Report Definition:`
Report definition input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Report Definition fields.

`SQL Statement>`
Specifies a valid SQL Query statement from which the DB2 result table will be generated.

Primary command **INPUT** (or **I**) may be run to display the contents of the result table generated by the SQL Query statement. (See "DB2 Result Table View")

It is quite likely that the input SQL statement will exceed the 60 characters provided by this panel input field. If so, the field may be expanded by positioning the cursor in the field input area and executing the EXPAND command

(default shift-F2). See the "DB2 SQL Expanded View" for details of this field view format.

On closing the expanded view of the field, the first 60 characters of the SQL Query will display within the input field.

SSN>

If specified, this field identifies the local DB2 sub-system (DB2 server) to which a connection will be made before the SQL Query statement is executed. This may be a different sub-system to the default sub-system to which a connection has already been made and will override an *ssn* value specified by DB2(*ssn*) in the INPUT section of the report definition.

If no DB2 sub-system name is specified in this field or in the report definition, then the default sub-system (set by the "DB2 Primary Option menu") is used. Note that, the default sub-system to which FileKit is connected is displayed in parentheses following "DB2" in the panel's title bar.

DB2 Row Selection:

Fields which together identify a window of rows to be fetched from the DB2 result table.

Start>

Specifies the number of the input DB2 result table row from which REPORT processing will start. Rows will be fetched sequentially from this row number. This will override a row number value specified by FROM ROW in the INPUT section of the report definition.

Rows that occur before the specified row number will be bypassed and not included in the number of rows count identified by an input limit (Max>) specification.

By default, REPORT processing starts from the first row of the result table.

Max>

Specifies the maximum number of rows that may be fetched from the DB2 result table. This will override a value specified by FOR ROWS in the INPUT section of the report definition.

Note that, if an ILIM input limit value is specified as an option in the report definition, then this will override the value specified in the Max> field.

Output Limit>

The **Output Limit>** input field value specifies the **maximum** number of detail line records that may be written to the output report. This value will override a value specified by the OLIM option in the report definition.

Once the number of output report detail lines reaches this limit, no further input records will be processed.

An output limit of "0" (zero) implies no output record limit and is set by default when no value is supplied and no OLIM option is set in the report definition.

Options:

Option input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Option fields.

DB2 SQL Expanded View

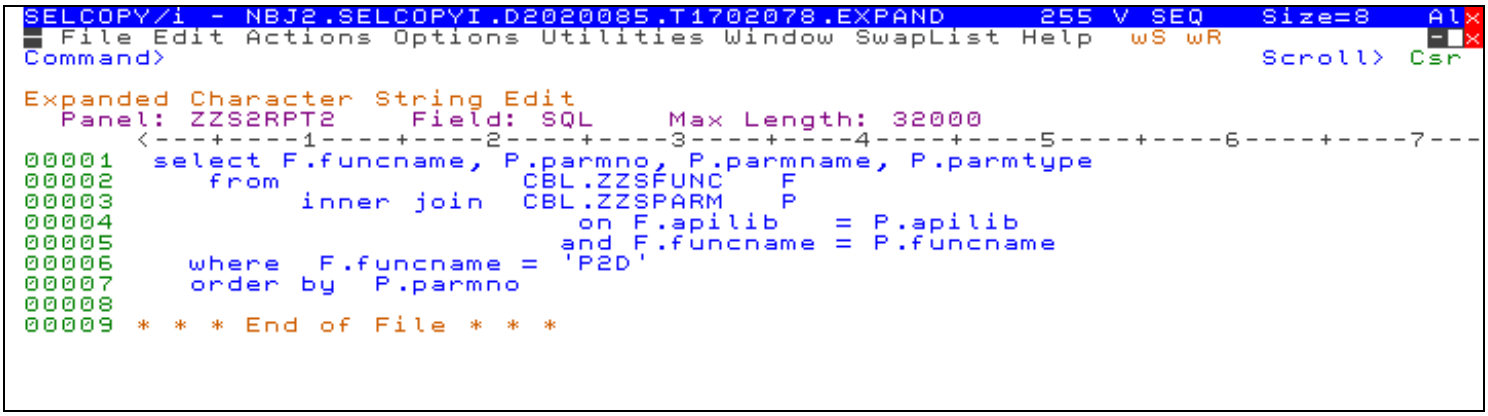


Figure 19. DB2 SQL Query Statement - Expanded View.

Overview:

This is a Text Editor edit view of the SQL statement panel field contents. It allows for a long SQL statement to be entered in the panel field.

The input value (SQL Query statement) may be typed and edited as required. The statement may stream over several consecutive lines of the display and Text Editor primary and line commands used to insert, delete, copy, move, replicate format and navigate the SQL statement text. (See publication "*FileKit Text Editor*" for details.)

When the view is closed (F3), the consecutive lines of text are joined together so that only a single blank character exists between the last non-blank character of a line and the first (blank or non-blank) character of the line that follows. Trailing blank lines are truncated.

DB2 Result Table View

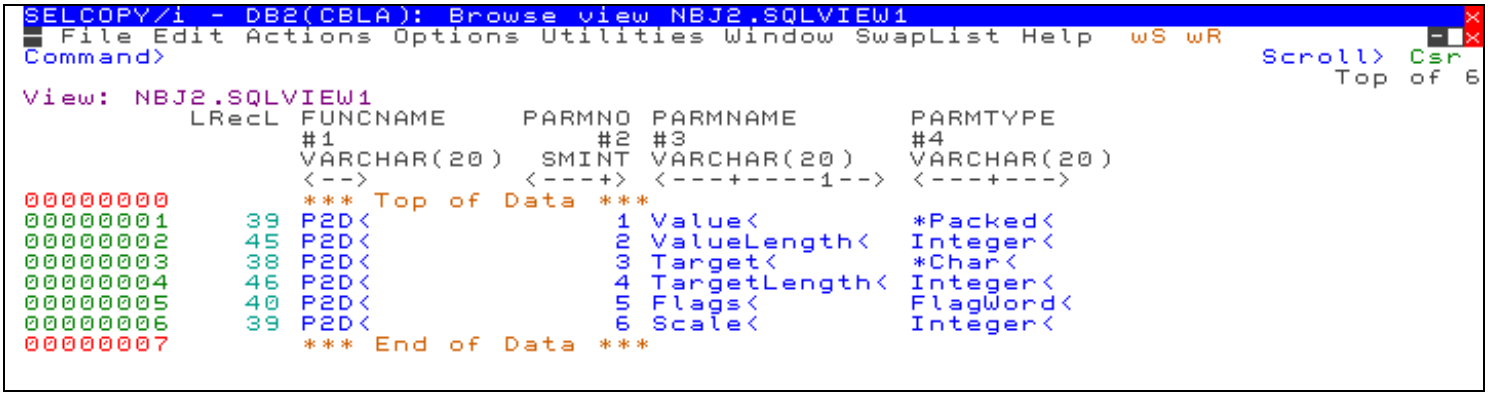


Figure 20. DB2 SQL Query Statement - Result Table view.

Overview:

The result table view is a FileKit Data Editor browse of the DB2 result table rows fetched on execution of an SQL Query statement. The SQL statement may have been supplied directory via an SQL input file or SQL panel input field, or may have been dynamically generated by FileKit.

The table rows are mapped by an SDO structure generated by FileKit from the SQLDA returned by DB2 on execution of the SQL Query. The rows are formatted using this SDO and displayed in columns.

This window provides a view of the rows that will be passed for processing by the REPORT Utility when it is executed.

SMF Report

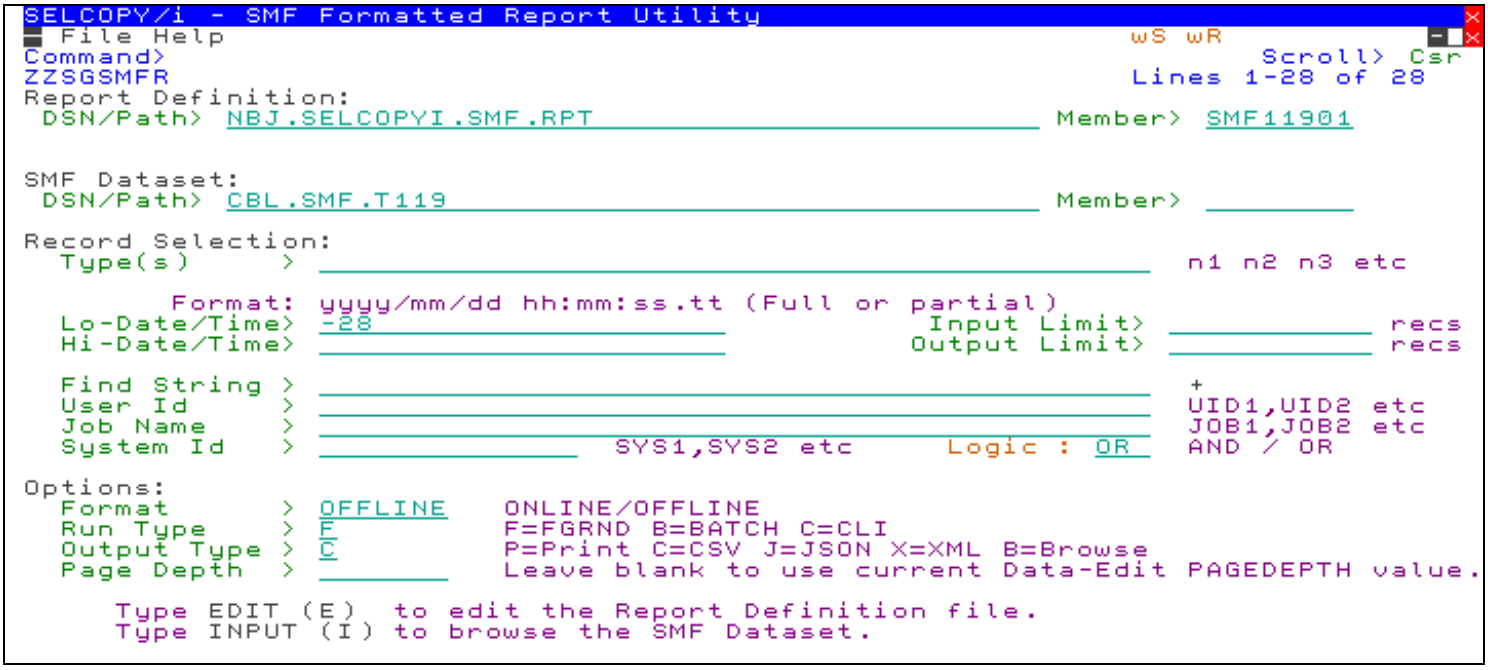


Figure 21. SMF Records REPORT Utility Panel (=11.4).

Overview:

The SMF Formatted Record Report Utility panel (ZZSGSMFR) is the FileKit display interface to the REPORT Utility where input is to be sourced from an IBM System Management Facilities (SMF) data set, library member or HFS/ZFS file.

The SMF record input may be read from a data source (data set) which contains the output from the SMF DUMP tool (IFASMFDP) or it may be read directly from an SMF log data set (typically a DSN of the format "**SYS1.xxxx.MANx)**"). Note that, SMF records read directly from the System Logger is **not** supported.

To open the "SMF Record Report Utility" panel, first select option 11. "Print/Report" from the FileKit primary options menu to open the "Print/Report Features" menu and then select option 4. "SMF Report". Alternatively, simply enter the primary command "**SMFRPT**" or fast path "=11.4" from any FileKit command prompt.

Primary Commands:

In addition to commands CLI (or CMX), Edit and JCL which are common to each REPORT utility panel, the Formatted Record Report Utility panel supports the following primary commands:

BInput
Open a Data Editor window view to browse the input SMF records data set, library member or HFS/ZFS file specified in the SMF Dataset DSN/Path/Member panel input fields. The record data in the browse display will be formatted using a **basic** (non-segmented) layout that maps only header fields and is applied to all SMF records types. See "*Basic Layout Browse (SMFBB)*" in the publication "*FileKit SMF Utilities*".

Input
Open a Data Editor window view to browse the input SMF records data set, library member or HFS/ZFS file specified in the SNF Dataset DSN/Path/Member panel input fields. The record data in the browse display will be formatted using **full** (segmented) layouts, one for each SMF record type and sub-type. See "*Full Layout Browse (SMFB)*" in the publication "*FileKit SMF Utilities*".

Panel Input Fields:

By default, field entries are populated with arguments and options that were entered the last time the panel was used.

Report Definition:
Report definition input fields are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of the Report Definition fields.

SMF Dataset:
Input fields which together identify an existing sequential or VSAM data set, GDG relative generation, HFS/ZFS file or PDS/PDSE library member containing the SMF data records from which the report is generated. This data

source will override the input data source provided in the INPUT section of the report definition.

Execute primary command **INPUT** (or **I**) to browse the input using full SMF record layouts, or **BINPUT** (or **BI**) to browse the input using the single, basic SMF record layout.

DSN/Path>
Identifies the fully qualified data set name of a sequential, VSAM, GDG or library data set or the file path of an HFS/ZFS file. An HFS/ZFS file path may be specified in full from the root directory or as path relative to the user's OMVS present working directory (displayed using primary command USS PWD).

Data set names beginning with "." (dot) will be treated as having the user's DSN prefix as defined by the User INI variable System.UserDSNPrefix.

A selectable list of data sets or HFS/ZFS files will be presented if the value entered contains wildcards characters "*" (asterisk) or "%" (percent).

Member>
If the DSN/Path> field contains the DSN of a PDS/PDSE library, then this field specifies the name of a library member. Otherwise, if DSN/Path> contains the DSN of a GDG, then this field specifies the relative generation number of a GDS (e.g. 0, -1, -10).

For a library data set, a selectable list of members will be presented if no member name is specified or a member name mask is entered. i.e. a member name containing wildcards characters "*" (asterisk) and/or "%" (percent). For a GDG, the relative generation 0 is used if no value is entered.

Record Selection:
Fields which specify selection criteria by which input SMF data records may be filtered. Only records that satisfy the specified selection criteria are selected for reporting.

See Record Filtering for full details on the relationship between input limit, output limit and content match criteria.

Type(s)>
The **Type(s)>** field is used to filter records based on the content of a record type field (**zRTY**) that exists in the header of all SMF records. Furthermore, it may be used to filter records based on a sub-type value found in the **zSTY** header field of certain SMF record types.

Type(s)> field input specifies one or more comma separated SMF record type identification values. These values may each be expressed as an SMF record type (*rectype*), an SMF record type range (*rectype*:*rectype*), or as an SMF record type with sub-type (*rectype-subtype* or *rectype#subtype*).

Specifying SMF record type identification values will override values specified by the SMFTYPES option in the report definition for **SMF Record Type matching**.

A description of each of the different SMF record type identification values is documented in SMF Type Values under *"Record Filtering"*.

If an SMF record contains a match on any of the supplied SMF record type identification values, then SMF Record Type matching will return a true result (1). Otherwise, if no match is found for any of the supplied values, a false result (0) is returned.

SMF Record Type matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

TYPES and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a TYPES specification exists, then the error message ERR066E is returned.

In the following example, a true result will be returned if the input SMF record type (zRTY field value) is 70 (any sub-type), if the SMF record type is 72 with sub-type (zSTY field value) of 3, or if the SMF record type is 62, 63 or 64.

```
Types(s)> 70, 72#3, 62:64
```

Lo-Date/Time>
The **Lo-Date/Time>** input field value specifies a complete or partial absolute timestamp, or a negative number of days that corresponds to a timestamp value which is relative to the current date. This value will override a value specified by the SMFDATELO option in the report definition.

Absolute and Relative timestamp specifications are described in detail by timestamp values under *"Record Filtering"*.

The start of every SMF record contains a common header which includes a timestamp (date and time) at which the record was written to the SMF log (zTME). If a low date/time threshold is specified, then only those SMF records with a timestamp later than or equal to this date and time will be passed to SMF content match criteria record filtering.

An absolute timestamp specification may be truncated to a minimum of 5 bytes ("*yyyy/*") in which case the truncated numeric digits will be set to "0". For example, "`2018/09`" is treated as "`2018/09/00 00:00:00.00`".

A relative timestamp, specified as number of days before the current date, will correspond to a date only. For example, if the current date is **2019/11/13** then "-28" would be equivalent to "`2019/10/16 00:00:00.00`".

`Hi-Date/Time>`
The **HI-Date/Time>** input field value specifies a complete or partial absolute timestamp, or a negative number of days that corresponds to a timestamp value which is relative to the current date. This value will override a value specified by the SMFDATEHI option in the report definition.

Absolute and Relative timestamp specifications are described in detail by timestamp values under *"Record Filtering"*.

The start of every SMF record contains a common header which includes a timestamp (date and time) at which the record was written to the SMF log (zTME). If a high date/time threshold is specified, then only those SMF records with a timestamp earlier than or equal to this date and time will be passed to SMF content match criteria record filtering.

An absolute timestamp specification may be truncated to a minimum of 5 bytes ("*yyyy/*") in which case the truncated numeric digits will be set to "9". For example, "2019/09/22 18" is treated as "2019/09/22 18:99:99.99".

A relative timestamp, specified as number of days before the current date, will correspond to a date only. For example, if the current date is **2020/03/05** then "`-5`" would be equivalent to "`2020/02/29 99:99:99.99`" since 2020 is a leap year.

`Input Limit>`
The **Input Limit>** input field value specifies the **maximum** number of records that may be read from the input file. This value will override a value specified by the ILIM option in the report definition.

Each input record is processed sequentially until this input record threshold is reached.

The input limit includes records which may subsequently be excluded from REPORT processing by a filter clause specified via a FILTER section in the report definition or, alternatively, via specification of content match criteria and/or High/Low date threshold values.

An input limit of "0" (zero) implies no input record limit and is set by default when no input limit is supplied and no ILIM option is set in the report definition.

`Output Limit>`
The **Output Limit>** input field value specifies the **maximum** number of detail line records that may be written to the output report. This value will override a value specified by the OLIM option in the report definition.

Once the number of output report detail lines reaches this limit, no further input records will be processed.

An output limit of "0" (zero) implies no output record limit and is set by default when no OLIM operand value is supplied and no OLIM option is set in the report definition.

`Find String>`
The **Find String>** field specifies one or more comma separated search string values. These values will override values specified by the FIND option in the report definition for **Unformatted Record Find String matching**.

The format of a find search string is described by search values under *"Record Filtering"*.

If a match on **any** of the search strings is located at **any** position within an unformatted input record, then Unformatted Record Find String matching will return a true result (1). Otherwise a false result (0) is returned.

Unformatted Record Find String matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

Unformatted Record Find String matching and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a find string specification exists, then the error message ERR066E is returned.

For example, the following will set a true condition if one of the strings "SYS1.MACLIB", "SYS1.MIGLIB", "SYS1.MODGEN" or "SYS1.MSGEN" (upper or lower case) exists at any location within the unformatted record.

```
Find String> SYS1.MACLIB, SYS1.MIGLIB, SYS1.MODGEN, SYS1.MSGEN
```

The panel entry field for **Field String>** displays only the first 45 characters of any input. To enter or display search values that extend beyond this display length, position the cursor in the input field area and press shift-F2 (**EXPAND**). A Text Editor window will be displayed allowing entry of FIND search values that may stream multiple text edit lines.

User Id>

The **User Id>** field is used to filter records based on the content of a user name field (**zUserId**) that exists at a fixed location within certain SMF records. This fixed position may be different for each of the SMF record types. The following SMF record types are those that contain a zUserId field:

004	014	020	030	035	042	062	065	068	110
005	015	025	032	036	060	063	066	069	118
006	017	026	034	040	061	064	067	080	119
010	018								

User Id> field input specifies one or more comma separated user name search values. These user name values will override values specified by the SMFUSERID option in the report definition for **SMF Record User Name matching**.

A user name value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*.

Unless the specified user name value contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the value will be truncated or padded with blanks to a length of 8 characters. Furthermore, if no percent ("%") or asterisk ("*") wildcards are specified and the value is expressed as an **unquoted** or **quoted** string, then all alpha characters in the string will be upper cased. For example, if SMF records contain a zUserId field value "ABC", "ABC1", "ABCXXX" and "XABC" then "USERID(abc)" would match "ABC" only, "USERID(abc*)" would match "ABC", "ABC1" and "ABCXXX", "USERID(%abc)" would match "XABC" only and "USERID(*abc*)" would match all 4 values.

If an SMF record zUserId field contains a match on **any** of the supplied user name values, then SMF Record User Name matching will return a true result (1). Otherwise, if no match is found for any of the supplied values or the SMF record does not contain a zUserId field, then a false result (0) is returned.

SMF Record User Name matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

Note that SMF Record User Name matching and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a user name specification exists, then the error message ERR066E is returned.

In the following example, a true result will be returned if the SMF record has a zUserId field that contains a user name of any length up to a maximum of 8 characters ending with 1, or a user name beginning with "ABC" followed by any single character followed by "DEFG".

```
User Id> *1, ABC%DEFG
```

Job Name>

The **Job Name>** field is used to filter records based on the content of a job name field (**zJobName**) that exists at a fixed location within certain SMF records. This fixed position may be different for each of the SMF record types. The following SMF record types are those that contain a zJobName field:

004	010	017	025	034	040	061	064	067	080
005	014	018	026	035	042	062	065	068	110
006	015	020	030	036	060	063	066	069	118

Job Name> field input specifies one or more comma separated job name separated job name search values. These job name values will override values specified by the SMFJOBNAME option in the report definition for **SMF Record Job Name matching**.

A job name value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*.

Unless a specified job name contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the job name value will be truncated or padded with blanks to a length of 8 characters. Furthermore, if no percent ("%") or asterisk ("*") wildcards are specified and the job name is an **unquoted** or **quoted** string, then all alpha characters in the string will be upper cased.

If an SMF record zJobName field contains a match on **any** of the supplied job name values, then SMF Record Job Name matching will return a true result (1). Otherwise, if no match is found for any of the supplied *jobname* values or the SMF record does not contain a zJobName field, then a false result (0) is returned.

SMF Record Job Name matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

Note that SMF Record Job Name matching and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a job name specification exists, then the error message ERR066E is returned.

In the following example, a true result will be returned if the SMF record has a zJobName field that specifically contains a job name "RSHD", contains a job name beginning with "GIM" or any job name of length 5.

```
Job Name> RSHD, GIM*, %%%%%
```

System Id>

The **System Id>** field is used to filter records based on the content of a system identifier field (**zSID**) that exists in the header of all SMF records.

System Id> field input specifies one or more comma separated system identification search values. These values will override values specified by the SMFSID option in the report definition for **SMF Record System Id matching**.

A system identifier value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*.

Unless the specified system identifier value contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the value will be truncated or padded with blanks to a length of 4 characters. Furthermore, if no percent ("%") or asterisk ("*") wildcards are specified and the value is expressed is an **unquoted** or **quoted** string, then all alpha characters will be upper cased.

If an SMF record zSID field contains a match on **any** of the supplied system identifier values, then SMF Record System Id matching will return a true result (1). Otherwise, a false result (0) is returned.

SMF Record System Id matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

Note that SMF Record System Id matching and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and system identifier specification exists, then the error message ERR066E is returned.

In the following example, a true result will be returned if the zSID field contains a system id value "XS1", a value beginning with "S0" followed by any single character followed by "1", or a value of up to 4 characters in length ending in "Z".

```
System Id> 'xs1', S0%1, '*Z'
```

Logic: AND | OR

The **Logic:** field input specifies the logical operation ("AND" or "OR") to be used when determining the result of content match criteria record filtering. This logical operation value will override a value specified by the SMFLOGIC option in the report definition.

The logical operation is used to combine the Boolean values (true or false) returned by each of the specified content match criteria elements:

- Unformatted Record Find String matching
- SMF Record Job Name matching
- SMF Record System Id matching
- SMF Record Type matching
- SMF Record User Name matching

Content matching criteria elements may be specified in the report definition input, and/or passed to the REPORT utility via panel input fields: Type(s), Find String, User Id, Job Name and System Id.

The "AND" or "OR" logical operation is performed between each of the Boolean values returned by the specified content matching criteria to produce an overall true (1) or false (0) result. If the overall result is true, the record satisfies the content match criteria and is passed to REPORT generation processing.

If logical operation **AND** is used then the result returned by **all** of the content checking criterion elements specified for the current REPORT execution must be 1 (i.e. true). If logical operation **OR** is used then **only one** of the values returned by the content checking criterion elements must be 1 (true) in order to return a true result for the record.

Note that other SMF record filtering controlled by high date/low date thresholds and input record limit, does not form part of the content checking criteria and so is not affected by the logical operation.

Options:

Option input fields **Run Type>**, **Output Type>** and **Page Depth>** are common to each of the REPORT Utility panels. See "Report Panels" for a description of use of these Option fields.

Format> ONLINE | OFFLINE

ONLINE indicates that the SMF dataset is in the format as written directly by SMF to the SMF log datasets (**SYS1.xxxx.MANx)**. Note that FileKit does not support reporting on SMF records directly from the System Logger.

ONLINE datasets include a **4-byte record descriptor word (RDW) prefix** at the start of each record, so record-type field mapping must be offset by this amount.

OFFLINE indicates that the SMF dataset is the format as written by the SMF DUMP tool (IFASMFDP) which does not include a 4-byte (RDW) record prefix.

Command Line Interface

The REPORT utility execution may be started using the FileKit primary command, **REPORT**.

The REPORT command may be passed to the FileKit command processor via one of the following:

1. Entered at any FileKit window command prompt.
2. As a line of text displayed using the FileKit Text Editor. (See Command File Execution.)
3. As SDEIN input to the FILEKITB (FileKit batch) program. (See Batch Execution.)

Command File Execution

If the utility is to be executed in the FileKit foreground environment, the REPORT command and its operands may be saved as text in a file (data set or library member) and started using the FileKit command execution facility. This involves placing the cursor on the text of the command in a Text Edit view of the file and then pressing the "Action" key (shift-F4 by default).

The user's personal "Home" file (edited using option 4 in the Primary Option menu) is created the first time the user starts FileKit. It exists to be updated by the user with useful or commonly used primary commands for execution using the "action" key. Because the file contains only text, it may also include any accompanying notes or comments about the commands.

To prime the command for execution using FileKit's command execution facility, it must be preceeded by a "less than"/"left chevron" symbol ("<"). Furthermore, the command's text may stream over multiple, consecutive file records. To do this, the command continuation "backslash" symbol ("\") must be specified as the last non-blank character of each record containing text that is to be continued on the next record.

For Example:

```
-USER123.FILEKIT.CMX      32752 V SEQ   Size=639   Alt=0,0;0                -+x
Command>                                                          Scroll> Csr
      <---+----1----+----2----+----3----+----4----+----5----+----6----+----7---
00001 ** USER123.FILEKIT.CMX ***       L=095 --- 2020/02/13 18:04:10 (USER123)
00002
00003 | Execute the REPORT Utility for SMF record input
00004 | ----------------------------------------------
00005 |
00006 | The following command reports on SMF TCP/IP 119 records and limits
00007 | the output to only 10 records of each subtype:
00008
00009 <REPORT  RUN   PRINT    RPTDEF('NBJ.FILEKIT.RPT(SMF119)')       \
00010       SMF-INPUT-BEG             CBL.SMF.GDG(-1)    OLIM(10)     \
00011       SMF-INPUT-END
00012
```

Batch Execution

If the utility is to be executed in batch, the REPORT command and its operands must be passed to the **FILEKITB** (FileKit Batch) program via the SDEIN DD input.

A template batch job containing relevant JCL statements and REPORT command may be generated by selecting run type option "B" (Batch) in any of the REPORT Utility panels provided for general formatted record reports, SMF record reports or DB2 table reports.

For Example:

```
-USER123.JCL(REP00004)     80 F PDSE   Size=69   Alt=0,0;25                -+x
Command>                                                        Scroll> Csr
      <---+----1----+----2----+----3----+----4----+----5----+----6----+----7--
00001 //U123JOB JOB ,,CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1),NOTIFY=&SYSUID
00002 //*
00003 //* ------------------------------------------------------------------
00004 //* REPORT: FileKit SMF Report Utility - 2019/11/07 12:00
00005 //* ------------------------------------------------------------------
00006 //*
00007 //REPORT   EXEC PGM=FILEKITB
00008 //STEPLIB  DD DISP=SHR,DSN=CBL.INST.CBL19293.SZZSLOAD
00009 //*
00010 //* --------- SORT work files beg ---------------------------------------
00011 //SORTIN   DD DISP=(NEW,PASS),SPACE=(TRK,(300,300)),
00012 //            DCB=(RECFM=VB,LRECL=1024,BLKSIZE=0,DSORG=PS)
00013 //SORTOUT  DD DISP=(NEW,PASS),SPACE=(TRK,(300,300)),
00014 //            DCB=*.SORTIN
00015 //SYSIN    DD DISP=(NEW,PASS),SPACE=(TRK,(001,001)),
00016 //            DCB=(RECFM=FB,LRECL=0080,BLKSIZE=0,DSORG=PS)
00017 //SYSOUT   DD SYSOUT=*
00018 //* --------- SORT work files end ---------------------------------------
00019 //*
00020 //MYREPORT DD   *
00021 HEAD:
00022   #TODAY  \  "z/OS TCP Daily Connections Report"  \  "PAGE:" #PAGE
00023
00024 COLUMNS:
00025   SMF119#02_TCPIP_Statistics.zSID                    'SysID'
00026   SMF119#02_Identification.zStack                    'Stack'
00027   SMF119#02_TCP_Connection_Termination.zRName        'Resource'
00028   SMF119#02_TCP_Connection_Termination.zConnectStart 'Start'
00029   SMF119#02_TCP_Connection_Termination.zConnectEnd   'End'
00030   SMF119#02_TCP_Connection_Termination.zInBytes      'Bytes In'  10 R
00031   SMF119#02_TCP_Connection_Termination.zOutBytes     'Bytes Out' 10 R
00032   SMF119#02_TCP_Connection_Termination.zTermCode     'Termination Desc'
00033
00034 SORT:
00035   SMF119#02_Identification.zStack
00036   SMF119#02_TCP_Connection_Termination.zRName
00037   SMF119#02_TCP_Connection_Termination.zConnectStart
00038
00039 BREAK:
00040   SMF119#02_TCP_Connection_Termination.zRName
00041
00042 FILTER:
00043   SMF119#02_TCP_Connection_Termination.zTermCode <> 'App_Close'
00044 /*
00045 //*
00046 //* --------- Optional overrides beg ----------------------------------
00047 //ZZSUSERI DD DUMMY           Deactivate USER=USER123 INI file options.
00048 //*ZZSUSERI DD DISP=SHR,DSN=USER123.FILEKIT.INI
00049 //*SDESDO   DD DISP=SHR,DSN=CBL.FILEKIT.SITE.SDO
00050 //*         DD DISP=SHR,DSN=CBL.INST.CBL19293.SZZSDIST.SDO
00051 //* --------- Optional overrides end ----------------------------------
00052 //*
00053 //SDEPRINT DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=0133)
00054 //SDEOUT   DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=1024)
00055 //SDEIN    DD   *
00056
00057    REPORT   RUN   PRINT
00058             RPTDEF(DD=MYREPORT)
00059           OUTPUTDD(SDEOUT)
00060      SMF-INPUT-BEG
00061           CBL.SMF.IBMSAMP.ZOSR22
00062           OFFLINE
00063              ILIM(050000)
00064      SMF-INPUT-END ;
00065
00066 /*
00067 * * * End of File * * *
```

JCL DD Statements

The executable program ZZSSMAIN (alias **FILEKITB**) is included by default in the CBL Product Suite load library (installed into SMP/E target library "*prefix*.SZZSLOAD").

The FILEKITB program is the batch interface to FileKit and may be used to execute any of FileKit's utility primary commands. This includes utilities such as File copy (FCOPY), File Search and Update (FSU), File Compare (COMPFILE) as well as the Report utility (REPORT). PGM=FILEKITB must be specified on the JCL EXEC statement of any batch job step that executes a FileKit utility.

A number of mandatory and optional JCL statements exist in batch jobs that execute the FILEKITB program. Additional statements may also be necessary for execution of the REPORT utility. This section itemises each of these JCL statements and describes their effect on REPORT utility batch processing.

SDEIN

SDEIN input is **mandatory** and contains the REPORT command to be passed to the FileKit command processor. The SDEIN input is not restricted to the REPORT command but may be used to execute any FileKit command or edit REXX macro in the batch environment.

Multiple commands may be specified in the same SDEIN input. If so, each command is executed in the sequence provided when execution of the previous command completes.

A FileKit command map span a number of consecutive input records and is only terminated by an unquoted semi-colon (";") symbol or the end of SDEIN input. Furthermore, SDEIN input may contain comment text which is imbedded within command string. Commment text occurs within unquoted slash-asterisk ("/*") and asterisk-slash ("*/") symbol pairs.

For example:

```
//SDEIN    DD  *
    REPORT  RUN  PRINT   PAGEDEPTH(74)
      RPTDEF('USER123.FILEKIT.RPT(ZZST1RPT)')          /* Report Definition.  */

      SDE-INPUT-BEG                              /* Structered input section. */
         USER123.SELCTRN.ZZST1DAT                      /* Input Data Records. */

         SDO-INPUT-BEG                           /* Structure object section. */
             COBOL   USER123.COPYLIB.COBOL(ZZST1CPC)   /* Structure Mapping.  */
         SDO-INPUT-END

      SDE-INPUT-END ;
/*
```

SDEOUT

SDEOUT is optional and, if present, will contain the output report text generated by the REPORT utility if REPORT operand **OUTDD** is not specified or has is specified as **OUTDD(SDEOUT)**.

If REPORT OUTDD(*ddname*) is specified, then the output report is written to *ddname*. If OUTDD is not specified and SDEOUT is not allocated, the output will be directed to SDEPRINT.

Note that report output records will be truncated if the length of the text exceed the defined LRECL value (or LRECL-4 if RECFM=V).

SDEPRINT

SDEPRINT is **mandatory** and will contain the FileKit execution diagnostic messages. If SDEIN is not allocated, it will also contain the output report text.

SDESDO

SDESDO is optional and specifies the SDO library path from which FileKit distributed SDO structure members may be found. In particular, SDO structure members used to map SMF records.

If SDESDO is not allocated, then SMF record mapping SDO members must exist in the library "*prefix*.SZZSDIST.SDO", where *prefix* is the library DSN prefix of the installation libraries (i.e. the SMP/E install target libraries).

SYSIN

SYSIN is **mandatory** if the REPORT utility input definition control statements include a SORT section.

The REPORT utility utilises the local SORT (DFSORT or SYNCSORT) program to sort the report detail records. The SYSIN input contains the SORT control statements which are **written** by the REPORT utility and then passed to the SORT program.

SYSPRINT

SYSPRINT is **mandatory** if the REPORT utility input definition control statements include a SORT section.

The SYSPRINT output will contain diagnostic messages (if any) written by the SORT utility during execution.

SORTIN

SORTIN is **mandatory** if the REPORT utility input definition control statements include a SORT section.

The SORTIN input contains the data records **written** by the REPORT utility and then passed to the SORT program for sorting.

SORTOUT

SORTOUT is **mandatory** if the REPORT utility input definition control statements include a SORT section.

The SORTOUT output will contain the sorted data records written by the SORT utility.

ZZSUSERI

ZZSUSERI is optional and specifies the DSN of the User INI file.

Like FileKit, when FILEKITB starts it establishes customised processing options from the site wide configuration data set (the SITE INI file). The DSN of the SITE INI options is identified by option INamDSN in the CBL Product Suite options module, CBLNAME.

Having established the site-wide options, processing option overrides set by the user's personal FileKit options data set (the USER INI file) are applied. In batch, the active user will be the USER=userid value specified on the JOB statement which defaults to be the userid of the submitting TSO/E user or job. If ZZSUSERI is not allocated, the DSN of the USER INI file is determined by the USERINIFILE option in the SITE INI file. This option specifies a DSN mask value based on the current userid.

To ensure an FILEKITB is user independent, a ZZSUSERI DD statement should be included and allocated to DUMMY or an existing INI file DSN.

ddname

Additional DD statements may be necessary if REPORT command operands reference DD name arguments. For example, DD=*ddname* may be specified for RPTDEF, OUTPUTDD, SMF-INPUT and SDE-INPUT operands.

For SMF-INPUT and SDE-INPUT, DD=*ddname* may be used to specify a *ddname* allocated to an input DASD or TAPE data set, or even a concatenation of data sets.

REPORT Command

Overview:

The REPORT primary command is the command line interface to the REPORT utility.

In addition to executing the utility in the foreground or batch to generate report output, the REPORT command may be executed in the foreground to:

1. List members of the default report definition source library.
2. Edit and optionally initialise a source report definition member.
3. From a Data Editor view of formatted data, add COLUMN section definitions to a report definition member for all fields in the focus line that have been selected for display. (The focus line is the line on which the cursor is positioned.)
4. Generate a JCL job to in order to generate the report utility in batch.

A REPORT command and operands may be generated for values entered in REPORT panel fields by selecting "C" (CLI) in the "Run Type" option field.

If REPORT is executed with no operands, the "Print/Report Features Menu" panel is opened allowing for selection of the required REPORT utility panel specific to input data type.

Examples:

Example 1. List Report Definition Members:

The following REPORT command may only be executed in the FileKit foreground.

```
REPORT  L
```

Opens a Library Member List window to display all members of the user's default report definitions library.

Example 2. Create Report Column Definitions from the Current Formatted Record View:

The following REPORT command may only be executed in the foreground whilst using the FileKit Data Editor to display formatted records.

```
REPORT  ADD   USER123.FILEKIT.RPT(MUSX301)
```

Edit the report definition member "USER123.FILEKIT.RPT(MUSX301)" and add column definitions for every record field displayed in the focus line of the current Data Editor view. The generated column definitions may be included in the COLUMNS section of the report definition member.

Example 3. Formatted Records Report:

The following REPORT command example is as it might appear in a text file for execution in the foreground using the FileKit ACTION key. See Command File Execution for details.

```
<REPORT  RUN   RPTDEF(USER123.FILEKIT.RPT(MUSX301))  \
         SDE-INPUT-BEG  \
            USER123.SOURCE.DATA  \
                \
            SDO-INPUT-BEG  \
                COBOL USER123.COPYBOOK.COBOL(T2VF002) \
            SDO-INPUT-END  \
                \
            OLIM(100)  \
         SDE-INPUT-END
```

Using the report definition member "USER123.FILEKIT.RPT(MUSX301)", produce a report of no more than 100 detail lines from records in dataset "USER123.SOURCE.DATA". The input records will be formatted using a FileKit SDO structure generated from COBOL copybook member "USER123.COPYBOOK.COBOL(T2VF002)". Because the REPORT command is to be executed in the foreground, no OUTDD is required. The report output will not be written to a DASD data set but will be displayed in a FileKit Data Edit window view instead. The report may be subsequently saved to DASD.

Example 4. SMF Record Report:

The following REPORT command example is as it might appear in DD SDEIN input to the FILEKITB (FileKit batch) program.

```
REPORT  RUN   RPTDEF(DD=RPTCTL)   OUTDD(RPTOUT)
         SMF-INPUT-BEG
            USER123.SMFSAMP.SMF030
            TYPES(30-5)
            DATELO( 2019/09/15 13:00)
            DATEHI( 2019/09/20 )
         SMF-INPUT-END
```

Using the report definition input allocated to DDname RPTCTL, produce a report of all **SMF Record-Type 30 SubType 5** records contained in dataset "USER123.SMFSAMP.SMF030" provided they fall within the DATELO/DATEHI timestamp range. The generated report is written to the data set allocated to DDname RPTOUT.

Example 5. SMF Record Report with Filters:

The following REPORT command example is as it might appear in DD SDEIN input to the FILEKITB (FileKit batch) program.

```
REPORT  RUN   RPTDEF(DD=RPTCTL)   OUTDD(RPTOUT)
         SMF-INPUT-BEG
            DD=SMFIN
            TYPES( 30 )
            JOBNAME(NBJ*,USER%%%)
            FIND('CBL.SMFSAMP.')
            LOGIC(AND)
            DATELO( 2020/01/01 )
            DATEHI( 2020/03/01 23:59:59.99)
         SMF-INPUT-END
```

Using the report definition input allocated to DDname RPTCTL, produce a report of all records of **SMF Record-Type 30** (any SubType) contained in the dataset(s) allocated to DD SMFIN provided they satisfy **all** of the following filter conditions:

1. Record was written within the DATELO/DATEHI timestamp range.
2. Record has a job name value beginning "NBJ" or a job name of length 7 characters beginning "USER".
3. Record contains the search string "CBL.SMFSAMP." at any position. (DSN qualifiers).

DD **SMFIN** may be allocated to a DASD or TAPE data set, or a concatenation of data sets.

The generated report is written to the data set allocated to DDname RPTOUT.

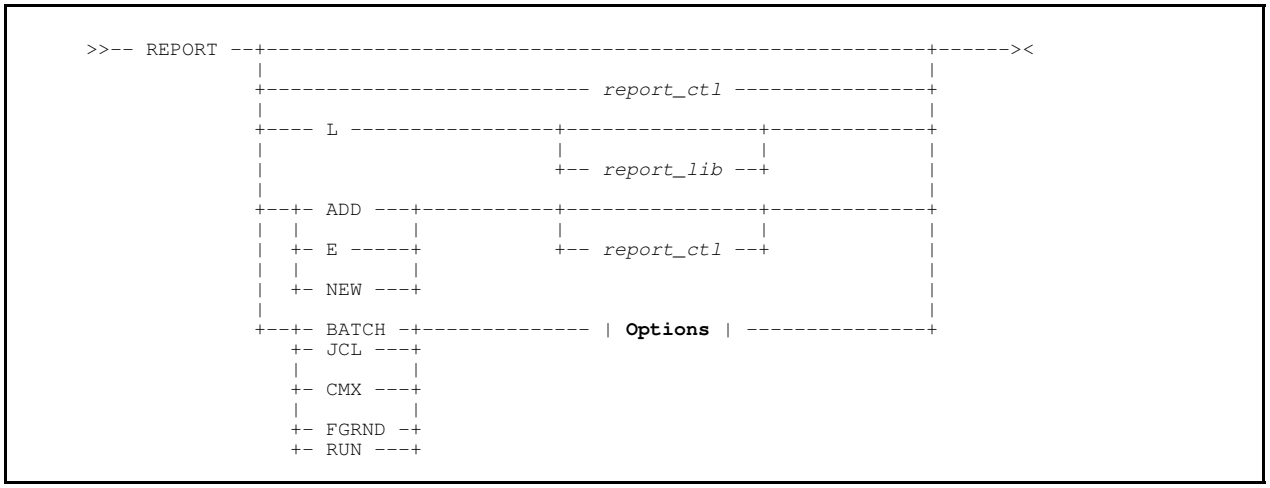
Example 6. DB2 Table Report:

The following REPORT command example is as it might appear in DD SDEIN input to the FILEKITB (FileKit batch) program.

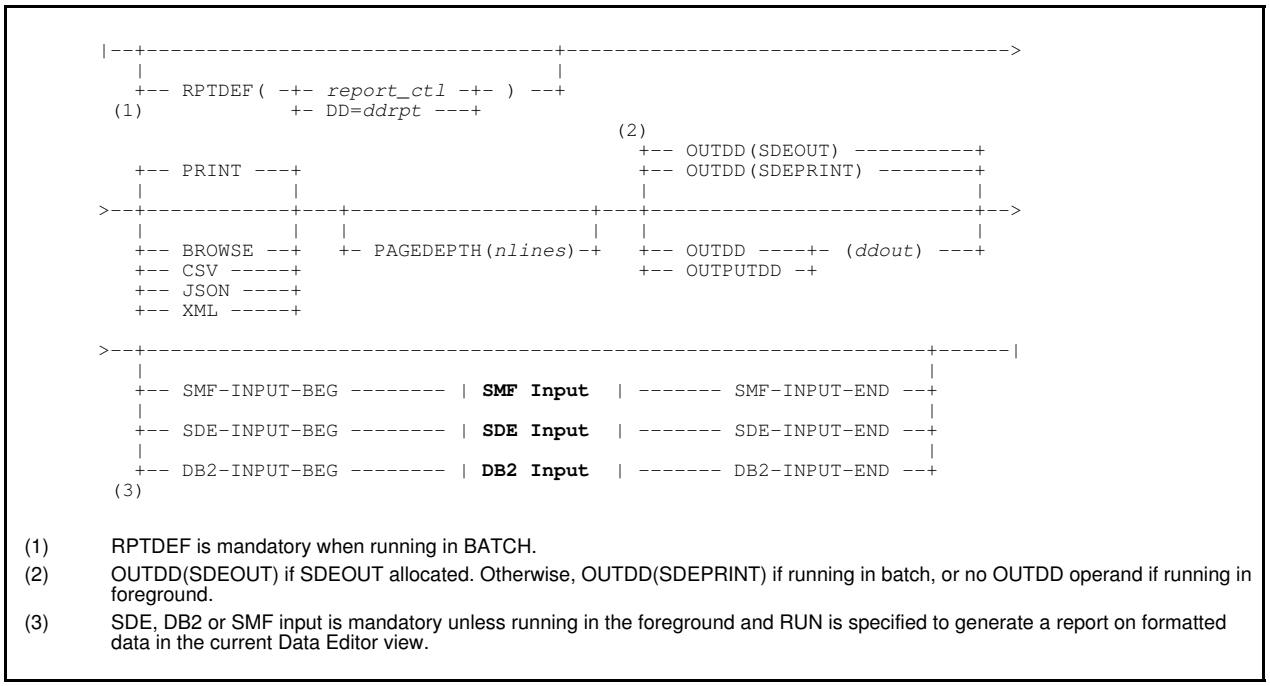
```
REPORT  RUN   RPTDEF(DD=DB2CTL)   OUTDD(DB2OUT)
         DB2-INPUT-BEG
            DB2  SSN(DBCG)  DSN8C10.EMP
         DB2-INPUT-END
```

Using the report definition input allocated to DDname DB2CTL, produce a report of all rows in the sample DB2 table "DSN8C10.EMP" on the local sub-system "DBCG". The generated report is written to the data set allocated to DDname DB2OUT.

Syntax:



Options:

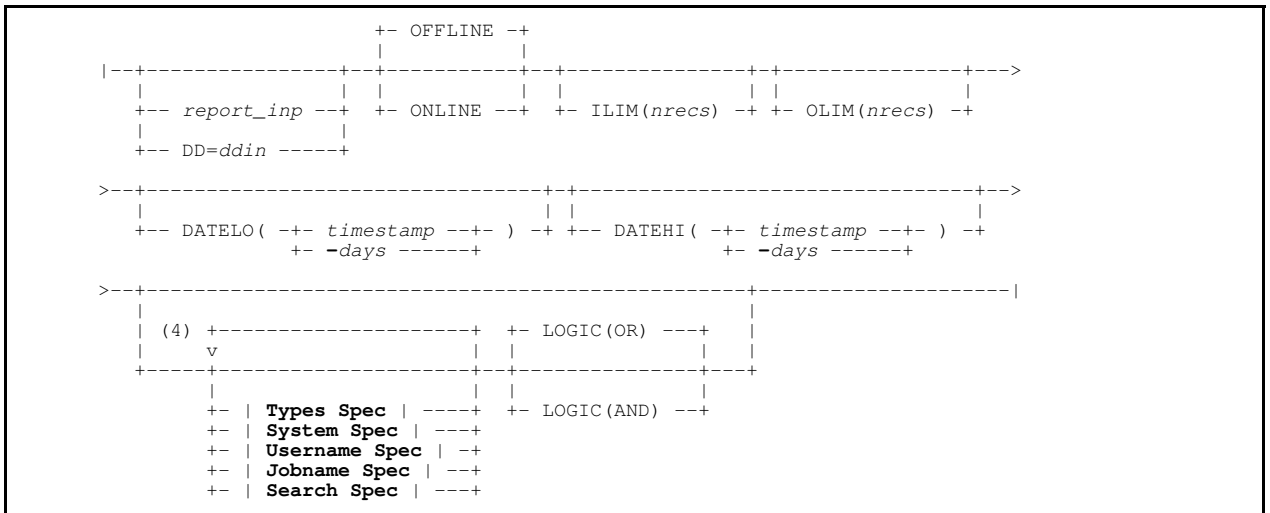


(1) RPTDEF is mandatory when running in BATCH.

(2) OUTDD(SDEOUT) if SDEOUT allocated. Otherwise, OUTDD(SDEPRINT) if running in batch, or no OUTDD operand if running in foreground.

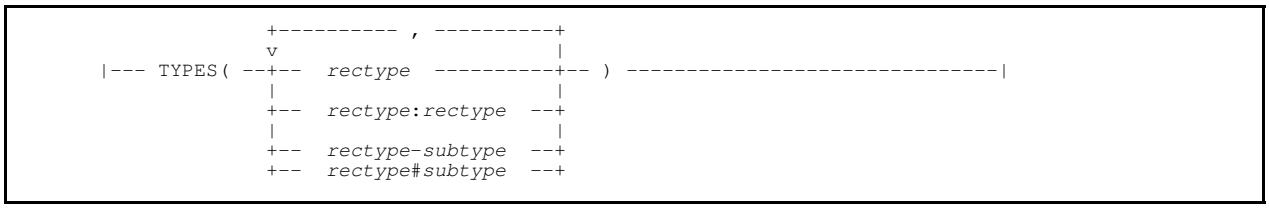
(3) SDE, DB2 or SMF input is mandatory unless running in the foreground and RUN is specified to generate a report on formatted data in the current Data Editor view.

SMF Input:

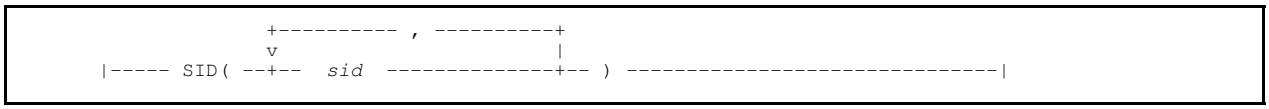


(4) Only 1 each of "***types_spec***", "***system_spec***", "***username_spec***", "***jobname_spec***" and "***search_spec***" may be specified.

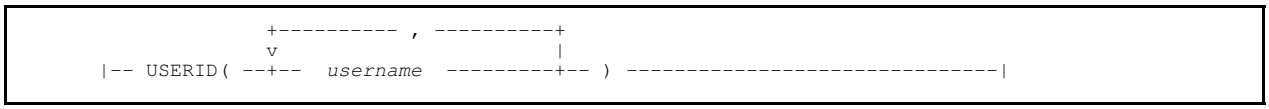
Types Spec:



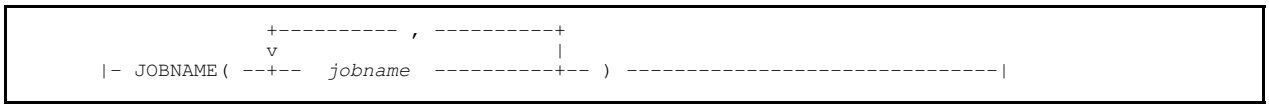
System Spec:



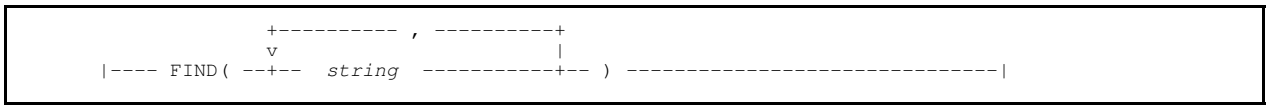
Username Spec:



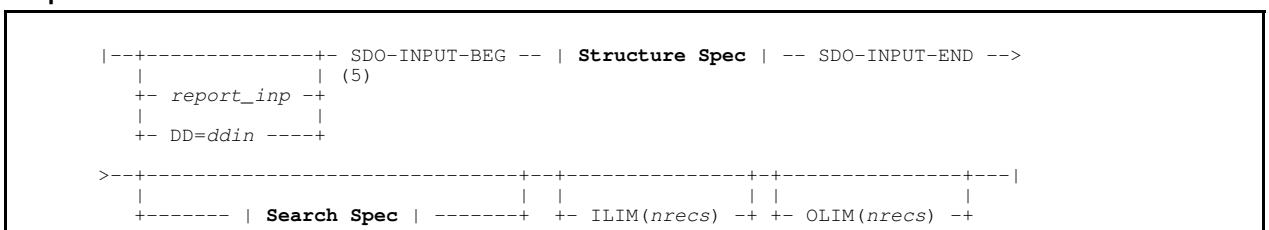
Jobname Spec:



Search Spec:

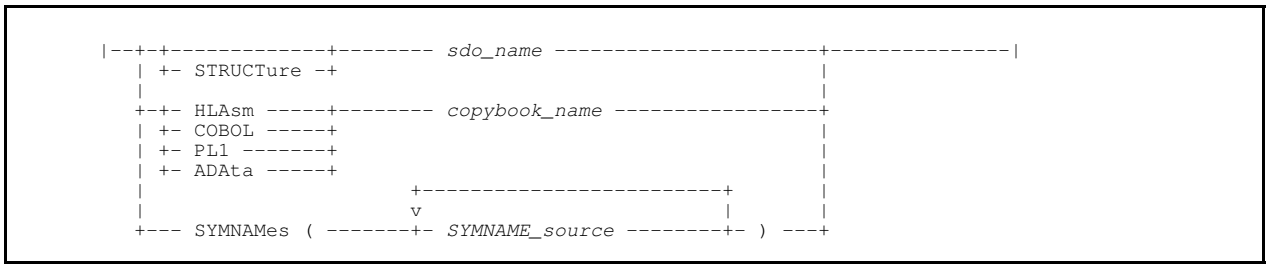


SDE Input:

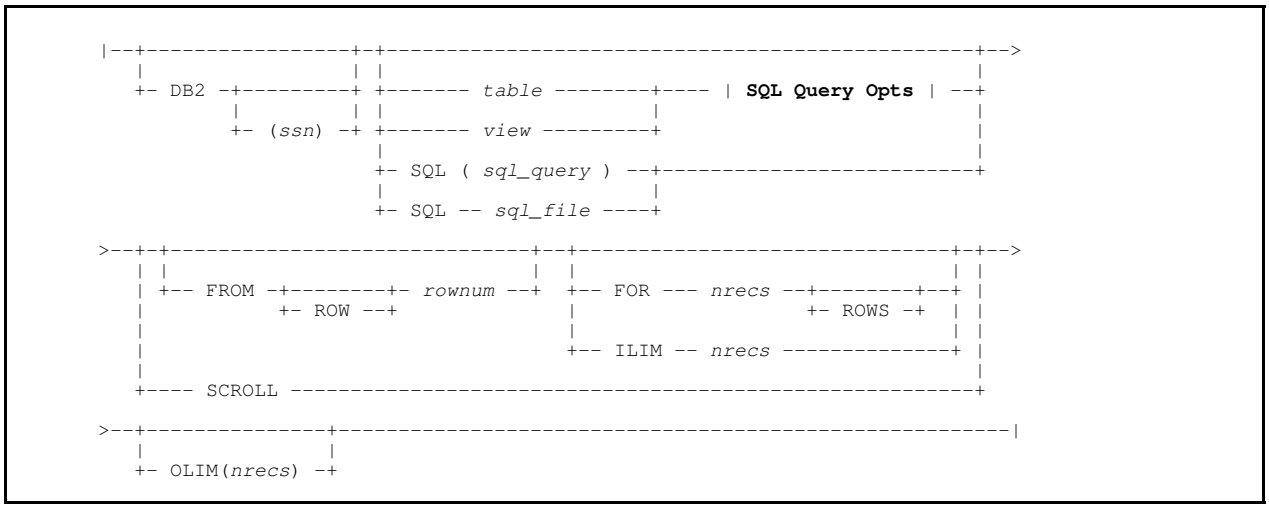


(5) SDO-INPUT-BEG/SDO-INPUT-END with a structure specification must be omitted if the report definition includes a MAP section to define input field mappings.

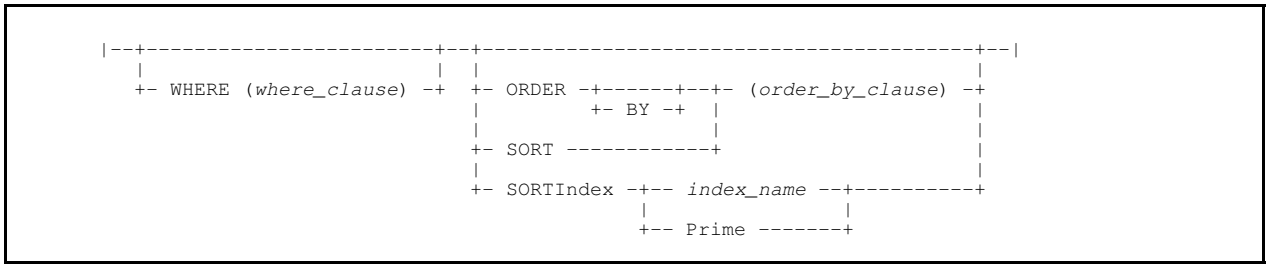
Structure Spec:



DB2 Input:



SQL Query Opts:



Parameters:

ADD | {**BATCH**|**JCL**} | **CMX** | **E** | {**FGRND**|**RUN**} | **L** | **NEW**

Operand that specifies the action to be performed by the REPORT command.

All actions are supported when REPORT is executed in the foreground. However, only action RUN may be used when REPORT is executed in batch.

ADD	Applicable only when executed within a Data Editor view of formatted records, "REPORT ADD" provides a convenient method of populating a report definition input member with column definitions based on fields within records being browsed or edited by the Data Editor. "REPORT ADD" will add one COLUMN definition statement to the *report_ctl* member for each field displayed in the focus line of the Data Editor view. (i.e. The line on which the cursor is positioned.) The Data Editor primary command **SELECT** may be used to open the "Select Display Fields for a Record Type" panel to select the fields for display and to specify the order in which fields are occur within the display. Column definitions will reflect this selection and field order. A "COLUMNS:" section header and other required sections should be manually added to *report_ctl* before using it as the report definition input to the REPORT utility.
BATCH **JCL**	Generate a job stream containing JCL suitable for submission to batch. The job comprises a single job step that executes the the FILEKITB (FileKit batch) program. See "Batch Execution" for details on execution of the REPORT utility in a batch environment.
CMX	Generate a REPORT primary command in a temporary text display. This option is supported purely for REPORT panel operation. The REPORT panels silently generate and execute a REPORT primary command in order to execute the REPORT utility. "REPORT CMX" is only sensible when executed via a panel since executing a REPORT primary command to generate another, almost identical REPORT primary command would be pointless.
E	Edit the report definition source dataset or library member identified by *report_ctl*.

FGRND **RUN**	Execute the REPORT utility to generate the report using the report definition source identified by the RPTDEF operand.
L	Open a library member list to display the members in the report definition source library specified by *report_lib*.
NEW	Edit the report definition source dataset or library member identified by *report_ctl* and delete the contents in preparation for writing a new report definition source. This action is primarily for use in conjunction with "REPORT ADD" when the default report definition source member "REPORT" is to be used for creating simple, one-off reports for the data in the current Data Editor view.

BROWSE | **CSV** | **JSON** | **PRINT** | **XML**

Operand that specifies the format of report output to be generated

BROWSE	Applicable only when executed in the foreground (not in batch), BROWSE will open a Data Editor browse window to display only those record types and fields selected for reporting in the COLUMNS and REQUIRED sections of the report definition input. The BROWSE option allows use of report definitions as templates for browsing formatted data. The browse view will display the values that would be extracted by the REPORT utility for report generation. For this reason, the BROWSE format is also useful for testing new report definitions. Note that fields included in the report definition that are mapped by a secondary record-type, will appear on the same line as fields mapped by the primary record-type provided the secondary record-type is not specified in the REPEAT section. Because the values of fields belonging to repeated record segments change for each segment occurrence, they are not displayed on the primary record segment line.
CSV	Generate Comma Separated Variable (CSV) output records for field values identified by COLUMN statements in the report definition file.
JSON	Generate JavaScript Object Notation (JSON) output records for field values identified by COLUMN statements in the report definition file.
PRINT	Generate a printed report output.
XML	Generate eXtensible Markup Language (XML) output records for field values identified by COLUMN statements in the report definition file.

DATEHI(*timestamp* | **-***days***)**

Applicable to SMF input only, DATEHI specifies a complete or partial absolute timestamp (*timestamp*), or a negative number of days (*-days*) that corresponds to a timestamp value which is relative to the current date. This value will override a value specified by the SMFDATEHI option in the report definition.

DATEHI must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands.

Absolute and Relative timestamp specifications are described in detail by timestamp values under *"Record Filtering"*.

The start of every SMF record contains a common header which includes a timestamp (date and time) at which the record was written to the SMF log (zTME). If DATEHI is specified, only those SMF records with a timestamp earlier than or equal to the DATEHI date and time will be passed to SMF content match criteria record filtering.

An absolute timestamp specification may be truncated to a minimum of 5 bytes ("*yyyy/*") in which case the truncated numeric digits will be set to "9". For example, "**DATEHI(2019/09/22 18)**" is treated as "**DATEHI(2019/09/22 18:99:99.99)**".

A relative timestamp, specified as number of days before the current date, will correspond to a date only. For example, if the current date is **2020/03/05** then "DATEHI(**-5**)" would be equivalent to "**DATEHI(2020/02/29 99:99:99.99)**" since 2020 is a leap year.

DATELO(*timestamp* | **-***days***)**

Applicable to SMF input only, DATELO specifies a complete or partial absolute timestamp (*timestamp*), or a negative number of days (*-days*) that corresponds to a timestamp value which is relative to the current date. This value will override a value specified by the SMFDATELO option in the report definition.

DATELO must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands.

Absolute and Relative timestamp specifications are described in detail by timestamp values under *"Record Filtering"*.

The start of every SMF record contains a common header which includes a timestamp (date and time) at which the record was written to the SMF log (zTME). If DATELO is specified, only those SMF records with a timestamp later than or equal to the DATELO date and time will be passed to SMF content match criteria record filtering.

An absolute timestamp specification may be truncated to a minimum of 5 bytes ("*yyyy/*") in which case the truncated numeric digits will be set to "0". For example, "**DATELO(2018/09)**" is treated as "**DATELO(2018/09/00 00:00:00.00)**".

A relative timestamp, specified as number of days before the current date, will correspond to a date only. For example, if the current date is **2019/11/13** then "DATELO(**-28**)" would be equivalent to "**DATELO(2019/10/16 00:00:00.00)**".

DB2 [**(***ssn***)**]

Applicable to DB2 table input only, specification of DB2 is optional and is only necessary if used to identify a specific DB2 sub-system.

If sepecified, DB2 must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands.

The (*ssn*) operand is optional and identifies the local DB2 sub-system name to which a connection will be made. This will override an *ssn* value specified by DB2(*ssn*) in the INPUT section of the report definition.

Before a connection can be made to the DB2 sub-system, the FileKit DB2 plan must have been bound to that sub-system.

Default for *ssn* is the DB2 sub-system name set in the FileKit DB2 Primary Options menu and saved in the User INI file.

DB2-INPUT-BEG ... DB2-INPUT-END

The DB2-INPUT-BEG and DB2-INPUT-END operand pair indicates DB2 format REPORT utility input. The operands also identify the start and end of a number of REPORT utility command line operands that are specific to DB2 table processing. i.e. DB2 specific operands must be entered after DB2-INPUT-BEG and before DB2-INPUT-END.

The REPORT utility uses FileKit functions to perform special processing for DB2 table input. Firstly, a connection is made with the specified DB2 sub-system via the Call Attachment Facility. Therafter, the program prepares and executes SQL statements to obtain information on and process the input DB2 result table. FileKit will also automatically generate an SDO structure in order to map input DB2 result table rows.

Note that if the REPORT option in the report definition is specified, then this will dictate the REPORT input format. If this option is SDE or SMF, then operands entered between DB2-INPUT-BEG / DB2-INPUT-END that are not applicable to the specified REPORT input format are ignored.

FIND(*string, ...***)**

Applicable to all input types except DB2 table input, the FIND operand specifies one or more comma separated search values (*string*). These string values will override values specified by the FIND option in the report definition for **Unformatted Record Find String matching**.

FIND must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands, or between the SDE-INPUT-BEG and SDE-INPUT-END command line operands.

Note: Filtering of DB2 table rows based on its contents may be achieved using a WHERE clause in the SQL query specified in the DB2 input operands.

The format of *string* is described by search values under *"Record Filtering"*.

If a match on **any** of the FIND search strings is located at **any** position within an unformatted input record, then Unformatted Record Find String matching will return a true result (1). Otherwise a false result (0) is returned.

Unformatted Record Find String matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

For non-SMF input, no other content match criterion is supported. Therefore, a record will be passed for REPORT processing if a true result is returned by Unformatted Record Find String matching.

For example, the following will set a true condition if one of the strings "SYS1.MACLIB", "SYS1.MIGLIB", "SYS1.MODGEN" or "SYS1.MSGEN" (upper or lower case) exists at any location within the unformatted record.

```
FIND(SYS1.MACLIB, SYS1.MIGLIB, SYS1.MODGEN, SYS1.MSGEN)
```

FIND and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a FIND specification exists, then error message ERR065E or ERR066E is returned.

FOR *nrecs* [**ROWS**]

Applicable to DB2 table input only, FOR *nrecs* ROWS specifies the maximum number of rows that may be fetched from the DB2 result table. This will override an *nrecs* value specified by FOR *nrecs* ROWS in the INPUT section of the report definition.

FOR *nrecs* ROWS must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands.

Note that, if an ILIM input limit value is specified, then this will override the *nrecs* value specified on FOR *nrecs* ROWS. However, any *nrecs* value specified by ILIM or FOR will be ignored if DB2 operand SCROLL is also specified to use a DB2 table scrollable cursor.

FROM [**ROW**] *rownum*

Applicable to DB2 table input only, FROM ROW *rownum* specifies the number of the input DB2 result table row from which REPORT processing will start. This will override a row number value specified by FROM ROW *rownum* in the INPUT section of the report definition.

FROM ROW *rownum* must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands.

Rows that occur before the specified row number will be bypassed and not included in the number of rows (*nrecs*) count identified by an input limit (ILIM) or FOR *nrecs* RECS specification.

Note that any specified *rownum* value will be ignored if DB2 operand SCROLL is also specified to use a DB2 table scrollable cursor.

By default, REPORT processing starts from the first row of the result table.

ILIM(*nrecs***)**

Specifies an input limit, the maximum number of records (or DB2 table rows) that will be read from the input data source. This value will override a value specified by the ILIM option in the report definition.

ILIM must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands, or between the SMF-INPUT-BEG and SMF-INPUT-END command line operands, or between the SDE-INPUT-BEG and SDE-INPUT-END command line operands.

For DB2 table input, the specified ILIM value will take precedence over any FOR *nrecs* ROWS specified in DB2 operands of the REPORT command or in the INPUT section of the report definition. The input limit will determine the number of rows fetched from the DB2 result table. Note, however, that any *nrecs* value specified by ILIM or FOR will be ignored if DB2 operand SCROLL is also specified to use a DB2 table scrollable cursor.

Each input record or DB2 row is processed sequentially until the input record threshold (*nrecs*) is reached. At this point, sorting occurs if a SORT section exists in the report definition, otherwise REPORT processing ends. When a SORT section is not present, then REPORT processing may end before the input limit is reached if a specified OLIM output limit threshold is reached first.

Where the input source is **not** a DB2 table, the input limit includes records which may subsequently be excluded from REPORT processing by other record filtering techniques. For example, use of a FILTER section in the report definition or, alternatively, specification of find search strings (FIND), high date (DATEHI) / low date (DATELO) thresholds, or SMF record type (TYPES), job name (JOBNAME), system name (SID) or user name (USERID) field matches.

ILIM(0) implies no input record limit and is set by default when no ILIM operand is supplied, no DB2 FOR *nrecs* ROWS specification exists, and no ILIM option is set in the report definition.

JOBNAME(*jobname, ...***)**

Applicable only to SMF input records, the JOBNAME operand specifies one or more comma separated job name search values (*jobname*). These job name values will override values specified by the SMFJOBNAME option in the report definition for **SMF Record Job Name matching**.

JOBNAME must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands.

A *jobname* value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*.

Unless *jobname* contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the *jobname* value will be truncated or padded with blanks to a length of 8 characters. Furthermore, if no percent ("%") or asterisk ("*") wildcards are specified and *jobname* is an **unquoted** or **quoted** string, then all alpha characters in the string will be upper cased.

A number of SMF record types contain a job name field **zJobName** at a fixed location within the record data. This fixed position may be different for each of the SMF record types. If an SMF record zJobName field contains a match on any of the supplied *jobname* values, then SMF Record Job Name matching will return a true result (1). Otherwise, if no match is found for any of the supplied *jobname* values or the SMF record does not contain a zJobName field, then a false result (0) is returned.

SMF Record Job Name matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

In the following example, a true result will be returned if the SMF record has a zJobName field that contains a job name "GIM" or beginning with "GIM" or any job name of length 5.

```
JOBNAME(GIM*, %%%%%)
```

The following SMF record types are known to contain a zJobName field.

004	010	017	025	034	040	061	064	067	080
005	014	018	026	035	042	062	065	068	110
006	015	020	030	036	060	063	066	069	118

JOBNAME and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a JOBNAME specification exists, then error message ERR066E is returned.

LOGIC(OR|AND)

Applicable to SMF input only, the LOGIC operand specifies the logical operation (**AND** or **OR**) to be used when determining the result of content match criteria record filtering. This logical operation value will override a value specified by the SMFLOGIC option in the report definition.

LOGIC must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands.

The logical operation is used to combine the Boolean values (true or false) returned by each of the specified content match criteria elements:

◊ Unformatted Record Find String matching (FIND)
◊ SMF Record Job Name matching (JOBNAME)
◊ SMF Record System Id matching (SID)
◊ SMF Record Type matching (TYPES)
◊ SMF Record User Name matching (USERID)

Content matching criteria elements may be specified in the report definition input, and/or passed to the REPORT utility via command line operands or panel input fields.

The "AND" or "OR" logical operation is performed between each of the Boolean values returned by the specified content matching criteria to produce an overall true (1) or false (0) result. If the overall result is true, the record satisfies the content match criteria and is passed to REPORT generation processing.

If logical operation **AND** is used then the result returned by **all** of the content checking criterion elements specified for the current REPORT execution must be 1 (i.e. true). If logical operation **OR** is used then **only one** of the values returned by the content checking criterion elements must be 1 (true) in order to return a true result for the record.

Note that other SMF record filtering controlled by high date (DATEHI) / low date (DATELO) thresholds and input record limit (ILIM), does not form part of the content checking criteria and so is not affected by the logical operation.

OLIM(*nrecs*)

Specifies the maximum number of detail line report records (*nrecs*) that may be written to the output dataset. This value will override a value specified by the OLIM option in the report definition.

OLIM must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands, or between the SMF-INPUT-BEG and SMF-INPUT-END command line operands, or between the SDE-INPUT-BEG and SDE-INPUT-END command line operands.

Once the number of output report detail lines reaches this limit, no further input records/DB2 table rows will be processed.

OLIM(0) implies no output record limit and is set by default when no OLIM operand value is supplied and no OLIM option is set in the report definition,

ONLINE|OFFLINE

Applicable to SMF input only, ONLINE indicates that SMF input records are being processed directly from an SMF log data set (**SYS1.xxxx.MANx**), OFFLINE indicates that records are being processed from an SMF archive data set. The specification of ONLINE or OFFLINE will override a value specified by the SMFONLINE option in the report definition.

Note that the REPORT utility does not support processing SMF records directly from the System Logger.

ONLINE/OFFLINE must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands.

Unlike records written to an archive data set by the SMF DUMP (IFASMFDL and IFASMFDP) utilities, records in an SMF log dataset are prefixed by an extra **4-byte record descriptor word (RDW)** and so record-type field mapping must be offset by 4 bytes. The OFFLINE/ONLINE specification will determine whether this offset is to be applied by the REPORT utility.

Beware that any application, including the REPORT utility, that processes records directly from an online SMF log data set, may prevent successful execution of an IFASMFDP CLEAR operation (usually triggered by the IEFU29 exit). This is because the IFASMFDP CLEAR operation requires exclusive access to the SMF dataset.

If no OFFLINE or ONLINE operand is supplied and no SMFONLINE option is set in the report definition, then **OFFLINE** is default.

{ **ORDER** [**BY**] | **SORT** } (*order_by_clause*)

Applicable to DB2 table input only where a DB2 *table* or *view* name is specified as input (as opposed to a fully formed SQL SELECT query).

ORDER BY or SORT specifies a DB2 SQL ORDER BY clause to be applied to the DB2 result table and included in the prepared SQL SELECT query statement generated by the REPORT utility. This ORDER BY clause will override a SORTINDEX, ORDER BY or SORT specification provided in the INPUT section of the report definition.

ORDER BY or SORT must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands.

See IBM publication "*DB2 SQL Reference*" for syntax of the *order_by_clause* which will fetch result table rows in the specified order.

If required, a SORT section may also be specified in the report definition to sort the report output record detail lines. This may be necessary if report detail lines are to be sorted based on the values of one or more *compute-field*.

Operands ORDER BY (or SORT) and SORTINDEX are mutually exclusive. If both are specified, the specified ORDER BY clause will take precedence.

OUTDD(*ddout*)
OUTPUTDD(*ddout*)

Specifies the DD name (*ddout*) that is allocated to the data set to which the generated output report will be written. This output DD name will override output to DD name SDEOUT or the output DD name, DSN or HFS/ZFS file path provided in the OUTPUT section of the report definition.

REPORT utility output data object is determined in the following order of precedence:

1. If the REPORT utility is executed using the REPORT primary command and the OUTDD (or OUTPUTDD) operand is specified, then output is to the DD name specified by operand OUTDD.
2. If DD name SDEOUT is allocated, DD=**SDEOUT**
3. The data object (*ddout*, *dsname* or *fileid*) specified by the OUTPUT section of the report definition.
4. If executing in batch, DD=**SDEPRINT**. (Allocation of SDEPRINT is mandatory for batch execution.)
5. For FileKit foreground execution only, an unsaved, in-storage file assigned a DSN "*userpfx*.REPORT.D*yyyyddd*.T*hhmmss*.TXT", where *userpfx* is the DSN prefix associated with the current FileKit user and *yyyyddd* and *hhmmss* is the current Julian date and time respectively. The in-storage data is displayed in a Text Editor view and may be saved to a specific DSN using "SAVE *dataset-name*".

PAGEDEPTH(*nlines*)

For printed report output only, PAGEDEPTH specifies the number of lines *nlines* per page. This value will override a value specified by the PAGEDEPTH option in the report definition.

If a PAGEDEPTH value is not specified and no PAGEDEPTH option exists in the report definition, then the default page depth will be the value assigned by the PAGEDEPTH Data Editor option. (See "*PAGEDEPTH - SET/QUERY/EXTRACT Option*" in the "*FileKit Data Editor (SDE)*" manual.)

report_ctl

Specifies the name of the report definition data set or library member. A *report_ctl* may be one of the following:

1. A sequential dataset DSN
2. A library DSN followed by a parenthesised member name
3. A member name. (Foreground execution only.)

A *report_ctl* object may be specified as the only operand on the REPORT command, in which case the default action is **RUN** and the default report output format is **PRINT**.

If executing in the FileKit foreground, then *report_ctl* may be omitted or specified simply as a member name. If omitted, the default for *report_ctl* is member name "REPORT". When no library DSN is specified, the default library "*userpfx*.FILEKIT.RPT" is used (see *report_lib* description). As for *report_lib*, this library data set will be allocated as new if it does not already exist.

For "REPORT ADD", "REPORT E" and "REPORT NEW" the specified member name may be created as new. In all other cases the report definition file specified by *report_ctl* must already exist.

report_inp | **DD=***ddin*

Used specifically for "REPORT RUN" execution for SDE and SMF format input, these operands identify the data source containing the input data records for report processing. This data source will override the input data source (DD name, DSN or HFS/ZFS file path) provided in the INPUT section of the report definition.

report_inp or DD=*ddin* must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands, or between the SDE-INPUT-BEG and SDE-INPUT-END command line operands. For foreground processing only, if no INPUT section exists in the report definition and no *report_inp* or DD=*ddin* is specified, then the REPORT utility will attempt to process non-excluded records from the current FileKit Data Editor view. If no current view exists or data does not include record-type field names specified in the report definition, then the appropriate error is returned.

The input may be specified as a sequential dataset, library dataset and member name or HFS/ZFS file path (*fileid*), or as an allocated DD name (*ddin*). If specified as a DD name, *ddin* must be prefixed with "**DD=**".

If DD=*ddin* is specified, *ddin* may be allocated to a DASD or TAPE dataset. Alternatively, it may be allocated to a concatenation of data sets, thus allowing records to be processed from multiple, consecutive input sources. e.g. Multiple generations of the same GDG.

report_lib

Used specifically by "REPORT L", *report_lib* identifies the library containing report definition members. A *report_lib* may be one of the following:

1. A library DSN
2. A library DSN followed by a parenthesised member mask
3. A member mask

A member mask may contain asterisk ("*") wildcard symbols which each represents 0 or more of any consecutive characters in the member name, and/or percent ("%") wildcard symbols which each represent any single character within the member name.

All members, or only those members that match the supplied member mask, will be included in the list of report definition source members.

If *report_lib* is not specified or is specified as a member mask without a library DSN, then a default library DSN of "*userpfx*.FILEKIT.RPT" is used, where *userpfx* is the user's default DSN high level qualifiers as set by the FileKit SITE INI file and assigned to environment variable "MyHLQ". If data set "*userpfx*.FILEKIT.RPT" does not yet exist, then it is automatically allocated as a new PDSE library with RECFM=VB, LRECL=16384, BLKSIZE=0 and CYLINDERS(1,1).

RPTDEF(*report_ctl* | **DD=***ddrpt***)**

The RPTDEF operand identifies the report definition source used for a "REPORT CMX", "REPORT JCL" or "REPORT RUN" execution. RPTDEF is mandatory for "REPORT RUN" execution in batch.

The report definition source may be specified as a sequential data set or library member (see *report_ctl* description) or as an allocated DD name (*ddrpt*). Note that DD= is mandatory if *ddrpt* is used.

If RPTDEF is omitted, then library member "REPORT" in the default report source library is used. i.e. "*userpfx*.FILEKIT.RPT(REPORT)", where *userpfx* is the current user's default DSN high level qualifiers.

SCROLL

Applicable only to DB2 table input, SCROLL indicates that a DB2 scrollable INSENSITIVE cursor is to be used to fetch DB2 rows.

SCROLL must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands.

If SCROLL is used, then once the cursor has been opened, only a relatively small number of rows will be kept in storage at any time. At open the results table is materialised (i.e. a temporary copy is made) which, for large tables, may mean that opening the cursor may take a long time and consume much resource.

Use of DB2 scrollable cursors may not be desirable and so is possible only if the DB2 administrator has set **DB2.SCROLL=YES** in the FileKit Site INI file.

SCROLL is incompatible with FOR, ILIM and FROM values specified via the INPUT section of the report definition, or passed as parameters by the FileKit REPORT panels or REPORT command. If specified, values provided for these operands will be ignored if SCROLL is also used.

SDE-INPUT-BEG ... SDE-INPUT-END

The SDE-INPUT-BEG and SDE-INPUT-END operand pair indicates SDE (Structure Data Edit) format REPORT utility input. The operands also identify the start and end of a number of REPORT utility command line operands that are specific to SDE record input processing. i.e. SDE specific operands must be entered after SDE-INPUT-BEG and before SDE-INPUT-END.

For this type of input, fields must be defined within the input records. This is done either by specifying a MAP section in the report definition file, or by providing an external record formatting structure (PL1 or COBOL copybook, HLASM DSECT or FileKit SDO). The name of a structure data set or library member may be specified via the INPUT section of the report definition, or passed as a parameter on the FileKit REPORT panels or REPORT command (see SDO-INPUT-BEG / SDO-INPUT-END below).

Note that if the REPORT option in the report definition is specified, then this will dictate the REPORT input format. If this option is DB2 or SMF, then operands entered between SDE-INPUT-BEG / SDE-INPUT-END that are not applicable to the specified REPORT input format are ignored.

SDO-INPUT-BEG ... SDO-INPUT-END

For SDE format input only, the SDO-INPUT-BEG and SDO-INPUT-END operands identify the start and end of REPORT utility command line operands that define the a record formatting structure. Record formatting structure operands must be entered after SDO-INPUT-BEG and before SDO-INPUT-END and will override a structure definition provided in the INPUT section of the report definition. Providing a record formatting structure will also override use of record field mappings defined by a MAP section in the report definition.

SDO-INPUT-BEG and SDO-INPUT-END operands must be entered between SDE-INPUT-BEG and SDE-INPUT-END command line operands.

The structure ultimately used by the REPORT utility to map the input records will be a FileKit structured data object (SDO). The name of an SDO may be passed directly to the REPORT utility or it may be automatically generated from an alternative source (e.g. a COBOL copybook). The following operands may be entered between SDO-INPUT-BEG and SDO-INPUT-END.

[**STRUCTURE**] *sdo_name*

Specifies *sdo_name*, the sequential DSN or library DSN and member name of the FileKit SDO structure used to map the input records. This is an *sdo_name* created via the FileKit Create Structure panels or via the CREATE STRUCTURE primary command.

{ **HLASM** | **COBOL** | **PL1** | **ADATA** } *copybook_name*

Specifies the format and name of the input record mapping source file (*copybook_name*). The *copybook_name* is a library DSN and member name and the format may be one of the following:

ADATA	The SYSADATA output generated by the assembley of an assembler source using the HLASM (High Level Assembler) program, or generated by the compilation of a COBOL or PL1 source using the Enterprise COBOL or Enterprise PL1 compiler.
COBOL	A copybook member containing COBOL data division - data description source.
HLASM	An Assembler source member containing DSECT definitions.
PL1	An %INCLUDE directive source member containing PL1 data declaration structures.

FileKit will interpret the record mapping source to generate a temporary SDO structure.

SYMNAMES(*SYMNAME_source ...* **)**

Specifies one or more *SYMNAME_source* entries, where *SYMNAME_source* is the name of a sequential data set or library and member name containing SYMNAMES symbol statements as supported by the SORT utility.

Symbol statements must include field definitions specified as position, length and format. Please refer to your SORT utility documentation (e.g. the IBM publication *"z/OS DFSORT Application Programming Guide"*) for details on the symbol statement.

FileKit will interpret the SYMNAMES field definitions to generate a temporary SDO structure containing a single record mapping (record-type).

SID(*sid, ...***)**

Applicable only to SMF input records, the SID operand specifies one or more comma separated system identification search values (*sid*). These system identification values will override values specified by the SMFSID option in the report definition for **SMF Record System Id matching**.

SID must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands.

A *sid* value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*.

Unless *sid* contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the *sid* value will be truncated or padded with blanks to a length of 4 characters. Furthermore, if no percent ("%") or asterisk ("*") wildcards are specified and *sid* is an **unquoted** or **quoted** string, then all alpha characters will be upper cased.

All SMF record types contain a system identifier field **zSID** in the record header. If an SMF record zSID field contains a match on any of the supplied *sid* values, then SMF Record System Id matching will return a true result (1). Otherwise, if no match is found for any of the supplied *sid* values, a false result (0) is returned.

SMF Record System Id matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

In the following example, a true result will be returned if the zSID field contains a system id value "XS1", a value beginning with "S0" followed by any single character followed by "1", or a value of up to 4 characters in length ending in "Z".

```
SID('xs1', S0%1, '*Z')
```

SID and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a SID specification exists, then error message ERR066E is returned.

SMF-INPUT-BEG ... SMF-INPUT-END

The SMF-INPUT-BEG and SMF-INPUT-END operand pair indicates SMF (z/OS System Management Facilities) format REPORT utility input. The operands also identify the start and end of a number of REPORT utility command line operands that are specific to SMF record input processing. i.e. SMF specific operands must be entered after SMF-INPUT-BEG and before SMF-INPUT-END.

The REPORT utility performs special processing for SMF record input and uses the standard SMF record mapping structures provided by FileKit to generate the required SDO structure.

Note that if the REPORT option in the report definition is specified, then this will dictate the REPORT input format. If this option is DB2 or SDE, then operands entered between SMF-INPUT-BEG / SMF-INPUT-END that are not applicable to the specified REPORT input format are ignored.

SORTINDEX { *index_name* | **PRIME** }

Applicable to DB2 table input only where a DB2 *table* or *view* name is specified as input (as opposed to a fully formed SQL SELECT query).

SORTINDEX specifies *index_name*, the name of an existing DB2 Index for the table being processed. The index identifies the key columns/expressions by which the table rows will be ordered for input to the REPORT utility. This SORTINDEX value will override a SORTINDEX, ORDER BY or SORT specification provided in the INPUT section of the report definition.

SORTINDEX must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands.

PRIME may be specified as an alternative to indicate that the primary index should be used.

If required, a SORT section may also be specified in the report definition to sort the report output record detail lines. This may be necessary if report detail lines are to be sorted based on the values of one or more *compute-field*.

Operands ORDER BY (or SORT) and SORTINDEX are mutually exclusive. If both are specified, the specified ORDER BY clause will take precedence.

table | *view* | **SQL(***sql_query***)** | **SQL** *sql_file*

Applicable to DB2 table input only, these operands define the DB2 result table to be used as input to the REPORT utility. This DB2 result table definition will override any result table definition provided in the INPUT section of the report definition.

A DB2 result table definition must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands.

These operands are mutually exclusive and may be specified as follows:

table	The name of a DB2 table (*table*) as defined in the SYSIBM.SYSTABLES catalog table.
view	The name of a DB2 view (*view*) as defined in the SYSIBM.SYSVIEWS catalog table.
SQL(*sql_query***)**	Specifies *sql_query*, a complete DB2 SQL query that generates a result table. For example, the SQL query may include clauses that select specific columns, join tables, filter and order the table rows.
SQL *sql_file*	Specifies *sql_file*, a sequential DSN or library DSN and member name in which a DB2 SQL query is saved. For example, this may be a library member containing a SQL query used as input to SPUFI or the FileKit EXECSQL utility.

Both *table* and *view* may be specified with either 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default for *location* is the local DB2 server and the default for *schema* is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID). Note that the user's SQLID is set via the FileKit DB2 Primary Options menu and saved in the User INI file.

If *table* or *view* is used, then FileKit will generate an SQL query clause (e.g. "SELECT * FROM *table*").

If one of the *SQL* type operands is used and the report definition input includes a FILTER section, then the filter clause will be ignored and warning message ZZSR064W returned. This is because the FILTER section will attempt to generate a WHERE clause to add to the SQL Query. However, the SQL Query passed to the REPORT utility is already fully formed.

The SQL query specified by the **SQL** operand or generated by FileKit is executed as a prepared DB2 SQL statement and the result table rows passed to the REPORT utility.

TYPES({*rectype* | *rectype*:*rectype* | {*rectype-subtype* | *rectype#subtype*} }, ...**)**

Applicable only to SMF input records, the TYPES operand specifies one or more comma separated SMF record type identification values (*rectype*, *rectype*:*rectype*, *rectype-subtype* or *rectype#subtype*). These SMF record type identification values will override values specified by the SMFTYPES option in the report definition for **SMF Record Type matching**.

TYPES must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands.

A description of each of the different SMF record type identification values is documented in SMF Type Values under *"Record Filtering"*.

All SMF record types contain an SMF record type field **zRTY** and some also contain a sub-type field **zSTY** in the record header. If an SMF record contains a match on any of the supplied SMF record type identification values, then SMF Record Type matching will return a true result (1). Otherwise, if no match is found for any of the supplied values, a false result (0) is returned.

SMF Record Type matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

In the following example, a true result will be returned if the input SMF record type (zRTY field value) is 42 (any sub-type), or if the SMF record type is 119 with sub-type (zSTY field value) of 21.

```
TYPES(42, 119#21)
```

TYPES and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a TYPES specification exists, then error message ERR066E is returned.

USERID(*username***, ...)**

Applicable only to SMF input records, the USERID operand specifies one or more comma separated user name search values (*username*). These user name values will override values specified by the SMFUSERID option in the report definition for **SMF Record User Name matching**.

USERID must be entered between the SMF-INPUT-BEG and SMF-INPUT-END command line operands.

A *username* value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*.

Unless *username* contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the *username* value will be truncated or padded with blanks to a length of 8 characters. Furthermore, if no

percent ("%") or asterisk ("*") wildcards are specified and *username* is an **unquoted** or **quoted** string, then all alpha characters in the string will be upper cased.

For example, if SMF records contain a zUserId field value "ABC", "ABC1", "ABCXXX" and "XABC" then "USERID(abc)" would match "ABC" only, "USERID(abc*)" would match "ABC", "ABC1" and "ABCXXX", "USERID(%abc)" would match "XABC" only and "USERID(*abc*)" would match all 4 values.

A number of SMF record types contain a user name field **zUserId** at a fixed location within the record data. This fixed position may be different for each of the SMF record types. If an SMF record zUserId field contains a match on any of the supplied *username* values, then SMF Record User Name matching will return a true result (1). Otherwise, if no match is found for any of the supplied *username* values or the SMF record does not contain a zUserId field, then a false result (0) is returned.

SMF Record User Name matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR".
3. All other specified SMF content match criteria each return a true result.

In the following example, a true result will be returned if the SMF record has a zUserId field that contains a user name of any length up to a maximum of 8 characters ending with 1, or a user name beginning with "ABC" followed by any single character followed by "DEFG".

```
USERID(*1, ABC%DEFG)
```

The following SMF record types are known to contain a zUserId field.

004	014	020	030	035	042	062	065	068	110
005	015	025	032	036	060	063	066	069	118
006	017	026	034	040	061	064	067	080	119
010	018								

USERID and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a USERID specification exists, then error message ERR066E is returned.

WHERE(*where_clause***)**

Applicable to DB2 table input only where a DB2 *table* or *view* name is specified as input (as opposed to a fully formed SQL SELECT query).

WHERE specifies a DB2 SQL WHERE clause to be used to filter the DB2 result table rows and is included in the prepared SQL SELECT query statement generated by the REPORT utility. This WHERE clause will override a WHERE clause specification provided in the INPUT section of the report definition.

WHERE must be entered between the DB2-INPUT-BEG and DB2-INPUT-END command line operands.

A DB2 where clause may also be provided via the FILTER section of the report definition. If both a FILTER section and a WHERE specification exists, then the contents of the FILTER section are ignored and warning message ERR064W is returned.

See IBM publication "*DB2 SQL Reference*" for syntax of the *where_clause* which will filter and include only table rows that match the where clause criteria.

REPORT Definition

The REPORT definition input contains the control statements that configure the required contents and layout of the report output.

This chapter details the format and syntax rules of report definition control statements. It also provides details on the syntax and actions performed by each of the report definition sections supported by the REPORT utility.

Syntax Rules

Records are read from the input REPORT definition file and processed one at a time in order of input sequence. The following describes how each of the REPORT definition input records is intepreted by the REPORT utility.

Statement Continuation

By default, every record read from the REPORT definition input file is identified as a control statement. The control statement text starts at the first character and ends at the last character of the record.

For practical or aesthetic purposes, it may be necessary to continue a control statement onto the next input record. To achieve this, an uncommented statement continuation symbol, **backslash** ("\"), should be entered as the last non-blank character of the input record containing the start of the control statement text. If the continuation symbol is part of comment data, then it will be ignored.

Input definition file record processing will join the character immediately before the continuation symbol with the first character of the next record. The continuation symbol itself is removed from the control statement text.

If the control statement needs to span multiple input records, then the uncommented continuation symbol must exist as the last non-blank character of all records over which the control statement spans, except the last.

For example, the following illustrates a single break definition statement that spans 13 records of the REPORT definition file.

```
BREAK:
  SMF119#02_TCP_Connection_Termination.zConnectStart  10 \
            AVERAGE    \
            NZAVERAGE  \
            MAXIMUM    \
            MINIMUM    \
            NZMINIMUM  \
            SPACEBEFORE(1)    \
            SPACEAFTER(1)      \
            FOOTING( \
                    <newline>  \
                    ' -- End of' zRNAME ' statistics for' zConnectStart (10) '--' \
                    <newline>  \
                   )
```

Statement Separation

An REPORT definition record may be split into a number of control statements using an uncommented control statement separation symbol, **semi-colon** (";"), which is not part of a quoted character string literal. The separation symbol marks the end of one control statement and the start of the next. If the separation symbol is part of comment data, then it will be ignored.

Input definition file record processing will terminate a control statement at the character immediately before the separation symbol and start the next control statement at the first character following the separation symbol. The separation symbol itself is removed from control statement text.

In the following example, the report definition input record is split into 3 column definitions belonging to the same report detail line for which values will be displayed.

```
COLUMNS:
  CUSTID ("Customer|Ref#");  CONTACT ("Contact|Name");  COMPANY ("Company|Name" CENTRE) 30
```

Comments

Comment text may be entered in REPORT definition file using either of the following 2 methods:

1. **Asterisk** ("*") as the first non-blank character on an input record.
2. Between unquoted **slash-asterisk** ("/*") and **asterisk-slash** ("*/") character pairs anywhere within the input records.

When a record is read that contains an asterisk ("*") as the first non-blank character, then the record is bypassed and processing continues with the next input record. Any statement separation or statement continuation specification will be ignored.

In the following example, the input records containing TOTAL and REPEAT parameters for the control break definition will be ignored. Statement continuation is picked up at the record containing the HEADING parameter that follows.

```
BREAK:
  SMF119#02_TCP_Connection_Termination.zRName   \
   *        TOTAL()  \
   *        REPEAT \
            HEADING( \
          <newline> 'This is a header for group:' zRNAME (RIGHT,10) \
          <newline> '---------------------------------------') \
            SPACEAFTER( PAGE )
```

When a slash-asterisk ("/*") character pair is encountered anywhere within the input records but is not part of a quoted character string literal, then it denotes the start of comment text. The comment text is ended by an asterisk-slash ("*/") character pair which is also not part of a quoted character string literal.

If a control statement contains comment text started by a slash-asterisk ("/*") character pair, then text on subsequent records is joined to the end of the control statement text until the comment text is ended by the asterisk-slash ("*/") character pair. Thus the comment text may span a number of input records.

Statement separation or statement continuation symbols that exist within the comment text will be ignored.

The following example contains numerous comment text specifications between the definition of a 3 line report page header. In the definition of the 3rd header line, the control statement is continued to included the "Duration" field value. Between the continued control statement is a string of comment text that spans 6 lines.

```
HEAD:
 /* 1st Page Header. */
  #TIMESTMP /       /* Time stamp is left adjusted in 1st page header. */

 /* 2nd Page Header. */
  "TCP/IP Connection Durations by Resource Name on:" zTME

 /* 3rd Page Header. */
  "Resource:" zRNAME  10 "First Connection:" zConnectStart  10   \
    /*  The "Duration" compute-field uses built-in functions Time2Secs()
    |   and Secs2Time() and is calculated as:
    |
    |  DURATION = Secs2Time( Time2Secs(zConnectEnd) - Time2Secs(zConnectStart) )
    |
    */  "Duration:" :Duration (8)   /* Last part of 3rd page header print-expression. */
```

Character String Literals

Character string literals are used to define fragments of text to be included in the report output. The value of a character string literal is constant and so remains unchanged throughout execution of the REPORT utility.

A character string literal is specified as *literal* and must be enclosed in either apostrophes ('*literal*') or quotation marks ("*literal*"). e.g.

```
"Contents of Album:"
```

The symbol selected to enclose *literal* may not be used as a character within the literal string unless it is escaped. An escaped occurrence of the enclosing symbol within *literal* will be treated as a character within the text string. The escape character is the enclosing symbol itself so, if quotation marks are used to enclose *literal*, then two adjacent quotation marks within *literal* ("") will represent a single occurrence of a quotation mark (") in the text string.

In the following example, apostrophe is used to enclose a *literal* value which contains both quotation marks and an apostrophe. The apostrophe following "John" is escaped so that it is treated as a single occurrence of the symbol.

```
'Report for "St. John''s Priory"'
```

Page Width

Printed report output lines have a maximum length equal to the page width value (plus 1 for the print ASA character).

The page width value is not specified as a report utility parameter but is instead calculated as the maximum length of the following report lines:

- Page heading lines (HEAD)
- Page footing lines (FOOT)
- Control break lines (BREAK)
- Column detail lines (COLUMNS)

The page width value is used by the REPORT utility to align portions of text in page heading and footing lines. For example, the following is a one line page heading definition.

```
HEAD:
  #TIMESTAMP  /  "Music Collection"  / "Page" #PAGE (RIGHT,5)
```

If the page width value is calculated as being 80, then the following printed report page heading generated will be length 81 with a page throw ASA character "1" in the first column.

```
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8.
12020/02/06 12:47              Music Collection                      Page     1
```

Note that, if a report line exceeds the maximum length of the allocated report output data set then it will be truncated.

Record Types

A record type is the name given to a group structure that maps fields to the data belonging to a record or record segment. A record type name is referenced throughout this manual as ***record-type***.

Record type mappings are defined within a FileKit structure definition object (SDO). When the REPORT utility starts, an SDO containing the required record type definitions is loaded into storage. Therefater, one of the record types contained in the SDO gets assigned to each input record or record segment as it is read from the input source.

FileKit SDO Structure

A FileKit SDO structure must be provided as input to the REPORT utility unless the report is generated from one of the following:

1. SMF records.
2. A DB2 result table.
3. A report definition that contains fields defined via the MAP section.

The specified FileKit SDO structure containing the required record types may be an already generated member of an SDO library or an in-storage object generated by FileKit from a COBOL, PL1 or HLASM source library member.

If the SDO is to be generated by FileKit, then the source library member must contain one or more data mapping group structures defined as either COBOL data description entries, PL1 structure definitions or HLASM DSECTs. A record type definition will be created for each COBOL 01 level group definition, PL1 declared structure or Assembler DSECT. If input records are to be mapped using different group structures obtained from more than one source member, then a permanent SDO library member should be created and used instead. (See "*Create Structure from COBOL/PL1 Copybooks*" in the "*FileKit Reference and User Guide*".)

Note that, for SMF record input, a temporary, in-storage SDO structure object with the name "REPORT" is automatically generated by FileKit for use by the REPORT utility. This SDO will contain only those record type definitions that match the *record-type* names specified in the COLUMNS and REQUIRED sections of the report definition file.

Similarly, for DB2 table input, FileKit uses information obtained from the SQLDA to generate a temporary, in-storage SDO containing a single record-type definition that maps all the rows of the input result table.

Record Type Assignment

As a record is read from the input data source, it is assigned one of the record types defined in the SDO structure.

The record-type used to map the data is selected based on the contents of the record and the selection criteria associated with the record-type. e.g. For SMF input records, the record-type "*SMF119#01_TCPIP_Statistics*" is selected to map the data if the record has a value of "119" in the "zRTY" (SMF record type) field and a value of "1" in the "zSTY" (SMF record sub-type) field.

If the input records are segmented (as is the case for SMF record types), then a primary record-type is assigned to the data at the start of the input record. This is known as the primary or base segement. Thereafter, record content and selection criteria determine the secondary record-type mappings assigned to the remainder of the record data. These are known as secondary record segements.

Record Type Specification

Unless the input SDO structure contains only one record type definition or has been generated for DB2 table input, then the record type name (*record-type*) to which a field belongs must be included in the *fieldname* specification for an input record field in the COLUMNS or REQUIRED sections of the Report Definition file. (i.e. *record-type.input-field*)

For example, the following report output column definition will uniquely identify input record field "TITLE" as belonging to the record type "CONTACT":

```
COLUMNS:
  CONTACT.TITLE
```

If a *record-type* has been identified in the COLUMNS or REQUIRED sections, then it may also be referenced in the REPEAT and RESET sections to trigger an output report and a reset to null of input field values.

If a field of the same name and data type exists in different record type definitions within the SDO, then for each input record assigned one of these record types, it may be desirable for the field's values to be reported in the same column output.

For example, the numeric fields "LENGTH", "WIDTH", "HEIGHT" may exist in 2 record type definitions "DIM_WINDOW" and "DIM_DOOR". In the report output, you may want the values of these 3 fields to be reported in the same 3 columns, regardless of whether the input record is mapped by "DIM_WINDOW" or "DIM_DOOR"

To achieve this, the *record-type* used in the *fieldname* specification may have a prefix that is common to both record type names followed by an asterisk ("*") wildcard. For this example, the column definitions would be as follows:

```
COLUMNS:
  DIM*.LENGTH
  DIM*.WIDTH
  DIM*.HEIGHT
```

Note that use of an asterisk wild card is valid only as a suffix on *record-type*.

Fields

Field specifications are the names of fields which contain values required for report generation. These field values may vary over the course of the REPORT program execution.

The field names may be referenced in sections of the REPORT definition input to identify report columns, statistics columns, sort fields and break fields. They may also be referenced in print expressions to include the current value of a field in an output text string.

Field specifications are categorised as one of the following:

1. An input record field.
2. A built-in field.
3. A computed field.

Input Record Fields

An input record field identifies the name of a field defined in the record-type structure used to map an input record or record segment. An input record field is referenced as *input-field* throughout this manual.

As a record is read from the input data set, it is assigned one of the record-types defined in the accompanying FileKit data mapping structure (SDO). Furthermore, if the record is segmented, each segment of the record data is assigned a record-type mapping and processed separately by the REPORT utility. (Note that most SMF records contain segmented data.)

Once record data has been mapped, values are extracted for all input record fields that are defined in the REPORT definition and also exist in the mapping for the record (or record segment) currently being processed. The values of all other fields defined in the record-type assigned to the current record are ignored.

Input Record Field Specification

The COLUMNS and REQUIRED sections of the Report Definition file define all the input record fields (*input-field*) to be used in the report generation. An error occurs if an *input-field* is referenced in another section but is not defined in the COLUMNS or REQUIRED section.

An *input-field* specification may comprise more than one qualifier name in order to specify the record-type and uniquely identify a field within the record-type mapping. Where *input-field* is qualified, then each qualifier corresponds to the name of a group in the field's record-type/sub-group/field name hierarchy. Each qualifier must be separated from the last using a dot/period (".") character.

Depending on the context in which it is used and the structure of the input records, an *input-field* identifier may be referenced in 1 of 2 ways:

Unqualified

This is the simplest form of *input-field* and is the name of the field within a record-type definition.

An unqualified *input-field* should be used when referenced as any of the following:

- The *fieldname* reference in a COLUMNS or REQUIRED section if the field name is unique within the record-type mapping.

  If more than one occurrence of the field name exists within the record-type mapping (e.g. within different group fields) then a qualified *input-field* must be used.

  The COLUMNS and REQUIRED sections define the input record fields for the REPORT utility execution. e.g.

```
COLUMNS:        /* Records mapped by structure containing record-types "ALBUM" and "TRACK". */
   ALBUM.TITLE         /* Unique field name "TITLE"   in record-type "ALBUM".   */
   TRACK.RUNTIME       /* Unique field name "RUNTIME" in record-type "TRACK".   */
```

- The *fieldname* reference in a BLANKWHENZERO, BREAK, MAP, SORT or STATISTICS section if the field is defined as an unqualified *input-field* in the COLUMNS or REQUIRED section.

- A variable in a REXX expression within the COMPUTE section if the *fieldname* is specified in the COLUMNS or REQUIRED section as *record-type.input-field* or *input-field* and *input-field* is also unqualified. e.g.

```
COMPUTE:              /* Update compute-field "ELAPSED" */
   ELAPSED = format(RUNTIME/1000,,3)     /* Value derived from input-field "RUNTIME". */
```

- A text substitution variable in a print expression if the *fieldname* is specified in the COLUMNS or REQUIRED section as *record-type.input-field* or *input-field* and *input-field* is also unqualified. e.g.

```
BREAK:                /* Trigger control break following change in "ALBUM" value. */
   ALBUM  HEADING("Tracks on Album:" ALBUM)     /* Substitute ALBUM name in header text. */
```

Fully Qualified

A fully qualified *input-field* identifier will contain a qualifier name for each level of group field within the record-type mapping to which the required field belongs. The last qualifier is the name of the field itself.

For example, within a record-type mapping "COPY" the same field name "DISP" exists within two separate group fields "SOURCE" and "TARGET". To uniquely identify these "DISP" fields, fully qualified *input-field* identifiers must be used:

```
SOURCE.DISP
TARGET.DISP
```

An fully qualified *input-field* should **only** be used for the following:

- The *fieldname* reference in a COLUMNS or REQUIRED section where specification of an unqualified *input-field* is not possible because the field name is **not unique** within the assigned record-type.

  The COLUMNS and REQUIRED sections define the input record fields for the REPORT utility execution. e.g.

  ```
  COLUMNS:      /* Structure containing multiple record-types including record-type "COPY". */
     COPY.SOURCE.BLKSIZE    /* Unique field name "SOURCE.BLKSIZE" in record-type "COPY". */
     COPY.TARGET.BLKSIZE    /* Unique field name "TARGET.BLKSIZE" in record-type "COPY". */
  ```

- The *fieldname* reference in a BLANKWHENZERO, BREAK, MAP, SORT or STATISTICS section if the field is defined as a fully qualified *input-field* in the COLUMNS or REQUIRED section.

- A variable in a REXX expression within the COMPUTE section if the *fieldname* is specified in the COLUMNS or REQUIRED section as *record-type.input-field* or *input-field* and *input-field* is also fully qualified. e.g.

  ```
  COMPUTE:              /* Update compute-field "NBLKRECS" */
     NBLKRECS = SOURCE.BLKSIZE % SOURCE.LRECL      /* Number of fixed length source records/block. */
  ```

- A text substitution variable in a print expression if the *fieldname* is specified in the COLUMNS or REQUIRED section as *record-type.input-field* or *input-field* and *input-field* is also fully qualified. e.g.

  ```
  BREAK:                /* Trigger control break following change in "SOURCE.DSN" value. */
     SOURCE.DSN  HEADING("Copy Input Dataset:" SOURCE.DSN)   /* SOURCE.DSN value in header. */
  ```

Computed Fields

A computed field represents a value that is computed from other field values (input record fields, built-in fields and/or other computed fields). A computed field is referenced as *compute-field* throughout this manual.

A *compute-field* value is established by a REXX routine specified by the Report Definition COMPUTE section. Therefore, *compute-field* is actually a REXX variable name whose value is the result returned by a valid REXX expression. In addition to the standard functions available in TSO/E REXX, the REPORT utility includes a number of built-in function. (See "*Appendix B. Built-in functions*" for details.)

The COMPUTE REXX routine is executed for each input record or record segment. Therefore, any change in input record field values or built-in field values are established before re-computing the *compute-field* values.

If a *compute-field* value needs to be initialised prior to processing the first input record (and so before the first execution of the COMPUTE REXX routine), then this can be done in the BROWSE-EXIT or INIT-EXIT REXX routines.

The default length of a *compute-field* value is the maximum of the widths specified on all references to the *compute-field* name in the REPORT definition. If no widths are specified, the default length is 9 (equal to the REXX NUMERIC DIGITS default).

A *compute-field* name may be used in any of the following sections:

COLUMNS	The *fieldname* identifier of a column definition for which values are displayed in the report.
BREAK	The *fieldname* for which a change in value will trigger a control break in the printed report.
BREAK FOOT HEAD	As a print expression element. A *compute-field* may be specified to represent a variable fragment of text in report page headings, page footings and control break lines.
SORT	The *fieldname* on which output report detail lines will be sorted.
STATISTICS	The *fieldname* identifying a report column for which statistical values may be generated.

Computed Field Specification

A *compute-field* name matches the name of a variable used in the COMPUTE section REXX routine.

To distinguish it from an record input field (*input-field*), whenever *compute-field* is referenced in report definition sections other than COMPUTE and BROWSE-EXIT, it must be specified with a colon (":") symbol prefix. This prefix character is used only to identify the field name as a *compute-field* and is **not** part of the *compute-field* name itself. e.g. A *compute-field* name "MyValue" is identified throughout the report definition as ":MyValue".

The following example demonstrates reference to a *compute-field* "TimeDiff" within a report definition.

```
COMPUTE:
  TimeDiff = Secs2Time( Time2Secs(ENDTIME) - Time2Secs(STARTTIME) )

COLUMNS:
  MEMBERNAME    "Member"
  STARTTIME     "Arrival"
  ENDTIME       "Departure"
  :TimeDiff     "Elapsed Time|(hhh:mm:ss.tt)"

SORT:
  :TimeDiff
```

Built-in Fields

The REPORT utility supports a number of built-in fields whose values may update over the course of the program execution. A built-in field is referenced as *built-in-field* throughout this manual.

Built-in fields include values for sequence number of the current output detail line, number of items in a control break group, page number, current date, day and time, input dataset name and input record number. See *"Appendix A. Built-in fields"* for supported *built-in-field* names and descriptions.

A *built-in-field* name may be used in any of the following sections:

BREAK FOOT HEAD	As a print expression element. A *built-in-field* may be specified to represent a variable fragment of text in report page headings, page footings and control break lines.
COLUMNS	The *fieldname* identifier of a column definition for which values are displayed in the report.
COMPUTE	As a variable in a REXX expression. A *built-in-field* may be used to resolve the value of a *compute-field*.

Built-in Field Specification

All *built-in-field* names begin with a hash ("#") character and must match the name of one of the standard built-in fields supported by the REPORT utility.

In the following, the page heading will contain the current date, time and day name at which the report was generated as well as the page number.

```
HEAD:
  #TODAY #DAYNAME / "Page" #PAGE (5,RIGHT)
```

Print Expressions

A print expression (referenced as *print-expression*) is used in BREAK, HEAD and FOOT sections to build a line of text to be displayed in a printed report.

A print expression contains a number of elements, each representing a fragment of text. Each print expression element may be a character literal string constant, a field name variable or a number of blanks. The order in which the elements occur in the expression defines the order in which their values occur in the output text.

For BREAK lines only, the print expression may also contain <NEWLINE> tags to allow a break line to span multiple report lines. For FOOT and HEAD sections, lines occupy multiple report lines if multiple footer/header line definitions are specified.

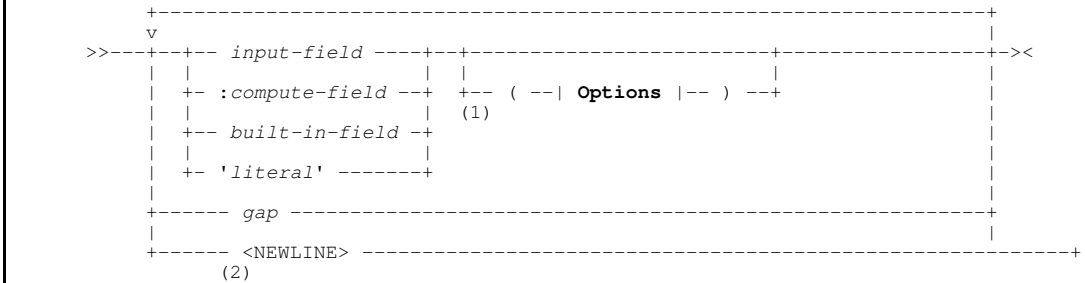
Examples:

Sample report definition with HEAD, FOOT and BREAK sections containing print expressions:

```
HEAD:
  #TIMESTMP  /  'Album Tracks for Artist:' ALBUM-ARTIST (25,RIGHT)  /

FOOT:
   /  'Page' #PAGE (5)                  /* Page number details right aligned.  */

REQUIRED:
  TRACK.ALBUM_ARTIST
  TRACK.TOTAL_TIME

COLUMNS:
  ALBUM.NAME         ('Album Name')
  TRACK.TRACK_NUM    ('Track Number' RIGHT)
  TRACK.NAME         ('Track Name')
  :DURATION          ('Running Time' RIGHT)   12  RIGHT

SORT:
  ALBUM.NAME
  TRACK.TRACK_NUM

COMPUTE:
  DURATION = Secs2Time(TOTAL_TIME)      /* e.g. 228.093 seconds = 00:03:48.093 */

BREAK:
  ALBUM.NAME    NOTOTAL  \
                FOOTING( \
                        'Number of tracks on album'    NAME (30)   0  ':'  #ITEMS \
                 <NEWLINE>   38    'Last track length:'    :DURATION (9,RIGHT) \
                       )
```

Syntax:

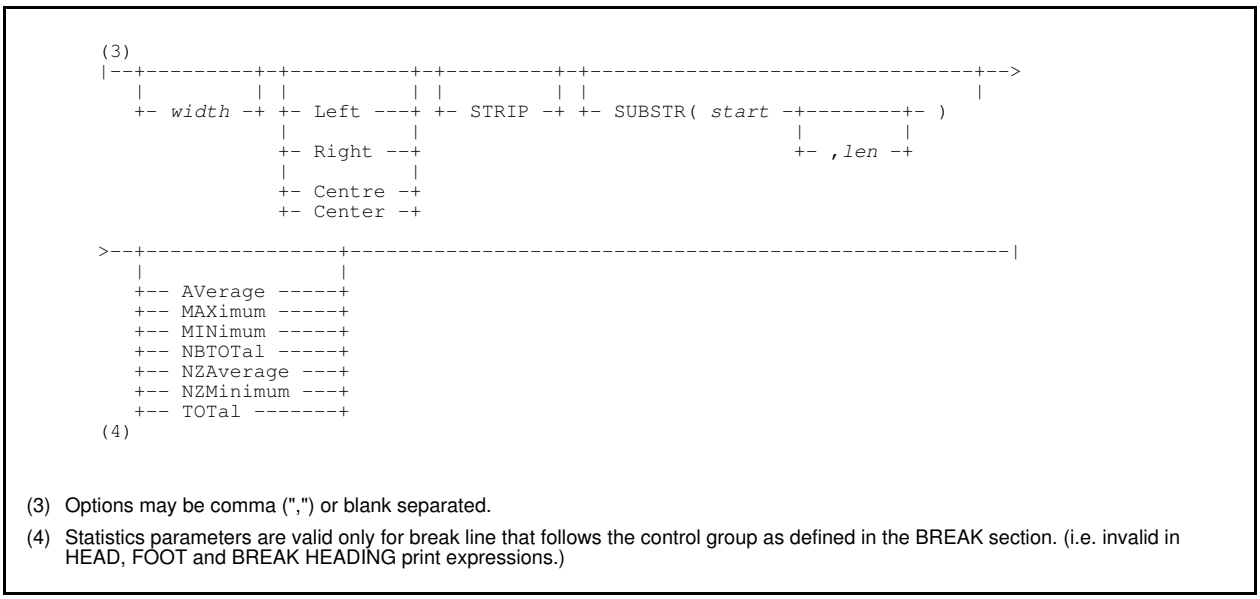
print-expression



(1) At least 1 blank space must exist before the open parenthesis in order to avoid being interpreted as a parenthesised input field subscript representing an array element.

(2) <NEWLINE> is valid only for break line text defined in the BREAK section.

Options:



(3) Options may be comma (",") or blank separated.

(4) Statistics parameters are valid only for break line that follows the control group as defined in the BREAK section. (i.e. invalid in HEAD, FOOT and BREAK HEADING print expressions.)

Parameters:

input-field
A field identifier that uniquely identifies an input record field.

The *input-field* specification must match an *input-field* specified in the COLUMNS or REQUIRED report definition sections, except that the record mapping structure name must be excluded. i.e. If the required field is defined in the COLUMNS section using a fully qualified *input-field*, then the same fully qualified *input-field* specification must be used in the *print-expression* but with its first (record-type) qualifier removed.

If *print-expression* is for a header line (HEAD or BREAK HEADING), then the value of an *input-field* element is obtained from the first report record that follows the heading. For footing lines (FOOT) and all other BREAK lines that appear after the control group (BREAK AVERAGE, FOOTING, MAXIMUM, MINIMUM, NZAVERAGE, NZMINIMUM and TOTAL), the value of an *input-field* element is obtained from the last report record written.

In the above example, **ALBUM-ARTIST** is a field defined in the input records mapped by record-type TRACK. It is not included as a column in the report output (COLUMNS section) so it must be referenced as a REQUIRED field. The page header line will replace *input-field* ALBUM-ARTIST with the artist name from the first detail line on the page.

Similarly, **NAME** is a field defined in the input records mapped by record-type ALBUM. The control break footing line will replace *input-field* NAME with the album name from the last detail line of the control group.

:*compute-field*
The name of a computational field defined in the COMPUTE section of the report definition. Note that *compute-field* must be prefixed by a colon (":") symbol to identify it as a computational field name.

Note that, a *compute-field* has default alignment "LEFT" and a minimum width of 9 characters. The default width is either 9, or a value greater than 9 and equal to the largest width value specified for the *compute-field* anywhere within the report definition.

If *print-expression* is for a header line (HEAD or BREAK HEADING) and a *compute-field* is based on *input-field* values, then the value of the *compute-field* element will be based on the *input-field* values obtained from the first report record that follows the heading. Otherwise, for footing lines and all other BREAK lines that appear after the control group, the value of the *compute-field* element will be based on *input-field* values obtained from the last report record written.

In the above example, **DURATION** is a computational field based on input record field **TOTAL_TIME**. The control break footing line will replace *compute-field* :DURATION with the duration value computed from input fields in the last detail line of the control group.

built-in-field
The name of a REPORT utility built-in field.
See Appendix A. built-in fields for the complete reference of built-in fields.

Note that all REPORT built-in-fields begin with a hash ("#") symbol.

'*literal*'
A literal text string enclosed in quotation marks (") or apostrophes ('). The text string will appear without the enclosing characters in the printed output.

In the above example, 'Album Tracks for Artist:', 'Page', 'Number of tracks on album', ':' and 'Last track length:' are all literal text strings.

width

A *width* value may be specified in parentheses "()" following the *input-field*, :*compute-field*, *built-in-field* or '*literal*' element to which it applies. The width value specifies the number of characters in the printed break line that will be reserved for the element's values (the element's print area).

The default width of a field name element is the maximum width of the field. For input fields, this is the maximum field length as defined in the structure; for built-in fields, it is the width assigned internally to the particular field; for compute fields, a defualt width of 9 is assumed. Alternatively, for literal elements, the default width is set to be the length of the literal value.

Unless STRIP is specified, the element value will be truncated or padded with blanks to fit the specified *width* number of characters.

At least 1 blank space must exist between the opening parenthesis ("(") and the preceding element specification. If other options (alignment, STRIP, SUBSTR) are also specified, then they must be enclosed within the same set of parentheses with blank or comma (",") separators.

In the above example, *input-field* ALBUM-ARTIST in the header is padded or truncated at 25 characters, *built-in-field* #PAGE in the footer at 5 characters and the *input-field* NAME in the break footer at 30 characters.

gap

Represents a number of blank characters to be printed between the previous and next element print areas. The default *gap* value is 1 (i.e. 1 blank will be inserted between element print areas).

If specified at the start of a print line, this number of blanks will occur before the start of the text. The default is not to insert blanks at the start of the print line so that text begins at the very beginning of the print line.

A zero (0) value may be used to suppress blanks and so join together 2 element print areas.

In the above example, a zero gap value is specified between the NAME and ':' elements in the break footing line definition so that no blanks are inserted between the area reserved for album name and the following colon (":") symbol. Also, a gap value of 38 exists at the start of the next print line to insert 38 blanks before the start of text that follows. This serves to align ":" following 'Last track length' with the ":" on the line before.

AVERAGE | MAXIMUM | MINIMUM | NBTOTAL | NZAVERAGE | NZMINIMUM | TOTAL

Applicable only to field elements (*input-field*, *compute-field* or *built-in-field*) which belong to a print expression that defines a break footing line of text. i.e. any BREAK line definition (other than BREAK HEADING) that appears after the control group.

One of these parameters may be specifies to indicate that the relevant statistical field break value will be output instead of the field value obtained from the last control group detail line. Statistics values are displayed below the column of values to which they apply. However, using this method to display statistical data allows positioning of values anywhere within a line that follows a control group of lines. What is more, it allows use of statistical values generated from fields not displayed as a column in the report detail lines (i.e. a field defined in the REQUIRED section as opposed to the COLUMNS section).

In the example above, the FOOTING line print expression for the ALBUM.NAME BREAK definition may be updated as follows so that, for detail lines in the last control group, the average value for compute-field "DURATION" is displayed instead of the value obtained from the last detail line of the control group.

```
BREAK:
  ALBUM.NAME    NOTOTAL  \
                FOOTING( \
                           'Number of tracks on album'    NAME (30)   0  ':'  #ITEMS \
                 <NEWLINE>   35    'Average track length:'    :DURATION (AVERAGE,9,RIGHT) \
                       )
```

LEFT | RIGHT | CENTRE | CENTER

An alignment (**LEFT**, **RIGHT**, **CENTRE** or **CENTER**) may be specified in parantheses "()" following the *input-field*, :*compute-field*, *built-in-field* or '*literal*' element to which it applies. The alignment specifies how a value represented by the element will be aligned within the print area reserved for that element's values.

For input fields, the default alignment is determined by the data type of the field element; for built-in fields, it is the alignment assigned internally to the particular field; for compute fields and literal elements the default alignment is "LEFT".

LEFT will align the value on the left of the element's print area and, if necessary, pad or truncate on the right of the value. **RIGHT** will align the value on the right of the element's print area and, if necessary, pad or truncate on the left of the value. **CENTRE** or **CENTER** will centralise the value within the element's print area and, if necessary, pad or truncate evenly on both the left and right of the value.

At least 1 blank space must exist between the opening parenthesis ("(") and the preceding element specification. If other options (width, STRIP, SUBSTR) are also specified, then they must be enclosed within the same set of parentheses with blank or comma (",") separators.

In the above example, (25,RIGHT) follows the ALBUM-ARTIST *input-field* element in the page header line definition ensuring that the artist name is right adjusted within a print area width of 25 characters. Similarly, (9,RIGHT) follows the :DURATION :*compute-field* element in the break footing line definition. This defines a print

area width of 9 characters in which the elapsed time value (format HH:MM:SS.mmm) are right aligned. This means that the duration values will appear in the format MM:SS.mmm with the number of hours value (HH:) on the left of the value being truncated.

`<NEWLINE>`
Valid only in BREAK line print expressions, <NEWLINE> starts a new print line in the printed text.

If no print expression elements exist before <NEWLINE>, then the contents of that line will be empty. Similarly, if <NEWLINE> is specified last in the print expression, the print line that follows will be blank.

`STRIP`
STRIP may be specified in parantheses "()" following the *input-field*, :*compute-field*, *built-in-field* or '*literal*' element to which it applies. STRIP specifies that all leading and trailing blanks will be stripped from the element value when it appears in the printed line.

If STRIP is specified, then the default or specified *width* value will define the **maximum** width of the element text instead of the size of the element print area. (The element print area will be equal to the length of the stripped text.)

At least 1 blank space must exist between the opening parenthesis ("(") and the preceding element specification. If other options (alignment, width, SUBSTR) are also specified, then they must be enclosed within the same set of parentheses with blank or comma (",") separators.

`SUBSTR(`*`start`*`[,`*`len`*`])`
SUBSTR may be specified in parantheses "()" following the *input-field*, :*compute-field*, *built-in-field* or '*literal*' element to which it applies. SUBSTR may be used when the required output element text is a sub-string of the element value.

SUBSTR specifies *start*, the position within the element value of the first output element text character, and optionally *len*, the number of element value characters. These parameters appear in parentheses immediately following the SUBSTR keyword and are separated by blanks or a comma (",").

If *len* is not specified, then the substring value will begin at the *start* character position and end at the last character in the element value. If *len* extends beyond the last character of the element value, then the element text will be padded with blanks.

Stripping of leading and trailing blanks, or alignment within the element's print area (*width*) will occur on the sub-string of the element's value.

Report Definition Sections

A REPORT definition is split into a number of sections. Each section begins with a section header and ends when a new section header or the end of the REPORT definition input is encountered.

The following describes the use and syntax of control statements for each section header within the REPORT definition.

A section header is recognised as being the first (or only), blank delimitted word on a report definition control statement that ends with a colon (":") character. If the word is not one of the supported section keywords, then all control statements that follow it will be ignored up to the next section header.

In the following example, the live report page header definition will be ignored since "**XHEAD**" is not a recognised section header. Syntax processing begins again at the "**HEAD**" section and so a temporary page header including the text "##UNDER DEVELOPMENT##" will be used.

```
XEAD:
  #TIMESTMP / "Access Report" / "PAGE" #PAGE (RIGHT,4)

HEAD:
  #TIMESTMP / "Access Report ##UNDER DEVELOPMENT##" / "PAGE" #PAGE (RIGHT,4)
```

Section headers may be specified more than once within the report definition in which case control statements that follow will be treated as a continuation of the original section header specification.

In the following example, the COLUMNS section is interrupted by the COMPUTE section. The column definition control statements in the second COLUMNS section are treated as if they had been inserted immediately following the "**zConnected**" column definition control statement. Note that it is not necessary for a computed field definition (in a COMPUTE section) to occur before its use in any other section. Therefore, the COMPUTE section containing the definition of **DURATION** may occur following the column definition **:DURATION** in the report definition.

```
COLUMNS:
  SMF119#02_TCP_Connection_Termination.zRName          ('RESOURCE')
  SMF119#02_TCP_Connection_Termination.zConnectStart   ('CONNECTION|START')
  SMF119#02_TCP_Connection_Termination.zConnectEnd     ('CONNECTION|END')

COMPUTE:
  DURATION = Secs2Time( Time2Secs(zConnectEnd) - Time2Secs(zConnectStart) )

COLUMNS:
  :DURATION                                            ('DURATION|HHH:MM:SS.SS')
  SMF119#02_TCP_Connection_Termination.zInBytes        ('INBOUND|BYTES')
  SMF119#02_TCP_Connection_Termination.zOutBytes       ('OUTBOUND|BYTES')
  SMF119#02_TCP_Connection_Termination.zTermCode       ('TERM CODE DESC')
```

Section Headers:

BLANKWHENZERO	BIZ BLANKIFZERO BWZ	Display zero column values as blank.
BREAK		Define report break columns and break line text.
BROWSE-EXIT	BROWSEEXIT	Exit routine executed at start of input record BROWSE processing.
COLUMNS		Define columns to appear in the report.
COMPUTE	COMP	Define computed field expressions.
DISPLAY-EXIT	DISPLAYEXIT	Exit routine executed on display of report output.
FILTER		Specify filter to be applied to input records.
FOOT		Define printed report page footing lines.
INIT-EXIT	INITEXIT	Exit routine executed at start of input record FILEIO processing.
INPUT		Define the input data object.
HEAD		Define printed report page header lines.
MAP		Define field mapping for input records.
OPTIONS	OPTION	Specify REPORT execution options.
OUTPUT		Define an output data object.
REPEAT		Specify input record types for which a new detail line will be generated.
REQUIRED	REQUIRE	Identify input record fields required for computed fields and non-column detail lines.
RESET		Specify input record types whose values are to be reset following output of a detail line.
SORT		Identify fields on which input records are to be sorted.
STATISTICS	STATS TOTAL TOTALS	Specify report columns for which statistical information will be gathered.
TRANSLATE		Specify input field value character translation table.
WHERE		Specify where-clauses to filter input record segments.

BLANKWHENZERO

Overview:

Specifies one or more columns for which any zero (0) value will be output as blank. This is of particular use in a printed report where a column contains mostly zero values. These values may be displayed as blanks thus making the non-zero values more prominent.

Each BLANKWHENZERO column fieldname occupies a separate report definition statement. A BLANKWHENZERO fieldname must represent an *input-field* and must match exactly one of the fieldnames specified in the COLUMNS section.

BLANKWHENZERO is applicable only to columns defined by input fields assigned a numeric data type. Specification of a fieldname which is of a non-numeric data type will have no effect on the output report.

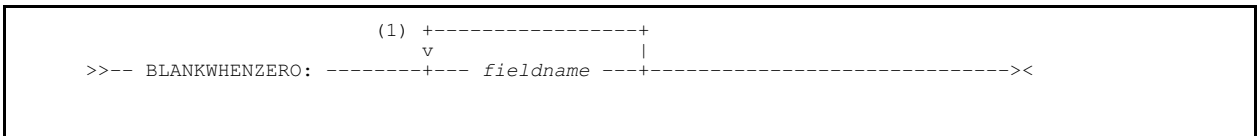
BLANKWHENZERO applies to all types of output (BROWSE, CSV, JSON, PRINT and XML).

Examples:

```
BLANKWHENZERO:
  SMF119#02_TCP_Connection_Termination.zInBytes
  SMF119#02_TCP_Connection_Termination.zOutBytes
```

Output report columns, defined by numeric input fields **zInBytes** and **zOutBytes** of the SMF type 119 sub-type 2 record for TCP Connection Termination, will output a blank in place of a zero value.

Syntax:



(1) Each *fieldname* must be specified on a separate control statement.

Synonyms:

BLANKWHENZERO	BWZ BLANKIFZERO BIZ

Parameters:

fieldname
An input record field name of numeric data type for which zero (0) values will be output as blanks. Each *fieldname* specification must also exist as an output column definition in the COLUMNS section.

BREAK

Overview:

The BREAK section specifies one or more control break definitions.

Although control break lines apply only to PRINT type output, break definitions may be used in conjunction with the DETAIL option to control the number of CSV, JSON or XML output lines written for each break key field value.

A report control break is a break in the printed detail line output.

Each break definition defines a break key field and the (input or compute) source field from which values are obtained. By default, a break key field has a format that matches that of its source field.

A control break occurs when there is a change in the value of the break key field. Consequently, the detail lines printed between each break are grouped by a common (break key) field value. These groups of detail lines constitute a **control group**.

Although not mandatory, a break key field is usually a field on which the report lines have been sorted, either prior to processing by the REPORT utility or via a report definition SORT control statement.

BREAK may also define an additional control break definition to customise the report output following the last detail line of the report. This control break is referenced using the pseudo key column field name, #GRAND, which is triggered after the last detail line is printed.

A control break definition may be used to specify a customised header to be printed before the detail lines belonging to a control group, as well as a customised number of blank lines and/or footer lines to be printed following a control group. Footer lines include lines containing statistical values (totals, averages, etc.) for columns nominated as statistics columns in the STATISTICS section. If a STATISTICS section does not exist, statistics will be generated for all columns of numeric data type by default.

Each control break definition occupies a single statement of the BREAK section and begins with the field name from which the break key field is constructed. Each control break is also assigned a level number within the break hierarchy. This is equal to 1 plus the sequence number (1,2,..,*n*) of the matching key column field name in the SORT section. Control break level 1 is reserved for the #GRAND control break.

When a control break is triggered, a control break is also triggered for all control break definitions with a higher break level number. Break output lines are printed first for the break with the highest level number and then for each descending level to the level number at which the control break was initially triggered.

Notes:

- A control group identifies all detail lines printed between two instances of a break triggered by a particular control break level.
- By default, each of the defined (or implied) statistics columns is underlined before control break lines are printed. For the #GRAND break, the equals ("=") symbol is used as the underline character and, furthermore, the printed underline is repeated following the break lines to mark the end of report text. For all other control breaks, the minus ("-") symbol is used as the underline character.

  Option BRKULINE(NO) will suppress all printed lines containing statistics column underlining.
- If the number of control break lines following the control group is greater than the number of lines remaining on the page, then the break lines are printed at the start of the new page following the page title, column headers and any repeated control break headings (see control break parameter REPEAT).

  Option SPLITBREAK(YES) will allow the block of break lines to wrap onto a new page.

Examples:

Example 1 - Single Control Break:

```
SORT:
  SMF030_Identification.zJOBNAME
  SMF030_Identification.zRST

BREAK:
  SMF030_Identification.zJOBNAME

STATISTICS:
  SMF030_IO_Activity.zTEP
  SMF030_IO_Activity.zTPT
  SMF030_IO_Activity.zTGT
```

The above example will trigger a control break following a change to the job name value in field **zJOBNAME**. By default, sub-total values will be printed following the nominated statistics columns **zTEP** (EXCP Count), **zTPT** (TSO/E terminal write count) and **zTGT** (TSO/E terminal read count).

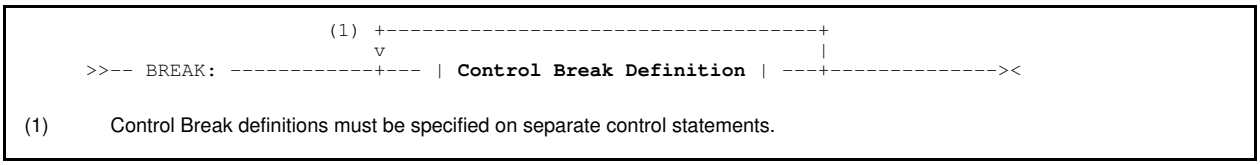
Example 1 - Multiple Control Breaks:

```
SORT:
 SMF119#02_TCP_Connection_Termination.zRName
 SMF119#02_TCP_Connection_Termination.zConnectStart

BREAK:
 SMF119#02_TCP_Connection_Termination.zRName   \
           TOTAL()  \
           HEADING( \
         <NEWLINE> 'This is a header for group:' zRNAME (RIGHT,10) \
         <NEWLINE> '--------------------------------------') \
           REPEAT \
           SPACEAFTER( page )

 SMF119#02_TCP_Connection_Termination.zConnectStart  10 \
           NZAVERAGE  \
           MAXIMUM    \
           NZMINIMUM  \
           SPACEBEFORE(1) \
           SPACEAFTER(3)

 #GRAND    FOOTING('End of TCP Connection Termination Report') \
           SPACEBEFORE(10)
```

3 control breaks are defined, 1 for each of the sorted key columns and 1 for the #GRAND grand totals output following the last detail output line. #GRAND always represents the 1st level control break. 2nd level control breaks occur when the value of field **zRName** changes, 3rd level control breaks when the value of field **zConnectStart** changes. A change in the zRName value will first trigger a 3rd level control break for zConnectStart before triggering the 2nd level control break for zRName.

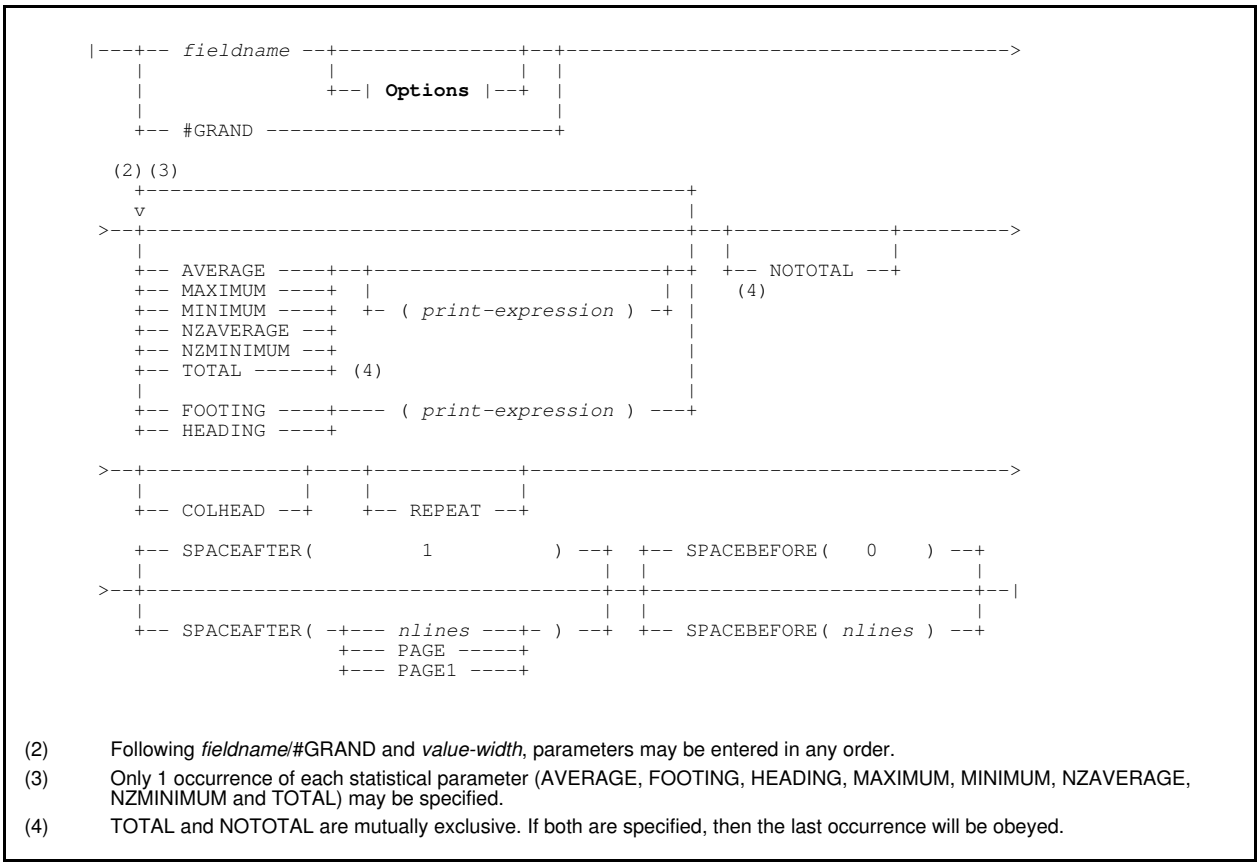
Each new resource name (zRName) control group will start on a new page of the report and will be preceeded by 3 header lines. A change in the first 10 characters of the zConnectStart timestamp value (i.e. the date portion) will trigger intermediate control breaks within each zRName control group.

Syntax:



(1) Control Break definitions must be specified on separate control statements.

Control Break Definition:

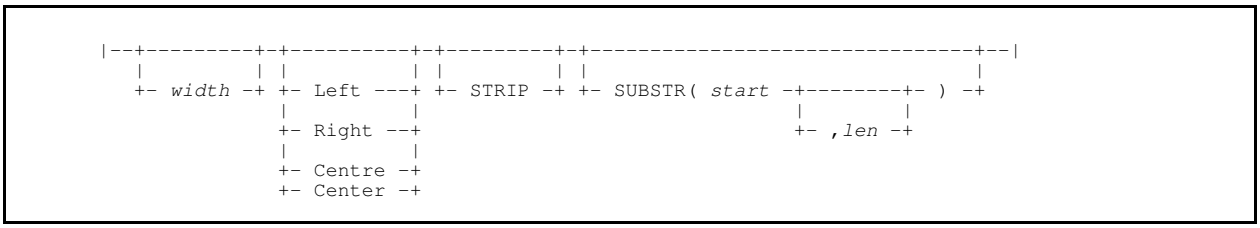


(2) Following *fieldname*/#GRAND and *value-width*, parameters may be entered in any order.

(3) Only 1 occurrence of each statistical parameter (AVERAGE, FOOTING, HEADING, MAXIMUM, MINIMUM, NZAVERAGE, NZMINIMUM and TOTAL) may be specified.

(4) TOTAL and NOTOTAL are mutually exclusive. If both are specified, then the last occurrence will be obeyed.

Options:



Parameters:

fieldname

The *fieldname* is an *input-field* or *compute-field* field name specification from which break key field values are obtained. A control break will be triggered whenever there is a change in the break key field value.

By default, the format of the break key field is the same as the default format of *fieldname* (i.e. it uses the field's default (maximum) field width and a value alignment determined by the data type of the field). Note that fields of numeric data types are right aligned, otherwise field values are left aligned by default.

The break key field format may be controlled using *width*, *alignment*, STRIP and/or SUBSTR options.

width

Specifies the width of the break key field. If *width* is greater than the default field width of *fieldname*, then blank padding will occur on the obtained field value. If it is less than this field width, then truncation of the *fieldname* value will occur.

For example, suppose an 8 character (zJOBNAME) input field contains left aligned job name values in a format which comprises a 4 character prefix followed by a 4 character sequence number. The following defines a left aligned break key field of width 4 so that the break key is the first 4 (truncated) characters of the job name value. This means that a control break will only be triggered when there is a change in the job name prefix.

```
zJOBNAME  4
```

#GRAND

#GRAND refers to the break that occurs at the end of the report only when all detail lines have been printed. There is only 1 #GRAND control group which includes all detail lines.

Unless GRANDTOTAL(NO) has been specified in the OPTIONS section, grand totals will be printed for statistics columns at the end of the report. The #GRAND control break provides the method by which the output header, statistics and footer lines may be configured for the #GRAND control break.

For example, the following will output the maximum value found in each statsitics column in addition to the default grand totals:

```
#GRAND  MAXIMUM
```

AVERAGE[(*print-expression*)]

Indicates that the control break print lines are to include a line containing the average value for each statistics column. Each statistics column's average value is calculated from **all** values belonging to that column within the control group. The average value is aligned below its statistics column in the output print line. The default is **not** to print average values at each break.

The optional, parenthesised *print-expression* may be used to specify the comment text to be printed before the average values. The print expression contains any number of text fragments specified as either literal text or field variables representing computed, built-in or input data values. See *print-expression* in this section for a detailed description.

For example, the following control break average specification:

```
zRName   AVERAGE('** Average values for' zRName '=')
```

will print the following text before the average values:

```
** Average values for XXXXX =
```

"XXXXX" is the resource name value in column field zRName as found in the **last** detail line of the control group.

If AVERAGE is specified without *print-expression*, then the following default comment text is used prefixed by a variable number of blanks equal to twice the break level value plus 1.

```
Average Value
```

A null literal may be specified for *print-expression* if comment text is to be suppressed.

After the comment text, the average values are aligned under each statistics column. If the comment text occupies the area below the first statistics column, then all the average values are printed on the next line.

CENTRE|CENTER|LEFT|RIGHT

Specifies the data alignment of the break key value within the break key field. The break value may be left adjusted, right adjusted or centralised within the defined break key field width (*width*).

If *fieldname* is an input field of numeric data type, then the default break key value alignment is RIGHT. Otherwise, the default alignment is LEFT.

Aligned values, padded or truncated to the data width (*width*), are used as the break key.

COLHEAD

Applicable only if printing of column headers has not been suppressed by the OPTIONS section entry COLHEAD(NO).

COLHEAD may be specified on a control break definition to indicate that, when the control break is triggered, column header lines are to be re-printed immediately prior to printing the next break heading line for any control break, or otherwise the first line of the next control group.

By default, column header lines are only printed at the start of a new page following any page header lines. The COLHEAD option allows the column header lines to be re-printed at lines within the page that follow report detail lines.

For example, the following defines 2 control breaks. The 1st (highest) level in the control break hierarchy is for input field zRName, the 2nd (lowest) level is for input field zRDate.

```
zRName   HEADING('Resource') NOTOTAL  FOOTING('End of Resource:' zRName)
zRDate   HEADING('Date')     TOTAL   COLHEAD
```

A change in the zRDate value will trigger the 2nd level control break and so will first print the TOTAL values for statistics columns, and then re-print the column header lines before printing the 2nd level control break heading "Date".

If a change in the zRName value occurs, then this will trigger first the 2nd level (zRDate) control break and then the 1st level (zRName) control break. The TOTAL values will be printed for the 2nd level (zRDate) control break, followed by the FOOTING text for the 1st level (zRName) control break. Because COLHEAD is flagged by the 2nd level break, column header lines are re-printed before printing the header text for the 1st (zRName) and then 2nd (zRDate) level control breaks.

FOOTING(*print-expression*)
Specifies text to be printed after the control group's detail lines and following the control break statistics lines. The default is **not** to print break footing lines.

A parenthesised *print-expression* is mandatory and specifies the footing text. The print expression contains any number of text fragments specified as either literal text or field variables representing computed, built-in or input data values. See *print-expression* in this section for a detailed description.

For example, the following control break footing specification:

```
DEPT   FOOTING('-#- End of' DEPT 'statistics for' DATE (10) '-#-')
```

will print the following text before the footing values:

```
-#- End of XXXX statistical for XXXX/XX/XX -#-
```

"XXXX" is the department name in column field DEPT and "XXXX/XX/XX" is the first 10 characters of the date value in column field DATE as found in the **last** detail line of the control group.

The "<NEWLINE>" item in *print-expression* may be used to output the break footing text over multiple report lines and also to print null (blank) lines.

HEADING(*print-expression*)
Specifies text to be printed before the control group's detail lines but following column header lines. The default is **not** to print break heading lines.

A parenthesised *print-expression* is mandatory and specifies the heading text. The print expression contains any number of text fragments specified as either literal text or field variables representing computed, built-in or input data values. See *print-expression* in this section for a detailed description.

For example, the following break heading parameters:

```
zRName   HEADING('Report for resource' zRName <NEWLINE> '---------------------------' )
```

will print the following break heading text at the start of the control group:.

```
Report for resource XXXXXXXX
---------------------------
```

"XXXXXXXX" is the resource name value in the column field zRName as found in the **first** detail line of the control group to be printed following the break heading.

If REPEAT is also specified on the control break definition, the heading text will be repeated at the start of each new page.

MAXIMUM[(*print-expression*)]
Indicates that the control break print lines are to include a line containing the maximum value for each statistics column. Each statistics column's maximum value is determined from **all** values belonging to that column within the control group. The maximum value is aligned below its statistics column in the output print line. The default is **not** to print maximum values at each break.

The optional, parenthesised *print-expression* may be used to specify the comment text to be printed before the maximum values. The print expression contains any number of text fragments specified as either literal text or field variables representing computed, built-in or input data values. See *print-expression* in this section for a detailed description.

For example, the following control break maximum specification:

```
zJOBNAME  MAXIMUM('** Maximum values for' zJOBNAME '('  0  #ITEMS 'Items)' )
```

will print the following text before the maximum values:

```
** Maximum values for XXXXX (n Items)
```

"XXXXX" is the job name value in column field zJOBNAME as found in the **last** detail line of the control group and "n" is the number of items in the control group.

If MAXIMUM is specified without *print-expression*, then the following default comment text is used prefixed by a variable number of blanks equal to twice the break level value plus 1.

```
Maximum Value
```

A null literal may be specified for *print-expression* if comment text is to be suppressed.

After the comment text, the maximum values are aligned under each statistics column. If the comment text occupies the area below the first statistics column, then the maximum values are printed on the next line.

MINIMUM[(*print-expression*)]
: Indicates that the control break print lines are to include a line containing the minimum value for each statistics column. Each statistics column's minimum value is determined from **all** values belonging to that column within the control group. The minimum value is aligned below its statistics column in the output print line. The default is **not** to print minimum values at each break.

  The optional, parenthesised *print-expression* may be used to specify the comment text to be printed before the minimum values. The print expression contains any number of text fragments specified as either literal text or field variables representing computed, built-in or input data values. See *print-expression* in this section for a detailed description.

  For example, the following control break minimum specification:

```
zTME 10  MINIMUM('Shortest elapsed times on' zTME (LEFT,10) )
```

  will print the following text before the minimum values:

```
Shortest elapsed times on XXXX/XX/XX
```

  "XXXX/XX/XX" is the ISO date value in column field zTME as found in the **last** detail line of the control group.

  If MINIMUM is specified without *print-expression*, then the following default comment text is used prefixed by a variable number of blanks equal to twice the break level value plus 1.

```
Minimum Value
```

  A null literal may be specified for *print-expression* if comment text is to be suppressed.

  After the comment text, the minimum values are aligned under each statistics column. If the comment text occupies the area below the first statistics column, then the minimum values are printed on the next line.

NOTOTAL
: Suppress control break totals.

  Unless option BRKTOTALS(NO), GRANDTOTAL(NO) or TOTALS(NO) has already been specified, then NOTOTAL may be used to suppress the default action of printing totals statistics for the control break.

  NOTOTAL and TOTAL are mutually exclusive parameters. If both are specified, then it is the parameter that is specified last in the control break definition which is obeyed.

NZAVERAGE[(*print-expression*)]
: NZAVERAGE is the similar to AVERAGE except that **non-zero** values are ignored when determining average values. i.e. The accumulated total for a statistics column in the control group is split evenly between the total number of non-zero items rather than the total number of items.

  NZAVERAGE indicates that the control break print lines are to include a line containing the non-zero average value for each statistics column. Each statistics column's non-average value is calculated from **all non-zero** values belonging to that column within the control group. The average value is aligned below its statistics column in the output print line. The default is **not** to print average values at each break.

  The optional, parenthesised *print-expression* may be used to specify the comment text to be printed before the average values. The print expression contains any number of text fragments specified as either literal text or field variables representing computed, built-in or input data values. See *print-expression* in this section for a detailed description.

  For example, the following control break non-zero average specification:

```
zSTART 10   NZAVERAGE('Non-zero average elapsed times on' zSTART (LEFT,10) 0 '...')
```

  will print the following text before the non-zero average values:

```
Non-zero average elapsed times on XXXX/XX/XX...
```

  "XXXX/XX/XX" is the ISO date value in column field zSTART as found in the **last** detail line of the control group.

  If NZAVERAGE is specified without *print-expression*, then the following default comment text is used prefixed by a variable number of blanks equal to twice the break level value plus 1.

```
Average of NON-ZERO Values
```

  A null literal may be specified for *print-expression* if comment text is to be suppressed.

  After the comment text, the average values are aligned under each statistics column. If the comment text occupies the area below the first statistics column, then the non-zero average values are printed on the next line.

NZMINIMUM[(*print-expression*)]
: NZMINIMUM is the similar to MINIMUM except that **non-zero** values are ignored when determining minimum values.

  NZMINIMUM indicates that the control break print lines are to include a line containing a non-zero minimum value for each statistics column. Each statistics column's non-zero minimum value is determined from **all non-zero** values belonging to that column within the control group. The non-zero minimum value is aligned below its statistics column in the output print line. The default is **not** to print non-zero minimum values at each break.

  The optional, parenthesised *print-expression* may be used to specify the comment text to be printed before the minimum values. The print expression contains any number of text fragments specified as either literal text or field variables representing computed, built-in or input data values. See *print-expression* in this section for a detailed description.

  For example, the following control break non-zero minimum specification:

```
zJOBNAME  NZMINIMUM('Lowest non-zero values:')
```

  will print the following text before the non-zero minimum values:

```
Lowest non-zero values:
```

  If NZMINIMUM is specified without *print-expression*, then the following default comment text is used prefixed by a variable number of blanks equal to twice the break level value plus 1.

```
Minimum of NON-ZERO Values
```

  A null literal may be specified for *print-expression* if comment text is to be suppressed.

  After the comment text, the minimum values are aligned under each statistics column. If the comment text occupies the area below the first statistics column, then the minimum values are printed on the next line.

REPEAT
: Applicable only if HEADING is also specified, REPEAT indicates that the control break heading text printed at the start of the control group is also to be printed at the start of each new page following the page title and column header lines.

  Default is to print break headings only at the start of each new control group.

SPACEAFTER(*nlines*|PAGE|PAGE1)
: Specifies *nlines*, the number of blank lines to be printed after the control break lines that follow the control group. The default is to print 1 blank line.

  If the number of lines remaining on the page is less than *nlines*, the surplus SPACEAFTER lines will be ignored (i.e. blank lines and will **not** be printed at the start of the new page).

  Alternatively, PAGE or PAGE1 may be specified to throw a new page following the last control break line. (PAGE1 will reset the page numbering on the new page back to 1.)

SPACEBEFORE(*nlines*)
: Specifies *nlines*, the number of blank lines to be printed following the control group but before the control break lines. The default is **not** to print any blank lines between the control group and the control break lines.

  Note that, unless option BRKULINE(NO) is specified, the control break lines begin with a line containing statistics column underline characters.

  The number of SPACEBEFORE blank lines are included in the number of control break lines that occur after the control group. Unless option SPLITBREAK(YES) has been specified, a new page occurs whenever this number of control break lines exceeds the number of lines remaining on the page.

STRIP
: Specifies that leading and trailing blanks that potentially exist in the value obtained from *fieldname* are to be stripped prior to being aligned in the break key field.

  Note that, if SUBSTR is also used, then the strip of leading and trailing blanks will occur on the substring value obtained from *fieldname*.

  By default, the value in the break key field is displayed in the totals statistics break line following the control group. The break key field definition parameters (*width*, *alignment*, STRIP and SUBSTR) are also used for the print element field definition within the generated totals line print expression. Therefore, if STRIP is used to define the break key field, the value displayed in the totals line will be stripped of leading and trailing blanks.

SUBSTR(*start*[,*len*])
: Specifies that the value obtained from *fieldname* is a substring of the field's value.

  A start position (*start*) and optional length value (*len*) is specified in parentheses "( )" immediately following the SUBSTR keyword. The *start* value is the position in the field of the first character obtained from *fieldname*, and *len* is the length of data to be obtained.

Note that, if *len* is not specified, then the substring value will begin at the *start* character position and end at the last character of the field value. If *len* extends beyond the last character of the input value, then the output value will be padded with blanks.

If both SUBSTR and STRIP are used, then the SUBSTR operation will occur first so that leading and trailing blank characters will be stripped fron the sub-string value and not from the original *fieldname* source field value.

The value obtained from the source field following a SUBSTR and/or STRIP is ultimately saved in the break key field using the key field's alignment and width.

`TOTAL[(`*`print-expression`*`) ]`
Indicates that the control break print lines are to include a line containing the total value for each statistics column. Each statistics column's total value is calculated from **all** values belonging to that column within the control group. The total value is aligned below its statistics column in the output print line.

The default is to print total values for each control break defined in the BREAK section and also for the #GRAND control break, whether or not it has been explicitly defined. Note that if both TOTAL and NOTOTAL are specified, then the occurrence that appears last within the control break definition will be obeyed.

Option BRKTOTALS(NO) will override this default and suppress totals for all, non-#GRAND control breaks specified in the BREAK section for which TOTAL is not explicitly defined.
Option GRANDTOTAL(NO) will suppress output of totals for the #GRAND break regardless of whether TOTAL has been explicitly defined.
Option TOTALS(NO) is equivalent to specifying both BRKTOTALS(NO) and GRANDTOTAL(NO).

The optional, parenthesised *print-expression* may be used to specify the comment text to be printed before the total values. The print expression contains any number of text fragments specified as either literal text or field variables representing computed, built-in or input data values. See *print-expression* in this section for a detailed description.

For example, the following control break totals specification:

```
REGION   TOTAL('** Totals **')
```

will print the following text before the totals values:

```
** Totals **
```

If totals are printed by default or TOTAL is specified without *print-expression*, then default comment text is used. For standard (non-#GRAND) control breaks:

```
Totals for XXXXX (n Items)
```

For the #GRAND control break:

```
Grand Totals (n Items)
```

Where "XXXXX" is the is the break key field value obtained from the last report detail line printed before the control break was triggered, and "n" is the number of items in the control group.

Both default text strings are indented so that the text starts in the same position for each ptint level. Furthermore, the text is prefixed by a variable number of equals ("=") symbols and a single blank. The number of these symbols is equal to 1 plus the total number of break levels minus the current break level, all muliplied by 2. For example, if the current break level is 2 and there are a total of 4 break levels (including the #GRAND break level) then the number of equal symbols prefixing the text is (1+4-2)*2 = 6.

A null literal may be specified for *print-expression* if comment text is to be suppressed.

After the comment text, the total values are aligned under each statistics column. If the comment text occupies an area which overlaps the area below the first statistics column, then all the total values are printed on the next report line.

`print-expression`
The *print-expression* is used to construct one or more lines of text to be included in each control break line of the report. The format of a *print-expression* is described under section header "Print Expression".

The print expression contains a number of elements, each representing a fragment of text in the control break line. Print expression elements may be a literal constant, a field name variable or a number of blanks and the order in which they occur in *print-expression* defines the order in which their values occur in the final output text.

In addition to elements which represent text fragments, the print expression for control break lines may also contain <NEWLINE> tags to allow a break line to span multiple report lines.

For HEADING break lines, the values of *input-field* elements are obtained from the first record in the control group which follows the heading. Similarly, if a *compute-field* value is derived from one or more *input-field* values, then the *compute-field* value will also reflect values obtained from the first record in the control group. All other break lines appear after the control group and so, for these control break lines, the value of an *input-field* (or *compute-field*) element is obtained from the last record in the control group.

A *print-expression* may contain **built-in-fields** elements which are described in detail in Appendix A. built-in fields. The following **built-in-fields** are of particular use in break line print expressions:

Built-in Field	Description
#ITEMS	The number of items (detail lines) in the control group. (Not applicable to HEADING break lines.)
#SEQUENCE	The report detail line sequence number. This is equivalent to the total number of items (detail lines) that have been printed to the report so far. (It is the running total that will be reported as the #GRAND break #ITEMS value.)

BROWSE-EXIT

Overview:

The BROWSE-EXIT section will trigger use of Data Editor browse processing to format input record data. It identifies the start of REXX program statements that are to be executed once only, immediately following initial browse of the structured data.

The exit allows for additional processing of the browsed data, not performed naturally by the REPORT utility. For example, you may wish to perform additional input record filtering based on data contained in more than one record. If the exit is used only to initialise REXX variable (*compute-field*) values, then consider using an INIT-EXIT section instead.

Example:

Example 1 - Use BROWSE-EXIT to Perform Advanced Record Filtering:

In the following, input records reflect a music database hierarchy whereby a record containing artist information (record-type "ARTIST") is followed by repeated groups of records. Each record group is comprised of one record containing recorded album information (record-type "ALBUM") followed by a record for each album track (record-type "TRACK").

The example demonstrates use of the BROWSE-EXIT section to exclude album record groups that have only one associated album track. Because this task requires logical processing, it can not be achieved using the FILTER section.

```
BROWSE-EXIT:
 "SET MSGMODE OFF"                        /* Suppress Data Editor messages. */
 "HIDE"                                   /* Hide excluded record shadow lines. */
 "TOP"                                    /* Position at "Top of Data" line. */

  do forever                              /* Loop to "End of Data" */
    "NEXT ALBUM"                          /* Locate the next record of type "ALBUM" */
     if rc<>0   then leave                /* No further ALBUM type records, so end processing. */
    "SET POINT .EXC1"                     /* Set a 1st  exclude line label on current record. */

  /* ** First record following ALBUM record. This should be a TRACK record. ** */
    "DOWN 1";    rr=rc                    /* Locate the next record (save the RC). */
    "extract /DRECTYPE/"                  /* Get the current record type. */

     select
      when rr<>0      then                /* "End of Data"  */
        do; "EXCLUDE ALL .EXC1 .EXC1"     /* Exclude the ALBUM record. */
            leave                         /* End processing. */
        end

      when drectype.1 <> "TRACK"          /* ALBUM not followed by TRACK (zero album tracks). */
        do; "EXCLUDE ALL .EXC1 .EXC1"     /* Exclude the ALBUM record. */
            "UP 1";   iterate             /* Go back 1 line in case current is "ALBUM" the restart loop. */
        end

      otherwise  "SET POINT .EXC2"        /* Set a last exclude line label on current record. */
     end

  /* ** First record following 1st TRACK record. ** */
    "DOWN 1";    rr=rc                    /* Locate the next record (save the RC).  */
    "EXTRACT /DRECTYPE/"                  /* Get the current record type.           */

     select
      when rr<>0      then                /* "End of Data"  */
        do; "EXCLUDE ALL .EXC1 .EXC1"     /* Exclude the ALBUM record. */
            leave                         /* End processing. */
        end

      when drectype.1 <> "TRACK"          /* 1st TRACK not followed by 2nd TRACK (only 1 album track). */
        do; "EXCLUDE ALL .EXC1 .EXC1"     /* Exclude the ALBUM & 1st TRACK record. */
            "UP 1"                        /* Go back 1 line in case current record is type "ALBUM". */
        end

      otherwise  nop                      /* No operation if >1 album track. */
     end
  end
```

The REPORT processing will logically "Browse" the input records. The exit routine performs a loop which processes the browsed records until End of Data is reached. Starting at the "Top of Data" record (i.e. before the first line of data), the loop will perform the following logic:

1. Locate the next "ALBUM" record and set a line label ".EXC1".
2. Scroll down 1 line.
3. Check whether the record-type of the new line is "TRACK".
4. If not, exclude the "ALBUM" record only (0 tracks). Otherwise set a line label ".EXC2".
5. Scroll down 1 line.
6. Check whether the record-type of the new line is "TRACK".
7. If not, only 1 track exists for the ALBUM so exclude the "ALBUM" and "TRACK" records.
8. Repeat until return code > 0 (End of input data).

Example 2 - Computed Field Initialisation:

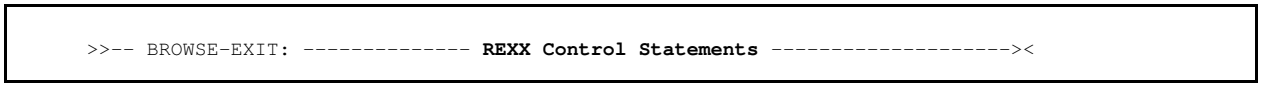
In the following, the BROWSE-EXIT section is used to initialise 2 *compute-field* variables at the start of input record processing.

The REXX statements in the COMPUTE section are executed before writing an output detail line. Assuming the input records are sorted on "JobName", the "JobCount" value will keep a running total of the different job names encountered in the input records.

Note, however, that if the BROWSE-EXIT section is present simply to initialise *compute-field* variables, then an INIT-EXIT section should be used instead.

```
BROWSE-EXIT:
  JobCount = 0                /* Initialise Compute-field variables. */
  JobOld   = ''

COMPUTE:
  if JobName <> JobOld  then
    do; JobCount = JobCount+1  /* Increment count of different Job names. */
        JobOld   = JobName     /* Update last job name encountered.  */
    end
```

Syntax:



Synonyms:

BROWSE-EXIT	BROWSEEXIT

Parameters:

REXX Control Statements
: Any number of valid REXX logical control statements may be specified to constitute an executable REXX routine. A REXX error will occur if invalid statements are entered.

  By default, CBLSDATA is the REXX environment and so any Data Editor primary commands may be executed as part of the logic. This may include commands such as EXTRACT (to extract information about the data in view), FIND, LOCATE, WHERE, NEXT, PREVIOUS, etc.

  The REXX routine ends at the start of the next REPORT section or at the end of the REPORT definition input, whichever is encoutered first.

COLUMNS

Overview:

The COLUMNS section defines the content and appearance of detail lines to be written to printed report, CSV, XML or JSON output.

Each statement in the COLUMNS section either identifies a column or a gap in the detail line. The order in which the statements are provided determines the order in which the columns and gaps appear in the detail lines. A column definition references a constant string 'literal' or a field name that either maps data in an input record, is a computed variable name or a REPORT utility built-in field (e.g. #RECNUM returns the input record number).

Each column definition may also specify column header text and, for printed report output, the header text alignment, column data width and data alignment.

A <NEWLINE> control statement will start a new detail line for the column definition statements that follow it. The column definitions start in the first position of the next detail line.

Any number of column definitions may be specified and, for printed report output, the number of detail lines is limited only by the defined page depth.

Input record fields used in the COLUMNS section to define report columns or otherwise listed in the REQUIRED section determine the fields for which values will be extracted from input records. These field *input-field* names may then be referenced in other REPORT definition sections. e.g. In print expressions used to generated page headings, footings and control break lines.

Examples:

SMF Record Input:

```
COLUMNS:
  SMF030_Identification.zJOBNAME     ('Job|Name'  CENTRE)
  3
  SMF030_Identification.zSIT         ('Job|Start')
  :Duration                          ('Job|Duration')  8
  SMF030_Identification.zRUD         ('RACF|User')
  SMF030_Identification.zJESJOB      ('JES|Jobname')
 '|'                                 ('')
  SMF030_EXCP.zEXP.zDDN              ('DDName')
  SMF030_EXCP.zEXP.zBLK              ('EXCP|Blocks'   RIGHT)  8  RIGHT
  SMF030_EXCP.zEXP.zBSZ.zBSZLarge    ('Largest|Block' RIGHT)  6  RIGHT
  SMF030_EXCP.zEXP.zCUA              ('Dev#')
```

The above example will output a single detail line of 9 columns.

Each column defined by a mapped field in the input data, is expressed by a field identifier. The field identifier is comprised of one or more qualifiers, each separated by a dot/period ("."). The first qualifier ("SMF030_Identification", "SMF030_EXCP") is the name of the record mapping structure (record-type) that maps the entire input record or, as in this case, a particular segment of the type 30 SMF record to which the required field definition belongs.

The last qualifier ("zJOBNAME", "zSIT", "zRUD", "zJESJOB", "zDDN", "zBLK", "zBSZLarge", "zCUA") is the name of the required field within the record mapping structure. Each intermediate qualifier (if present) is the name of a group field containing either the required field or another group field to which the required field belongs.

In the above example, field identifier "SMF030_EXCP.zEXP.zBSZ.zBSZLarge" uniquely identifies a field name "zBSZLarge" defined in group field "zBSZ" which is itself defined in group field "zEXP" within the record mapping structure "SMF030_EXCP". (See publication *"FileKit SMF Utilities"* for details of the FileKit SMF record mapping structures and their filed name definitions.)

The statement containing "3" indicates that 3 blanks are to be included between the zJOBNAME and zSIT columns. ":Duration" references a vatiable named "Duration" whose value is re-assigned before each detail line is written via execution of the COMPUTE section REXX statements. The literal '|' will insert a vertical line between each zJESJOB and zDDN value in the detail line. The header above the vertical line is suppressed.

Multiple Column Lines:

```
COLUMNS:
     'Artist:';   ARTIST 30;    'Album: ';   ALBUM 30
 <NEWLINE>
     'Track Info:'
 <NEWLINE>
    1; 'Track#:'; TRACK-NUM 3; 0; '/'; 0;  DISC-NUMBER 1
    3; 'Title:' ; NAME 30;     3; ' Duration:';    :Elapsed     11 RIGHT
 <NEWLINE>
   58;                             'Released:'; 2; RELEASE-DATE 10
```

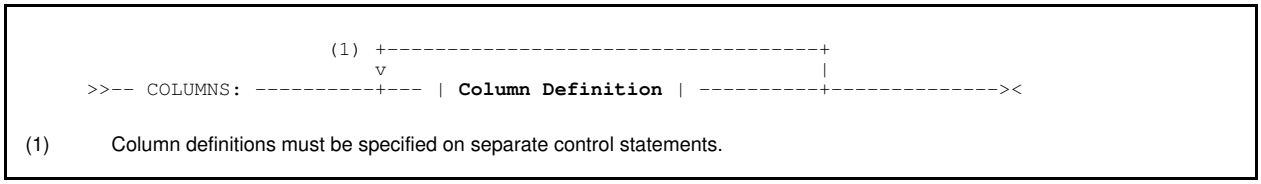
The above example uses the semi-colon (";") default statement separator symbol to include more than one statement on a single line of the report definition input.

The 3 <NEWLINE> tags mean that 4 lines of column data will be generated for each output record.

The input records are mapped by a single record mapping structure (record-type) and so specification of the record-type name in the field identifier is unnecessary. Also, since the defined field names are unique within the record-type structure, the field identifiers need only have one qualifier. This is simply the name of each required input record field ("ARTIST", "ALBUM", "TRACK-NUM", "DISC-NUMBER", "NAME", "RELEASE-DATE").

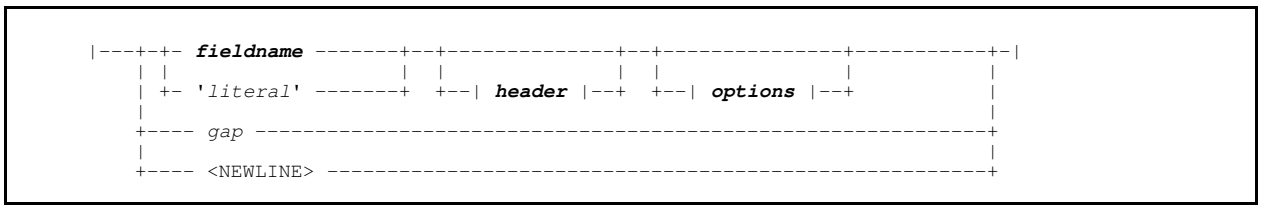
Integer width values follow the field identifiers to restrict the width of the output columns. A statement containing only an integer (gap) value defines the number of blanks to be inserted between the column before and after the specified value in the output report line.

Syntax:

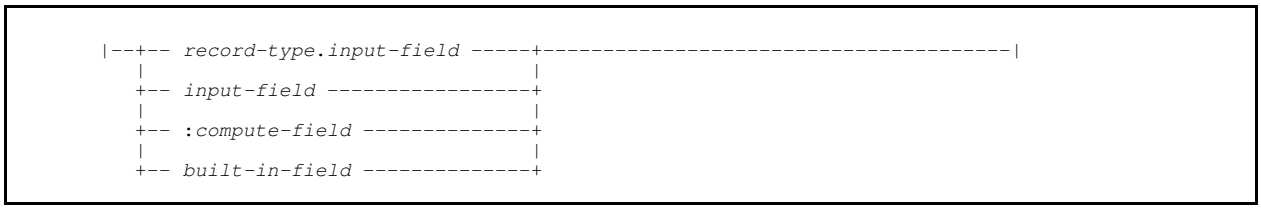


(1) Column definitions must be specified on separate control statements.

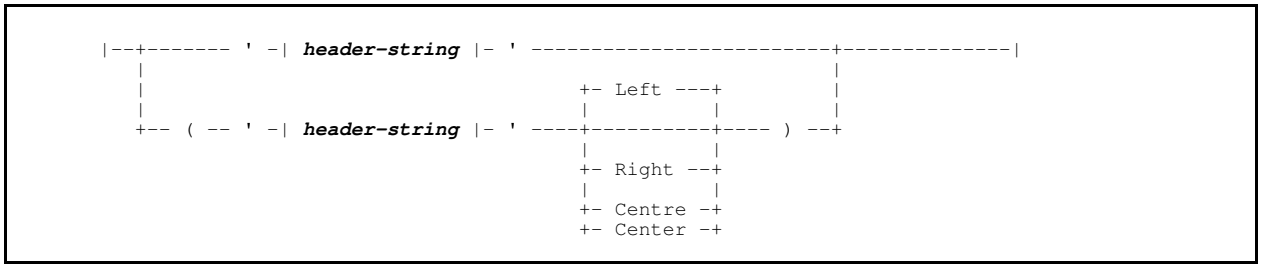
Column Definition:



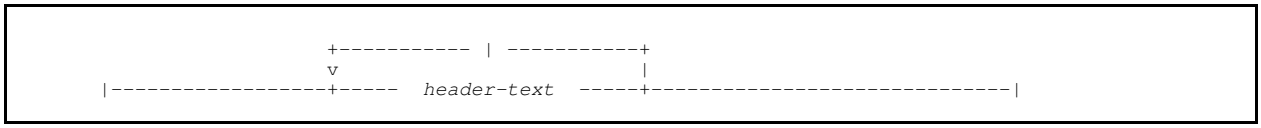
***fieldname*:**



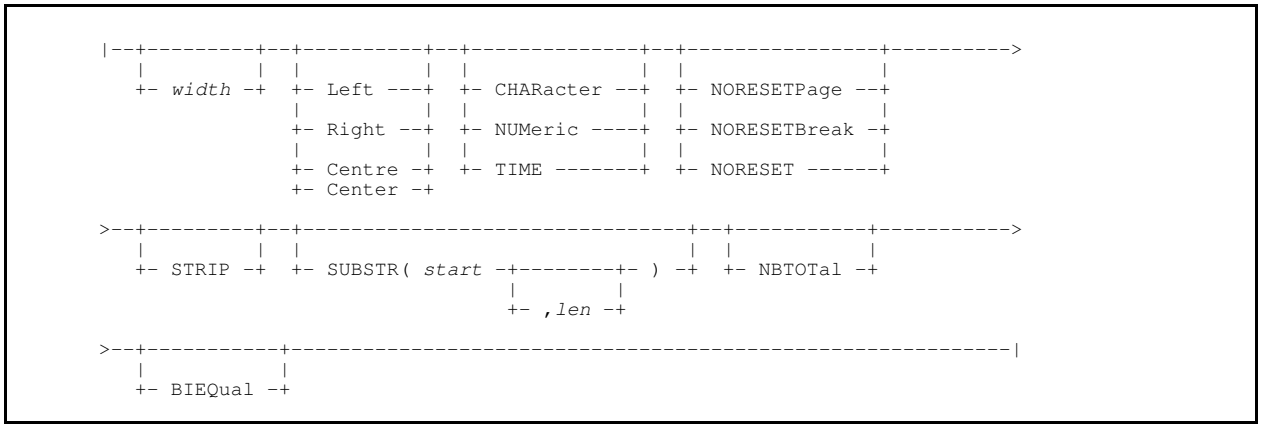
***header*:**



***header-string*:**



***options*:**



Parameters:

fieldname
A *fieldname* is the name of an input record field, computed field or built-in program field used to identify a column of data in the output detail line and from which column values are obtained.

Field values may change after processing the current input record or record segment. Therefore, a column identified by *fieldname* may contain a different value in each report output detail line.

A *fieldname* must be one of the following field types:

input-field
This *fieldname* format identifies a field whose values are extracted from within the input records.

Use of *input-field* without a preceding *record-type* specification applies only when only one record-type structure exists for mapping the input record or input is a DB2 table. In these cases *input-field* should be used.

See description of input record fields for details on *input-field* specification.

record-type.input-field
Like *input-field* alone, this *fieldname* format identifies a field within the input records.

Where the FileKit SDO structure associated with the input records contains more than one record-type definition, then the *record-type* name must be included before the *input-field* specification to identify the record-type mapping to which the field belongs. The *record-type* and *input-field* name specifications must be separated by a dot/period (".") character.

Note that when reporting on SMF records, the REPORT utility will dynamically generate an SDO structure to map the required SMF record types. This structure will always contain more than one record-type and so a *record-type* name is mandatory on an SMF input record field specification. Furthermore, the *record-type* names specified in the COLUMNS and REQUIRED sections of an SMF report definition input, are used by the REPORT utility to identify which SMF record-type mappings are required to build the dynamic SDO structure. See publication "*FileKit SMF Utilities*" for SMF record-type and field names.

Use of *record-type* is unnecessary where the SDO structure contains only one record-type definition.

:*compute-field*
This *fieldname* format identifies a field whose value is computed using REXX statements specified in the COMPUTE section.

To distinguish it from an *input-field*, a *compute-field* must be specified with a colon (":") symbol prefix. This prefix character is used only to identify the field name as a *compute-field* and is **not** part of the *compute-field* name itself. e.g. A column name referenced by a computed field name "MyValue" is identified using ":MyValue" in the column definition control statement.

See description of computed fields for details on *compute-field* specification.

built-in-field
This *fieldname* format identifies a REPORT utility built-in field.

All *built-in-field* names begin with a hash ("#") character and are documented in Appendix A. Built-in fields.

See description of built-in fields for details on *built-in-field* specification.

Where *fieldname* is used in other REPORT definition sections to reference a column definition, then the *fieldname* format must exactly match that specified in the COLUMNS section. e.g. The *fieldname* specifications in the following refere to the same input record field but do not match exactly and so an error is returned.

```
COLUMNS:
  SMF014_INPUT_or_RDBACK_Dataset.JFCB.Ind2.DISP

SORT:
  SMF014_INPUT_or_RDBACK_Dataset.DISP
```

Statements in the following REPORT definition sections contain a *fieldname* reference that must exactly match either a *fieldname* specified as a COLUMNS section column definition or a REQUIRED section fieldname specification.

BLANKWHENZERO	Field contains numeric values which are replaced with a blank (x'40') character if the value is zero (0).
BREAK	Field will trigger a report break if there is a change in its value.
SORT	Field values are used to sort the report output detail lines.
STATISTICS	Field contains values for which statistical values are generated. (COLUMNS section only.)

gap
Applicable only to printed report output and ignored for CSV, JSON and XML output, *gap* specifies the number of blank characters to be inserted between report columns.

A *gap* value occupies a single control statement in the COLUMNS section. It may also be specified before the first column and/or after the last column to force padding at the beginning and/or end of the column header and data lines.

The default is 1.

header
Definition of the column header text which is used as follows:

- For a printed report, the column header text will be printed on each new page of the report after the page headers and above the column values.
- For CSV output, the column header is included in the first output record with a variable number that corresponds to its column values occurring in subsequent CSV records.
- For XML the column header text is used as the tag name. A start and end tag is generated and used to enclose each of the column's values. Any invalid character symbols in the tag name is translated to an underscore ("_") character.
- For JSON the column header text is used as the JSON field name occuring before the colon (":") in a name/value pair.

If no *header* is specified then a default header is used. If the column is defined as an *input-field*, the default header is the last or only qualifier of the field identifier (i.e. the field name) is used. Otherwise the default column header will be the field name specified as *compute-field*, *built-in-field* or *literal*.

The *header* is specified as a character literal *header-string* which must be enclosed in either quotation marks ("") or apostrophes("""). Header text alignment may be specified by enclosing both the *header-string* and the alignment specification within parentheses "()".

If specified in parentheses, a null *header* value ("") will suppress column heading text. For printed report output, this means that no header is produced for the column and so the column header underline is also suppressed. A null *header* will create a null entry in the header record of CSV output and would result in a null name tag for XML and JSON output. Beware that a null tag name is not valid for XML and JSON.

A null *header* value which is not specified in parentheses will be ignored and the default column header used.

For printed report output, a *header-string* specification may span more than one report line. To do this, the quoted *header-string* may be split into a number of *header-text* elements each separated by the column header break symbol which, by default, is the vertical bar symbol ("|"). Each *header-text* element will appear on the next line of the report and has the effect of reducing the width of the column header. If *header-string* is not split into *header-text* elements, the column header width is the length of the *header-string* text. Otherwise the column header width is the length of the longest *header-text* element.

In the following example, the quoted *header-string* is split into 3 *header-text* elements.

```
#RECNUM     ('Input|Record|Number')
```

This will display in a printed report as follows:

```
Input
Record
Number
------
```

The column header break symbol is set by option COLHEADBRK and is assigned to **vertical bar ("|")** by default. This symbol may be included within the quoted *header-string* text to break the *header-string* into a number of *header-text* elements. Each *header-text* element will be aligned above the column values and printed on a new line of the report.

Exceptions to this rule occur when:

1. The *header-string* has a length of 1 character and that character is the column header break symbol. In this case, the column header break symbol is treated as being the only *header-text* element.
2. A column header break symbol is immediately preceded (escaped) by another column header break symbol. In this case, the pair are treated as a single occurrence of the symbol and treated as text within a *header-text* element. e.g. If the column header break symbol is default ("|"), a *header-string* of "YES||NO" has a single *header-text* element which is printed on a single line as "YES|NO".
3. Output is to CSV records, in which case any occurrence of a column header break symbol is translated to a blank character.
4. Output is to JSON or XML records, in which case any occurrence of a column header break symbol is translated to an underscore ("_") character.

'*literal*'
A *literal* is a character string constant value that will be repeated in every detail line. The character text must be enclosed in either quotation marks (") or apostrophes(').

In the following, the vertical bar symbol ("|") will be repeated on every report line so having the effect of producing a vertical line between the column before and after.

```
COLUMNS:
   SMF110#01_Performance_Class.zSTART    ('Start' CENTRE)
  '|'                                    ('|')
   SMF110#01_Performance_Class.zSTOP     ('Stop'  CENTRE)
```

width
Specifies the column data width. Column values will be truncated or blank padded to this width accordingly.

The default width is the maximum number of characters that would be required to display the widest value represented by the column field definition. For example, if *input-field* represents an unsigned, 2-byte integer field in the input data, the default width is 5 because the highest value represented by *input-field* is 65535.

For column definitions identified by a *literal* string or a *built-in-field* representing a character string of fixed length, the default *width* is the length of the character string. For a *compute-field*, the default width is either 9, or a value greater than 9 and equal to the largest width value specified for the same *compute-field* anywhere within the report definition. For a *built-in-field* representing a numeric value, a default *width* of 9 is used.

For CSV, JSON and XML output, *width* will determine the output width of the value. For a printed report, the output column width will be the larger of the column header width and *width* value.

BIEQUAL
Specifies that a blank column detail line value is to be displayed when the value for the column entry matches that for the same column in the previous report detail line. This option is applicable only to PRINT report output.

Synonyms are BLANKIFEQual, BLANKWHENEQual and BWEQual.

CENTRE|CENTER|LEFT|RIGHT
Specifies the data alignment of the field, literal or header-string value. The value text may be left adjusted, right adjusted or centralised within the defined data width (*width*).

An alignment may be specified for printed report column header text (*header*) as well as the column data values. For column data definitions of NUMERIC data type, the default column data value text alignment is RIGHT. For all other types of column definition and also for column headers, the default text alignment is LEFT.

Aligned values, padded or truncated to the data width (*width*), will be written to the column display area of printed report or, alternatively, to the CSV, JSON and XML record output.

If specified for column headers, the *header* must be enclosed in parentheses "()" and include the required alignment keyword within. The *header-string* (or all its individual *header-text* elements) will be aligned within an area equal to the column header width.

CHARACTER|NUMERIC|TIME
Specifies the data type of the values assigned to *fieldname*.

For an *input-field*, the data type is automatically determined based on the field mapping information provided by the record-type structure. However, you may wish to override this. For example, if an *input-field* has a source data type of CHARACTER but contains numeric values, you may wish to set data type "NUMERIC" so that the field is included as one of those eligible for statistics (totals, averages, etc.) generation.

However, for a *compute-field*, there is no defined data type on which the REPORT utility can base a default assignment. Therefore, it assigns a data type based on the data type of the field's value at the time the first report detail line is written. This is determined as follows:

1. If the value is in a time format then the *fieldname* is "TIME".
   The REPORT utility identifies a time format as ***n:n:n.n***, ***n:n:n***, ***n:n*** or ***n:n.n*** where ***n*** represents 1 or more decimal digits).
2. If the value is a REXX numeric value then the *fieldname* is "NUMERIC".
3. Otherwise the *fieldname* is "CHARACTER".

This method is a best effort and may not return the desired result. Therefore, it is recommended that a specific data type is provided for a *compute-field* definition.

The data type of a *built-in-field* is assigned internally by the REPORT utility and should not require a data type specification.

`NBTOTAL`

Specifies that a count of the number of non-blank values will be maintained for each control break level. If no **BREAK** section exists, then the count exists for the default (#GRAND) control break only.

The NBTOTAL value for any report break, will occupy the statistics value position below the column data used to display the accumulated total value for columns of numeric or time data type. Therefore, the NBTOTAL value is displayed only if the control break definition includes a TOTAL control break print line, which it does by default unless NOTOTAL is specified.

NBTOTAL may be specified on an *input-field* or *compute-field* COLUMN definition of any data type. However, if the field is of numeric or time data type, the NBTOTAL value will replace the accumulated total values for that column. For non-character type fields, all values are non-blank unless identified as a column field in section **BLANKWHENZERO**. In this case, the field's zero values are translated to blanks and so would not be included in the count of non-blank values.

Specification of NBTOTAL will include the field column in the list of fields for which statistics values are displayed by default on each control break. However, if a **STATISTICS** section exists, then statistics values (including the NBTOTAL value) will only be displayed for column fields referenced in the STATISTICS section.

`<NEWLINE>`

Applicable only to printed report output and ignored for CSV, JSON and XML output, <NEWLINE> may be specified at the start of a COLUMNS statement or on a statement of its own to trigger a break in the column detail line. Any column values that follow the <NEWLINE> specification will be written to the start of a new print output line.

If <NEWLINE> is used, the resulting multiple column detail lines written for the same output record means that the print of column headers is suppressed by default. OPTION COLHEAD(YES) may be used to force output of column headers for columns in the first column detail line only.

`NORESET`

Applicable only to *input-field* column defintions, NORESET will exclude the field from value reset processing.

Value reset processing occurs following output of a report detail line at which point the REPORT utility sets a null value to each *input-field* specified in the COLUMN or REQUIRED sections if either of the following is true:

- ◊ No REPEAT: section has been specified.
- ◊ Both REPEAT: and RESET: sections have been specified, and output has been triggered by input of a record (or record-segment) with a record-type mapping that matches one specified in the RESET: section.

If NORESET is specified on an *input-field* definition in the COLUMN or REQUIRED section, then this field's values are never reset.

`NORESETBREAK`

NORESETBREAK is applicable only to *input-field* column defintions and is effective only if control breaks exist (i.e.a BREAK: section has been specified).

NORESETBREAK will allow the field's value to be reset to null by value reset processing (see NOREST description). However, the field's value will be re-instated if the next output detail line is the first in a break control group. Therefore, the last input record value assigned to the column field will be displayed in place of blank characters in the first report detail line of a control group.

`NORESETPAGE`

NORESETPAGE is applicable only to *input-field* column defintions.

NORESETPAGE will allow the field's value to be reset to null by value reset processing (see NOREST description). However, the field's value will be re-instated if the next output detail line is the first on a new page. Therefore, the last input record value assigned to the column field will be displayed in place of blank characters in the first report detail line of a new page.

`STRIP`

Specifies that leading and trailing blanks that potentially exist in the field value, are to be stripped prior to being aligned in the column display.

Note that, if an alignment (LEFT, RIGHT, CENTRE or CENTER) is specified on an *input-field* column definition and that alignment does not match the default alignment for the *input-field* data type, then STRIP is automatically implied.

For example, values in character fields may be padded with blanks to a fixed length but an alignment of RIGHT may required so that the lsat non-blank character appears in the last position of the column display area. Since character fields are LEFT aligned by default, then specification of RIGHT will also imply STRIP.

`SUBSTR(`*`start`*`[,`*`len`*`])`

Used when the required output field values are a sub-string of the input values. SUBSTR specifies *start*, the position within the input value of the first output field character, and optionally *len*, the number of characters in the output field.

If *len* is not specified, then the substring value will begin at the *start* character position and end at the last character in the input value. If *len* extends beyond the last character of the input value, then the output value will be padded with blanks. The substring value will ultimately be aligned according to the specified or default data alignement to a length specified by *width*.

If both SUBSTR and STRIP are used, then the SUBSTR operation will occur first so that leading and trailing blank characters will be stripped fron the sub-string value.

COMPUTE

Overview:

The COMPUTE section identifies the start of REXX program statements that are executed immediately following input of a data record but after input field values have been extracted from the new input record (or record segment).

Although other tasks may be performed within the REXX statements, this section is used primarily to update values assigned to user-defined *compute-field* variables using standard REXX expressions, operators and functions.

A *compute-field* variable may be based on the current value of an input record field (*input-field*). If so, *input-field* must have been defined in either the COLUMNS or REQUIRED section.

By default, *input-field* values are each assigned to a REXX variable that has the same name as the *fieldname* used to identify the field within the COLUMNS or REQUIRED sections, but with any *record-type* (record mapping name) qualifier removed.

For example, the value assigned to an *input-field* identified as "SMF014_INPUT_or_RDBACK_Dataset.JFCB.VOLS" in the COLUMNS section, is assigned to the REXX variable "JFCB.VOLS". If, however, the same input record field is identified as "SMF014_INPUT_or_RDBACK_Dataset.VOLS", which is valid since field name "VOLS" is unique within the "SMF014_INPUT_or_RDBACK_Dataset" record mapping structure, then the field values will be assigned to the REXX variable "VOLS". See also the FIELDNAME option which may be used to force field value assignments to qualified (LONG) or unqualified (SHORT) REXX variable names.

Exceptions to the default REXX variable naming convention are as follows:

1. An input field which is an individual element in an array of field entries. Fields of this type are referenced using a numeric subscript in parentheses immediately following the field name. For example, "SMF014_INPUT_or_RDBACK_Dataset.SMFEXCP(1)" identifies the first array element of field SMFEXCP.

   A REXX compound variable will be set for an array element so that the field name is the stem and the numeric subscript is the tail. Therefore, for array element "SMFEXCP(1)", the REXX variable "SMFEXCP.1" is set. Additional parts are added to the tail for each dimension of a multi-dimension array. For example, the 2-dimensional array element "ROOM(2,3)" would set the field value to REXX variable "ROOM.2.3".

2. An input field which contains the hyphon/minus symbol ("-"). In particular, COBOL supports field names which contain this symbol. However, REXX does not support "-" within variable names and is instead interpreted as a minus operator for which REXX will attempt to perform a subtraction function.

   To avoid this problem, the REPORT utility instead assigns the field value to a REXX variable name equal to the field name but with all minus ("-") symbols translated to underscore ("_"). For Example, the value in field name "TOTAL-TIME" is assigned to REXX variable "TOTAL_TIME".

The value assigned to *input-field* REXX variable is the last value extracted from the input data prior to calling the COMPUTE REXX statements. This extracted value is unaffected by any STRIP or SUBSTR parameter specified on the field definition in the COLUMNS or REQUIRED section. Similarly, the value assigned to any *built-in-field*, used as a REXX variable name within the COMPUTE section, will be the prevailing value of the field at the time the REXX statements are called.

If required, *input-field* and *built-in-field* values may also be updated within the COMPUTE section REXX statements.

Note that, if a *compute-field* needs to be assigned an initial value, then this may be done in the BROWSE-EXIT section. For example, a *compute-field* variable "XNUM" may be a counter value referenced in a BREAK line and so must be initialised to zero ("0") before the COMPUTE section containing the REXX statement "XNUM=XNUM+1" is executed.

The REPORT utility supports a number of built-in REXX functions that may be used in the COMPUTE section. See Appendix B. Built-in functions for descriptions of these.

Example:

```
REQUIRED:
  SMF030_Common_Address_Space_Work.zTME

COMPUTE:
  Duration = ''
  if zSIT<>''      then
    do; if zTME<>''
          then   Duration = Secs2Time( Time2Secs(zTME) - Time2Secs(zSIT) )

        zSIT  =  translate(zSIT , "-:," , "/:.")    /* Date and Time chars. */
    end

COLUMNS:
  SMF030_Identification.zJOBNAME
  SMF030_Identification.zSIT
  :Duration
```

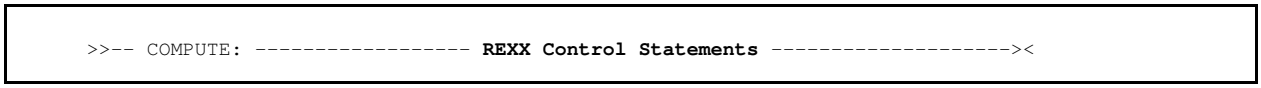
The above example calculates the time elapsed between a start and end timestamp and assigns the value to the REXX variable "Duration". It then updates the "zSIT" input record timestamp value so that it displays in the format "yyyy-mm-dd HH.MM.SS,tt".

This variable name is identified as :Duration (a *compute-field*) in the COLUMNS section. Each column detail line will contain a job name ("zJobName"), followed by the job execution start time ("zSIT"), followed by the job execution time (":Duration").

Note that the input record field containing the end time (zTME) is not included in the column detail line and so must be referenced in the REQUIRED section in order that its value is retrieved from the input records.

The Duration value is dependent upon non-null values for both zTME and zSIT otherwise a null Duration value will be returned. "Secs2Time" and "Time2Secs" are REPORT utility built-in functions to convert a number of seconds to an elapsed time and a timestamp to number of seconds respectively.

Syntax:



Synonyms:

COMPUTE	COMP

Parameters:

REXX Control Statements
: Any number of valid REXX logical control statements may be specified to constitute an executable REXX routine. A REXX error will occur if invalid statements are entered.

  The REXX routine ends at the start of the next REPORT section or at the end of the REPORT definition input, whichever is encoutered first.

DISPLAY-EXIT

Overview:

The DISPLAY-EXIT section is applicable only to foreground execution of the the REPORT utility.

On completion of REPORT utility execution in FileKit foregroud, the output report is displayed automatically. If an OUTDD was specified on the REPORT input then, the report is displayed in a Data Editor browse view, otherwise the report is displayed in a Text Editor edit view.

The DISPLAY-EXIT section identifies the start of REXX program statements that are executed once only following display of the output report.

The exit may be used to issue command to the Text or Data Editor as appropriate.

Example:

Example 1 - Apply Colour Highlighting to Displayed Report:

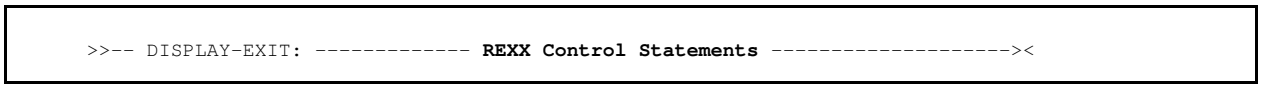
The example demonstrates use of the DISPLAY-EXIT section to highlight report lines based on their content, and to highlight individual keyword strings in the text.

```
DISPLAY-EXIT:
 if address() = "CBLEDIT"                /* If this is a Text Editor EDIT view. */
  then
   do; "lcolour /==/      yellow"         /* Highlight lines containing "==" in yellow. */
       "scolour /Minimum/ red"            /* Highlight string "Minimum" in red. */
       "scolour /Maximum/ blue"           /* Highlight string "Maximum" in blue. */
       "scolour /Average/ green"          /* Highlight string "Average" in green. */
   end
                                  /* Otherwise, this is a Data Editor BROWSE view. */
  else "rcolour unmapped yellow when (record << '==')"
                                          /* Highlight lines containing "==" in yellow. */
```

A test on the name of the prevailing REXX environment is performed to determine whether the Text Editor (REXX environment "CBLEDIT") or the Data Editor (REXX environment "CBLSDATA") is being used to display the report output.

For the Text Editor, commands LCOLOUR and SCOLOUR are supported to perform line colouring and string colouring respectively. For the Data Editor, line (row) colouring can be performed using command RCOLOUR.

Syntax:



Synonyms:

DISPLAY-EXIT	DISPLAYEXIT

Parameters:

REXX Control Statements

Any number of valid REXX logical control statements may be specified to constitute an executable REXX routine. A REXX error will occur if invalid statements are entered.

If the REXX environment is "CBLEDIT" then Text Editor commands may be executed. Otherwise, if the REXX environment is "CBLSDATA" then Data Editor primary commands may be executed as part of the logic.

The REXX routine ends at the start of the next REPORT section or at the end of the REPORT definition input, whichever is encoutered first.

FILTER

Overview:

The FILTER section specifies an SDE filter clause used to include (or exclude) processing of input records containing data that matches particular criteria.

For SMF record input, a filter clause may be used in place of REPORT utility parameters that filter input records based on subsystem Id (SID), User Id (USERID) and Job Name (JOBNAME). Any specification of these parameters will be overidden and ignored if the REPORT definition contains a FILTER section filter clause.

For all types of input records, a filter clause will override any record filtering specified by find string(s) in the REPORT utility parameter FIND.

The FILTER section allows for more flexible record filtering than can be achieved simply using REPORT utility input parameters. For example, input records may be selected based on the contents of more than one field. Also, a filter clause may involve more complex conditional logic using balanced parentheses and logical operators AND and OR.

A filter clause may span several REPORT definition control statments and so statement continuation is not necessary.

Examples:

Example 1 - Apply Filter Conditions for Multiple Record Mappings:

In the following example, records are ordered so that a unique customer record (mapped by record-type CUSTOMER) is followed by zero or more records detailing invoices raised for that customer (mapped by record-type INVOICE).

The FILTER section is used to select only invoices raised in the finacial year 2016/17 (i.e. between 1st April 2016 and 31st March 2017) for specific customers.

```
FILTER:
  INCLUDE  CUSTOMER
    WHERE (CUSTNO  IN (6281,7532) )

  INCLUDE  INVOICE
    WHERE (CUSTREF IN (6281,7532) )  &  ISSUE_DATE  BT  ('2016/04/01','2017/03/31') )
```

Example 2 - Filter SMF Records:

The following example uses the FILTER section to select SMF records of Type 14 only when the DDNAME does not begin with "SYS" and the DSN begins with either "CBL", "JGE" or "NBJ" or when the SMF Type 14 record is for a dataset with SMS Management class "CBLHSM".

SMF records are split into sections (segments) where each segment is mapped by a discrete FileKit SMF record-type structure. The first segment of an SMF record is mapped by a base (primary) record-type structure with all remaining record segments being mapped by secondary record-type structures.

Field references are comprised of one or more qualifiers, each separated by a dot/period (".") . The first qualifier is the name of the primary or secondary segment record-type mapping structure and the last qualifier is the name of the required field within the record-type structure. Each intermediate qualifier (if present) is the name of a group field containing either the field itself or another group field to which the required field belongs.

Any field referenced within the filter clause expression which occurs in the primary record-type mapping, may simply be referenced by its field name. Otherwise, if the field is mapped by a secondary record-type, the field reference must include the secondary segment record-type name qualifier.

The fields **SMFTIOE5** and **DSN** belong to the primary segment record-type structure ("SMF014_INPUT_or_RDBACK_Dataset") and so do not require specification of the record-type 1st level qualifier. However, **zMCN** belongs to the secondary segment record-type structure ("SMF014#2_SMS_Class") and so must be referenced with the record-type 1st level qualifier.

```
FILTER:
  (                  SMFTIOE5 \>> 'SYS'
   and
       (             DSN       >> 'CBL.CBLI'
         or          DSN       >> 'JGE'
         or          DSN       >> 'NBJ'
       )
  )
 or
  ( SMF014#2_SMS_Class.zMCN   =   'CBLHSM' )
```

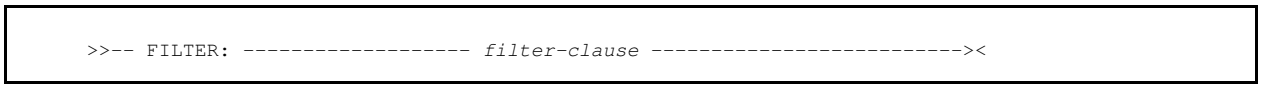
Example 3 - Apply a DB2 SQL Search Condition:

The following example uses the FILTER section to pass a DB2 SQL WHERE clause search condition to the SQL query used to create the input DB2 result table.

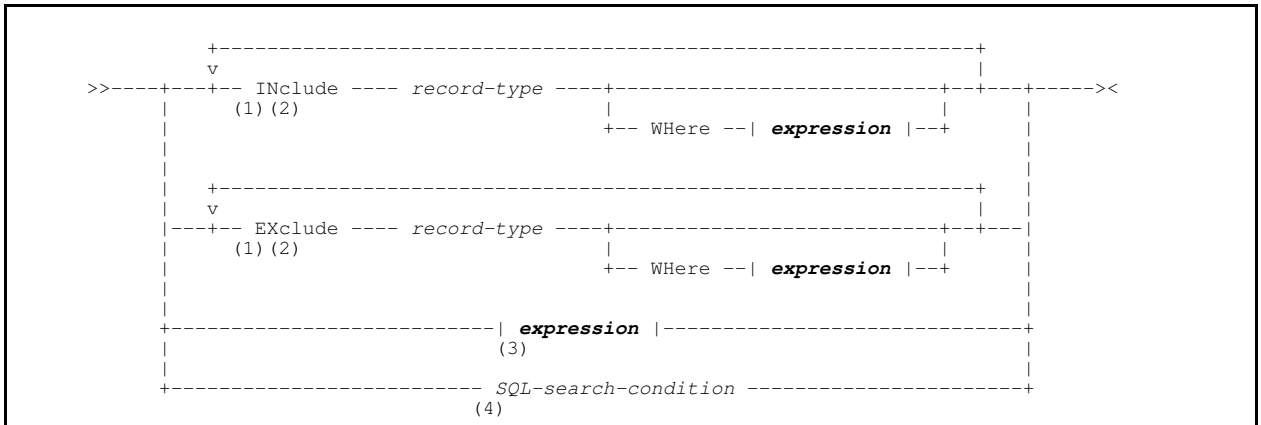
Input is the SYSIBM.SYSTABLES table and the search condition selects only those rows containing tables belonging to the CBLI350 database or tables whose name contains "SELCTRN".

```
FILTER:
   NAME LIKE '%SELCTRN%'  OR  DBNAME = 'CBLI350'
```

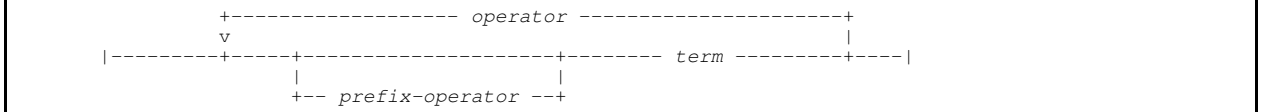
Syntax:



***filter-clause*:**



***expression*:**



(1) A *filter-clause* may not contain both INCLUDE and EXCLUDE format sub-clauses.

(2) A *filter-clause* that uses the INCLUDE or EXCLUDE format to nominate specific record-type mappings is not valid if the report is generated from DB2 table rows or SMF input records.

(3) A *filter-clause* containing only an *expression* will include records of any *record-type* mapping that matches the expression criteria. (Not valid if the report is generated from DB2 table rows.)

(4) A *SQL-search-condition* specification is valid only for reports generated from DB2 table input.

Parameters:

filter-clause
A filter clause will include or exclude records that are mapped by a particular record type that match an optionally provided expression.

EXCLUDE *record-type*
Specifies one instance of an EXCLUDE sub-clause which names the *record-type* for which associated input records may be excluded. For DB2 table and SMF record input, use of EXCLUDE *record-type* in the *filter-clause* is invalid and will return error ZZSR036E.

Any number of EXCLUDE sub-clauses may be specified. Although achievable in a single *expression*, the same *record-type* name may be specified on more than one EXCLUDE sub-clause to provide alternative selection criteria expressions for the same record-type.

Input records will be tested against each EXCLUDE sub-clause in the order specified until either a match is found or all EXCLUDE sub-clause specifications have been exhausted. Input records that are not mapped by *record-type* will fail the individual EXCLUDE sub-clause specification.

If the input record is mapped by *record-type*, then the record data is tested against the WHERE *expression* that follows. If no WHERE *expression* is specified or the input record data satisfies the criteria specified by *expression*, then the record passes the filter criteria and is excluded from REPORT processing.

If the input record does not satisfy any of the EXCLUDE sub-clause specifications then it fails the filter criteria and is included in REPORT processing.

`INCLUDE record-type`
Specifies one instance of an INCLUDE sub-clause which names the *record-type* for which associated input records may be included. For DB2 table and SMF record input, use of INCLUDE *record-type* in the *filter-clause* is invalid and will return error ZZSR036E.

Any number of INCLUDE sub-clauses may be specified. Although achievable in a single *expression*, the same *record-type* name may be specified on more than one INCLUDE sub-clause to provide alternative selection criteria expressions for the same record-type.

Input records will be tested against each INCLUDE sub-clause in the order specified until either a match is found or all INCLUDE sub-clause specifications have been exhausted. Input records that are not mapped by *record-type* will fail the individual INCLUDE sub-clause specification.

If the input record is mapped by *record-type*, then the record data is tested against the WHERE *expression* that follows. If no WHERE *expression* is specified or the input record data satisfies the criteria specified by *expression*, then the record passes the filter criteria and is included in REPORT processing.

If the input record does not satisfy any of the INCLUDE sub-clause specifications then it fails the filter criteria and is excluded from REPORT processing.

`WHERE expression`
Applicable only if EXCLUDE *record-type* or INCLUDE *record-type* is specified, the WHERE keyword indicates the start of the record selection *expression*.

If WHERE *expression* is not specified, all input records mapped by *record-type* will satisfy the sub-clause.

expression
An SDE (Data Editor) *expression* that returns a Boolean value (1="true", 0="false") or a numerical value. A numerical result is treated as being Boolean in nature so that a value of zero (0) is a "false" condition and any non-zero value is a "true" condition. If "true", the input record passes the sub-clause criteria.

For example, the following *expression* will return "1" (a true condition) if the field "FORENAME" (length 16) contains "John".

```
STRIP(FORENAME) = c'John'
```

Note that for SMF record input, an *expression* **must** be specified without INCLUDE/EXCLUDE WHERE parameters. For SDE record input, an *expression* without INCLUDE/EXCLUDE WHERE parameters is commonly used when all records are mapped by a single *record-type*.

If specified without INCLUDE/EXCLUDE WHERE parameters, only one *expression* may be specified which then applies to **all** input records regardless of record mapping (*record-type*).

ZZSD061E is returned if *expression* contains the name of an *input-field* not defined in the *record-type* used to map the input record. Therefore, if *expression* is specified without INCLUDE/EXCLUDE WHERE parameters, *input-field* would have to exist in all base (or primary segment) record-types used to map input record data.

An *expression* consists of one or more *terms* (with or without a *prefix-operator*) and zero or more *operators*.

- A **term** may be a literal string, numerical value, input field specification (*input-field*), function call or another *expression*. REPORT utility *compute-field* or *built-in-field* definitions are not supported as terms in a filter clause *expression*.
- A **prefix-operator** is optional and applies to the term that follows. It may be a unary plus ("+"), unary minus ("-") or a logical NOT ("¬") symbol.
- An **operator** acts on the pair of terms between which it is positioned.

An expression and individual terms within an expression are evaluated from left to right. However, the order in which operators are actioned depends on their defined level of precedence and the presence of parentheses.

An operator with a higher precedence level will be actioned before operators with a lower precedence level. This process is repeated until the entire expression is evaluated. e.g. In the following expression, where operator2 has a higher precedence level than operator1:

```
term1  operator1  term2  operator2  term3
```

The sub-expression `term2 operator2 term3` will be evaluated first.

When parentheses are encountered, the entire sub-expression between the parentheses is evaluated immediately when the term is required. In this way parentheses may be used to force the action of an operator with a lower precedence level before that with a higher level. e.g. Logical NOT has a higher precedence level than logical AND, therefore `¬1&0` evaluates to 0, however `¬(1&0)` evaluates to 1.

The order of operator precedence is as follows (highest level at the top):

Prefix operators	`¬ (Logical NOT)  Unary +  Unary -`
Arithmetic Power	`**`
Arithmetic Multiply and Divide	`*  /  %  //`
Arithmetic Add and Subtract	`+  -`
Relational operators	`=  \=  >  <  >=  <=  <<` `\<<  >>  \>>`
Logical AND	`&`
Logical OR	`|`

See publication *"FileKit Data Editor (SDE)"* for a detailed description of expressions including operator descriptions and available functions.

SQL-search-condition

Applicable only to DB2 table input, *SQL-search-condition* is a standard DB2 SQL search condition. The unaltered *SQL-search-condition* string is included on the WHERE clause of an SQL query generated by FileKit to create the input result table.

See IBM publication *"DB2 for z/OS SQL Reference"* for details on *SQL-search-condition* syntax and the SQL Query WHERE clause.

Only DB2 table rows that satisfy the search condition will be passed as input to the REPORT utility.

FOOT

Overview:

The FOOT section applies only to PRINT type output and specifies the contents of one or more lines to be output at the bottom of each page. The FOOT section is ignored for non-PRINT report output.

By default, no footing lines are produced for printed reports and so column detail records and break output records may occupy the last lines of the printed page. If footing lines are defined, a number of lines equal to the number of footing lines plus 1 is reserved at the bottom of each page. (1 blank line is always printed before the first footing line.)

A footing line definition may be split into 1, 2 or 3 partitions using the slash ("/" character). Each partition is represented by a *print-expression* comprised of one or more text fragment definitions and optional gap values. Each fragment of text may be a character string literal or a value obtained from a field in the input record, a computed field (REXX variable name) or a built-in report field.

Depending on the number of partitions, each partition is either left adjusted, centralised or right adjusted within the width of the page. Note that, before a printed report is generated, the page width is set so that it is the maximum length of all the report page heading, page footing, break and column detail lines. Therefore, the text in adjusted footing partitions will not overlap and the REPORT processing ensures that at least 2 blanks separate each partition.

Each footing line definition occupies a single statement of the REPORT FOOT section. Use of the statement continuation character, backslash ("\"), may be necessary in order to stream a single footing line definition over more than one REPORT definition input record.

A new footing line definition is started for each new statement in the FOOT section. Footing line definitions end at the start of the next REPORT section or at the end of the REPORT definition input, whichever is encoutered first.

Examples:

Example 1 - Single Footing Line:

```
FOOT:
  #TIMESTMP                                                    \
  / "=== End of Report (Size:"  #SEQUENCE (RIGHT,5)  0  ") ==="   \
  / "Tel:" PhoneNum
```

The above example will output 1 footing line and a preceding blank line on each page of a printed report.

The footing definition statement contains 3 partitions. The statement continuation character is used to specify the second and third partition print definitions on separate REPORT definition records.

The first partition contains 1 text fragment, the built-in field **#TIMESTMP**, which gets substituted with the current date and time.

The second partition contains 3 text fragments, a character string literal followed by the *built-in-field* (**#SEQUENCE**) and a second string litral. **#SEQUENCE** is substituted with the current number of report detail lines, a value which is right adjusted in an area of 5 characters. Note that a gap of zero (0) overrides the default of 1 blank character so that the second string literal immediately follows the sequence number value.

The third partition contains 2 text fragments, a character string literal followed by an *input-field* (**PhoneNum**). This *input-field* will be substituted with the value of **PhoneNum** obtained from the last record reported on the page.

Text belonging to the first partition will be left adjusted, text belonging to the second partition will be centralised and text belonging to the third partition will be right adjusted within the page width.

Example 2 - Multiple Footing Lines:

```
FOOT:
  "TCP/IP Connection Report:"   zConnectStart
  / "PAGE" #PAGE (8)
```

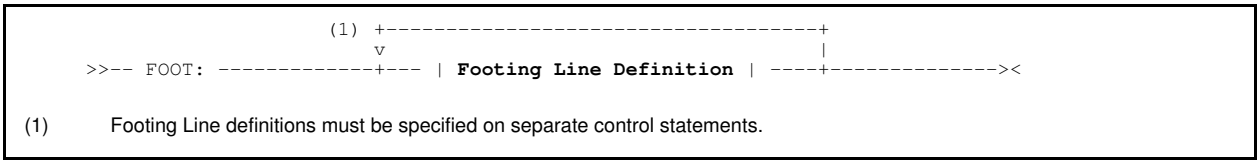
The above example will output 2 footing lines and a preceding blank line on each page of a printed report.

The first footing definition contains only 1 partition with 2 text fragments, a string literal followed by an *input-field* (**zConnectStart**). This *input-field* will be substituted with the value of **zConnectStart** obtained from the last record reported on the page. Because the first line contains only one partition, the partition text will be centralised in the page width.

The second footing definition contains 2 partitions. The first has no text fragments and the second 2 text fragments, a string literal followed by the *built-in-field* (**#PAGE**). The (null) text belonging to the first partition will be left adjusted and the text belonging to the second partition will be right adjusted within the page width.

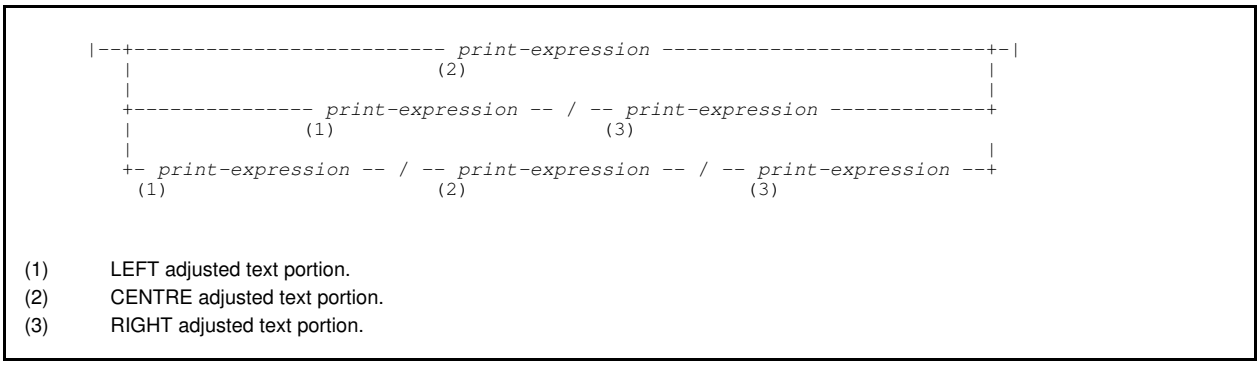
#PAGE, will be substituted with the current page number and will occupy 8 characters. Because the page number value is of numeric data type, the value is automatically right adjusted within the 8 characters it occupies, with non-significant leading zeros replaced with blank characters.

Syntax:



(1) Footing Line definitions must be specified on separate control statements.

Footing Line Definition:



(1) LEFT adjusted text portion.
(2) CENTRE adjusted text portion.
(3) RIGHT adjusted text portion.

Parameters:

print-expression

A *print-expression* defines a portion of text in the footing line. The format of a *print-expression* is described under "Print Expression".

A footing line may be partitioned so that all text belonging to each partition is either left, centre or right aligned within the derived page width. Up to 3 partitions may be defined where each partition is represented by a *print-expression*.

If more than 1 *print-expression* is specified (to define multiple partitions), then each *print-expression* must be separated from the next using a slash ("/") character. Note that a *print-expression* may be null and so "/" may be specified as the first and/or last character of the footing line definition.

Alignment of partitions is based on the number of partitions defined.

1. Partition text will be **left** aligned if its *print-expression* is the first of 2 or 3 partition definitions.
2. Partition text will be **centre** aligned if its *print-expression* is the second of 3 partition definitions or is the only *print-expression* specified.
3. Partition text will be **right** aligned if its *print-expression* is the last of 2 or 3 partition definitions.

HEAD

Overview:

The HEAD section applies only to PRINT type output and specifies the contents of one or more lines to be output at the start of each page. The HEAD section is ignored for non-PRINT report output.

A number of lines equal to the number of heading lines plus 1 is reserved at the top of each page (1 blank line is always printed after the last heading line.)

By default, if no HEAD section exists, then a single heading line containing the timestamp at which the report was generated and the page number will be generated. The format of this heading line is:

```
#TIMESTMP / 'PAGE' #PAGE (4)
```

A heading line definition may be split into 1, 2 or 3 partitions using the slash ("/" character). Each partition is represented by a *print-expression* comprised of one or more text fragment definitions and optional gap values. Each fragment of text may be a character string literal or a value obtained from a field in the input record, a computed field (REXX variable name) or a built-in report field.

Depending on the number of partitions, each partition is either left adjusted, centralised or right adjusted within the width of the page. Note that, before a printed report is generated, the page width is set so that it is the maximum length of all the report page heading, page footing, break and column detail lines. Therefore, the text in adjusted heading partitions will not overlap and the REPORT processing ensures that at least 2 blanks separate each partition.

Each heading line definition occupies a single statement of the REPORT HEAD section. Use of the statement continuation character, backslash ("\"), may be necessary in order to stream a single heading line definition over more than one REPORT definition input record.

A new heading line definition is started for each new statement in the HEAD section. Heading line definitions end at the start of the next REPORT section or at the end of the REPORT definition input, whichever is encoutered first.

Examples:

Example 1 - Single Heading Line:

```
HEAD:
   "Mike's Music Collection:"  #DAYNAME #TODAY
```

The above example will output 1 heading line followed by a blank line at the top of each page in a printed report.

The heading definition statement contains 1 partition comprised of 3 fragments, a character string literal followed by 2 *built-in-field* elements (**#DAYNAME** and **#TODAY**). #DAYNAME is substituted with the current day name (e.g. Wednesday) and #TODAY is substituted with the current date in the format "yyyy/mm/dd".

The partition text is centrailsed within the page width.

Example 2 - Multiple Heading Lines:

```
HEAD:
  #TIMESTMP  /  "PAGE" #PAGE (6)
  "TCP/IP Connection Durations by Resource Name on:" zTME
  "Resource:" zRNAME  10 "First Connection:" zConnectStart  \
                                        10 "Duration:" :Duration (8)
```

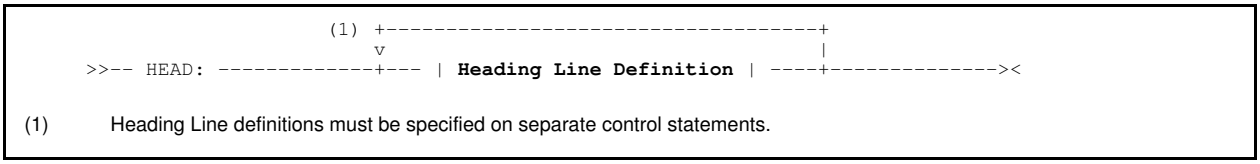
The above example will output 3 heading lines followed by a blank line at the top of each page in a printed report.

The first heading definition contains 2 partitions. The first partition has 1 text fragment, the *built-in-field* (**#TIMESTMP**) and the second partition has 2 text fragments, a string literal followed by the *built-in-field* (**#PAGE**). The text belonging to the first partition will be left adjusted and the text belonging to the second partition will be right adjusted within the page width.

The second heading definition contains only 1 partition with 2 text fragments, a string literal followed by an *input-field* (**zTME**). This *input-field* will be substituted with the value of **zTME** obtained from the first detail record to be reported on the page. Because the line contains only one partition, the partition text will be centralised within the page width.

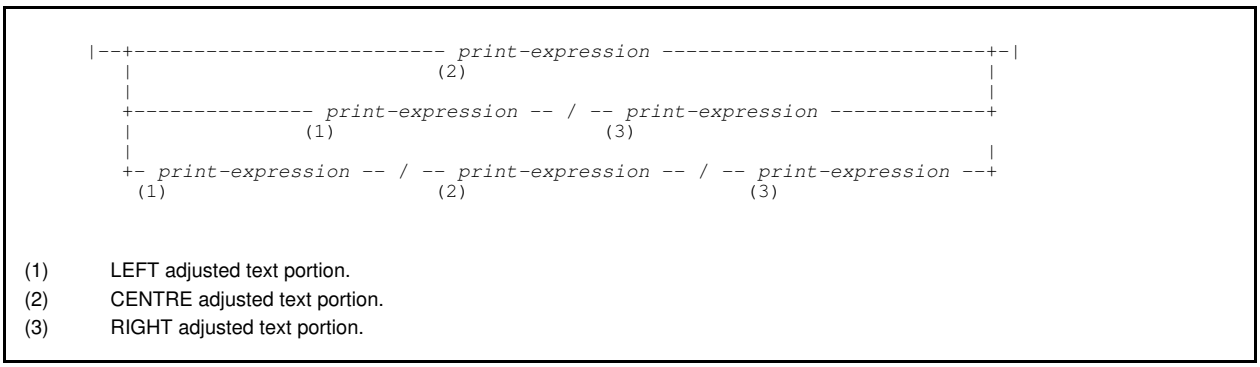
The third heading definition statement also contains only 1 partition and the statement continuation character is used to continue the heading definition statement onto a second REPORT definition record. The partition has 6 text fragments and 2 gap values. The *input-field* elements (**zRNAME** and **zConnectStart**) will be substituted with values obtained from the first detail record to be reported on the page. The *compute-field* (**Duration**) will be substituted with a value of length 8 which has been calculated based on input field values obtained from the first detail record reported on the page. The gap values (both 10) insert a gap of 10 blanks between the field value before and the next string literal element that follows.

Syntax:



(1) Heading Line definitions must be specified on separate control statements.

Heading Line Definition:



(1) LEFT adjusted text portion.
(2) CENTRE adjusted text portion.
(3) RIGHT adjusted text portion.

Parameters:

print-expression

A *print-expression* defines a portion of text in the heading line. The format of a *print-expression* is described under "Print Expression".

A heading line may be partitioned so that all text belonging to each partition is either left, centre or right aligned within the derived page width. Up to 3 partitions may be defined where each partition is represented by a *print-expression*.

If more than 1 *print-expression* is specified (to define multiple partitions), then each *print-expression* must be separated from the next using a slash ("/") character. Note that a *print-expression* may be null and so "/" may be specified as the first and/or last character of the heading line definition.

Alignment of partitions is based on the number of partitions defined.

1. Partition text will be **left** aligned if its *print-expression* is the first of 2 or 3 partition definitions.
2. Partition text will be **centre** aligned if its *print-expression* is the second of 3 partition definitions or is the only *print-expression* specified.
3. Partition text will be **right** aligned if its *print-expression* is the last of 2 or 3 partition definitions.

INIT-EXIT

Overview:

The INIT-EXIT section identifies the start of REXX program statements that are to be executed once only, immediately prior to input of the first data record or DB2 table row.

The exit allows for initialisation of REXX variable (*compute-field*) values, for reference in REXX program statments identified in the COMPUTE: section. Unlike BROWSE-EXIT, presence of INIT-EXIT in the report definition will not trigger use of the REPORT utility's Data Editor browse processing of input records, and so may not be used as a mechanism to "exclude" input records from report processing.

Examples:

Example 1 - Computed Field Initialisation:

In the following, the INIT-EXIT section is used to initialise a *compute-field* variable at the start of input record processing.

The REXX statements in the COMPUTE section are executed following each input record (or record segment). Each new "zJobName" SMF input field value is added to "JobArray" (a blank delimitted array of job names) if it does not already exist in the array.

```
INIT-EXIT:
  JobArray = ''                 /* Initialise Compute-field variables. */

COMPUTE:
  if wordpos(zJobName,JobArray) = 0     /* First occurrence of this Job Name ? */
    then  JobArray = JobArray JobName   /* Add it to array of Job Names. */
```

Syntax:

```
>>-- INIT-EXIT: ---------------- REXX Control Statements ---------------------><
```

Synonyms:

INIT-EXIT	INITEXIT

Parameters:

`REXX Control Statements`

Any number of valid REXX logical control statements may be specified to constitute an executable REXX routine. A REXX error will occur if invalid statements are entered.

The REXX routine ends at the start of the next REPORT section or at the end of the REPORT definition input, whichever is encoutered first.

INPUT

Overview:

The INPUT section specifies the source data for the generated report.

The REPORT Utility can generate output from one of a number of input data sources, specifically a data set, library member, HFS/ZFS file or DB2 result table. The syntax of the input definition depends on the data source, supporting operands that are specific to DB2, SDE or SMF format input.

The input format (DB2, SDE or SMF) is identified by one of the following:

- The REPORT option specified in the report definition.
- The FileKit REPORT Utility panel used to launch the report generation. (REPORT Utility panels are specific to DB2, SDE or SMF input.)
- Specification of format indicator operand pairs *xxx*-INPUT-BEG and *xxx*-INPUT-END (where *xxx* is DB2, SDE or SMF) on the REPORT primary command.

Error ZZS062E is returned if the REPORT option is specified with a different input format to that specified by the REPORT primary command or implied by the Utility panel from which the report is generated.

Operand values set in the INPUT section are merged with those obtained from the REPORT primary command or REPORT Utility panels. Therefore, input definition operands required for successful operation may be omitted in the INPUT section provided they are specified by these other sources. For example, when generating a report via the FileKit panels, a DB2 result table definition may be omitted from the report definition INPUT section so long as it is specified in the FileKit DB2 REPORT panel input field(s).

Note that operand values obtained from the REPORT primary command or REPORT Utility panels will override values specified on the same operand in the report definition INPUT section. Thus, REPORT input source may be temporarily changed simply be specifying a different input source as parameters on the REPORT command or Utility panels.

An input definition occupies a single statement of the REPORT INPUT section. Use of the statement continuation character, backslash ("\"), may be necessary in order to stream an input source definition over more than one report definition input record.

Examples:

Example 1 - SDE Input:

```
INPUT:
                'CBL.INST.CBL21042.SZZSSAM2(ZZSDF1DR)' \
    USING COBOL 'CBL.INST.CBL21042.SZZSSAM1(ZZSCF1DR)'
```

The above example will input data records from the **ZZSDF1DR** member of the CBL supplied sample data library, and map record fields using the COBOL copybook member **ZZSCF1DR** provided in the CBL supplied sample job library.

Example 2 - SMF Input:

```
INPUT:      DD=SMFCAT
```

The above example will input SMF data records from the DD name **SMFCAT** which may have been allocated to a single data set, library member or HFS/ZFS file path, or to a concatenation of data sets.

Example 3 - DB2 Input (Table):

```
INPUT:
    DB2(CBLA) CBL.ZZSFUNC                \
              FROM ROW 31   FOR 50 ROWS  \
         WHERE( FUNCNAME LIKE 'B%')      \
      ORDER BY( FUNCNAME DESC )
```

The above example will input 50 rows of a DB2 reults table starting at input row number 31. A connection is made to the "CBLA" DB2 sub-system before the result table is generated using an SQL SELECT Query that includes the specified table name, WHERE clause and ORDER BY clause.

Example 4 - DB2 Input (SQL):

```
INPUT:
    DB2(CBLA)                                                          \
    SQL(                                                               \
         SELECT   F.FUNCNAME,  P.PARMNO,  P.PARMNAME,  P.PARMTYPE      \
            FROM            CBL.ZZSFUNC      F                         \
                 INNER JOIN CBL.ZZSPARM      P                         \
                   ON  F.FUNCNAME=P.FUNCNAME  AND  F.APILIB=P.APILIB  \
            WHERE F.FUNCNAME = 'P2D'                                   \
         ORDER BY P.PARMNO                                             \
       )
```

The above example will input all rows of a DB2 reults table generated using a fully-formed SQL Query statemnt that involves a join of 2 tables.

Syntax:

```
>>-- INPUT: ----------------- | Input Definition | --------------------------><
```

Input Definition:

```
>>---+------ | SDE Input | -----------------+-----------------------------------><
     |                                      |
     +------ | SMF Input | -----------------+
     |                                      |
     +------ | DB2 Input | -----------------+
```

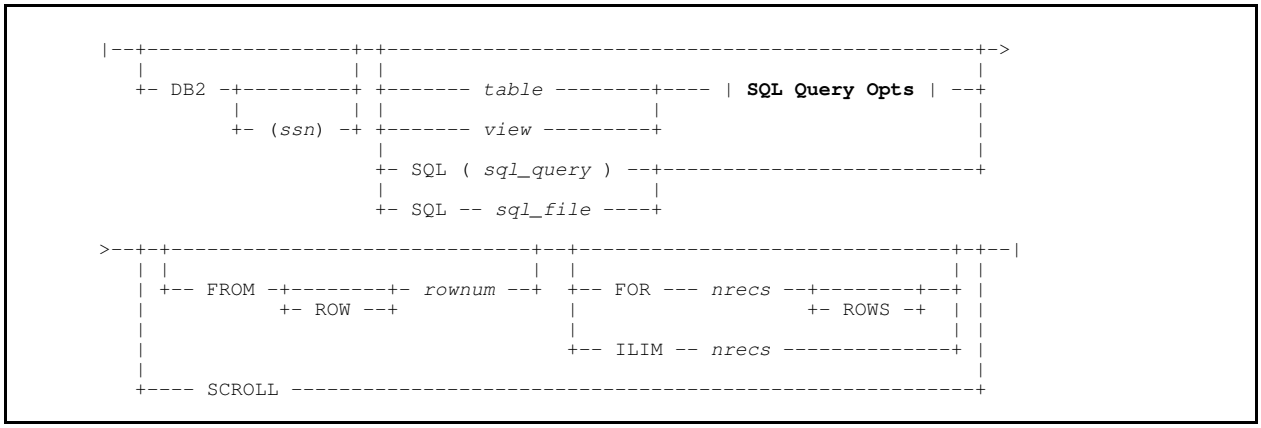
SDE Input:

```
|--+---------------+-+-------------------------------------------------------+--|
   |               | |                                                       |
   +- report_inp -+ +- USING --+-+-------------+--- sdo_name -----------+-+
   |              |            | +- STRUCTure -+                        |
   +- DD=ddin ----+            |                                        |
                               +-+- HLAsm -----+--- copybook_name -----+
                               | +- COBOL -----+                        |
                               | +- PL1 -------+                        |
                               | +- ADAta -----+                        |
                               |                +----------------+      |
                               |                v                |      |
                               +--- SYMNAMes ( --+- SYM_source --+- ) -+
```

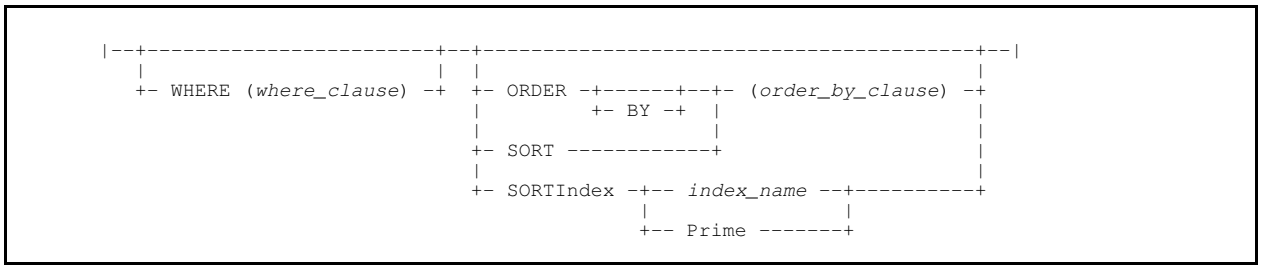
SMF Input:

```
|--+---------------+-------------------------------------------------------------|
   |               |
   +- report_inp -+
   |              |
   +- DD=ddin ----+
```

DB2 Input:



SQL Query Opts:



Parameters:

DB2 [**(***ssn***)**]
Applicable to DB2 table input only, specification of DB2 is optional and is only necessary if used to identify a specific DB2 sub-system.

The (*ssn*) operand is optional and identifies the local DB2 sub-system name to which a connection will be made. This value may be overridden by an *ssn* value specified on the REPORT primary command or via the "SSN>" input field of the FileKit DB2 REPORT Utility panel from which the report is generated.

Before a connection can be made to the DB2 sub-system, the FileKit DB2 plan must have been bound to that sub-system.

Default for *ssn* is the DB2 sub-system name set in the FileKit DB2 Primary Options menu and saved in the User INI file.

FOR *nrecs* [**ROWS**]
Applicable to DB2 table input only, FOR *nrecs* ROWS specifies the maximum number of rows that may be fetched from the DB2 result table. This value may be overridden by a FOR *nrecs* value specified on the REPORT primary command or via the "Max>" input field of the FileKit DB2 REPORT Utility panel from which the report is generated.

Note that, if an input limit (ILIM) value is specified, it will override the *nrecs* value specified on FOR *nrecs* ROWS. However, any *nrecs* value specified by ILIM or FOR will be ignored if DB2 operand SCROLL is also specified to use a DB2 table scrollable cursor.

By default, REPORT processing will include all rows of the DB2 result table.

FROM [**ROW**] *rownum*
Applicable to DB2 table input only, FROM ROW *rownum* specifies the number of the input DB2 result table row from which REPORT processing will start. This value may be overridden by a FROM *rownum* value specified on the REPORT primary command or via the "Start>" input field of the FileKit DB2 REPORT Utility panel from which the report is generated.

Rows that occur before the specified row number will be bypassed and not included in the number of rows (*nrecs*) count identified by an input limit (ILIM) or FOR *nrecs* RECS specification.

Note that any specified *rownum* value will be ignored if DB2 operand SCROLL is also specified to use a DB2 table scrollable cursor.

By default, REPORT processing starts from the first row of the result table.

{ **ORDER** [**BY**] | **SORT** } (*order_by_clause*)
Applicable to DB2 table input only where a DB2 *table* or *view* name is specified as input (as opposed to a fully formed SQL SELECT query).

ORDER BY or SORT specifies a DB2 SQL ORDER BY clause to be included in the prepared SQL SELECT query statement generated by the REPORT utility and used to obtain the DB2 result table. This clause may be overridden by a SORTINDEX value or ORDER BY clause specified on the REPORT primary command, or an ORDER BY clause constructed via the FileKit "Create DB2 SELECT/ORDER BY Clause" sub-panel opened from the DB2 REPORT Utility panel from which the report is generated.

See IBM publication "*DB2 SQL Reference*" for syntax of the *order_by_clause* which will fetch result table rows in the specified order.

If required, a SORT section may also be specified in the report definition to sort the report output record detail lines. This may be necessary if report detail lines are to be sorted based on the values of one or more *compute-field*.

Operands ORDER BY (or SORT) and SORTINDEX are mutually exclusive. If both are specified, the ORDER BY clause will be used.

report_inp | **DD=***ddin*
Applicable to SDE and SMF format input, *report_inp* or DD=*ddin* identifies the data source from which input records are obtained for report processing. This data source may be overridden by a *report_inp* or DD=*ddin* specification on the REPORT primary command, or by entering a file object name in the "DSN/Path>" input field of the FileKit REPORT Utility panel from which the report is generated.

A *report_inp* value may be quoted or unquoted and is either a sequential dataset name, library dataset and member name or a HFS/ZFS file path. A *ddin* value is an allocated DD name which must be prefixed with "**DD=**".

If DD=*ddin* is specified, *ddin* may be allocated to any data source represented by *report_inp* including a DASD or TAPE dataset. Alternatively, it may be allocated to a concatenation of data sets, thus allowing records to be processed from multiple, consecutive input sources. e.g. Multiple generations of the same GDG.

If DD=*ddin* is specified, *ddin* may be allocated to any data source represented by *report_inp* including a DASD or TAPE dataset. Alternatively, it may be allocated to a concatenation of data sets, thus allowing records to be processed from multiple, consecutive input sources. e.g. Multiple generations of the same GDG.

Unless included via the REPORT command or FileKit REPORT Utility panel, specification of an input data source (*report_inp* or DD=*ddin*) is **mandatory**.

SCROLL
Applicable only to DB2 table input, SCROLL indicates that a DB2 scrollable INSENSITIVE cursor is to be used to fetch DB2 rows.

If SCROLL is used, then once the cursor has been opened, only a relatively small number of rows will be kept in storage at any time. At open the results table is materialised (i.e. a temporary copy is made) which, for large tables, may mean that opening the cursor may take a long time and consume much resource.

Use of DB2 scrollable cursors may not be desirable and so is possible only if the DB2 administrator has set **DB2.SCROLL=YES** in the FileKit Site INI file.

SCROLL is incompatible with FOR ROWS, input limit (ILIM) and FROM ROWS values specified via the INPUT and OPTIONS sections, or passed as parameters by the FileKit REPORT panels or REPORT command. If specified, values provided for these operands will be ignored if SCROLL is also used.

SORTINDEX { *index_name* | **PRIME** }
Applicable to DB2 table input only where a DB2 *table* or *view* name is specified as input (as opposed to a fully formed SQL SELECT query).

SORTINDEX specifies *index_name*, the name of an existing DB2 Index for the table being processed. The index identifies the key columns/expressions by which the table rows will be ordered for input to the REPORT utility. Alternatively, PRIME may be specified to indicate that the primary index should be used. This SORTINDEX value may be overridden by a SORTINDEX value or ORDER BY clause specified on the REPORT primary command, or an ORDER BY clause constructed via the FileKit "Create DB2 SELECT/ORDER BY Clause" sub-panel opened from the DB2 REPORT Utility panel from which the report is generated.

If required, a SORT section may also be specified in the report definition to sort the report output record detail lines. This may be necessary if report detail lines are to be sorted based on the values of one or more *compute-field*.

Operands ORDER BY (or SORT) and SORTINDEX are mutually exclusive. If both are specified, the ORDER BY clause will be used.

table | *view* | **SQL(***sql_query***)** | **SQL** *sql_file*
Applicable to DB2 table input only, each of these operands define the source of a DB2 result table from which input rows are obtained for report processing.

This result table source definition may be overridden by a *table*, *view*, SQL(*sql_query*) or SQL *sql_file* specification on the REPORT primary command. Alternatively, it may be overridden by entering a DB2 result table source in the DB2 Table/View "Name>" input field, Input SQL File "DSN/Path>" input field, or "Statement>" input field of the

particular FileKit DB2 REPORT Utility panel from which the report is generated.

The following operands are mutually exclusive and may be used to specify the DB2 result table source:

table	The name of a DB2 table or alias (*table*) as defined in the SYSIBM.SYSTABLES catalog table.
view	The name of a DB2 view (*view*) as defined in the SYSIBM.SYSVIEWS catalog table.
SQL(*sql_query***)**	Specifies *sql_query*, a complete DB2 SQL query that generates a result table. For example, the SQL query may include clauses that select specific columns, join tables, filter and order the table rows.
SQL *sql_file*	Specifies *sql_file*, a sequential DSN or library DSN and member name in which a DB2 SQL query is saved. For example, this may be a library member containing a SQL query used as input to SPUFI or the FileKit EXECSQL utility.

Both *table* and *view* may be specified with either 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default for *location* is the local DB2 server and the default for *schema* is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID). Note that the user's SQLID is set via the FileKit DB2 Primary Options menu and saved in the User INI file.

If *table* or *view* is used, then FileKit will generate an SQL query clause (e.g. "SELECT * FROM *table*").

If one of the *SQL* type operands is used and the report definition input includes a FILTER section, then the filter clause will be ignored and warning message ZZSR064W returned. This is because the FILTER section will attempt to generate a WHERE clause to add to the SQL Query. However, the SQL Query passed to the REPORT utility is already fully formed.

The SQL query specified by the **SQL** operand or generated by FileKit is executed as a prepared DB2 SQL statement and the result table rows passed to the REPORT utility.

Unless a DB2 result table source definition is to be provided via the REPORT command or FileKit DB2 REPORT Utility panels, then specification of a DB2 result table source in the INPUT section is **mandatory**.

USING

For SDE format input only, USING identifies the record formatting structure definition used to map input record fields. Providing a record formatting structure will override use of record field mappings defined by a MAP section.

This structure definition may be overridden by a structure definition specified via the REPORT primary command, or by entering a file object name and type in the "Structure/Copybook overlay:" input fields of the FileKit REPORT Utility panel from which the report is generated.

The structure ultimately used by the REPORT utility to map the input records will be a FileKit structured data object (SDO). The name of an existing SDO may be passed directly to the REPORT utility or it may be generated automatically from an alternative source (e.g. a COBOL copybook). The following operands are mutually exclusive and may be used to specify the structure definition:

[**STRUCTURE**] *sdo_name*

Specifies the name of an existing FileKit (SDO) structure file *sdo_name*, which has been generated via the FileKit Create Structure panels or CREATE STRUCTURE primary command. The *sdo_name* may be quoted or unquoted, and is the name of a sequential data set or a library DSN and member name.

{ **HLASM** | **COBOL** | **PL1** | **ADATA** } *copybook_name*

Specifies the format and name of the input record mapping source file (*copybook_name*). The *copybook_name* may be quoted or unquoted and is a library DSN and member name. The format of the copy book source may be one of the following:

ADATA	The SYSADATA output generated by the assembley of an assembler source using the HLASM (High Level Assembler) program, or generated by the compilation of a COBOL or PL1 source using the Enterprise COBOL or Enterprise PL1 compiler respectively.
COBOL	A copybook member containing COBOL data division - data description source.
HLASM	An Assembler source member containing DSECT definition(s).
PL1	An %INCLUDE directive source member containing PL1 data declaration structure(s).

FileKit will interpret the record mapping source to generate a temporary SDO structure.

SYMNAMES(*SYM_source ...* **)**

Specifies one or more *SYM_source* entries, where *SYM_source* is the name of a sequential data set or library DSN and member name containing SYMNAMES symbol statements as supported by the SORT utility.

Symbol statements must include field definitions specified as position, length and format. Please refer to your SORT utility documentation (e.g. the IBM publication *"z/OS DFSORT Application Programming*

Guide") for details on the symbol statement.

FileKit will interpret the SYMNAMES field definitions to generate a temporary SDO structure containing a single record mapping (record-type).

Unless a MAP section exists or a structure definition is included via the REPORT command or FileKit REPORT Utility panel, then specification of USING and a record formatting structure definition is **mandatory**.

WHERE (*where_clause***)**

Applicable to DB2 table input only where a DB2 *table* or *view* name is specified as input (as opposed to a fully formed SQL SELECT query).

WHERE specifies a DB2 SQL WHERE clause to be included in the prepared SQL SELECT query statement generated by the REPORT utility and used to obtain the DB2 result table. This clause may be overridden by a WHERE clause specified on the REPORT primary command, or a WHERE clause constructed via the FileKit "Row Selection by Column Value" sub-panel opened from the DB2 REPORT Utility panel from which the report is generated.

A DB2 where clause may also be provided via the FILTER section of the report definition. If both a FILTER section and a WHERE specification exists, then the contents of the FILTER section are ignored and warning message ERR064W is returned.

See IBM publication "*DB2 SQL Reference*" for syntax of the *where_clause* which will filter and include only table rows that match the where clause criteria.

MAP

Overview:

Applicable to SDE input only, the MAP section is used to identify individual fields within the input records when no structure/copybook has been provided via the INPUT section or REPORT utility parameters.

Having been defined in the MAP section, a field definition name may be used in any other section that support an *input-field* reference.

For anything but very simple record field maps that apply to all the input records, it is recommended that an existing structure is provided via a copybook or a FileKit generated structure object (SDO).

The field mapping may be specified using one of the following four mutually exclusive methods:

SYMNAME Definitions

The syntax of a SYMNAME definition resembles that used for SORT utility SYMNAMES input.

Each SYMNAME definition specifies a field name, a fixed position within the record, a length and a data type format. Note that a field's start position is a byte number in the record data which, for RECFM=V variable length input records, does **not** include the length of the 4-byte RDW. If required, SYMNAME definitions permit overlapping of fields.

Unlike standard SYMNAME field definitions, use of the equals symbol ("=") to represent a value specified in the previous field definition, is not permitted in MAP field definitions. Otherwise, MAP field definitions support most of the commonly used data formats supported by SYMNAME field definitions plus some extra formats not supported by SYMNAME.

Please refer to your SORT utility documentation (e.g. the IBM publication *"z/OS DFSORT Application Programming Guide"*) for details on the SYMNAMES symbol statement.

Contiguous Field Definitions

The syntax of a field definition matches that used for record-type field definitions specified by the CREATE STRUCTURE primary command.

The first field definition defines a field starting at position 1 of the record data (excluding any RDW). Unless defining a UNION of fields which redifnes fields in the same area of record data, the position of each successive field follows the last byte of the field defined immediateley before.

Field definitions support more source data type formats than SYMNAME definitions.

Non-contiguous Field Definitions

Syntax identical to that of a contiguous field definition except that an OFFSET value is supplied so that each field starts at a fixed offset within the record data.

Like SYMNAME definitions, fields may overlap and need not be defined in the order the occur within the record. However, since field offsets are fixed, this method may not be used if the field location varies (e.g. if it follows a variable length field or a variable number of array field elements).

If a non-contiguous field definition is used, then **all** field definitions must include an OFFSET specification.

CREATE STRUCTURE Primary Command

A fully-formed FileKit **CREATE STRUCTURE** primary command used to generate a FileKit SDO structure.

Defining fields using this method has the advantage that a structure may be created which contains more than one record mapping for different format input records, and potentially created using multiple copybook/structure sources. The record mapping used to map a input record is based on values in the record itself (identified via a USE WHEN clause).

Beware that, unless operand TEMPORARY is specified, the CREATE STRUCTURE operation will attempt to create a permanent copy of the structure. The SDO structure will be written to a DASD data set, library member or HFS/ZFS fileid based on the specified structure name. Similarly, once a permanent or temporary structure has been created, the REPLACE operand must be used to prevent error ZZSD017E. This would occur on subsequent executions of the report generation that contains the CREATE STRUCTURE command.

Refer to primary command CREATE STRUCTURE in the *"FileKit Data Editor"* reference manual for syntax and description of command operands.

All input following the MAP section header is joined to become a single REPORT definition control statement. Therefore, the specified definitions or CREATE STRUCTURE command may stream over several lines without having to specify a line continuation symbol ("\") at the end of each line.

If a MAP section exists and the REPORT utility is passed a structure reference (via the INPUT section, REPORT utility panel or primary command), then the MAP section will be ignored. Specification of a MAP section is invalid for SMF record or DB2 table input and will return error ZZSR049E.

Examples:

Example 1 - SYMNAME Definitions:

In the following example, the MAP section contains SYMNAME definitions used to define 6 fields that map the input record data. These field names may be referenced in other REPORT sections.
Asterisk ("*") is used in the field definitions to indicate a field position that immediately follows the field define before it (i.e. field "**Region**" has start position 71, following field "**Company**").

```
MAP:
   RefId,1,4,BI
   Company,11,60,CH
   Region,*,10,CH
   Zip_Code,121,8,CH
   Emp_Total,*,4,BI
   Web_URL,*,80,CZ
```

Example 2 - Contiguous Field Definitions:

In the following example, the MAP section contains contiguous field definitions used to define fields that map the input record data. These field names may be referenced in other REPORT sections.

Unnamed filler fields are defined in order to map fields that follow later in the record data. The filler fields map data fields that are not referenced by this report definition. A "UserName" structure field is included which comprises 2 fields, "LastName" and "FirstName". Note that, if field name "LastName" was not unique within the MAP structure, then the field would have to be referenced as "UserName.LastName" throughout the report definition.

```
MAP:
   RefID        int(2)  unsigned  remark "Unique Reference Number"
  ,            char(17)           remark "Filler"
  ,Host        char(12)           remark "Host Name"
  ,            char(6)            remark "Filler"
  ,UserName  struct(
                     LastName   char(30)
                    ,FirstName  char(30)
                   )
  ,            char(32)           remark "Filler"
  ,IPv4          ip(4)            remark "Host IP Address"
```

Example 3 - Non-Contiguous Field Definitions:

The following is the same as the above example except that field offset values are used. Because the fields need not follow on from each other, there is no need to specify filler fields. However, the field offsets need to be calculated beforehand.

```
MAP:
   RefID        int(2)  unsigned  remark "Unique Reference Number"  offset(0)
  ,Host        char(12)           remark "Host Name"                offset(18)
  ,UserName  struct(
                     LastName   char(30)                           offset(36)
                    ,FirstName  char(30)                           offset(66)
                   )                                               offset(36)
  ,IPv4          ip(4)            remark "Host IP Address"         offset(128)
```

Example 4 - CREATE STRUCTURE command:

The following example uses the CREATE STRUCTURE command to generate a temporary SDO structure containing 2 record mappings based on 01 level group definitions "ContactType1" and "ContactType2" found in copy book source members "XMCNTX1" and "XMZC21" respectively. Both members exist in the library "NBJ.COBOL.COPY".

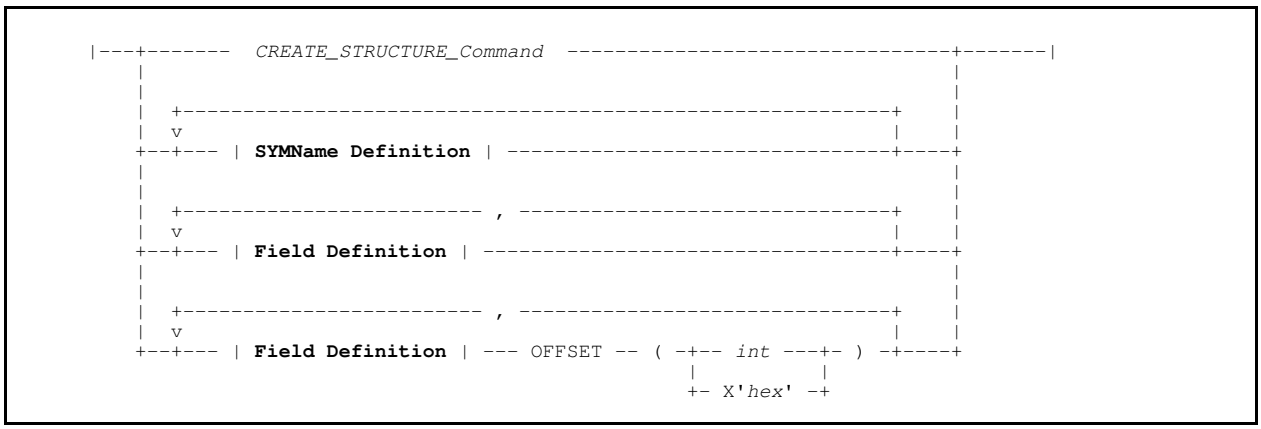
Input records are mapped by fields in record mapping "ContactType2" if the "CT" field within the record data has a value of "2", otherwise they are mapped by record mapping "ContactType1" by default.

Any field name defined in either of the record mapping structures can be referenced within the report definition.

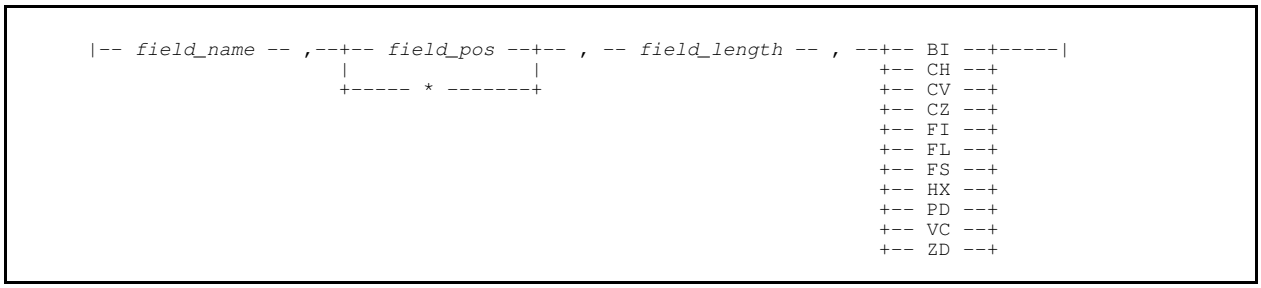
```
MAP:
   CREATE STRUCTURE   TEMPSTRUCT
        LIBRARY (NBJ.COBOL.COPY)
        RECORD  (NAME          ContactType1
                 SOURCE COBOL  XMCNTX1
                 DEFAULT
                )
        RECORD  (NAME          ContactType2
                 SOURCE COBOL  XMZC21
                 PRIMARY
                 USE WHEN  CT='2'
                )
        TEMPORARY REPLACE
```

Syntax:

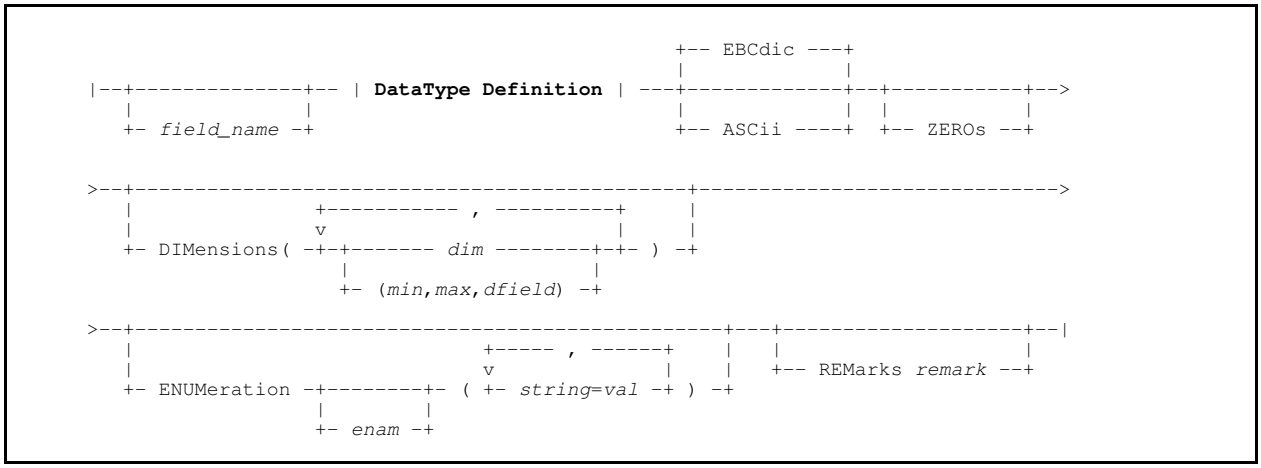
Field Definition Statement:



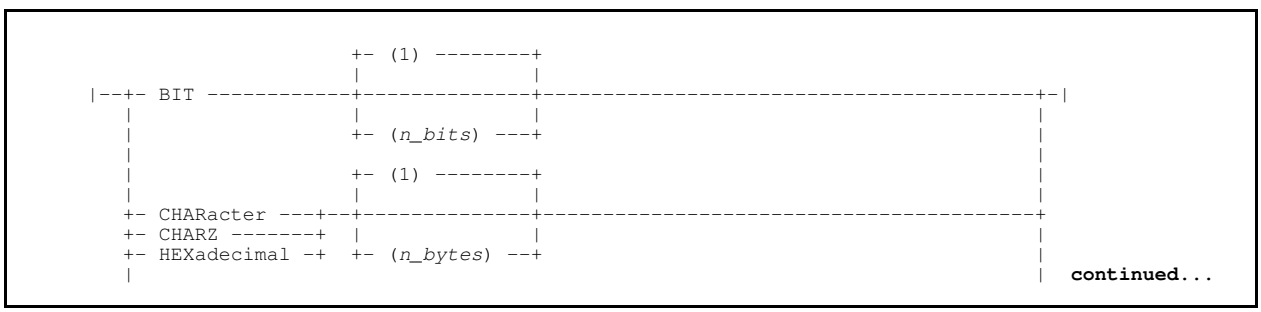
SYMNAME Definition:



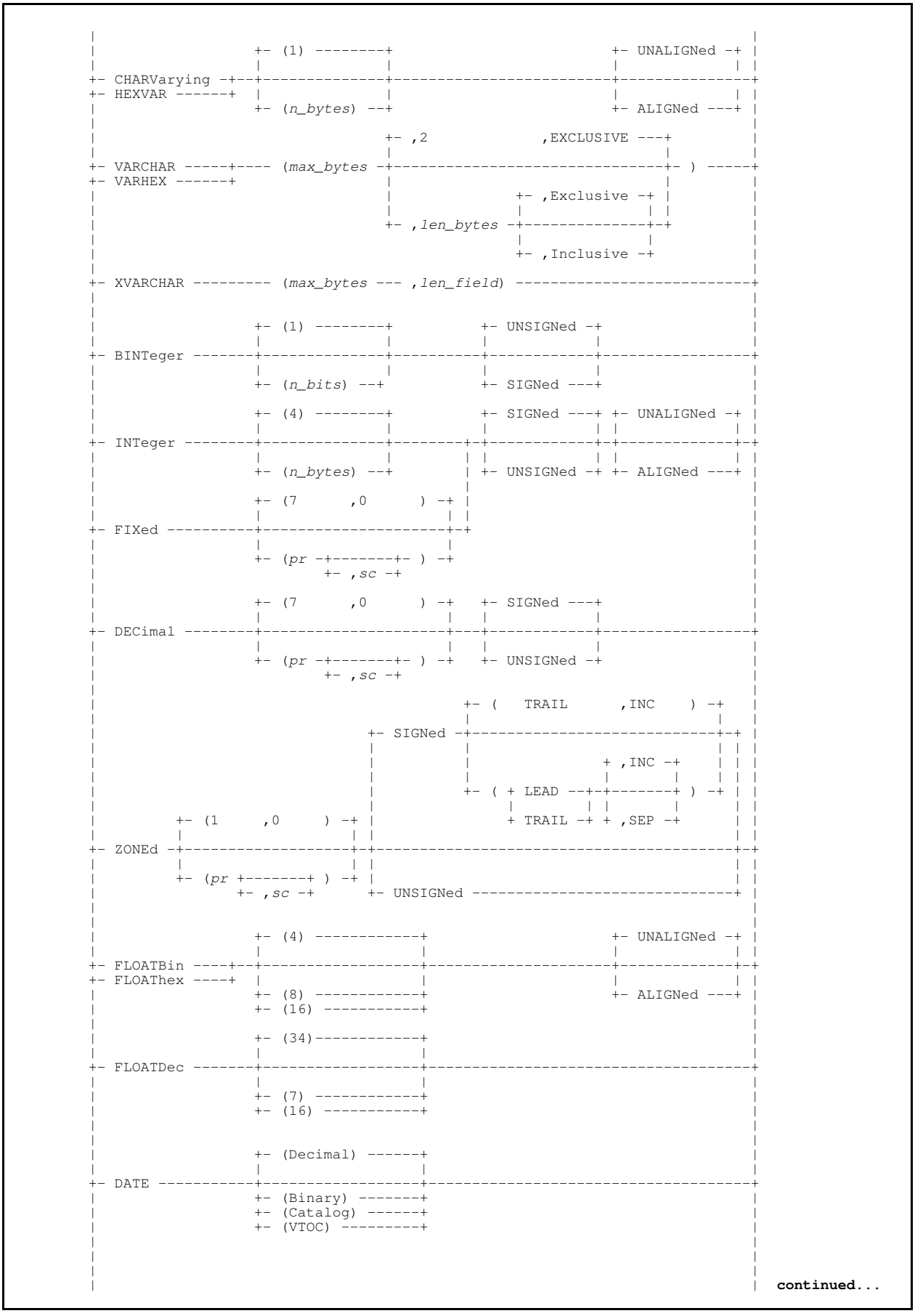
Field Definition:



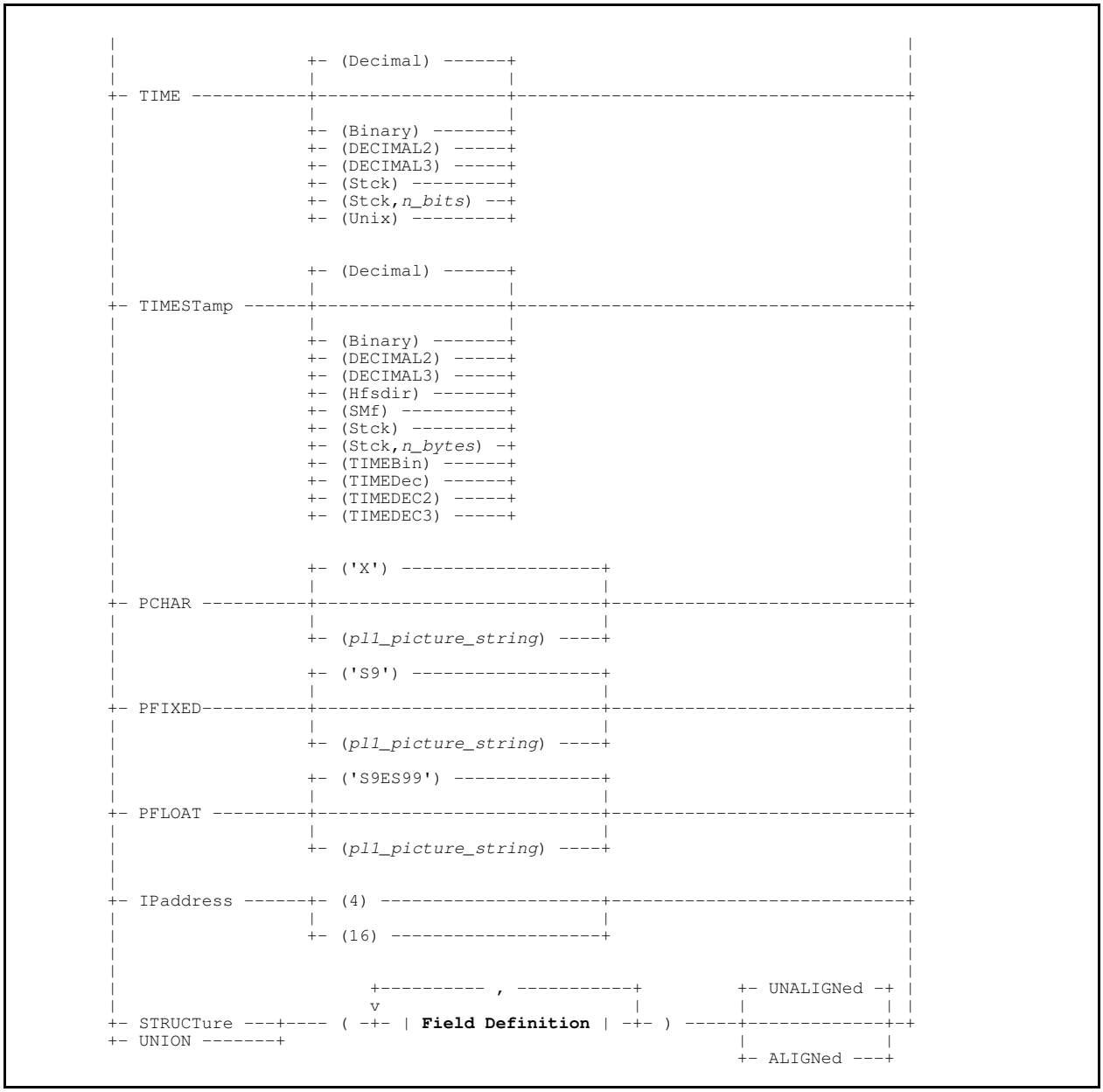
DataType Definition:



DataType Definition (Continued):



DataType Definition (Continued):



Parameters:

field_name
The name of the field to be assigned to the field in the input record data.

The *field_name* may be referenced as a *input-field* in any of the other REPORT defintion sections.

For Field Definition syntax, the *field_name* may be omitted. For example, when defining a filler field that is not referenced in the REPORT, but is necessary to pad to the next required input field position.

field_pos | *****
For SYMNAME Definition syntax, *field_pos* or "*" specifies the position number of the field within the input record data.

If asterisk ("*") is used, then the position will be the position of the character following the previous field definition. If no previous field definition exists, then position 1 is used.

If required, *field_pos* may be a position within an already defined field.

field_length
For SYMNAME Definition syntax, *field_length* specifies the length of the field within the input record data.

BI | **CH** | **CV** | **CZ** | **FI** | **FL** | **FS** | **HX** | **PD** | **VC** | **ZD**
For SYMNAME Definition syntax, this operand specifies the format (data type) of the field data as follows:

Format	Description
BI	Binary Integer (unsigned)
CH	Character of length *field_length*
CV	Character Varying with 2-byte length prefix so that the field has a fixed length of *field_length*+2
CZ	Zero (x'00') delimitted Character of maximum length *field_length*
FI	Fixed-Point Integer (signed)
FL	Hexadecimal Floating-Point (signed)
FS	Signed numeric with optional leading floating sign
HX	Character of length *field_length* displayed as hexadeimal
PD	Packed Decimal (signed)
VC	Variable Character with 2-byte length prefix so that the field has a variable length up to a maximum of *field_length*+2.
ZD	Zoned Decimal (signed)

CREATE_STRUCTURE_Command
A fully-formed FileKit CREATE STRUCTURE primary command to be used to create a FileKit SDO structure comprising one or more record mappings structures.

Refer to primary command CREATE STRUCTURE in the *"FileKit Data Editor"* reference manual for syntax and description of command operands.

DataType Definition
For Field Definition syntax, the datatype definition specifies both the length and format of data in the field. The data type will determine how the data will be interpreted by the REPORT utility.

As detailed below, a wider range of data types is supported for Field Definition syntax than for SYMNAME Definition syntax.

BINTEGER (*n_bits***)** [**SIGNED** | **UNSIGNED**]
A signed or unsigned binary integer value occupying a number of bits (*n_bits*). The binary value is converted to decimal for display and reporting.

A BINTEGER field is always bit aligned so that it begins at the bit immediately following the field defined before it. If a BINTEGER field is defined as SIGNED and *n_bits* is greater than 1, then negative values are represented as the two's complement of the equivalent positive value. The binary data is big-endian so that the sign is determined by the first (leftmost) bit value (0=positive, 1=negative). If *n_bits* is 1, then the field will always be treated as UNSIGNED.

By default the field is UNSIGNED and the default value for *n_bits* is 1.

BIT (*n_bits***)**
A bit value is interpreted and displayed as a binary string of "1" and "0" values occupying a number of bits (*n_bits*). The default value for *n_bits* is 1.

A BINTEGER field is always bit aligned so that it begins at the bit immediately following the field defined before it.

CHARACTER (*n_bytes***)**
A fixed-length character string occupying a number of bytes (*n_bytes*). The default value for *n_bytes* is 1.

A CHARACTER field is always byte aligned so that it begins at the byte immediately following the field defined before it.

CHARVARYING (*n_bytes***)**
A variable-length character string of maximum length (*n_bytes*). The character string is padded with blanks to occupy the fixed number of bytes (*n_bytes*). The default value for *n_bytes* is 1.

A CHARVARYING field is equivalent to a PL/1 field which is declared as being CHARACTER VARYING. An INTEGER field occupies the first 2 bytes and the character data the remaining *n_bytes*, thus the overall length of a CHARVARYING field is *n_bytes*+2. The actual length of the variable length character string within the fixed length area is held in the 2-byte INTEGER field prefix.

A CHARVARYING field may be UNALIGNED so that it begins at the byte immediately following the field defined before it, or ALIGNED. If ALIGNED is specified, the field will be aligned on a halfword boundary. The area of record between the last byte of the previous field and the first byte of the aligned CHARVARYING field contains unreferenced slack bytes. Default is UNALIGNED.

CHARZ (*n_bytes***)**
A variable-length character string of maximum length (*n_bytes*). The character string is terminated with a null character (x'00') and padded with blanks to occupy a fixed number of bytes (*n_bytes*+1), thus the overall length of a CHARZ field is *n_bytes*+1. The default value for *n_bytes* is 1.

A CHARZ field is equivalent to a PL/1 field which is declared as being CHARACTER VARYINGZ.

A CHAZ field is always byte aligned so that it begins at the byte immediately following the field defined before it.

DATE (BINARY | CATALOG | DECIMAL | VTOC)
A date field with the source data in one of the following formats:

BINARY	A BINARY date field has a 6-byte length and is in the format X'*yyyy*,*mmmm*,*dddd*'. The *yyyy*, *mmmm* and *dddd* are each 2-byte INTEGER field values representing year number, month of year number and day of month number respectively. For example, a DATE(BINARY) field containing X'07DB,000A,0014' corresponds to 20th October 2011.
CATALOG	A CATALOG date field corresponds to the format of date fields commonly used in z/OS ICF Catalog data sets. This date field has a 4-byte length and is in the format X'*yydd*,*dFcc*'. The *yy* is a 1-byte DECIMAL field with no sign indicator containing a year of century number, *dddF* is a 2-byte DECIMAL field representing day of year number and *cc* is the century indicator, a 1-byte INTEGER field with a value of either "0" or "1". A century indicator value of "0" corresponds to "19" and a value of 1 corresponds to "20". For example, a DATE(CATALOG) field containing X'0819,7F01' corresponds to 16th July 2008.
DECIMAL	A DECIMAL date field has a 4-byte length and is in the format X'*ccyy*,*dddF*'. The *yy* is a 1-byte DECIMAL field with no sign indicator containing a year of century number, *dddF* is a 2-byte positive DECIMAL field representing day of year number and *cc* is the century indicator, a 1-byte INTEGER field with a value of either "0" or "1". A century indicator value of "0" corresponds to "19" and a value of 1 corresponds to "20". For example, a DATE(DECIMAL) field containing X'0111,045F' corresponds to 14th February 2011.
VTOC	A VTOC date field corresponds to the format of date fields commonly used in a DASD Volume Table of Contents (VTOC). This date field has a 3-byte length and is in the format X'*yydddd*'. The *yy* is a 1-byte INTEGER field representing the number of years since 1900 and *dddd* is a 2-byte INTEGER field representing the day of year number. For example, a DATE(VTOC) field containing X'6900E7' corresponds to 19th August 2005.

The default date format is DECIMAL.

DECIMAL (*pr*[,*sc*]**)** [**SIGNED** | **UNSIGNED**]
A signed or unsigned packed decimal value occupying a number of bytes based on the precision (*pr*) value. The number of bytes is equal to the integer part of the result obtained from (*pr*+2)/2.

The precision (*pr*) value is the total number of decimal digits represented by the field which can range from 1 to 31. The scale (*sc*) value is the number of fractional digits (digits following the decimal point) which can range from 0 to the precision value.

A DECIMAL field is always byte aligned so that it begins at the byte immediately following the field defined before it.

The last 4 bits of the (low order) byte represents the sign of the value. X'A', X'C', X'E' and X'F' indicate a positive number, X'B' and X'D' indicate a negative number. FileKit (and the REPORT utility) will correctly interpret any of these sign representations and display the value correctly. However, when using the FileKit Data Editor to input a value to a DECIMAL field defined with SIGNED, X'C' and X'D' will be used for positive and negative values respectively. Otherwise, if the DECIMAL field is defined with UNSIGNED, FileKit will use X'F' and so all values will be positive.

By default the field is SIGNED, the default value for precision (*pr*) is 7 and the default scale (*sc*) is 0.

FIXED (*pr*[,*sc*]**)** [**SIGNED** | **UNSIGNED**]

A signed or unsigned binary value representing a decimal fixed point (rational) numeric value. The field occupies a number of bytes based on the precision (*pr*) value as follows:

Precision (*pr*)	**#Bytes**
1 to 4	2 (halfword)
5 to 9	4 (fullword)
10 to 19	8 (doubleword

The precision (*pr*) value is the total number of decimal digits represented by the field which can range from 1 to 19. The scale (*sc*) value is the number of fractional digits (digits following the decimal point) which can range from 0 to the precision value.

The binary value is converted to decimal before the scaling factor is applied. For example, if a field defined as FIXED(3,2) contains the value X'0200', then FileKit will convert the value to decimal ('512') before applying the scaling factor, resulting in decimal value '5.12'.

A FIXED field may be UNALIGNED, so that it begins at the byte immediately following the field defined before it, or ALIGNED. If ALIGNED, then the field will begin at the next halfword, fullword or doubleword boundary within the record data for FIXED fields of length 2, 4 and 8 respectively. The area of record between the last byte of the previous field and the first byte of the aligned FIXED field contains unreferenced slack bytes.

If a FIXED field is defined as SIGNED, then negative values are represented as the two's complement of the equivalent positive value. The binary data is big-endian so that the sign is determined by the first (leftmost) bit value (0=positive, 1=negative).

By default the field is SIGNED and UNALIGNED, the default value for precision (*pr*) is 7 and the default scale (*sc*) is 0.

FLOATBIN (4|8|16**)**

A signed binary (IEEE 754) floating-point format value representing a decimal (rational) numeric value. The binary floating-point field occupies 4 (short), 8 (long) or 16 (extended) bytes.

A FLOATBIN field may be UNALIGNED, so that it begins at the byte immediately following the field defined before it, or ALIGNED. If ALIGNED, then the field will begin at the next fullword, doubleword or quadword boundary within the record data for FLOATBIN fields of length 4, 8 and 16 respectively. The area of record between the last byte of the previous field and the first byte of the aligned FLOATBIN field are unreferenced slack bytes.

By default the field is UNALIGNED, the default value field length is 4.

FLOATDEC (7|16|34**)**

A signed decimal floating-point format value representing a decimal (rational) numeric value. The decimal floating-point field occupies a number of bytes based on the specified number of significant digits represented by the field value. A significant digit value of 7 implies 4 bytes (short), 16 implies 8 bytes (long), and 34 implies 16 bytes (extended).

A FLOATDEC field is always byte aligned so that it begins at the byte immediately following the field defined before it.

By default field length is 34.

FLOATHEX (4|8|16**)**

A signed hexadecimal floating-point format value representing a decimal (rational) numeric value. The hexadecimal floating-point field occupies 4 (short), 8 (long) or 16 (extended) bytes.

A FLOATHEX field may be UNALIGNED, so that it begins at the byte immediately following the field defined before it, or ALIGNED. If ALIGNED, then the field will begin at the next fullword, doubleword or quadword boundary within the record data for FLOATHEX fields of length 4, 8 and 16 respectively. The area of record between the last byte of the previous field and the first byte of the aligned FLOATHEX field are unreferenced slack bytes.

By default the field is UNALIGNED, the default value field length is 4.

HEXADECIMAL (*n_bytes***)**

A fixed-length string occupying a number of bytes (*n_bytes*) and displayed in hexdecimal format. The default value for *n_bytes* is 1.

Hexadecimal format displays each byte of the field as 2 hexadecimal digit characters (0-F) so that the displayed data length is twice the field length.

A HEXADECIMAL field is always byte aligned so that it begins at the byte immediately following the field defined before it.

HEXVAR (*n_bytes***)**
A variable-length string of maximum length (*n_bytes*) which is displayed in hexdecimal format. The string is padded with blanks to occupy the fixed number of bytes (*n_bytes*). The default value for *n_bytes* is 1.

HEXVAR and CHARVARYING field types are analogous, and differ only in the how the data is displayed. Hexadecimal format displays each byte of the string as 2 hexadecimal digit characters (0-F) so that the displayed data length is twice the field length.

An INTEGER field occupies the first 2 bytes of the HEXVAR field and the string data the remaining *n_bytes*, thus the overall length of a HEXVAR field is *n_bytes*+2. The actual length of the variable length string within the fixed length area is held in the 2-byte INTEGER field prefix.

A HEXVAR field may be UNALIGNED so that it begins at the byte immediately following the field defined before it, or ALIGNED. If ALIGNED is specified, the field will be aligned on a halfword boundary. The area of record between the last byte of the previous field and the first byte of the aligned HEXVAR field contains unreferenced slack bytes. Default is UNALIGNED.

INTEGER (*n_bytes***)** [**SIGNED** | **UNSIGNED**]
A signed or unsigned binary value representing a decimal whole number (integer) numeric value. The field occupies a number of bytes (*n_bytes*).

By default, an INTEGER field is UNALIGNED, so that it begins at the byte immediately following the field defined before it. For INTEGER type with *n_bytes* length 2, 4 or 8, the field may be ALIGNED so that it begins at the next halfword, fullword or doubleword boundary respectively. The area of record data between the last byte of the previous field and the first byte of the aligned INTEGER field contains unreferenced slack bytes.

If an INTEGER field is defined as SIGNED, then negative values are represented as the two's complement of the equivalent positive value. The binary data is big-endian so that the sign is determined by the first (leftmost) bit value (0=positive, 1=negative).

By default the field is SIGNED and UNALIGNED, and the default value for *n_bytes* is 4.

IPaddress (4|16**)**
A field interpreted as an IP address occupying either **4** or **16** bytes.

IPADDRESS(4)	An IPv4 address comprising 4, 1-byte unsigned INTEGER fields. The value will display as 4, 3-digit decimal values each separated by a "." (dot/period) with an overall length of 15 bytes. For example, 192.168.001.064
IPADDRESS(16)	This format will detect whether the 16-byte source represents an IPv4 or IPv6 address. If the first 10 bytes of the source are X'00' and the next 2 bytes are X'FF', then the field is determined to be an IPv4 address. The junior 4 bytes of the source value will be processed as for IPADDRESS(4) and the displayed value will be left justified within a 39-byte display area. Otherwise, the source is determined to be an IPv6 address. The value will display as 8, 4-digit hexadecimal values each separated by a ":" (colon) with overall length of 39 bytes. For example, 0123:4567:89AB:CDEF:0123:4567:89AB:CDEF

PCHAR (*pl1_picture_string***)**
A character field represented by a PL/1 style PICTURE string (*pl1_picture_string*) occupying a number of bytes equal to the length of the PICTURE string.

The *pl1_picture_string* must be enclosed in quotation marks (") or apostrophes ('), and may contain only picture string characters "X", "A" and "9" and repetition factors "(*n*)". See publication "*Enterprise PL1 for z/OS Language Reference*" for details on picture characters for character data.

The default PCHAR character data picture string is 'X'.

PFIXED (*pl1_picture_string***)**
A numeric character field represented by a PL/1 style PICTURE string (*pl1_picture_string*) that describes a decimal fixed-point numeric value. The field occupies a number of bytes equal to the maximum length of the numeric character data item that can be represented by the PICTURE string characters.

The *pl1_picture_string* must be enclosed in quotation marks (") or apostrophes ('), and may contain only valid numeric character picture string characters applicable to fixed-point values (for example: "9", "V", "Z", "+", "-", "S", "B", ".", "CR") and repetition factors "(*n*)". The *pl1_picture_string* must **not** contain characters "A", "X", "E" or "K". See publication "*Enterprise PL1 for z/OS Language Reference*" for details on picture characters for character numeric data.

The default PFIXED numeric character data picture string is 'S9'.

PFLOAT (*pl1_picture_string***)**
A numeric character field represented by a PL/1 style PICTURE string (*pl1_picture_string*) that describes a decimal floating-point numeric value. The field occupies a number of bytes equal to the maximum length of the numeric character data item that can be represented by the PICTURE string characters.

The *pl1_picture_string* must be enclosed in quotation marks (") or apostrophes ('), and may contain only valid numeric character picture string characters applicable to floating-point values (for example: "9", "V", "Z", "+", "-", "S", "B", ".", "CR", "E", "K") and repetition factors "(*n*)". The *pl1_picture_string* must **not** contain characters "A" or "X". See publication "*Enterprise PL1 for z/OS Language Reference*" for details on picture characters for character numeric data.

The default PFLOAT numeric character data picture string is 'S9ES99'.

STRUCTURE (Field Definition `[, ...]`**)**
A STRUCTURE field is a group of one or more fields, each specified using a Field Definition syntax and separated from the next field definition using a comma (","). One or more of these fields may itself be a STRUCTURE field, thus allowing definition of multiple levels of nested group fields.

The STRUCTURE field occupies a number of bytes equal to the total length of the fields defined in the structure plus any slack bytes inserted due to alignment.

A STRUCTURE field may be UNALIGNED, so that it begins at the byte immediately following the field defined before it, or ALIGNED. If ALIGNED is specified then all fields within the structure will also be aligned according to their data type. The STRUCTURE field itself will be aligned according to the data type of the first field in the structure. For example, if the first field of the structure is FLOATHEX(16) then the structure will start at the next quadword boundary position, if the first field is INTEGER(4) then the structure will start at the next fullword boundary position, and if it is CHARACTER then it will start at the next byte position. The area of record between the last byte of the previous field and the first byte of the aligned STRUCTURE field will contain unreferenced slack bytes. Default is UNALIGNED

TIME (BINARY | **DECIMAL** | **DECIMAL2** | **DECIMAL3** | **STCK** | **UNIX)**
A time of day field with the source data in one of the following formats:

BINARY	A BINARY time field has a 4-byte length and is in the format `X'`*nnnn*`,`*nnnn*`'`. The field contains a 32-bit unsigned binary value representing the number of one hundredths of a second (0.01) that have elapsed since midnight. For example, a TIME(BINARY) field containing `X'004A,D2A3'` (decimal '4903587') corresponds to time of day '13:37:15.87'.
DECIMAL	A DECIMAL time field has a 4-byte length and is in the format as returned by the TIME macro with option DEC, `X'`*HHMM*`,`*SShh*`'`. The *HH*, *MM*, *SS* and *hh* are each a 1-byte DECIMAL field with no sign indicator. The values represent hour of day, minute of hour, second of minute and hundredths of second respectively. For example, a TIME(DECIMAL) field containing `X'1337,1587'` corresponds to time of day '13:37:15.87'.
DECIMAL2	A DECIMAL2 time field has a 4-byte length and is in the format `X'00`*HH*`,`*MMSS*`'`. The *HH*, *MM* and *SS* are each a 1-byte DECIMAL field with no sign indicator. The values represent hour of day, minute of hour and second of minute respectively. For example, a TIME(DECIMAL2) field containing `X'0013,3715'` corresponds to time of day '13:37:15'.
DECIMAL3	A DECIMAL3 time field has a 4-byte length and is in the format `X'0`*HHM*`,`*MSSC*`'`. The field is a positively signed, 4-byte DECIMAL field where the leftmost (high order) 4-bits of the packed decimal value are 0. The pairs of 2 packed decimal digits that follow are: *HH* (hour of day), *MM* (minute of hour) and *SS* (second of minute) values. For example, a TIME(DECIMAL3) field containing `X'0133,715C'` corresponds to time of day '13:37:15'.
STCK	A STCK time field has an 8-byte length and is a 64-bit unsigned binary elapsed time value in the system TOD clock format. See publication "*z/Architecture Principles of Operation*" for details on system TOD clock format.

STCK,*n_bits*	A STCK time field with an *n_bits* value has a 4-byte length. It represents an unsigned binary elapsed time value obtained from the rightmost (low order) 32-bits of the system TOD clock value which has been shifted right a number of bits specified by *n_bits*. This STCK format is often found in SMF record for fields representing elapsed time values. The following shows the number of microsecond (us) time units represented by one bit in the 32-bit TOD clock value which has been shifted a *n_bits* number of bits to the right as specified by TIME(STCK,*n_bits*). TIME(STCK,**12**) 1us TIME(STCK,**16**) 16us TIME(STCK,**19**) 128us TIME(STCK,**22**) 1024us TIME(STCK,**32**) 1048576us
UNIX	A UNIX time field is a 4-byte unsigned INTEGER value representing the number of seconds elapsed since midnight. For example, a TIME(UNIX) field containing X'0000,BF8B' (decimal '49035') corresponds to time of day '13:37:15'.

The default time format is DECIMAL.

TIMESTAMP (BINARY | DECIMAL | DECIMAL2 | DECIMAL3 | HFSDIR | SMF | STCK | ...
... TIMEBIN | TIMEDEC | TIMEDEC2 | TIMEDEC3)

A date and time field with the source data in one of the following formats:

BINARY	A BINARY timestamp field is equivalent to a DATE(BINARY) field followed by a TIME(BINARY) field but referenced as a single timestamp value. It has a 10-byte length and is in the format X'*yyyy*,*mmmm*,*dddd*,*nnnn*,*nnnn*'.
DECIMAL TIMEDEC	A DECIMAL timestamp field is equivalent to a DATE(DECIMAL) field followed by a TIME(DECIMAL) field but referenced as a single timestamp value. It has an 8-byte length and is in the format X'*ccyy*,*dddF*,*HHMM*,*SShh*'.
DECIMAL2 TIMEDEC2	A DECIMAL2 (synonym TIMEDEC2) timestamp field is equivalent to a DATE(DECIMAL) field followed by a TIME(DECIMAL2) field but referenced as a single timestamp value. It has an 8-byte length and is in the format X'*ccyy*,*dddF*,00*HH*,*MMSS*'.
DECIMAL3 TIMEDEC3	A DECIMAL3 (synonym TIMEDEC3) timestamp field is equivalent to a DATE(DECIMAL) field followed by a TIME(DECIMAL3) field but referenced as a single timestamp value. It has an 8-byte length and is in the format X'*ccyy*,*dddF*,0*HHM*,*MSSC*'.
HFSDIR	An HFSDIR timestamp field has a 4-byte length and is in a format used by the system for HFS directory timestamps. This is a 32-bit unsigned binary value containing a number of seconds elapsed since 1970/01/01 00:00:00.
SMF	A SMF timestamp field is equivalent to a TIME(BINARY) field followed by a DATE(DECIMAL) field but referenced as a single timestamp value. It has an 8-byte length and is in the format X'*nnnn*,*nnnn*,*ccyy*,*dddF*'.
STCK	A STCK timestamp field has a length 8-bytes and is an unsigned 64-bit binary system TOD clock value.
STCK,*n-bytes*	A STCK timestamp field occupies a number of bytes (*n_bytes*) and has a value equivalent to the leftmost (high order) *n_bytes* of a system TOD clock value. STCK,4 is commonly used as the timestamp format in a number of SMF records fields.
TIMEBIN	A TIMEBIN timestamp field is equivalent to a DATE(DECIMAL) field followed by a TIME(BINARY) field but referenced as a single timestamp value. It has an 8-byte length and is in the format X'*ccyy*,*dddF*,*nnnn*,*nnnn*'.

The default timestamp format is DECIMAL.

UNION (Field Definition [, ...]**)**
A UNION field is used to define one or more fields at the same area of the record data. Each field in the union starts at the same record position as the start of the UNION field and redefines the interpretation of the data at that position.

A field within the union is defined using Field Definition syntax, entered between the parentheses that follow UNION and is separated from the next field definition using a comma (","). Each comma separated field definition identifies a field union of fields.

The UNION field occupies a number of bytes equal to the length of the longest field defined in the union.

A UNION field may be UNALIGNED, so that it begins at the byte immediately following the field defined before it, or ALIGNED. If ALIGNED is specified then the UNION field and all fields within the union will be aligned according to the data type of the first field in the union. For example, if the first field defined in the union is FLOATHEX(8) then all fields in the union will start at the same doubleword boundary position, if it is INTEGER(2) then all fields will start at the same halfword boundary position, and if it is CHARACTER then all fields will start at the next byte position. The area of record between the last byte of the previous field and the first byte of the aligned UNION field will contain unreferenced slack bytes.

A field within the union may have a data type of STRUCTURE in which case a group of fields may be defined to occupy the same area as the other field(s) in the union. Default is UNALIGNED

VARCHAR (*max_bytes*[,*len_bytes*[,**EXCLUSIVE** | **INCLUSIVE**]]**)**
A variable-length character string occupying a variable number of bytes up to a maximum of *max_bytes* or *max_bytes*+*len_bytes*, as defined by INCLUSIVE and EXCLUSIVE respectively.

A VARCHAR field comprises an INTEGER field of length *len_bytes* (default 2-bytes) followed by a variable number of character bytes. The value in the INTEGER field specifies the number of bytes of character data. The *max_bytes* length value includes the INTEGER length field if INCLUSIVE is specified, in which case the maximum length of the character data is *max_bytes*-*len_bytes*. If EXCLUSIVE is specified, the *max_bytes* length does **not** include the INTEGER length field.

A VARCHAR field is always byte aligned so that it begins at the byte immediately following the field defined before it. The start positions of fields that follow a VARCHAR field may vary for each record mapped by the record mapping structure.

By default, the *max_bytes* value does not include the INTEGER length field (EXCLUSIVE) which itself has a default (*len_bytes*) length of 2.

VARHEX (*max_bytes*[,*len_bytes*[,**EXCLUSIVE** | **INCLUSIVE**]]**)**
A variable-length string occupying a variable number of bytes up to a maximum of *max_bytes* or *max_bytes*+*len_bytes*, as defined by INCLUSIVE and EXCLUSIVE respectively. The string is displayed in hexdecimal format.

VARHEX and VARCHAR field types are analogous, and differ only in the how the data is displayed. Hexadecimal format displays each byte of the string as 2 hexadecimal digit characters (0-F) so that the displayed data length is twice the string length.

A VARHEX field comprises an INTEGER field of length *len_bytes* (default 2-bytes) followed by a variable number of data bytes. The value in the INTEGER field specifies the number of bytes of data. The *max_bytes* length value includes the INTEGER length field if INCLUSIVE is specified, in which case the maximum length of the data is *max_bytes*-*len_bytes*. If EXCLUSIVE is specified, the *max_bytes* length does **not** include the INTEGER length field.

A VARHEX field is always byte aligned so that it begins at the byte immediately following the field defined before it. The start positions of fields that follow a VARHEX field may vary for each record mapped by the record mapping structure.

By default, the *max_bytes* value does not include the INTEGER length field (EXCLUSIVE) which itself has a default (*len_bytes*) length of 2.

XVARCHAR (*max_bytes*,*len_field***)**
A variable-length character string occupying a variable number of bytes up to a maximum of *max_bytes*.

An XVARCHAR field comprises variable number of character bytes of length specified by another field mapped within the same record (*len_field*). This length field must be of numeric data type and contain a whole-number value which is less than the *max_bytes*.

An XVARCHAR field is always byte aligned so that it begins at the byte immediately following the field defined before it. The start positions of fields that follow an XVARCHAR field may vary for each record mapped by the record mapping structure.

ZONED (*pr*[,*sc*]**)** [**SIGNED** [(**TRAILING** | **LEADING** [, **INCLUDED** | **SEPARATE**])] | **UNSIGNED**]
A signed or unsigned binary zoned-decimal character value representing a decimal fixed point (rational) numeric value. The field occupies a number of bytes equal to the precision (*pr*) or precision plus 1 (*pr*+1) if a separate sign byte (SEPARATE) is used.

The precision (*pr*) value is the total number of zoned-decimal digits represented by the field. The scale (*sc*) value is the number of fractional digits (digits following the decimal point) which can range from 0 to the precision value.

A ZONED field is always byte aligned so that it begins at the byte immediately following the field defined before it.

If a ZONED field is defined as UNSIGNED, then the field value is always positive and each character is a decimal digit where the 4-byte zone portion of each byte is X'F'. For example, a ZONED(4) UNSIGNED field with value 123 is X'F0F1F2F3'. If the field is defined as SIGNED then the field has a format based on the combination of the TRAILING/LEADING and INCLUDED/SEPARATE attributes. See table below.

LEADING INCLUDED	The sign is **included** in the ZONED value so that the 4-byte zone portion of the **first** character is X'C' if the value is positive or X'D' if the value is negative. For example, a ZONED(4) SIGNED(LEAD,INC) with value 416 is X'C0F4F1F6' and -794 is X'D0F7F9F4'.
TRAILING INCLUDED	The sign is **included** in the ZONED value so that the 4-byte zone portion of the **last** character is X'C' if the value is positive or X'D' if the value is negative. For example, a ZONED(4) SIGNED(TRAIL,INC) with value 416 is X'F0F4F1C6' and -794 is X'F0F7F9D4'.
LEADING SEPARATE	The sign occupies a **separate** byte in the ZONED field increasing the length of the field by 1 (i.e. *pr*+1). The sign character ("+" for a positive value and "-" for a negative value) is positioned **before** the character digits. For example, a ZONED(4) SIGNED(LEAD,SEP) with value 416 is X'4EF0F4F1C6' (+0416) and -794 is X'60F0F7F9F4' (-0794).
TRAILING SEPARATE	The sign occupies a **separate** byte in the ZONED field increasing the length of the field by 1 (i.e. *pr*+1). The sign character ("+" for a positive value and "-" for a negative value) is positioned **after** the character digits. For example, a ZONED(4) SIGNED(TRAIL,SEP) with value 416 is X'F0F4F1C64E' (0416+) and -794 is X'F0F7F9F460' (0794-).

By default, a ZONED field has attributes SIGNED(TRAILING,INCLUDED). The default value for precision (*pr*) is 1 and the default scale (*sc*) is 0.

DIMENSIONS ({*dim* | **(***min*,*max*,*dfield***)**} [, ...] **)**
For Field Definition syntax, DIMENSIONS indicates that the field is a template for an array of fields.

The array may comprise a fixed number of field elements specified by the integer value *dim*, or may have a variable number of field elements as specified by the whole number value contained in the field *dfield* which is defined at another location within the record data. If there are a variable number of array elements, then a *min* and *max* integer value must be specified to define a fixed minimum and maximum number of array elements.

Each field element in the array has the same filed name and definition, and are individually identified via a sequence number in a parenthesised subscript. For example, for a field name "`Result`" defined with `DIMENSIONS(4)`, then the array elements would each be referenced as `Result(1)`, `Result(2)`, `Result(3)` and `Result(4)`.

A single specification of *dim* or *dfield* defines a single array dimension. However, if necessary, an array may be multi-dimensional so that each array field element is itself an array of field elements. A number of array dimension definitions may be specified with each definition separated from the next by a comma (","). The subscript values, used to identify an individual element of a multi-dimensional array, is also multi-dimensional so that a comma separated sequence number exists for each dimension. For example, if field name "`Result`" was defined with `DIMENSIONS(2,2)`, then the array elements would each be referenced as `Result(1,1)`, `Result(1,2)`, `Result(2,1)` and `Result(2,2)`.

A mixture of fixed and variable length arrays may be used for multi-dimensional arrays. For example, field "`BoxDim`" defined with `DIMENSIONS((0,10,NBoxes),3)` defines a 2-dimensional array. The first array dimension is variable and has a number of elements equal to the whole number value in the "`NBoxes`" (number of boxes) field. Each of these BoxDim array field elements is an array of 3 field elements (denoting the length, width and height of each box). If "`NBoxes`" has a value 2, then the array field elements would be `BoxDim(1,1)`, `BoxDim(1,2)`, `BoxDim(1,3)`, `BoxDim(2,1)`, `BoxDim(2,2)` and `BoxDim(2,3)`.

Note that the start positions of fields that follow a field involving a variable number of array elements may vary for each record mapped by the record mapping structure.

EBCDIC | **ASCII**
For Field Definition syntax, EBCDIC and ASCII apply only to fields defined with one of the character data types (CHARACTER, CHARZ, CHARVARYING, VARCHAR or XVARCHAR) and specifies the encoding of the character data.

Default encoding is EBCDIC.

ENUMERATION [*enam*] **(***string*=*val* [, ...] **)**
For Field Definition syntax, ENUMERATION applies only to fields defined with one of the integer data types (BINTEGER or INTEGER) and is used to create an enumeration definition for the field.

An enumeration definition is a list of comma separated integer value (*val*) to character name (*string*) equivalencies. The definition may be assigned a name (*enam*) which is necessary if it is be used by another field definition. If the field requires the same named enumeration definition as that defined on another field, then ENUMERATION may be specified without a list of specifications and simply reference the other enumeration by name (*enam*).

Integer values that match an enumerator value will display as the equivalent enumeration string. Similarly, a value entered in the field using the Data Editor must be one of the enumerated strings. However, any reference to the field's value (for example in an expression or a search string) may use the enumerated string or its equivalent integer value.

For example, a field defined as `BINTEGER(1)` with `ENUMERATION SWITCH ("Off"=0,"On"=1)` will display values "On" or "Off" instead of "1" and "0". Furthermore, another integer type field may be assigned the same enumeration simply by specifying `ENUMERATION SWITCH`.

Note that *string* must be enclosed in quotation marks (") or apostrophes (') if it contains comma (",") or blank characters. A *string* may also be specified without an equivalent *val*, in which case *val* will default to be 1 plus the value in the previous equivalency in the list.

OFFSET (*int* | X'*hex*'**)**
For Non-Contiguous Field Definitions only, OFFSET specifies the absolute offset of the field from the start of the record data, expressed as a decimal (*int*) or hexadecimal (X'*hex*') value.

Field definitions that use OFFSET may occur in any order. If OFFSET is specified, then it must also be used on all other field definitions.

REMARKS *remark*
For Field Definition syntax, REMARK specifies a comment string (*remark*) to be associated with the field.

The comment string must be enclosed in quotation marks (") or apostrophes (').

Note that REPORT will use a field's *remark* text as the default column header if the field is identified as a report column *input-field* in the COLUMNS section.

ZEROS
For Field Definition syntax, ZEROS applies only to fields defined with one of the numeric data types (BINTEGER, DECIMAL, FIXED, INTEGER or ZONED) and is used to include non-significant leading zeros in the displayed value.

By default, all non-significant leading zeros are replaced with blank characters.

OPTIONS

Overview:

The OPTIONS section specifies options that affect the REPORT utility execution and determine the format of the report output.

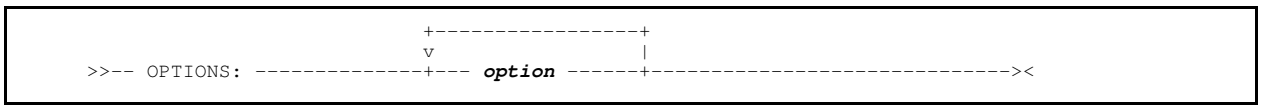
Multiple option values may be specified on the same control statement and/or multiple control statements under the OPTIONS section header.

Example:

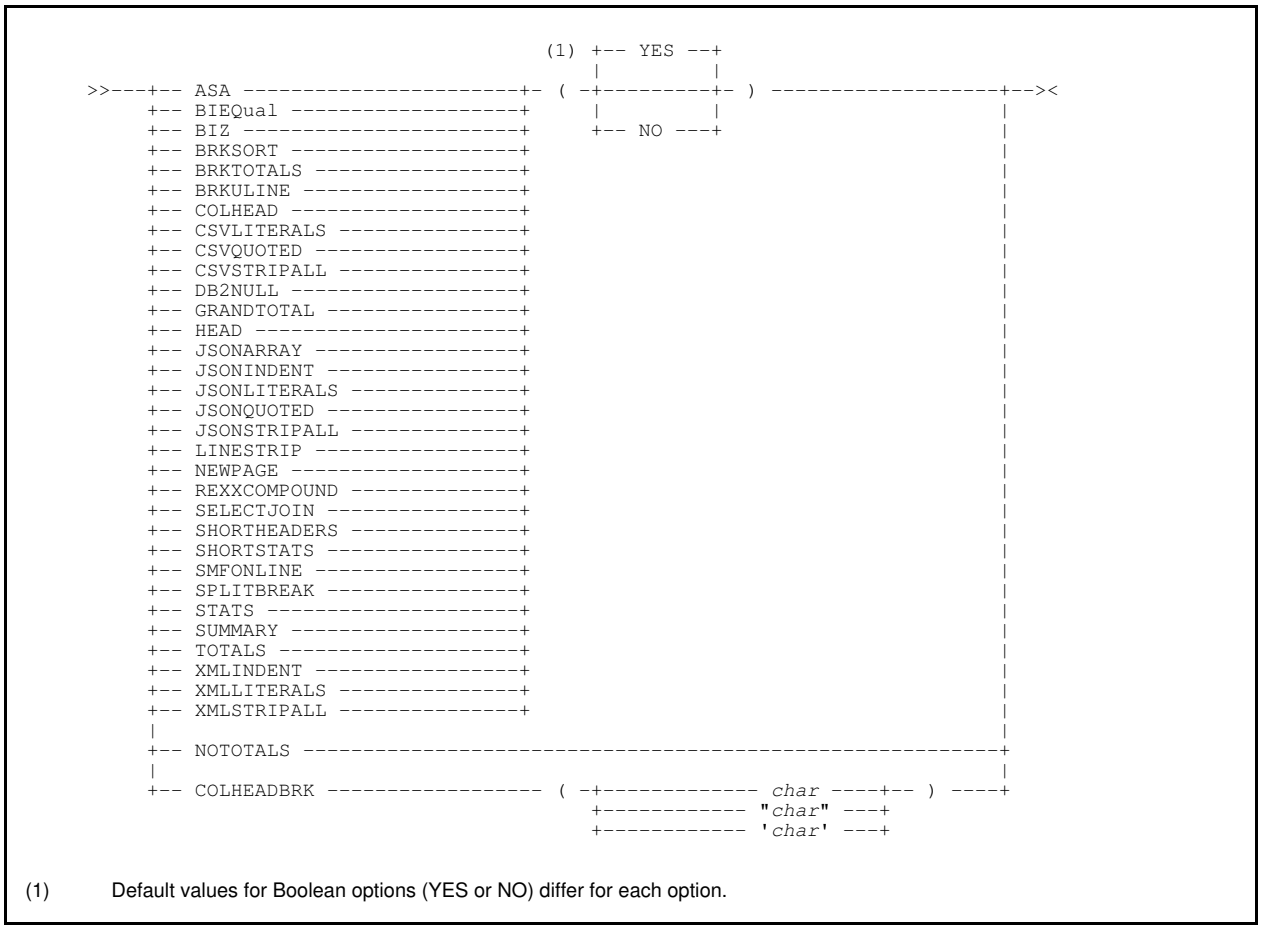
In the following example, the OPTIONS section is used to restrict printed report output. A maximum of 5 detail records will be written for each break control group, underlining before statistics values will be suppressed and output of statistics column totals will be also be suppressed.

```
OPTIONS:
   DETAIL(5)  NOBRKULINE
   NOTOTALS
```

Syntax:

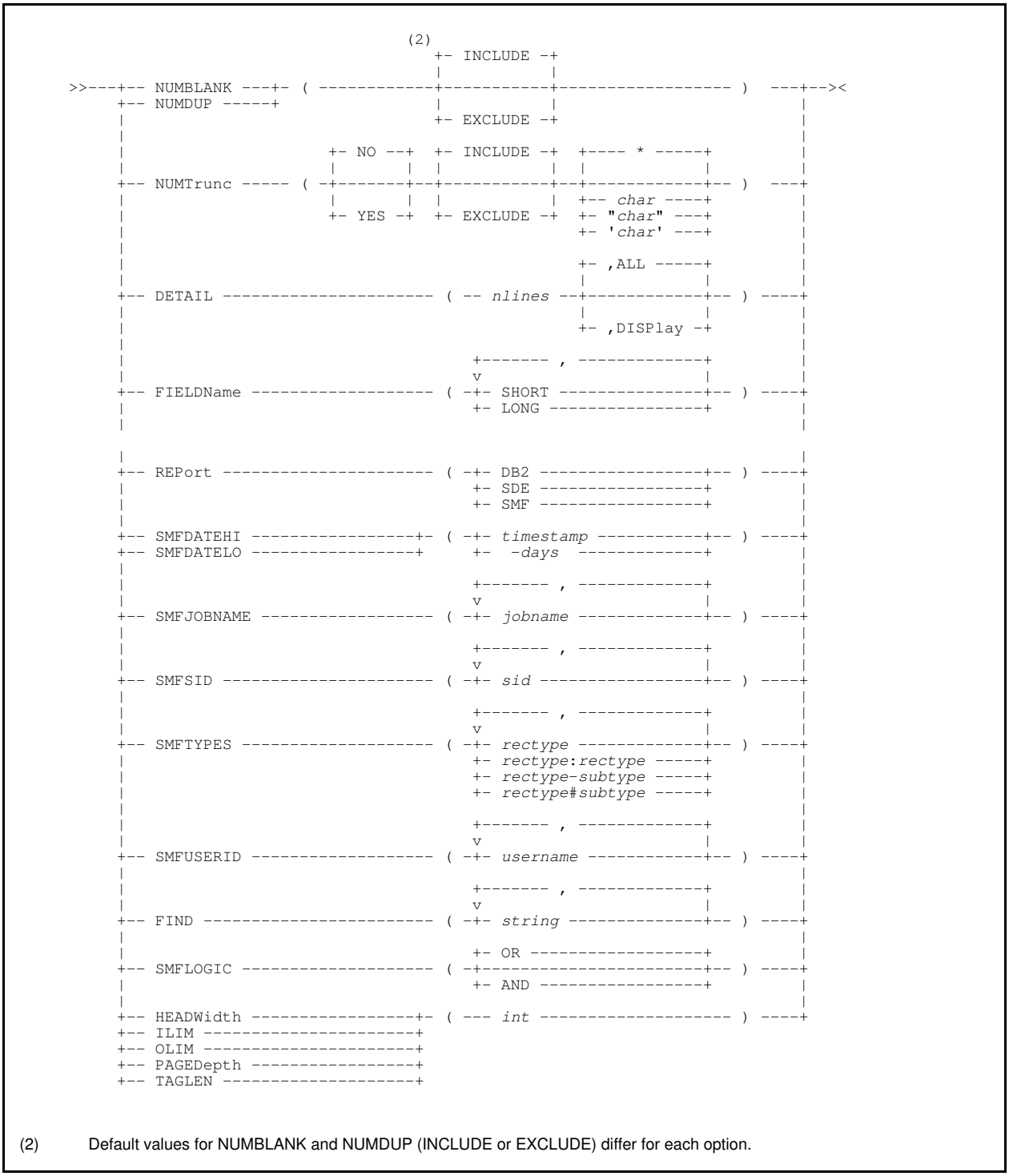


***option* (1/2):**



(1) Default values for Boolean options (YES or NO) differ for each option.

***option* (2/2):**



(2) Default values for NUMBLANK and NUMDUP (INCLUDE or EXCLUDE) differ for each option.

Synonyms:

OPTIONS	OPTION

Parameters:

ASA [**(YES | NO)**]
Applicable only to printed output, ASA specifies whether or not the first position of each report line is reserved for mainframe EBCDIC printer ASA/ANSI carriage control symbols.

By default, ASA print symbols are generated. The following ASA characters are used:

ASA		Description
HEX	CHAR	
X'40'	(blank)	Space one line and print (single spacing).
X'F1'	1	Skip to a new page and print.

BIEQual [**(YES | NO)**]
BIEQUAL or one of its synonyms (BLANKIFEQUAL, BLANKWHENEQUAL or BWEQUAL) specifies whether or not an entry in a report detail line will be displayed as blank if the column value matches that in the same column of the previous report detail line.

The operand value specified on option BIEQUAL will apply to **all** column entries. Specify BIEQUAL on individual entries in the COLUMNS section if the option BWEQUAL(YES) is to apply only to values in specific columns.

BIEQUAL(NO) is default and so equal values on successive detail lines are **not** converted to blanks.

BIZ [**(YES | NO)**]
BIZ or one of its synonyms (BLANKIFZERO, BLANKWHENZERO or BWZ) specifies whether or not a zero (0) value returned for **any** numeric input field defined in the COLUMNS section is converted to a blank in the report output.

By default, zero values are not converted. If a BLANKWHENZERO section exists, then the BIZ option is ignored and **only** zero values in the numeric columns specified in the BLANKWHENZERO section control statements are converted to blank.

BRKSORT [**(YES | NO)**]
Applicable only to printed output, BRKSORT specifies whether or not an interdependency exists between BREAK and SORT field definitions.

A #GRAND break may be specified regardless of the setting of BRKSORT. Whether or not it is specified in the BREAK section, the #GRAND break is always treated as the 1st break level in the control break sequence hierarchy.

For **BRKSORT(NO)**, a control break may be defined based on any input, computed or built-in field, regardless of whether the break field is a one on which input records will be sorted (via the SORT section). The order in which break fields are defined in the BREAK section determines the hierarchical sequence of break levels that follow the #GRAND break. BRKSORT(NO) is default for REPORT processing.

BRKSORT(YES) may be used to allow only break fields that are also specified as sort key fields within the SORT section. The number of control breaks defined may be less than or equal to the number of sort key fields. If multiple control breaks are defined, then the order in which key field names are specified in the SORT section, and not the order in which control breaks are defined in the BREAK section, determines the hierarchical sequence of break levels that follow the #GRAND break. Error ZZSR024E is returned if a break field is used which is not a specified sort key field.

If a SORT section exists, then the REPORT utility will perform its own sort of the input records based on the sort key fields. Therefore, if input records are already in the required sequence or if DB2 table input is used, then BRKSORT(YES) should not be used.

BRKTOTALS [**(YES | NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, BRKTOTALS specifies whether or not default TOTAL lines are generated for all control break definitions other than the #GRAND control break.

BRKTOTALS(YES) is default and so the REPORT utility automatically generates a TOTAL break line specification for each BREAK control statement that does not have the *fieldname* "#GRAND" and on which TOTAL is not already specified. Therefore, a break line containing the statistic column sub-totals will be printed following each control break group in the report output.

BRKTOTALS(NO) will override this default for all control breaks except the #GRAND break. If BRKTOTALS(NO) is specified, the statistics column totals line will still be printed for a control break if the TOTAL parameter has been explicitly specified on the control break definition.

See GRANDTOTAL for controlling default output of grand total values for an explicitly defined or utility generated #GRAND break definition.

The value of BRKTOTALS is also set by option TOTALS.

BRKULINE [**(YES** | **NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, BRKULINE specifies whether or not statistics column values are to be underlined in the printed report.

BRKULINE(YES) is default and so a line containing hyphen/minus symbols ("-") or equals symbols ('=') is printed following each control break group, serving to underline the column values for which statistic data (totals, averages maximums, etc.) is reported.

Hyphen/minus symbols are used for the underlining that follows standard control break groups and equals symbols are used following the #GRAND control break group. For the #GRAND control break group only, the underline is repeated following the last break line to contain a reported statistics value.

BRKULINE(NO) will suppress all statistic column underlining.

COLHEAD [**(YES** | **NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, COLHEAD specifies whether or not column headers are included in the printed output.

COLHEAD(YES) will print column headers and is default if the report contains single line column detail lines.

COLHEAD(NO) will suppress the print of column headers and is default if the report contains multi-line column detail lines. (i.e. if **<NEWLINE>** is used in the COLUMNS section).

If COLHEAD(YES) is used for a report that contains multi-line column detail lines, then only the column headings belonging to columns defined in the **first** detail line will be printed. Any heading text specified or implied for column definitions the follow the <NEWLINE> tag will therefore be ignored.

COLHEADBRK(*char* | "*char*" | '*char*'**)**
Applicable only to printed report output and ignored for CSV, JSON and XML output, COLHEADBRK specifies the character to be used as the column header break symbol used in column definitions in the COLUMNS section.

The text of a column header is supplied as a quoted character string literal on a column definition. The column header break symbol may be included in the quoted string to indicate a break in the header. When printed, the text that follows a header break symbol occupies a new column header line and is aligned below the header text that comes before it.

COLHEADBRK('|') is default indicating that the vertical bar character ('|') is the column header break symbol. Note that the assigned (or default) character used as the header break symbol prevails until another COLHEADBRK option is encountered in the report definition.

Note that the column header break symbol will be treated as being part of the header text if it is escaped by another header break symbol immediately following. e.g. In the following, the printed header text for column RESPONSE will be "YES|NO", not "YES" on the first header line and "NO" on the second.

```
OPTIONS:  COLHEADBRK('|')
COLUMNS:  RESPONSE   'YES||NO'
```

CSVLITERALS [**(YES** | **NO)**]
Applicable only to CSV report output, CSVLITERALS specifies whether or not *literal* values specified in the in the COLUMNS section are included as values in the CSV output detail lines.

CSVLITERALS(NO) is default and will exclude column field *literal* values from the CSV output.

CSVQUOTED [**(YES** | **NO)**]
Applicable only to CSV report output, CSVQUOTED specifies whether or not **all** values are to be enclosed in quotation marks (").

CSVQUOTED(YES) is default and so values are always enclosed in quotation marks.

CSVQUOTED(NO) indicates that values are not enclosed in quotation marks unless it is necessary to do so in order to output a valid value (e.g. if a value contains the CSV separator character - default comma (",").

CSVSTRIPALL [**(YES** | **NO)**]
Applicable only to CSV report output, CSVSTRIPALL specifies whether or not leading and trailing blanks are to be stripped from all values, so that the comma separator immediately follows the last non-blank character on all but the last value in the output line.

CSVSTRIPALL(NO) is default and so values are not stripped of leading and trailing blanks. Each value will be of a fixed length equal to the specified (or default) field width.

CSVSTRIPALL(YES) indicates that all CSV values are to be stripped of leading and trailing blanks.

DB2NULL [**(YES** | **NO)**]
For DB2 table input only, DB2NULL specifies whether or not the default Data-Edit NULL value output indicator character is displayed for a null value in a DB2 column defined with NULL. (See the **NULLCHAR** Data-Edit SET/QUERY/EXTRACT option).

DB2NULL(NO) is default and so DB2 NULL values are displayed as blanks.

DB2NULL(YES) will display the NULLCHAR null value character (default is the underscore symbol "_").

DETAIL(*nlines*[,**ALL**|**DISPLAY**]**)**
DETAIL specifies the maximum number of detail lines (*nlines*) to be reported in each control break group and whether statistics are to be generated for all detail lines in the control group (ALL) or only those displayed (DISPLAY). If there are no control breaks in the report, DETAIL will specify the maximum number of report detail lines to be printed.

Note that, although break lines are not written for CSV, JSON and XML output, BREAK definitions may be specified and the DETAIL option used to limit the number of CSV, JSON and XML lines reported for each consecutive occurrence of a value in the break key field.

If ALL (the default) is selected:

1. The number of detail lines (#ITEMS) in the control group includes both reported and unreported lines.
2. Reported statistics values (e.g. totals, averages, etc.) are calculated based on values in both reported and unreported lines.
3. Field references (*fieldname*) in break line print expressions that follow the control group, are substituted with values from the last detail line of the control group, whether or not that line is reported

If DISPLAY is selected:

1. The number of detail lines (#ITEMS) in the control group includes only reported lines. (i.e. #ITEMS is always less than or equal to *nlines*.),
2. Reported statistics values (e.g. totals, averages, etc.) are calculated based on values in reported lines only.
3. Field references (*fieldname*) in break line print expressions that follow the control group, are substituted with values from the last reported detail line of the control group.

Break lines, column headings, page headings and page footings are unaffected by the DETAIL option. DETAIL(0) will suppress report output of all detail lines.

In the following example, the DETAIL option is used to report the names of the first 5 tracks on an album. Beacuase ALL is default, the value of the #ITEMS built-in field, reported following each control group break, will reflect the total number of tracks on the album.

```
OPTIONS:  NOTOTALS  DETAIL(5)
HEAD:     #TIMESTMP / "First 5 Album Tracks" / "PAGE" #PAGE (RIGHT,4)
COLUMNS:  ALBUM;  TRACK-NUM 4;  NAME 50
SORT:     ALBUM;  TRACK-NUM
BREAK:    ALBUM
```

Default is to print all detail lines.

FIND(*string, ...***)**
Applicable only to SDE and SMF type input (not DB2), FIND specifies one or more comma separated search strings (*string*) which are used to perform **Unformatted Record Find String matching** for input record filtering.

Note: Filtering of DB2 table rows based on its contents may be achieved using a WHERE clause in the DB2 operands of the INPUT section.

The format of *string* is described by search values under *"Record Filtering"*.

If a match on **any** of the FIND search strings is located at **any** position within an unformatted input record, then Unformatted Record Find String matching will return a true result (1). Otherwise a false result (0) is returned.

Unformatted Record Find String matching is one of the content match criteria for **SMF** record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR" (see SMFLOGIC).
3. All other specified SMF content match criteria each return a true result. (see SMFJOBNAME, SMFSID, SMFTYPES and SMFUSERID)

For **non-SMF** input, no other content match criterion is supported. Therefore, a record will be passed for REPORT processing if a true result is returned by Unformatted Record Find String matching.

For example, the following will set a true condition if one of the strings "SYS1.MACLIB", "SYS1.MIGLIB", "SYS1.MODGEN" or "SYS1.MSGEN" (upper or lower case) exists at any location within the unformatted record.

```
FIND(SYS1.MACLIB, SYS1.MIGLIB, SYS1.MODGEN, SYS1.MSGEN)
```

FIND and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a FIND specification exists, then error message ERR065E or ERR066E is returned.

Note that all search strings specified in the FIND option will be overridden by FIND search strings entered as parameters via the following:

- The Formatted Record Report panel
- The SMF Report panel
- The FIND operand of the REPORT primary command

FIELDNAME([**SHORT**][**,LONG**]**)**]

FIELDNAME forces the REPORT utility to assign input field values to REXX variables that match the **SHORT** (unqualified) format of the field name, the **LONG** (qualified) format of the field name or both formats.

By default, the REPORT utility will assign an input field value to a REXX variable name that exactly matches the *fieldname* used in the COLUMNS or REQUIRED section to identify the input field. The variable name may be simple (short), matching the name of the field, or compound (long), containing dots/periods that delimit each field name group qualifier.

For example, the following column definitions are based on input fields "RECFM" and "DSN" which both belong to the group structure "JFCB" in SMF record type 14 mapping "SMF014_INPUT_or_RDBACK_Dataset". However, the *fieldname* specification for "RECFM" is unqualified (does not include the "JFCB" group structure name), whereas "DSN" is qualified. By default, field values will be assigned to REXX variables "RECFM" and "JFCB.DSN".

```
SMF014_INPUT_or_RDBACK_Dataset.RECFM
SMF014_INPUT_or_RDBACK_Dataset.JFCB.DSN
```

These variables may be referenced in the REXX routine identified by the COMPUTE section to establish *compute-field* values.

Where an input field has a name that is non-unique within the record structure, a fully-qualified *fieldname* reference is necessary to specify the group structure hierarchy to which the field belongs and so accurately identify the required input field. Since a qualified *fieldname* uses a dot/period to delimit each group structure name, the field value will be assigned to a variable that includes these qualifying dots/periods (i.e. a REXX compound variable).

To reference an input field identified by a fully-qualified *fieldname* using just its unqualified field name, then option FIELDNAME(SHORT) should be specified. In addition to assigning the field value to a REXX compound variable name, this option will force the REPORT utility to also assign the field value to a simple variable name that matches the lowest level qualifier in the *fieldname* specification (i.e. the field name itself).

FIELDNAME(LONG) may be specified to force the REPORT utility to assign field values to compound variable names. This may be useful if all field definitions have been provided using unqualified *fieldname* specifications but the field exists within a group structure of the record type structure.

Note that, use of FIELDNAME(SHORT or LONG) may be unnecessary. For input fields defined via the COLUMNS and/or REQUIRED sections, the REPORT utility will assign field values to **both** the long and short formats of the REXX input field variables when the following conditions are both true:

1. A mixture of qualified and unqualified *fieldname* specifications are used.
2. At least one of the unqualified field names identifies a field in a group structure.

GRANDTOTAL[**(YES**|**NO)**]

Applicable only to printed report output and ignored for CSV, JSON and XML output, GRANDTOTAL specifies whether or not a default TOTAL line is generated for the #GRAND control break.

GRANDTOTAL(YES) is default and so the REPORT utility automatically generates a TOTAL break line specification for the BREAK control statement with *fieldname* "#GRAND" if TOTAL is not already specified. Therefore, a break line containing the statistic column grand totals will be printed at the end of the report output.

If no BREAK section exists and GRANDTOTAL(YES) is in effect, then a #GRAND break is generated to report grand totals whether or not the report output is sorted.

GRANDTOTAL(NO) will suppress generation of grand totals and a grand totals break line whether or not the TOTAL parameter is explicitly specified on the #GRAND control break definition.

See BRKTOTALS for controlling default output of sub-total values for explicitly defined break definitions with a file name other than #GRAND.

The value of GRANDTOTAL is also set by option TOTALS.

HEAD[**(YES**|**NO)**]

Applicable only to printed report output and ignored for CSV, JSON and XML output, HEAD specifies whether or not page header lines are to be included in the report output.

HEAD(YES) is default and will print a page header at the top of each new page, whether or not one has been defined via the HEAD section. Page header lines are printed at the start of each new page. If no HEAD section exists but HEAD(YES) is in effect, then the default page header will be generated.

HEAD(NO) will suppress output of page header lines and the blank line that follows. The report column headers will occupy the first line of each new page. If option COLHEAD(NO) is in effect, then a report detail line will occupy the first line of each new page.

HEADWIDTH(*int***)**
Applicable only to printed report output and ignored for CSV, JSON and XML output, HEADWIDTH (or its synonym HW) specifies the width of the page header and footer lines within the report output.

Page header and footer segments of text are aligned within the header and footer lines using the header width value. By default, the header width will be equivalent to the page width value which is derived from the maximum lengths of text generated in page header, page footer, column detail and control break lines.

ILIM(*int***)**
Specifies an input limit, the maximum number of records (or DB2 table rows) that will be read from the input data source.

For DB2 table input, the specified ILIM value will take precedence over any **FOR** *nrecs* **ROWS** specification in the INPUT section of the report definition. The input limit will determine the number of rows fetched from the DB2 result table. Note, however, that any value specified by ILIM or FOR will be ignored if DB2 operand SCROLL is also specified to use a DB2 table scrollable cursor.

Each input record or DB2 row is processed sequentially until the input record threshold (*int*) is reached. At this point, sorting occurs if a SORT section exists in the report definition, otherwise REPORT processing ends. When a SORT section is not present, then REPORT processing may end before the input limit is reached if a specified OLIM output limit threshold is reached first.

Where the input source is **not** a DB2 table, the input limit includes records which may subsequently be excluded from REPORT processing by other record filtering techniques. For example, use of a FILTER section in the report definition or, alternatively, specification of SMF Low/High date thresholds (SMFDATELO/SMFDATEHI) or content match criteria (FIND, SMFJOBNAME, SMFSID, SMFTYPES and SMFUSERID).

ILIM(0) implies no input record limit and is set by default when no ILIM operand is supplied and no DB2 FOR *nrecs* ROWS specification exists.

Note that the record input limit specified in the ILIM option will be overridden by an ILIM value entered as a parameter via the following:

- The Formatted Record Report panel
- The SMF Report panel
- The ILIM operand of the REPORT primary command

JSONARRAY[**(YES**|**NO)**]
Applicable only to JSON report output, JSONARRAY specifies whether field values for each report line are part of a single JSON object, or one object within an array of objects.

REPORT generates a JSON object literal (in braces "{}") comprising a key/value pair for each column field in the output.

JSONARRAY(NO) is the default and so the generated JSON object literal is the "value" in a JSON object comprising a single key/value pair, where "key" is the record sequence number expressed as a string (in quotation marks).

```
{
    "000000001" : {"NAME" : "Daniel Ricciardo",     "COUNTRY" : "Australia" }
   ,"000000002" : {"NAME" : "Lando Norris",     "COUNTRY" : "United Kingdom" }
}
```

JSONARRAY(YES) will generate the JSON object literal as one object value in an array of comma separated object values enclosed in square brackets ("[]"). The array itself is the "value" in a JSON key/value pair, where "key" is the string "FileKit_Report".

```
{"FileKit_Report" :
 [
    {"NAME" : "Daniel Ricciardo",     "COUNTRY" : "Australia" }
   ,{"NAME" : "Lando Norris",     "COUNTRY" : "United Kingdom" }
 ]
}
```

JSONINDENT [**(YES** | **NO)**]
Applicable only to JSON report output, JSONINDENT specifies whether or not each key/value pair is to appear on its own line of the JSON output.

JSONINDENT(NO) is the default so that all key/value pairs belonging to the same report detail line will be written to the same line of the JSON output.

JSONINDENT(YES) indicates that all key/value pairs belonging to the same report detail line will be written to concurrent lines of the JSON output and indented beneath the opening and closing JSON object string braces ("{}").

```
{
    "000000001" :
    {"NAME" : "Daniel Ricciardo",
     "COUNTRY" : "Australia"
    }
   ,"000000002" :
    {"NAME" : "Lando Norris",
     "COUNTRY" : "United Kingdom"
    }
}
```

JSONLITERALS [**(YES** | **NO)**]
Applicable only to JSON report output, JSONLITERALS specifies whether or not *literal* values specified in the in the COLUMNS section are included as the "string" value in a key/value pair of the JSON output.

JSONLITERALS(NO) is default and will exclude column field *literal* values from the JSON output.

Note that the column entry header value is used as the "key" which is the same as the *literal* value by default.

JSONQUOTED [**(YES** | **NO)**]
Applicable only to JSON report output, JSONQUOTED specifies whether or not **all** values are to be treated as JSON strings and so enclosed in quotation marks (").

JSONQUOTED(YES) is default and so values are always enclosed in quotation marks.

JSONQUOTED(NO) indicates that only non-numeric values are treated as strings and so enclosed in quotation marks. Numeric values are **not** enclosed in quotation marks.

JSONSTRIPALL [**(YES** | **NO)**]
Applicable only to JSON report output, JSONSTRIPALL specifies whether or not leading and trailing blanks are to be stripped from all values. This is particularly relevant to quoted JSON string values where leading trailing blanks would be treated as part of the string value.

JSONSTRIPALL(NO) is default and so values are not stripped of leading and trailing blanks. Each value will be of a fixed length equal to the specified (or default) field width.

JSONSTRIPALL(YES) indicates that all JSON values are to be stripped of leading and trailing blanks.

LINESTRIP [**(YES** | **NO)**]
LINESTRIP specifies whether or not trailing blank characters are to be stripped from the lines of text written to the REPORT output.

LINESTRIP(NO) is default and so trailing blanks will be preserved when writing report lines. This is important when writing reports comprising only report detail lines (no page or report breaks) to be used as fixed length data input to another application.

NEWPAGE [**(YES** | **NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, NEWPAGE indicates whether the printed report will span multiple pages or will occupy a single page of unrestricted page depth.

NEWPAGE(YES) is default and will trigger a new page when the number of lines per page (page depth) value is reached. Note that REPORT utility may print blank filler lines at the end of a page in order to prevent multi-line report detail lines or control break lines belonging to a single control break definition from being split over 2 pages. (See option SPLITBREAK .)

NEWPAGE(NO) will ignore the specified or default page depth value and suppress the start of a new page. The report will comprise a single page with page heading lines at the start (if HEAD(YES) is set) and page footing lines at the end (if a FOOT section exists). If NEWPAGE(NO) is used, the printed report will never contain blank filler lines.

NOTOTALS
Equivalent to TOTALS(NO). See TOTALS option.

NUMBLANK [**(INCLUDE** | **EXCLUDE)**]
NUMBLANK specifies whether or not numeric field values in the report PRINT output that are displayed as blanks as a result of a BIEQUAL specification, are to be included in or excluded from column statistics calculations.

NUMBLANK(EXCLUDE) is default and so values appearing as blanks in columns containing numeric data will **not** be included in total, average, minimum and maximum statistics calculations performed for that column.

NUMDUP [**(INCLUDE**|**EXCLUDE)**]
NUMDUP specifies whether or not numeric field values in the report PRINT output that are a duplicate of the column value found on the previous detail line, are to be included in or excluded from column statistics calculations.

A duplicate column value is an *input-field* value that has not been reset and no new input value has been obtained before the detail line is written. This may occur when output is triggerred by a REPEAT record-type that corresponds to a secondary record segment, and values in the primary record segment are not reset.

NUMDUP(INCLUDE) is default and so duplicate numeric values are included in total, average, minimum and maximum statistics calculations performed for that column.

NUMTRUNC [**(YES**|**NO** [**INCLUDE**|**EXCLUDE**] [*char* | "*char*" | '*char*'] **)**]
Specifies whether truncation of displayed numeric values will occur. If not, it defines the character used to fill the display of a field containing numeric values that would be truncated, and also determines whether or not these values are included in or excluded from column statistics calculations. Number truncation may occur if the width of the display field is not sufficient to display all digits of the numeric value.

For NUMTRUNC(YES), the value will be abbreviated to fit within the display width. Number compression may result in the loss of the least significant digits. If these digits are non-zero and the value is not expressed with an exponent, then the value becomes an approximation and so is prefixed with the appropriate inequality symbol ("<" or ">"). If the display width is such that the numeric value cannot be abbreviated without loss of integrity, the display area will be filled with the number truncation character "*char*".

For NUMTRUNC(NO), if stripping of non-significant zero digits still does not allow the value to fit within the field display width, then the display area is filled with the number truncation charcater "*char*".

Number compression will perform the following steps until the number fits within the display width. Note that, for normalised floating point values that include an exponent, fractional digits are those belonging to the mantissa.

1. Strip non-significant leading and fractional zeroes.
2. Strip least significant, non-zero fractional digits and decimal point "." if no fractional digits remain. If the value does not contain an exponent, then the first character of the display area will be an inequality symbol.
3. For fixed-point values only (i.e. no exponent), strip least significant, whole number digits in multiples of 3 replacing them with "K", then "M", "G", "T", "P" respectively. If any of these stripped digits are non-zero, then the first character of the display area will be an inequality symbol.

INCLUDE and EXCLUDE indicates that numeric values that are replaced with the numeric truncation character "*char*" in the output, will be included in or excluded from column statistics calculations respectively. INCLUDE is default.

This applies to numeric fields defined as detail line column entries (COLUMNS), fields specified in page heading (HEAD) or footing (FOOT) print expressions, break key field definitions (BREAK), sort key field definitions (SORT), required field definitions (REQUIRED) and fields occuring in BREAK line print expressions.

NUMTRUNC(NO,INCLUDE,'*') is default indicating that numeric values that are too long to fit in the display field are **not** truncated, are included in statistics calculations and that asterisk characters ('*') are used to fill the field display area.

OLIM(*int***)**
OLIM specifies the maximum number of detail line report records (*int*) that may be written to the output dataset.

Once the number of output report detail lines reaches this limit, no further detail lines will be written and so end of report processing is triggered. The input of records/DB2 table rows or sorted detail lines (if a SORT section exists) will end.

OLIM(0) implies no output record detail line limit and is set by default when no OLIM option value is specified.

Note that the output detail line limit specified in the OLIM option will be overridden by an OLIM value entered as a parameter via the following:

◊ The Formatted Record Report panel
◊ The DB2 Report - Table/View panel
◊ The DB2 Report - SQL Query Control File panel
◊ The DB2 Report - SQL Query Statement panel
◊ The SMF Report panel
◊ The OLIM operand of the REPORT primary command

PAGEDEPTH(*int***)**
Applicable only to printed report output and ignored for CSV, JSON and XML output, PAGEDEPTH (or its synonym PD) specifies the number of lines (*int*) on each report page. If not specified, the page depth will be the value assigned by the PAGEDEPTH Data Editor option. (See "*PAGEDEPTH - SET/QUERY/EXTRACT Option*" in the "*FileKit Data Editor (SDE)*" manual.)

Note that the page depth value specified in the PAGEDEPTH option will be overridden by a PAGEDEPTH value entered as a parameter via:

- The Formatted Record Report panel
- The DB2 Report - Table/View panel
- The DB2 Report - SQL Query Control File panel
- The DB2 Report - SQL Query Statement panel
- The SMF Report panel
- The PAGEDEPTH operand of the REPORT primary command

PAGEPAD [**(AUTO** | **YES** | **NO)**]
PAGEPAD specifies whether or not blank lines are to be written to the last page of a PRINT output report to pad the page to the specified (or defaulted) PAGEDEPTH.

AUTO is the default and will pad the last page with blank lines only if it is **not** the first (and therefore only) page of the report. No additional page padding is necessary if a page footing has been defined via a report definition FOOT: section.

REPORT(DB2 | **SDE** | **SMF)**
REPORT specifies the type of input (DB2, SDE or SMF) from which the report is generated.

DB2	A DB2 result table.
SDE	A data set or HFS/ZFS file where fields are mapped by a copybook structure or via fields definitions in the MAP section.
SMF	An on-line SMF data set, or a data set containing SMF DUMP (IFASMFDL or IFASMFDP) output.

The REPORT option determines the format of the syntax specified in the INPUT section.

If the REPORT option is not specified, then the input type is determined by the report generation panel used to create the report, or via specification of a *xxx***-INPUT-BEG** / *xxx***-INPUT-END** operand pair on the REPORT primary command (where *xxx* is one of "DB2", "SDE" or "SMF"). If, however, the REPORT primary command is executed without specifying one of these operand pairs **and** the REPORT option is not specified, then the input type defaults to be SDE.

Note that, if the REPORT input type implied by the Utility panel, or via a *xxx***-INPUT-BEG** / *xxx***-INPUT-END** operand pair on the REPORT primary command, is different to that specified by the REPORT option, then only input field/operand values that are applicable to the REPORT option specification will be used. All other non-applicable operands will be silently ignored.

REXXCOMPOUND [**(YES** | **NO)**]
REXXCOMPOUND specifies whether or not the REXX variable names, generated for input-fields identified using a qualified field name, inherit the dot/period (".") field name qualifier separator character and so define a REXX compound symbol variable name.

When option FIELDNAME(LONG) is set (default) and a COMPUTE: and/or DISPLAY-EXIT: section REXX procedure exit exists, then the REPORT utility will assign input-field values to REXX variables of the same name as that used to identify the input-field (minus the record-type name). These variables may then be referenced in the REXX procedures. For example, an input-field identified by "SMF030_Identification.zJOBNAME" in the COLUMNS: or REQUIRED: section will assign the current value for this field to the REXX variable name "zJOBNAME".

If the input-field is identified using a qualified name (so that the name includes an owning group field name), then the name of the REXX variable will contain the "." (dot/period) qualifier separator, making it a REXX compound variable name. For example an input-field identified by "SMF030_EXCP.zEXP.zBSZ.zBSZLarge" will assign the current value for this field to REXX compound variable name "zEXP.zBSZ.zBSZLarge".

It is possible that use of a compound variable may cause problems if any of the qualified names in the tail are also used elsewhere in procedure as simple variable names. In this case REXXCOMPOUND(NO) may be specified to ensure that the REPORT utility uses only simple variable names. These names are still based on the input-field identifier, but use "_" (underscore) symbols in place of "." (dot/period) qualifier separator symbols. For example, the input-field identified by "SMF030_EXCP.zEXP.zBSZ.zBSZLarge" will assign the current value for this field to REXX simple variable name "zEXP_zBSZ_zBSZLarge".

REXXCOMPOUND(YES) is default and REXX compound variable names are possible.

SELECTJOIN [**(YES** | **NO)**]
Applicable only to SMF type input or SDE input of segmented records. SELECTJOIN specifies whether or not values for *input-fields* that belong to a secondary record segment whose record-type is **not** identified in a report definition REPEAT: section. are to be joined to the primary segment input.

For efficiency, the REPORT utility bypasses processing of secondary record segments that are not specified in the REPEAT: section, and instead joins the specified segment field values to the primary segment data.

SELECTJOIN(NO) will stop this default action so that secondary segments of all record-types identified in the report definition are processed, and field values are no longer joined to the primary segment.

The effect of SELECTJOIN can only been witnessed if BROWSE output is requested on invocation of the REPORT utility (via the utility panels or primary command).

SHORTHEADERS [**(YES** | **NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, SHORTHEADERS specifies whether or not the shortened form of generated (**default**) column headers is used.

In the COLUMNS section, the REPORT utility will generate a column header for any column definition where no column header text is specified. This default header includes the name of the field used to define the column. For headers generated for columns defined by input fields, this header may also include text obtained from any REMARK comments assigned to the input field definition.

Note that a field definition REMARK comment text may exist if the FileKit SDO record mapping structure has been created using the Direct Definition form of the **CREATE STRUCTURE** primary command. See "*FileKit SDE Data Editor*" reference manual for details of the CREATE STRUCTURE command.

SHORTHEADERS(NO) is default and so the REPORT utility will construct default column headers using REMARK comment text if available. The generated header text is split over several lines in order to best preserve the column data width. However, if a word in the REMARK text is longer than the column data width, then the width of the column is extended accordingly. The name of the input field will occupy the last line of the default column header.

SHORTHEADERS(YES) will use the shortened form of the default column header which is simply the name of the field from which the column is defined.

Field definitions in FileKit SMF record mapping SDO structures are assigned REMARK text. Therefore, SMF reports will by default contain column headers that describe the column contents.

SHORTSTATS [**(YES** | **NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, SHORTSTATS specifies whether or not a shortened form of a statistics value will be displayed if the length of the statistics value is greater than the field display area width.

If SHORTSTATS(NO) is set and the statistics value exceeds the width of the field display area, the display field will be filled with the default number truncation filler character ("*"). See option NUMTRUNC.

If SHORTSTATS(YES) is set (the default), a statistics value that exceeds the width of the field display area will be displayed in a shortened form that fits within the field display area.

For a time/duration, the value is truncated on the right preserving the left, most significant characters. A greater than symbol (">") prefix is added to indicate that the value has been truncated. For example, a total of duration field values may be "138:26:22" (138 hours, 26 minutes, 22 seconds) but the width of the display field is only 5. Therefore, the shortened value would display as ">138:".

The sortened form of a numeric statistics value is an approximation of the original numeric value. The number displayed may include a decimal point and will have a multiplier suffix of either **K**, **M**, **G**, **T** or **P** representing a number of thousands (10^3), millions (10^6), billions (10^9), trillions (10^{12}) or quadrillions (10^{15}) respectively. The length of the number and the suffix used will depend on the number of digits in the original value and the width of the field display area. Unless the shortened value exactly matches the original value, the shortened value will also have a greater than symbol (">") prefix indicating that the original value is in excess of the shortened value.

If the shortened number also exceeds the width of the field display area field, then the field will be filled with the number truncation filler symbol. The following table demonstrates how a statistics value will appear when shortened for different display field area widths.

Display	Values					
10	1000000	1100000	1183000	3451230	9893000000	9893400000
6	1000K	1100K	1183K	>3451K	9893M	>9893M
5	1000K	1100K	1183K	>3.4M	9893M	>9.8G
4	1M	1.1M	>1M	>3M	>9G	>9G
3	1M	>1M	>1M	>3M	>9G	>9G
2	1M	**	**	**	**	**

SMFDATEHI(*timestamp | -days***)**
Applicable only to SMF type input (not DB2 or SDE), SMFDATEHI specifies a complete or partial absolute timestamp (*timestamp*), or a negative number of days (*-days*) that corresponds to a timestamp value which is relative to the current date. SMF records with a timestamp later than this upper limit will be excluded from REPORT processing.

Absolute and Relative timestamp specifications are described in detail by timestamp values under *"Record Filtering"*.

The start of every SMF record contains a common header which includes a timestamp (date and time) at which the record was written to the SMF log (zTME). If a SMFDATEHI value is specified, only those SMF records with a timestamp earlier than or equal to this high date and time will be passed on to SMF content match criteria record filtering.

An absolute timestamp specification may be truncated to a minimum of 5 bytes ("*yyyy/*") in which case the truncated numeric digits will be set to "9". For example, "**DATEHI(2019/09/22 18)**" is treated as "**DATEHI(2019/09/22 18:99:99.99)**".

A relative timestamp, specified as number of days before the current date, will correspond to a date only. For example, if the current date is **2020/03/05** then "DATEHI(**-5**)" would be equivalent to "**DATEHI(2020/02/29 99:99:99.99)**" since 2020 is a leap year.

Note that the upper limit timestamp specified in the SMFDATEHI option will be overridden by the high date and time values entered as parameters via the following:

◊ The SMF Report panel
◊ The DATEHI operand of the REPORT primary command

SMFDATELO(*timestamp | -days***)**
Applicable only to SMF type input (not DB2 or SDE), SMFDATELO specifies a complete or partial absolute timestamp (*timestamp*), or a negative number of days (*-days*) that corresponds to a timestamp value which is relative to the current date. SMF records with a timestamp earlier than this lower limit will be excluded from REPORT processing.

Absolute and Relative timestamp specifications are described in detail by timestamp values under *"Record Filtering"*.

The start of every SMF record contains a common header which includes a timestamp (date and time) at which the record was written to the SMF log (zTME). If a SMFDATELO value is specified, only those SMF records with a timestamp later than or equal to this low date and time will be passed on to SMF content match criteria record filtering.

An absolute timestamp specification may be truncated to a minimum of 5 bytes ("*yyyy/*") in which case the truncated numeric digits will be set to "0". For example, "**DATELO(2018/09)**" is treated as "**DATELO(2018/09/00 00:00:00.00)**".

A relative timestamp, specified as number of days before the current date, will correspond to a date only. For example, if the current date is **2019/11/13** then "DATELO(**-28**)" would be equivalent to "**DATELO(2019/10/16 00:00:00.00)**".

Note that the lower limit timestamp specified in the SMFDATELO option will be overridden by the low date and time values entered as parameters via the following:

◊ The SMF Report panel
◊ The DATELO operand of the REPORT primary command

SMFJOBNAME(*jobname, ...***)**
Applicable only to SMF type input (not DB2 or SDE), SMFJOBNAME specifies one or more comma separated job name search values (*jobname*) which are used to perform **SMF Record Job Name matching** for input record filtering.

A *jobname* value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*. Unless *jobname* contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the *jobname* value will be truncated or padded with blanks to a length of 8 characters. Furthermore, if no percent ("%") or asterisk ("*") wildcards are specified and *jobname* is an **unquoted** or **quoted** string, then all alpha characters in the string will be upper cased.

A number of SMF record types contain a job name field (**zJobName**) at a fixed location within the record data. This fixed position may be different for each of the SMF record types. The following SMF record types are those that contain a zJobName field:

004	010	017	025	034	040	061	064	067	080
005	014	018	026	035	042	062	065	068	110
006	015	020	030	036	060	063	066	069	118

If an SMF record zJobName field contains a match on **any** of the supplied *jobname* values, then SMF Record Job Name matching will return a true result (1). Otherwise, if no match is found for any of the supplied *jobname* values or the SMF record does not contain a zJobName field, then a false result (0) is returned.

SMF Record Job Name matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR". (see SMFLOGIC).
3. All other specified SMF content match criteria each return a true result. (see SMFFIND, SMFSID, SMFTYPES and SMFUSERID)

Note that SMF Record Job Name matching and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a job name specification exists, then error message ERR066E is returned.

In the following example, a true result will be returned if the SMF record has a zJobName field that specifically contains a job name "RSHD", contains a job name beginning with "GIM" or any job name of length 5.

```
Job Name> RSHD, GIM*, %%%%%
```

Job names specified in the SMFJOBNAME option will be overridden by job name values entered as parameters via the following:

- The SMF Report panel
- The JOBNAME operand of the REPORT primary command

SMFLOGIC (OR | AND)

Applicable only to SMF type input (not DB2 or SDE), SMFLOGIC specifies the logical operation (**AND** or **OR**) to be used when determining the result of content match criteria record filtering.

The logical operation is used to combine the Boolean values (true or false) returned by each of the specified content match criteria elements:

- Unformatted Record Find String matching (FIND)
- SMF Record Job Name matching (SMFJOBNAME)
- SMF Record System Id matching (SMFSID)
- SMF Record Type matching (SMFTYPES)
- SMF Record User Name matching (SMFUSERID)

Content matching criteria elements may be specified as options in the report definition input, and/or passed to the REPORT utility via command line operands or panel input fields.

The "AND" or "OR" logical operation is performed between each of the Boolean values returned by the specified content matching criteria to produce an overall true (1) or false (0) result. If the overall result is true, the record satisfies the content match criteria and is passed to REPORT generation processing.

If logical operation **AND** is used then the result returned by **all** of the content checking criterion elements specified for the current REPORT execution must be 1 (i.e. true). If logical operation **OR** is used then **only one** of the values returned by the content checking criterion elements must be 1 (true) in order to return a true result for the record.

Note that other SMF record filtering controlled by high date (SMFDATEHI) / low date (SMFDATELO) thresholds and input record limit (ILIM), does not form part of the content checking criteria and so is not affected by the logical operation.

The default value for SMFLOGIC is "OR". However, note that the logical operation specified by the SMFLOGIC option will be overridden by the logical operation value entered as a parameter via the following:

- The SMF Report panel
- The LOGIC operand of the REPORT primary command

SMFONLINE [**(YES | NO)**]

Applicable only to SMF type input (not DB2 or SDE), SMFONLINE specifies whether or not SMF input records are being processed directly from an SMF log data set (**YES**) or from an SMF archive data set (**NO**).

Note that the REPORT utility does not support processing SMF records directly from the System Logger.

Unlike records written to an archive data set by the SMF DUMP (IFASMFDL and IFASMFDP) utilities, records in an SMF log dataset are prefixed by an extra **4-byte record descriptor word (RDW)** and so record-type field mapping must be offset by 4 bytes. The OFFLINE/ONLINE specification will determine whether this offset is to be applied by the REPORT utility.

Beware that any application, including the REPORT utility, that processes records directly from an online SMF log data set, may prevent successful execution of an IFASMFDP CLEAR operation (usually triggered by the IEFU29 exit). This is because the IFASMFDP CLEAR operation requires exclusive access to the SMF dataset.

The default value for SMFONLINE is NO. However, the SMFONLINE option value will be overridden by the Format value entered as a parameter via:

- The SMF Report panel
- The ONLINE/OFFLINE operand of the REPORT primary command

SMFSID(*sid, ...***)**
Applicable only to SMF type input (not DB2 or SDE), SMFSID specifies one or more comma separated system identification search values (*sid*) which are used to perform **SMF Record System Id matching** for input record filtering.

A *sid* value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*. Unless *sid* contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the *sid* value will be truncated or padded with blanks to a length of 4 characters. Furthermore, if no percent ("%") or asterisk ("*") wildcards are specified and *sid* is an **unquoted** or **quoted** string, then all alpha characters will be upper cased.

All SMF record types contain a system identifier field **zSID** in the record header. If an SMF record zSID field contains a match on **any** of the supplied *sid* values, then SMF Record System Id matching will return a true result (1). Otherwise a false result (0) is returned.

SMF Record System Id matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR". (see SMFLOGIC).
3. All other specified SMF content match criteria each return a true result. (see SMFFIND, SMFJOBNAME, SMFTYPES and SMFUSERID)

Note that SMF Record System Id matching and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a System Id specification exists, then error message ERR066E is returned.

In the following example, a true result will be returned if the zSID field contains a system id value "XS1", a value beginning with "S0" followed by any single character followed by "1", or a value of up to 4 characters in length ending in "Z".

```
SID('xs1', S0%1, '*Z')
```

System names specified in the SMFSID option will be overridden by system identification values entered as parameters via the following:

- The SMF Report panel
- The SID operand of the REPORT primary command

SMFTYPES(*{rectype | rectype:rectype | {rectype-subtype | rectype#subtype} }, ...***)**
Applicable only to SMF type input (not DB2 or SDE), SMFTYPES specifies one or more comma separated SMF record type identification values (*rectype*, *rectype*:*rectype*, *rectype-subtype* or *rectype#subtype*) which are used to perform **SMF Record Type matching** for input record filtering.

A description of each of the different SMF record type identification values is documented in SMF Type Values under *"Record Filtering"*.

All SMF record types contain an SMF record type field **zRTY** and some also contain a sub-type field **zSTY** in the record header. If an SMF record contains a match on any of the supplied SMF record type identification values, then SMF Record Type matching will return a true result (1). Otherwise a false result (0) is returned.

SMF Record Type matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR". (see SMFLOGIC).
3. All other specified SMF content match criteria each return a true result. (see SMFFIND, SMFJOBNAME, SMFSID and SMFUSERID)

Note that TYPES and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a TYPES specification exists, then error message ERR066E is returned.

In the following example, a true result will be returned if the input SMF record type (zRTY field value) is 42 (any sub-type), or if the SMF record type is 119 with sub-type (zSTY field value) of 21.

```
TYPES(42, 119#21)
```

Note that all values specified in the SMFTYPES option will be overridden by types values entered as parameters via the following:

- The SMF Report panel
- The TYPES operand of the REPORT primary command

SMFUSERID(*username, ...***)**
Applicable only to SMF type input (not DB2 or SDE), SMFUSERID specifies one or more comma separated user name search values (*username*) which are used to perform **SMF Record User Name matching** for input record filtering.

A *username* value may be specified as an **unquoted**, **quoted** or **character literal** string and may contain one or more **wildcard** characters as described by search values under *"Record Filtering"*. Unless *username* contains an asterisk ("*") wildcard, which represents zero or more occurrences of any character, then the *username* value will be truncated or padded with blanks to a length of 8 characters. Furthermore, if no percent ("%") or asterisk ("*") wildcards are specified and *username* is an **unquoted** or **quoted** string, then all alpha characters in the string will be upper cased.

A number of SMF record types contain a user name field **zUserId** at a fixed location within the record data. This fixed position may be different for each of the SMF record types. The following SMF record types are those that contain a zUserId field:

004 005 006 010	014 015 017 018	020 025 026	030 032 034	035 036 040	042 060 061	062 063 064	065 066 067	068 069 080	110 118 119

If an SMF record zUserId field contains a match on **any** of the supplied *username* values, then a true result (1) will be returned for **SMFUSERID** matching. Otherwise, if no match is found for any of the supplied *username* values or the SMF record does not contain a zUserId field, then a false result (0) is returned.

SMF Record User Name matching is one of the content match criteria for SMF record filtering. If a true result is returned, then the SMF record will be passed for REPORT processing only if at least one of the following is true:

1. No other SMF content match criteria is specified.
2. The SMF content match criteria logical operation is "OR". (see SMFLOGIC).
3. All other specified SMF content match criteria each return a true result. (see SMFFIND, SMFJOBNAME, SMFSID and SMFTYPES)

Note that SMF Record User Name matching and other content match criterion are invalid if a filter expression is provided via the FILTER section of the report definition. If both a FILTER section and a user name specification exists, then error message ERR066E is returned.

In the following example, a true result will be returned if the SMF record has a zUserId field that contains a user name of any length up to a maximum of 8 characters ending with 1, or a user name beginning with "ABC" followed by any single character followed by "DEFG".

```
SMFUSERID(*1, ABC%DEFG)
```

User names specified in the SMFUSERID option will be overridden by user name values entered as parameters via the following:

- The SMF Report panel
- The USER operand of the REPORT primary command

SPLITBREAK[**(YES**|**NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, SPLITBREAK specifies whether or not the printed lines belonging to a control break may be split over a new page.

SPLITBREAK(NO) is default and so the REPORT utility will determine whether or not the break lines that follow a control group all fit in the lines remaining on the current page. If not, the lines remaining on the page are left blank and the break lines are printed on a new page.

SPLITBREAK(YES) will allow the control break lines to span more than 1 page.

If a block of control break lines are split over 2 pages, then any re-print of break HEADING lines that would occur as a result of a break definition REPEAT option, will be suppressed during break line printing. Heading lines will still be re-printed if the page throw occurs during detail line printing.

STATS[**(YES**|**NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, STATS specifies whether or not generation of statistics values (totals, averages, etc.) will occur for statistics columns.

STATS(YES) is default and so statistics values will be generated for the statistics columns as appropriate.

Statistics columns are named in the STATISTICS section or otherwise identified as columns of numeric data type defined in the COLUMNS section. By default, statistics values will be generated for each statistics column and displayed beneath the column's values at each control break and at the end of the report.

Note that STATS(NO) will **not** suppress the text printed for TOTAL, AVERAGE, NZAVERAGE, MAXIMUM, MINIMUM and NZMINIMUM break lines specified in a BREAK section.

The report line generated to underline statistics columns and so separate the column values from the statistics values, will be empty. If required, this blank line may be suppressed using option BRKULINE(NO).

SUMMARY [**(YES** | **NO)**]
Applicable only to printed report output and ignored for CSV, JSON and XML output, SUMMARY specifies whether or not a summary report is to be generated.
SUMMARY(NO) is deafult and so a full report of the detail lines is produced.

Option SUMMARY(YES) is similar to option DETAIL(0) in that all report detail lines will be suppressed but all page headings, column headings, break lines and page footings are unaffected and will be printed as normal. However, unlike DETAIL(0), SUMMARY(YES) will set SPACEBEFORE(0) and SPACEAFTER(0) for all BREAK section break definitions, overriding any explicit specification. Similarly, the REPEAT option to output break HEADING lines at the start of each new page is ignored for summary output.

Therefore, no blank lines will appear before or after each block of break lines in the printed output unless blank lines are printed as a result of trailing null lines in a break FOOTING specification. e.g.
`FOOTING("End of statistics for:" RESNAME <NEWLINE>)`

TAGLEN(*int***)**
Applicable only to JSON and XML output and ignored for CSV and printed report output, TAGLEN limits the XML tag or JSON name generated from the COLUMN definition header to a maximum of length *int* characters. The *int* value may be between 1 and 235.

By default, the REPORT utility restricts the length of XML tag names and JSON names (in a name/value pair) to a maximum of 235 characters, i.e. TAGLEN(235).

TOTALS [**(YES** | **NO)**] | **NOTOTALS**
Applicable only to printed report output and ignored for CSV, JSON and XML output, TOTALS specifies whether or not generated TOTAL break lines are to be automatically generated for control breaks, including the default end-of-report #GRAND totals control break.

TOTALS(YES) is default and is equivalent to specifying both options BRKTOTALS(YES) and GRANDTOTAL(YES). TOTALS(NO) or NOTOTALS is equivalent to specifying both options BRKTOTALS(NO) and GRANDTOTAL(NO).

XMLINDENT [**(YES** | **NO)**]
Applicable only to XML report output, XMLINDENT specifies whether or not each XML tagged report field value is to appear on its own line of the XML output.

XMLINDENT(NO) is the default so that all XML tagged values belonging to the same report detail line will be written to the same line of the XML output.

```
<?xml version="1.0"?>

<INPUT>
 <FileKit_Report>  <NAME>Daniel Ricciardo</NAME> <COUNTRY>Australia</COUNTRY> </FileKit_Report>
 <FileKit_Report>  <NAME>Lando Norris</NAME> <COUNTRY>United Kingdom</COUNTRY> </FileKit_Report>
</INPUT>
```

XMLINDENT(YES) indicates that all XML tagged values belonging to the same report detail line will be written to concurrent lines of the XML output and indented within the report line tags.

```
<?xml version="1.0"?>

<INPUT>
 <FileKit_Report>
   <NAME>Daniel Ricciardo</NAME>
   <COUNTRY>Australia</COUNTRY>
 </FileKit_Report>
 <FileKit_Report>
   <NAME>Lando Norris</NAME>
   <COUNTRY>United Kingdom</COUNTRY>
 </FileKit_Report>
</INPUT>
```

XMLLITERALS [**(YES** | **NO)**]
Applicable only to XML report output, XMLLITERALS specifies whether or not *literal* values specified in the in the COLUMNS section are included as the tagged values in the XML output.
XMLLITERALS(NO) is default and will exclude column field *literal* values from the XML output.

Note that the column entry header value is used as the XML tag name which is the same as the *literal* value by default.

XMLSTRIPALL [**(YES** | **NO)**]
Applicable only to XML report output, XMLSTRIPALL specifies whether or not leading and trailing blanks are to be stripped from all values.

XMLSTRIPALL(NO) is default and so values are not stripped of leading and trailing blanks. Each value will be of a fixed length equal to the specified (or default) field width.

XMLSTRIPALL(YES) indicates that all XML values are to be stripped of leading and trailing blanks.

OUTPUT

Overview:

The OUTPUT section specifies the destination of the generated report.

The report destination specified by the OUTPUT section may be a new or existing data set, library member or HFS/ZFS file. To prevent truncation of record data, the defined LRECL of an existing data set or library member should be large enough to contain the longest output record generated by the REPORT utility.

The REPORT utility identifies the specified output data object as being either a valid DD name, a DSN with or without a library member reference, or an HFS/ZFS file id path. If the output data object already exists, then its contents will be replaced with the generated report output.

When the output data object is a new sequential or library data set, then the REPORT utility will catalog the new DSN with attributes: LRECL=16384, RECFM=VB, BLKSIZE=0 (SMS system determined blocksize) and SPACE=(TRK,(5,5)). UNIT, MGMTCLAS, DATACLAS and STORCLAS attributes will also be set if default values have been set in the FileKit System INI file.

When the output data object is a new HFS/ZFS file, then the REPORT utility will allocate the specified file path with attributes: PATHDISP=(KEEP,KEEP) and PATHMODE=(SIRWXU,SIRWXG,SIROTH).

The REPORT utility output data object is determined in the following order of precedence:

1. If the REPORT utility is executed using the REPORT primary command and the OUTDD (or OUTPUTDD) operand is specified, then output is to the DD name specified by operand OUTDD.
2. If DD name SDEOUT is allocated, DD=**SDEOUT**
3. The data object (*ddout*, *dsname* or *fileid*) specified by the OUTPUT section of the report definition.
4. If executing in batch, DD=**SDEPRINT**. (Allocation of SDEPRINT is mandatory for batch execution.)
5. For FileKit foreground execution only, an unsaved, in-storage file assigned a DSN "*userpfx*.REPORT.D*yyyyddd*.T*hhmmss*.TXT", where *userpfx* is the DSN prefix associated with the current FileKit user and *yyyyddd* and *hhmmss* is the current Julian date and time respectively. The in-storage data is displayed in a Text Editor view and may be saved to a specific DSN using "SAVE *dataset-name*".

Therefore, output report destination specified in the OUTPUT section is ignored if either the REPORT primary command is executed with operand OUTDD (or OUTPUTDD), or DD name SDEOUT is allocated.

Examples:

Example 1 - DD Output:

```
OUTPUT:
   DD=SMFCSV
```

The above example specifies that the report output is to be written to the data set, library member or HFS/ZFS file allocated to DD name SMFCSV.

Example 2 - Data Set Output:

```
OUTPUT:
   CBL.SMFRACF.REPORT.D2021132
```

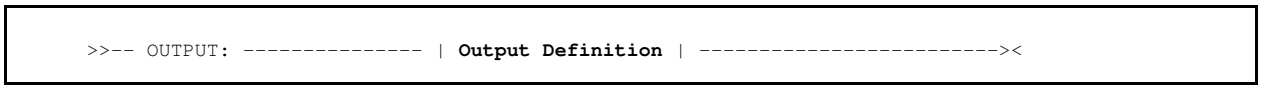
The above example specifies that the report output is to be written to the new or already cataloged data set "CBL.SMFRACF.REPORT.D2021132".

Example 3 - HFS/ZFS File Output:

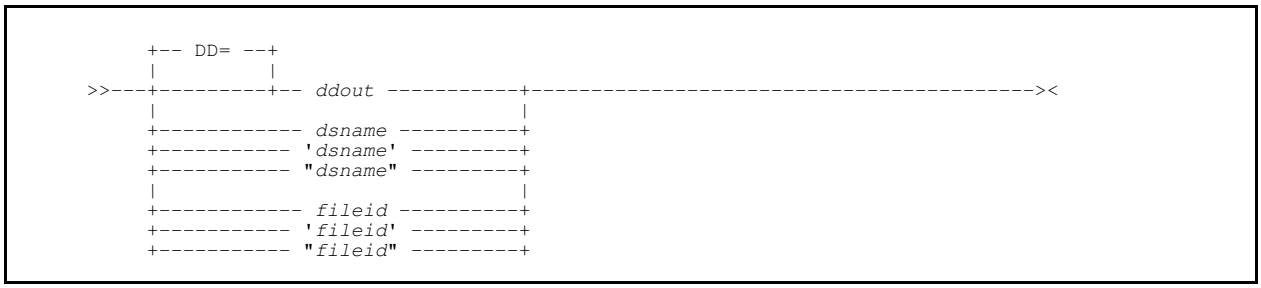
```
OUTPUT:
   "/XS01/home/john/TCPIP-Connect-Report.txt"
```

The above example specifies that the report output is to be written to the new or existing HFS/ZFS file "TCPIP-Connect-Report.txt" in directory "/XS01/home/john".

Syntax:



Output Definition:



Parameters:

[**DD=**]*ddout*
: Specifies the DD name (*ddout*) that is allocated to the data set, library member or HFS/ZFS file to which the generated output report will be written.

dsname | '*dsname*' | "*dsname*"
: Specifies an unquoted or quoted data set name or library name and member (*dsname*) to which the generated output report will be written.

fileid | '*fileid*' | "*fileid*"
: Specifies an unquoted or quoted HFS/ZFS file path to which the generated output report will be written.

REPEAT

Overview:

The REPEAT section specifies the record-type mapping(s) of the records (or record segments) that trigger output of a report detail line.

Every input record is assigned a named record mapping (record-type). The record-type is selected from the available record-types defined in the SDE Data Editor structure (SDO) based on USE WHEN selection criteria or else the DEFAULT record-type. The structure is either provided by the user on input or automatically generated by the REPORT utility for SMF record or DB2 table processing.

If input records are segmented, then each segment of an input record is mapped by a separate record-type mapping. The record-type used to map a segment is determined based on USE WHEN selection criteria applied to data in the segment or in a previous segment. The first segment of a segmented record is mapped by a **primary** (or base) record-type. All other segments are mapped by a **secondary** record-type. For the purpose of terminology, a non-segmented input record or DB2 table row may be considered to be mapped entirely by a **primary** record-type.

By default, the REPORT utility will trigger output of a new detail line immediately upon input of the second or subsequent record, record segment or DB2 table row to be mapped by a **primary** record-type. The values of fields used in the report output record will reflect the values obtained from the previous input record, record segments or DB2 table row.

Depending on the structure of input record data, it is possible for the value of an input field to be updated more than once between 2 occurrences of a report output record. Since the output record reflects the current value of an input field, the report would not include all the other values assigned to the field since the last output record was processed. To ovecome this, a REPEAT section should be specified to override the REPORT utility default.

Instances where field values may not be reported and so a REPEAT section is necessary, are as follows:

1. Input field values are obtained from multiple records of different primary record-types. In this case, output should be delayed until a record of a specific record-type is read.

2. Input field values are obtained from a potentially repeating segment of secondary record-type within the same record. In this case, output should be forced following input of each segment of the repeating secondary record-type.

Examples:

Example 1 - SMF Record Input with Repeating Secondary Segment:

```
COLUMNS:
  SMF030_Identification.zJOBNAME     ('Job|Name'  CENTRE)
  SMF030_Identification.zSIT         ('Job|Start')
 '|'                                 ('')
  SMF030_EXCP.zEXP.zDDN              ('DDName')
  SMF030_EXCP.zEXP.zBLK              ('EXCP|Blocks'   RIGHT)  8  RIGHT
  SMF030_EXCP.zEXP.zBSZ.zBSZLarge    ('Largest|Block' RIGHT)  6  RIGHT
  SMF030_EXCP.zEXP.zCUA              ('Dev#')

REPEAT:
  SMF030_EXCP
```

In the above example, a number of record-types are used to map the individual sections (segments) of input SMF type 30 records.

For each SMF 30 record, there are potentially many occurrences of the EXCP segments mapped by secondary record-type "SMF030_EXCP". Therefore, in order to output a report detail line containing the DDname (zDDN), Number of EXCP blocks (zBLK), largest block (zBSZLarge) and device number (zCUA) recorded for each occurrence of an EXCP segment, the REPEAT section is used to trigger a new report line for each segment with record-type "SMF030_EXCP".

The "zJOBNAME" and "zSIT" columns are defined in the primary segment record-type. These columns will contain the same values in each report line generated from a single SMF type 30 record. See the RESET section for description on resetting non-repeated input field values to null.

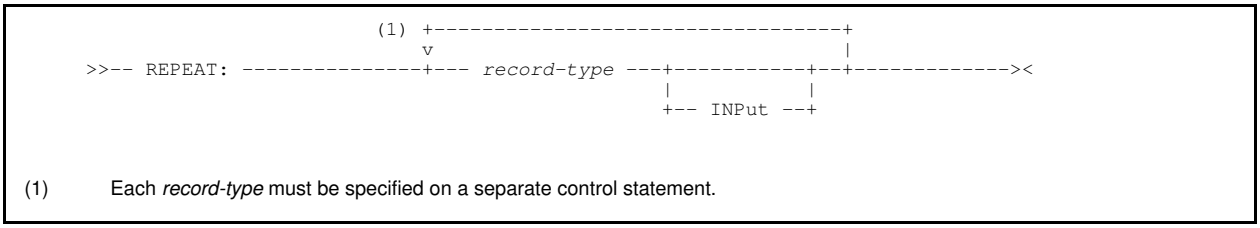
Example 2 - Multiple Non-Segmented Records:

```
COLUMNS:
   TRACK.TRACK-NUM      'Track|Number'      RIGHT
   ARTIST.ARTIST        'Artist'         30
   ALBUM.ALBUM          'Album'          30
   TRACK.NAME           'Track|Name'     30
   TRACK.TOTAL-TIME     'Track|Duration'
   TRACK.RELEASE-YYYY   'Year|Released'     RIGHT

REPEAT:
   TRACK
```

In the above example, the input records are arranged in a hierarchy so that a record for a recording artist (mapped by record-type "ARTIST") is followed by one or more records, each detailing an album recorded by that artist (mapped by record-type "ALBUM"). Each ALBUM record is itself followed by one or more records, each detailing a track on the album (mapped by record-type "TRACK").

Specification of the "TRACK" record-type in the REPEAT: section will delay output of a report detail line until after a "TRACK" record has been read and processed. The name of the artist (ARTIST) and the name of the album (ALBUM) are set from the last ARTIST and ALBUM records to have been processed and so will be repeated for all tracks in the report output that belong to the same album.

Syntax:



(1) Each *record-type* must be specified on a separate control statement.

Parameters:

record-type
Identifies the type of input record or record segment that will trigger output of a new report detail line.

A *record-type* is the name of a mapping structure used to map an input record (or record segment). The *record-type* must match the first qualifier of one of the *input-field* identifiers used to define a column or work field in the COLUMNS or REQUIRED sections respectively.

One or more *record-type* names may be specified, each on separate control statements following the REPEAT section header in the report definition input.

INPUT
Applicable only to input records that are segmented (e.g. SMF type input), INPUT indicates that the specified segment *record-type* is repeating but must **not** trigger output of a new report detail line.

Where a secondary segment *record-type* repeats within a record but is not selected as one of the segments for which an output detail line is written, then field values are obtained from the **first** occurrence of the repeating segment only. The INPUT operand allows fields in a repeating segment to be re-evaluated for each occurrence of the segment type, but without triggering output of a new report detail line. Therefore, the field values of the repeating segment when a report detail line is written, will be those obtained from the **last** occurrence of a segment encountered before the detail line is triggered.

REQUIRED

Overview:

The REQUIRED section is used to identify additional input record or computed field names from which values will be extracted.

By default, the REPORT Utility will only extract values for record input fields that have been specified as column *fieldname* identifier in a COLUMNS section column definition. However, there may be instances where an input record field or computed field value is required for report processing but is not to be included in the report detail lines. In these cases, the input record field or compute field must be defined in the REQUIRED section. The values of input record fields that are **not** identified in either the COLUMNS or REQUIRED sections will not be extracted from the input records.

If an input record field or computed field is already used as a column definition *filedname* identifier, then it does not need to be specified in the REQUIRED section.

The following report definition sections may reference input record fields and computed fields that are not necessarily identified as a column definition source:

BREAK	The *fieldname* for which a change in value will trigger a control break in the printed report.
BREAK FOOT HEAD	As a print expression element. An input record field or a compute field may be specified to represent a variable fragment of text in report page headings, page footings and control break lines.
COMPUTE	As a variable in a REXX expression. Specified as *input-field*, the input field value may be used to resolve the value of a *compute-field*
SORT	The *fieldname* on which output report detail lines will be sorted.

Note that, where a computed field is referenced in one of these sections but not defined in either of the COLUMNS or REQUIRED sections, then a definition for the compute field with default width, data-type and alignment values will be generated automatically in the REQUIRED section.

Examples:

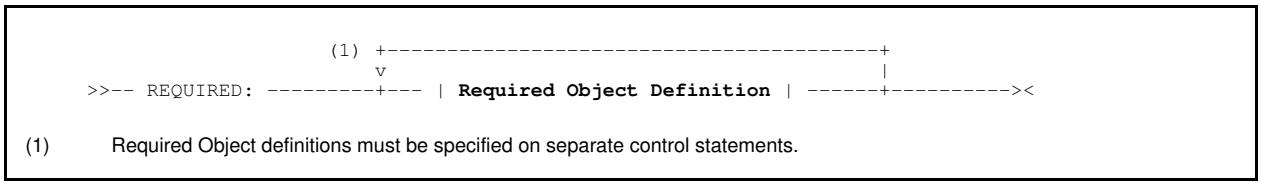
```
REQUIRED:
  SMF119#02_TCP_Statistics.zTME                        10
  SMF119#02_TCP_Connection_Termination.zConnectStart
  SMF119#02_TCP_Connection_Termination.zConnectEnd

HEAD:
  "TCP/IP Connections Reported on:" zTME

COMPUTE:
  DURATION = Secs2Time( Time2Secs(zConnectEnd) - Time2Secs(zConnectStart) )

COLUMNS:
  SMF119#02_TCP_Connection_Termination.zRName          ('RESOURCE')
  :DURATION           ('CONNECTION|DURATION|HHH:MM:SS.SS' RIGHT) 12 RIGHT
```

The above example indicates to the REPORT utility that in addition to input record field "zRName", which is included as a column in report detail lines, it must also extract the values for input record fields "zTME", "zConnectStart" and "zConnectEnd".

The 16-character timestamp value in "zTME" is truncated at 10 characters so that only the date portion is extracted (e.g. 2020/01/23). The 10 character value of "zTME", extracted from the first output record to be processed on a new printed report page, will be substituted in place of "zTME" in the page heading.

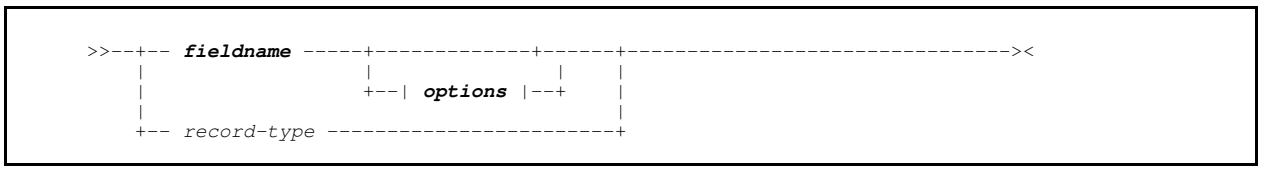
For each output record, the prevailing values for "zConnectStart" and "zConnectEnd" are used in the COMPUTE section to resolve the value of *compute-field* "DURATION". This is an elapsed time (format "*hh:mm:ss.ttt*" which is reported as a column value in in the detail line.

Syntax:

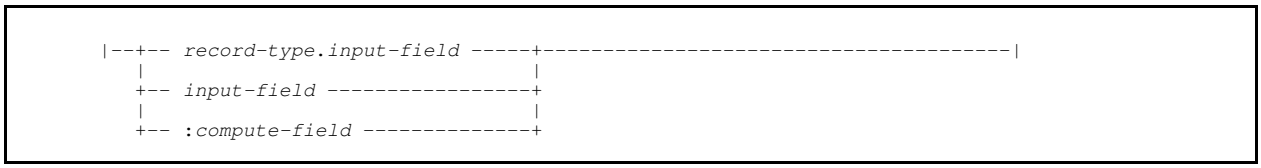


(1) Required Object definitions must be specified on separate control statements.

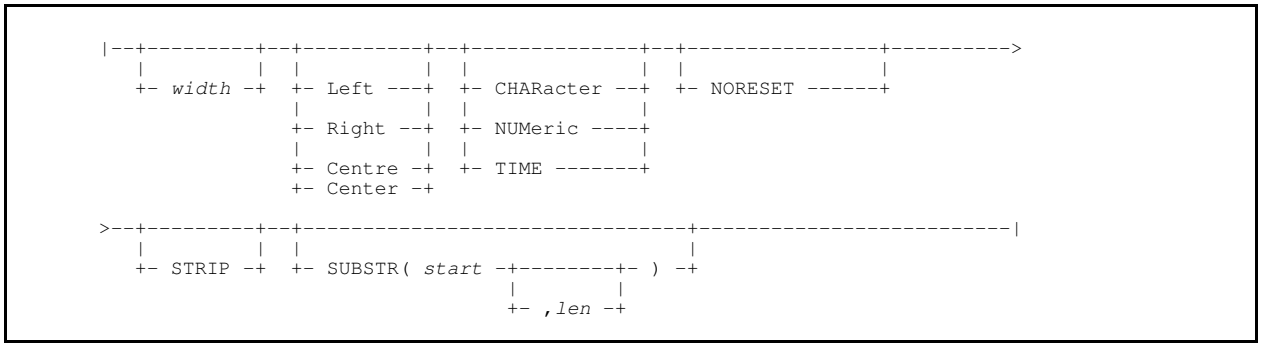
Required Object Definition:



***fieldname*:**



***options*:**



Synonyms:

REQUIRED	REQUIRE

Parameters:

fieldname

A *fieldname* identifies the name of an input record field or computed field that may be referenced in other report sections and from which values will be extracted.

Field values may change after processing the current input record or record segment. Therefore, *fieldname* may have a different value in each report output record used to generate the report.

A *fieldname* must be one of the following field types:

input-field

This *fieldname* format identifies a field whose values are extracted from within the input records.

Use of *input-field* without a preceding *record-type* specification applies only when only one record-type structure exists for mapping the input record or input is a DB2 table. In these cases *input-field* should be used.

See description of input record fields for details on *input-field* specification.

record-type.input-field
Like *input-field* alone, this *fieldname* format identifies a field within the input records.

Where the FileKit SDO structure associated with the input records contains more than one record-type definition, then the *record-type* name must be included before the *input-field* specification to identify the record-type mapping to which the field belongs. The *record-type* and *input-field* name specifications must be separated by a dot/period (".") character.

Note that when reporting on SMF records, the REPORT utility will dynamically generate an SDO structure to map the required SMF record types. This structure will always contain more than one record-type and so a *record-type* name is mandatory on an SMF input record field specification. Furthermore, the *record-type* names specified in the COLUMNS and REQUIRED sections of an SMF report definition input, are used by the REPORT utility to identify which SMF record-type mappings are required to build the dynamic SDO structure. See publication "*FileKit SMF Utilities*" for SMF record-type and field names.

Use of *record-type* is unnecessary where the SDO structure contains only one record-type definition.

:*compute-field*
This *fieldname* format identifies a field whose value is computed using REXX statements specified in the COMPUTE section.

To distinguish it from an *input-field*, a *compute-field* must be specified with a colon (":") symbol prefix. This prefix character is used only to identify the field name as a *compute-field* and is **not** part of the *compute-field* name itself. e.g. A column name referenced by a computed field name "MyValue" is identified using ":MyValue" in the required field definition control statement.

See description of computed fields for details on *compute-field* specification.

record-type
Applicable only to SMF record reporting, *record-type* specifically identifies the name of a record segment record-type definition found in the general SDO structure that maps all segments of a particular SMF record type or sub-type. See publication "*FileKit SMF Utilities*" for the segment record-type names of all supported SMF record types.

For efficient storage usage and improved execution time, the REPORT utility will dynamically generate an SDO structure containing only the record-type definitions it needs to map the SMF record segments referenced within the report definition.

Specification of an SMF record field name in either the COLUMNS: or REQUIRED: section must reference the segment record-type mapping to which the field belongs. Therefore, it is not usually necessary to specify *record-type* alone in the REQUIRED: section.

width
Specifies the field data width. Input record field values will be truncated or padded to this width accordingly.

The default width is the maximum number of characters that would be required to display the widest value represented by the input field definition. For example, if *fieldname* represents an unsigned, 2-byte integer field in the input data, the default width is 5 because the highest value represented by *input-field* is 65535.

For column definitions identified by a *compute-field*, the default width is either 9, or a value greater than 9 and equal to the largest width value specified for the same *compute-field* anywhere within the report definition.

CENTRE|CENTER|LEFT|RIGHT
Specifies the data alignment of the field value. The value text may be left adjusted, right adjusted or centralised within the defined data width (*width*).

For column data definitions of NUMERIC data type, the default column data value text alignment is RIGHT. For all other types of column definition, the default text alignment is LEFT.

Aligned values, padded or truncated to the data width (*width*) will be used in place of the field's reference within a printed report heading, footing or control break record, unless data alignment is specified for the field within the HEAD, FOOT or BREAK section definition.

CHARACTER|NUMERIC|TIME
Specifies the data type of the values assigned to *fieldname*.

For an *input-field*, the data type is automatically determined based on the field mapping information provided by the record-type structure. However, you may wish to override this. For example, if an *input-field* has a source data type of CHARACTER but contains numeric values, you may wish to set data type "NUMERIC" so that the field is included as one of those eligible for statistics (totals, averages, etc.) generation.

However, for a *compute-field*, there is no defined data type on which the REPORT utility can base a default assignment. Therefore, it assigns a data type based on the data type of the field's value at the time the first report detail line is written. This is determined as follows:

1. If the value is in a time format then the *fieldname* is "TIME".
   The REPORT utility identifies a time format as *n:n:n.n*, *n:n:n*, *n:n* or *n:n.n* where *n* represents 1 or more decimal digits).
2. If the value is a REXX numeric value then the *fieldname* is "NUMERIC".
3. Otherwise the *fieldname* is "CHARACTER".

This method is a best effort and may not return the desired result. Therefore, it is recommended that a specific data type is provided for a *compute-field* definition.

The data type of a *built-in-field* is assigned internally by the REPORT utility and should not require a data type specification.

`NORESET`
Applicable only to *input-field* column defintions, NORESET will exclude the field from value reset processing.

Value reset processing occurs following output of a report detail line at which point the REPORT utility sets a null value to each *input-field* specified in the COLUMN: or REQUIRED: sections if either of the following is true:

- ◊ No REPEAT: section has been specified.
- ◊ Both REPEAT: and RESET: sections have been specified, and output has been triggered by input of a record (or record-segment) with a record-type mapping that matches one specified in the RESET: section.

If NORESET is specified on an *input-field* definition in the COLUMN: or REQUIRED: section, then this field's values are never reset. A field in the REQUIRED: section is not included in the report detail line output, however NORESET may be useful if the value is referenced as a REXX variable within the COMPUTE: section.

`STRIP`
Specifies that leading and trailing blanks that potentially exist in the field value, are to be stripped.

`SUBSTR(`*`start`*`[,`*`len`*`])`
Used when the required field values are a sub-string of the input values. SUBSTR specifies *start*, the position within the field value of the first output field character, and optionally *len*, the number of characters in the output field value.

If *len* is not specified, then the substring value will begin at the *start* character position and end at the last character in the input value. If *len* extends beyond the last character of the input value, then the output value will be padded with blanks. The substring value will ultimately be aligned according to the specified or default data alignement to a length specified by *width*.

Note that, if STRIP has been specified, then SUBSTR will operate on the stripped input value.

RESET

Overview:

The RESET section is only applicable if a REPEAT section also exists. Without a REPEAT section, any entries specified in a RESET section will be ignored.

The RESET section specifies the names of record (or record segment) record-type mapping(s) in the REPEAT section that, having triggered an output record, will also trigger a reset of **all** *input-field* values. An *input-field* is identified via the COLUMNS or REQUIRED section.

By default, when an output record is triggered by a match on a REPEAT section **record-type** name, *input-field* values are not reset to null. Therefore, a column's value will be repeated in the next output record unless updated by input of another record or segment which is assigned the same **record-type** to which the column field belongs.

Note that if no REPEAT section exists, output of the current report line will occur immediately following input of the next data record (or the next data record's primary segment if input records are segmented). In this case, all *input-field* values are automatically reset to null following output of the current report line and before then being re-assigned values from the new input record.

An individual *input-field* may be excluded from any reset processing by specifying parameter NORESET on its definition in the COLUMNS: or REQUIRED: sections.

Examples:

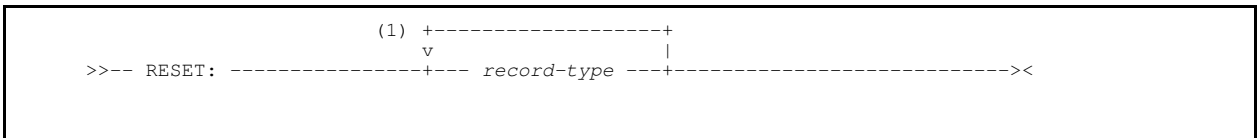
```
COLUMNS:
  SMF030_Identification.zJOBNAME     ('Job|Name'  CENTRE)
 '|'                                 ('')
  SMF030_EXCP.zEXP.zDDN              ('DDName')
  SMF030_EXCP.zEXP.zBLK              ('EXCP|Blocks')

REPEAT:
  SMF030_EXCP

RESET:
  SMF030_EXCP
```

In the above example, a number of record-types are used to map the individual sections (segments) of input SMF type 30 records.

In each SMF 30 record there is an identification segment of primary record-type "SMF030_Identification" for which there are potentially many occurrences of the EXCP segment mapped by secondary record-type "SMF030_EXCP". Reference to "SMF030_EXCP" in the REPEAT section ensures that a new report record is generated for each EXCP segment.

The "zJOBNAME" column is defined in the primary segment record-type. Normally, this column would contain the same values in each report line generated from the same input SMF type 30 record. However, reference to "SMF030_EXCP" in the RESET section means that all input field values are reset following output of each report line. Therefore, the "zJOBNAME" column in the first report line generated for an SMF type 30 record will contain a non-null value but will contain null values in all subsequent report lines for the same SMF record.

Syntax:



(1) Each *record-type* must be specified on a separate control statement.

Parameters:

record-type

Identifies the type of input record or record segment that having triggered output of a new report detail line, will also trigger reset of all input filed values to null.

A *record-type* is the name of a mapping structure used to map an input record (or record segment). To have any affect, the *record-type* name must match a *record-type* name specifeid in the REPEAT section.

One or more *record-type* names may be specified, each on separate control statements following the RESET section header in the report definition input.

SORT

Overview:

Presence of SORT section control statements indicate that the report records are to be sorted.

A number of sort control statements may be specified, each identifying a sort key field which is based on a defined *input-field*, *built-in-field* or *compute-field*. The order in which the control statements are specified dictates the hierarchy of the sort fields.

If no sorting is performed, then by default records are processed in the order they are read from the input data set.

If the REPORT utility is executed in a batch, JCL DD statements must exist for DDnames **SYSIN**, **SYSOUT**, **SORTIN** and **SORTOUT**. When executed in TSO/E foreground, the utility will dynamically allocate these DDnames to temporary data sets.

When sorting is required, the REPORT utility will:

1. Write interim records to DD SORTIN. These records contain the output report record data and so field values are in their printable character representation.
2. Execute use the standard SORT utility installed on the local z/OS host to write sorted records to DD SORTOUT.
3. Process the sorted records from DD SORTOUT to genertae the final report output.

In addition to the sort field name, the SORT control statement allows specification of the following optional parameters:

1. The sort order (ascending or descending).
2. The width of the sort field value. For use where the sort field occupies only the first characters of the specified column or required field for the specified width. Only valid for left adjusted values (e.g. character strings or time stamps).
3. The sort field data format as supported by the local SORT utility. "CSF", "SFF", "UFF" and "CH" are the data formats supported by DFSORT and SYNCSORT and applicable to report record fields. By default, the REPORT utility will select the appropraite sort format for the type of data containined in the sort field.

Examples:

Example 1 - Sort by a Single Field:

```
COLUMNS:
   TRACK.TRACK-NUM      'Track|Number'                RIGHT
   ARTIST.ARTIST        'Artist'                   30
   ALBUM.ALBUM          'Album'                    30
   TRACK.NAME           'Track|Name'               30

SORT:
   ARTIST.ARTIST  ( DESCENDING )
```

In the above example, record output lines will be sorted in descending order of artist name.

Example 2 - Sort by Multiple Fields:

```
COLUMNS:
 SMF119#02_TCP_Connection_Termination.zRName         'Resource'
 SMF119#02_TCP_Connection_Termination.zConnectStart  'Connection|Start'
 SMF119#02_TCP_Connection_Termination.zConnectEnd    'Connection|End')
 :DURATION                                           'Connection|Duration'      12 RIGHT
 SMF119#02_TCP_Connection_Termination.zInBytes       'Inbound|Bytes'
 SMF119#02_TCP_Connection_Termination.zOutBytes      'Outbound|Bytes'
 SMF119#02_TCP_Connection_Termination.zTermCode      'Termination|Description'

SORT:
 SMF119#02_TCP_Connection_Termination.zRName
 SMF119#02_TCP_Connection_Termination.zConnectStart   10
 :DURATION  (DESCENDING)
```

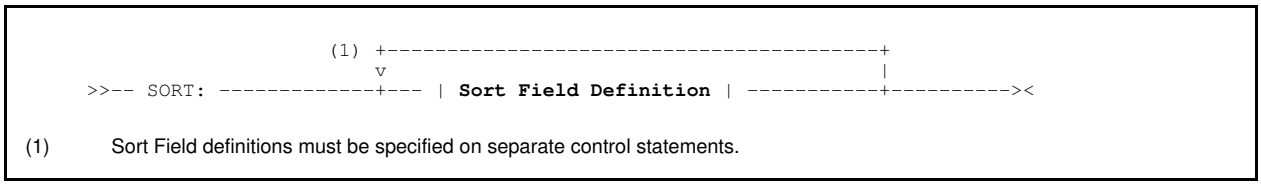
In the above example, report records contain details of TCP/IP connections obtained from SMF record type 119 sub-type 2 input records.

A hierarchy of SORT field definitions are used to sort the report records. The records are sorted first by TCP/IP resource name ("zRName") in ascending alphabetical order.

The secondary sort field ("zConnect") is a timestamp date and time of length 22 and format "*yyyy/mm/dd hh:MM:ss.tt*". However, only the first 10 characters are used as the sort field and so, for each resource name, the records are ordered by ascending order of date.

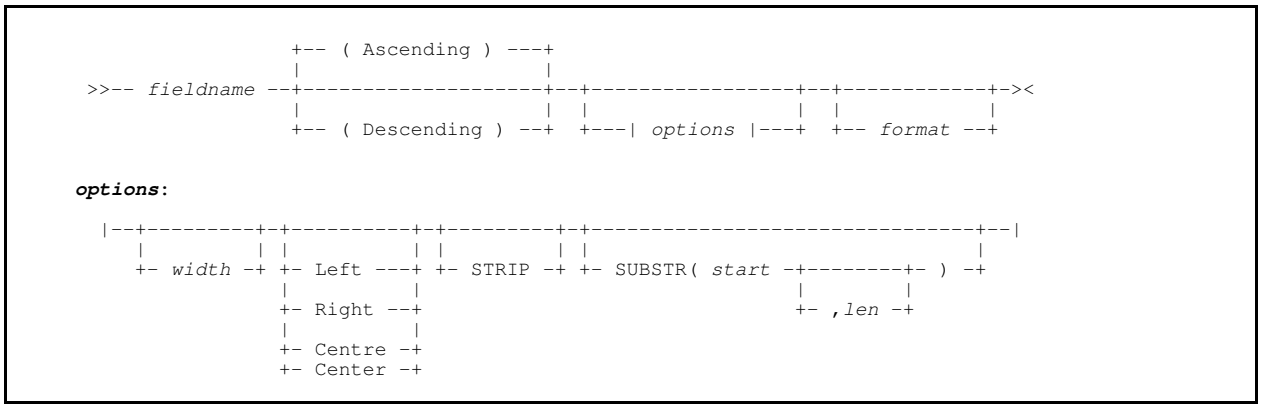
The final sort field (":DURATION") is a computed field containing the time elapsed for the connection in the format "*hhh:MM:ss.tt*". For each resource name connection on the same date, the report records are sorted by descending order of duration.

Syntax:



(1) Sort Field definitions must be specified on separate control statements.

Sort Field Definition:



Parameters:

fieldname
Identifies the field on which the sort key field is based. Values from this field will be used to sort the output report records.

The *fieldname* is an *input-field*, *compute-field* or *built-in-field* name specification.

(ASCENDING | DESCENDING)
Specified in parentheses, ASCENDING or DESCENDING defines the order in which values in the key field are sorted.

Default is ASCENDING.

width
Specifies the width of the sort key field. If *width* is greater than the default field width of *fieldname*, then blank padding will occur on the obtained field value. If it is less than this field width, then truncation of the *fieldname* value will occur.

The following example shows how report records may be grouped for UserIds (length 8), that have the same 4 character prefix and also sorted so that, for each group of UserIds, the login times occur in descending order:

```
UserId    (ASCENDING)   4
LoginTime (DESCENDING)
```

format
A DFSORT/SYNCSORT data format assigned by the REPORT utility to the fields on which report records are to be sorted.

Since output report record field values are in a printable character representation, the only sort data formats that are appliacble to these fields are **CH**, **CSF** (or **FS**), **SFF** and **UFF**.

By default, the REPORT utility will assign a sort data format based on field's data type within the input source record as follows:

Data Type	Data Format	Description
Time Date TimeStamp	UFF	Unsigned Free Form numeric format extracts decimal digits (0-9) from right to left anywhere in the field to form a positive number. Characters other than 0-9 are ignored.
Numeric	SFF	Signed Free Form numeric format extracts decimal digits (0-9) from right to left anywhere in the field to form a positive or negative number. If minus ("-") is found anywhere in the field, the number is treated as negative, otherwise it is treated as positive. Characters other than 0-9 are ignored.
Character	CH	CHaracter EBCDIC unsigned.

The *format* may be set to override one of these defaults where local knowledge of the data dicatates a more appropriate sort data format. For example, it is possible that a character data type sort field contains only signed integer values in which case the field values should be right aligned and data format "CSF" used as the sort data format.

`CENTRE | CENTER | LEFT | RIGHT`
Specifies the data alignment of the sort key value within the sort key field. The sort value may be left adjusted, right adjusted or centralised within the defined break key field width (*width*).

If *fieldname* is an input field of numeric data type, then the default sort key value alignment is RIGHT. Otherwise, the default alignment is LEFT.

Aligned values, padded or truncated to the data width (*width*), are used as the sort key.

`STRIP`
Specifies that leading and trailing blanks that potentially exist in the value obtained from *fieldname* are to be stripped prior to being aligned in the sort key field.

Note that, if SUBSTR is also used, then the strip of leading and trailing blanks will occur on the substring value obtained from *fieldname*.

`SUBSTR(`*`start`*`[,`*`len`*`])`
Specifies that the sort key value is a substring of the value obtained from *fieldname*.

A start position (*start*) and optional length value (*len*) is specified in parentheses "( )" immediately following the SUBSTR keyword. The *start* value is the position in the field of the first character obtained from *fieldname*, and *len* is the length of data to be obtained.

Note that, if *len* is not specified, then the substring value will begin at the *start* character position and end at the last character of the field value. If *len* extends beyond the last character of the *fieldname* value, then the value will be padded with blanks.

If both SUBSTR and STRIP are used, then the SUBSTR operation will occur first so that leading and trailing blank characters will be stripped fron the sub-string value and not from the original *fieldname* source field value.

Once SUBSTR and/or STRIP has been performed on the *fieldname* value, it is saved in the sort key field using the key field alignment.

In the following example, DateAdded is a left adjusted input timestamp field of length 28 in the format *yyyy-mm-dd hh:mm:ss.nnn*. To sort the report records by DateAdded, first in ascending order of date and then in descending order of time, then the SUBSTR option would be used as follows:

```
DateAdded (ASCENDING)  SUBSTR(1,10)
DateAdded (DESCENDING) SUBSTR(12)
```

STATISTICS

Overview:

The STATISTICS section applies only to PRINT type output and specifies one or more column field names for which statistical values (totals, averages, maximums, etc.) may be maintained and reported.

If no STATISTICS section is defined, then the default statistics columns are those whose values are obtained from either *input-fields* or *compute-fields* that are of a numeric data type. (See parameter NUMERIC in the COLUMNS section for a description of how the REPORT utiliy determines the data type of a *compute-field*.)

In a printed report, statistical values appear aligned below each statistics column's values at every control break and/or at the end of report (#GRAND) break. Unless suppressed, statistics totals are generated by default for each of these report breaks. Other statistical values that may be generated are determined by parameters on break definitions in the BREAK section. Specifically, these are the average value, maximum value, mimimum value, average of non-zero values and minimum of non-zero values.

Each statistics column definition occupies a single statement of the STATISTICS section and must contain an *input-field*, *compute-field* or *built-in-field* name specification that exactly matches a column field name identified in the COLUMNS section.

Examples:

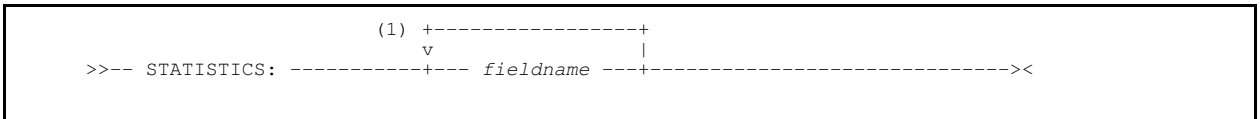
```
REQUIRED:
   TRACK.TOTAL-TIME

COMPUTE:
   Elapsed = format(TOTAL_TIME/1000,,3)

COLUMNS:
   TRACK.ALBUM
   TRACK.TRACK-NUM       'Track|Number'
   TRACK.NAME            'Track|Name'
   :Elapsed              'Track|Duration|(seconds)'

STATISTICS:
   :Elapsed

SORT:
   TRACK.ALBUM

BREAK:
   TRACK.ALBUM    AVERAGE  MAXIMUM
```

In the above example, the column definition for input record field "TRACK-NUM" is of numeric data type. All other column definitions are of non-numeric data type and so, by default, "TRACK-NUM" would be the only statistics column.

However, the STATISTICS section overrides this default and selects the *compute-field* "Elapsed" as the only statistics column. The ":Elapsed" field contains elapsed time values in seconds (format "*sssss.ttt*").

In addition to the implied #GRAND break which will, by default, output a grand total for "Elapsed" following the last report record, the BREAK section defines a control break which will occur every time there is a change in the "ALBUM" name. When this control break is triggered, the average, maximum and total values for "Elapsed" values in the control group will be displayed in the break lines. (Note that the total is displayed since control break definition parameter "TOTAL" is default.)

Syntax:



(1) Each *fieldname* must be specified on a separate control statement.

Synonyms:

STATISTICS	STATS TOTAL TOTALS

Parameters:

fieldname
Identifies a field for which statistical values may be generated.

The *fieldname* is an *input-field*, *compute-field* or *built-in-field* name specification that exactly matches a report *fieldname* identified in either the COLUMNS section.

One or more *filednames* may be specified, each on separate control statements following the STATISTICS section header in the report definition input.

TRANSLATE

Overview:

The TRANSLATE section provides the ability to translate characters in values extracted from **all** input record fields to other characters. Use of a *compute-field* is preferrable to restrict character translation to values in specific input fields.

The TRANSLATE section may be comprised of 2 control statements each containing a quoted string of characters representing a translate table. The first is the input translate table, the second the output translate table. If no output translate table is specified, it defaults to null.

A character at a position within the input table string corresponds to a character at the same position in the ouput table string. If the output table string is null or there is no corresponding position in the output table string (because the length of the input table string is greater), then the corresponding character is blank (x'40').

If a character in the input field exists in the input table, then it will be translated to the corresponding character in the output table. Otherwise, the input field character is not translated.

Note that if more than 2 statements are specified containing translate table strings, then the first and last string specified will be used as the input and output translate tables.

Examples:

```
COLUMNS:
   SMF014_INPUT_or_RDBACK_Dataset.zJobName   'Job Name'
   SMF014#3_Step_Info.zSPN                   'Step Name'
   SMF014#3_Step_Info.zPGN                   'PgmName'
   SMF014_INPUT_or_RDBACK_Dataset.SMFTIOE5   'DDName'
   SMF014_INPUT_or_RDBACK_Dataset.DSN        'Dataset Name'  44

TRANSLATE:
   '0004'x
   '.'
```

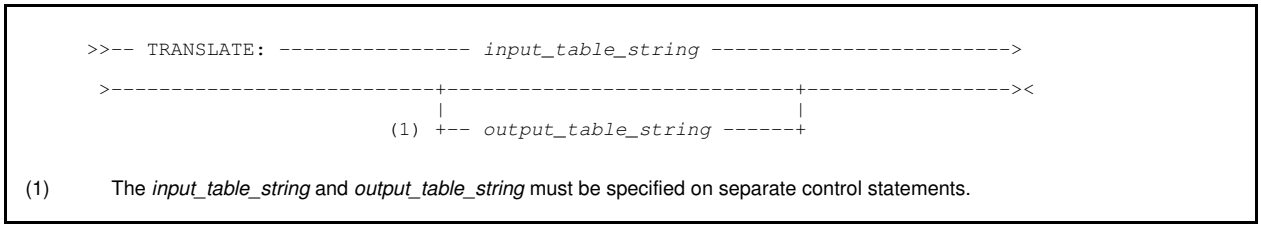
The above example reports on SMF type 14 records.

The TRANSLATE section is used to translate any occurrence of the unprintable character x'00' in values extracted from the input fields to dot/period (".").

Similarly, any occurrence of unprintable character x'04' is translated to a blank character (x'40'). Note that x'04' has no corresponding character in the output table string and so translates to blank (x'40').

Translation of x'04' is necessary since SMF type 14 records include a dataset name ("DSN") obtained from a JFCB control block. If input allocation is for a volume table of contents (VTOC) then the JFCB DSN contains 44 unprintable x'04' characters.

Syntax:



(1) The *input_table_string* and *output_table_string* must be specified on separate control statements.

Parameters:

input_table_string
: Identifies the input translate table string of characters. This may be specified as a quoted character string or a hexadecimal string. For example, '4B'X.

output_table_string
: Identifies the output translate table string of characters. This may be specified as a quoted character string or a hexadecimal string. Default is a null string and so a character in *input_table_string* will be translated to blank.

WHERE

Overview:

The WHERE section is used to define where-clauses which filter input records or record segments.

Each where-clause comprises a data edit expression and the name of the record-type mapping to which it applies. An input record or record segment which is mapped by this record-type must then satisfy the expression in order to be included in REPORT utility processing. Note that including a where-clause that references a record-type not used in the REPORT definition.

A number of expression/record-type combinations may be specified to provide filtering criteria for more than one of the record-type mappings referenced in the REPORT definition. Records or segments mapped using a record-type not identified in a WHERE section will always be included in REPORT utility processing. If more than one WHERE expression exists for the same record-type, then the last one specified will be used.

The function of the WHERE section is similar to that of the FILTER section but with the following differences:

1. WHERE section filtering may be used in addition to content match criteria filtering that are specified using options FIND, SMFTYPES, SMFSID, SMFUSERID and SMFJOBNAME (or their command line override equivalents).
2. For segmented record input (e.g. SMF records), the WHERE section will filter record segments of a specific record type mapping within the record, whereas the FILTER section will filter the entire record.
3. For segmented record input, WHERE section specifications may reference any primary or secondary segment record-type on which the associated expression will be applied. FILTER section INCLUDE/EXCLUDE specifications must reference a primary segment record-type definition (i.e. one that maps data that occurs first in the record). Note, that FILTER expressions may still test field values in the secondary segment record-types that follow the primary record-type using fully qualified field names (i.e. *rectype.fieldname*).
4. Unlike the WHERE section, FILTER will honour multiple INCLUDE (or EXCLUDE) specifications that reference the same record-type name. Doing so provides alternative criteria for selecting the record. This may however be easily achieved in a single expression in the WHERE section simply by enclosing each of the alternate selection criteria expressions in parentheses and then joining them with a separating logical OR ("|") symbol.

WHERE section syntax may span several REPORT definition control statments and so statement continuation is not necessary.

Examples:

Example 1 - Filter Single Record-Type:

```
COLUMNS:
   RELEASE-DATE  10
   ARTIST        31
   TRACK-NUM      2  CENTRE
   ALBUM         30
   NAME          40
   TOTAL-TIME     6

WHERE:
   TRACK-NUM = 1  & (  ARTIST = "Bruce Springsteen"
                     | ARTIST = "Journey"  )
```

In the example above, only one record-type definition exists in the mapping structure. Therfore, no record-type specification is required in the COLUMNS section definitions or the WHERE clause.

Example 2 - Filter SMF Record Segments:

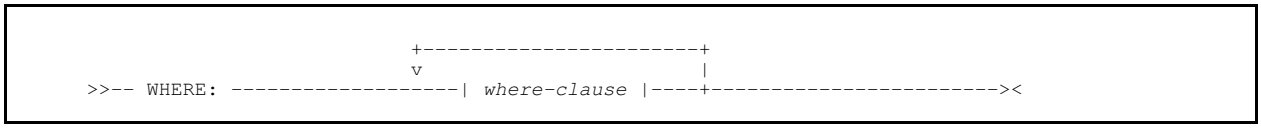
```
COLUMNS:
  SMF099#06_SRM_Service_Class_Period_Summary.zTME
  SMF099#06_Product_Information.zSLV
  SMF099#06_Period_Data_Section.zECLASS_NAME
  SMF099#06_Period_Data_Section.zGOALTYPE
  SMF099#06_Period_Data_Section.zGOALVAL

WHERE:
  zGOALVAL > 0   IN  SMF099#06_Period_Data_Section
```

The above example reports on SMF type 99, sub-type 6 records which are a summary of System Resource Manager (SRM) service class period activity for a particular policy interval.

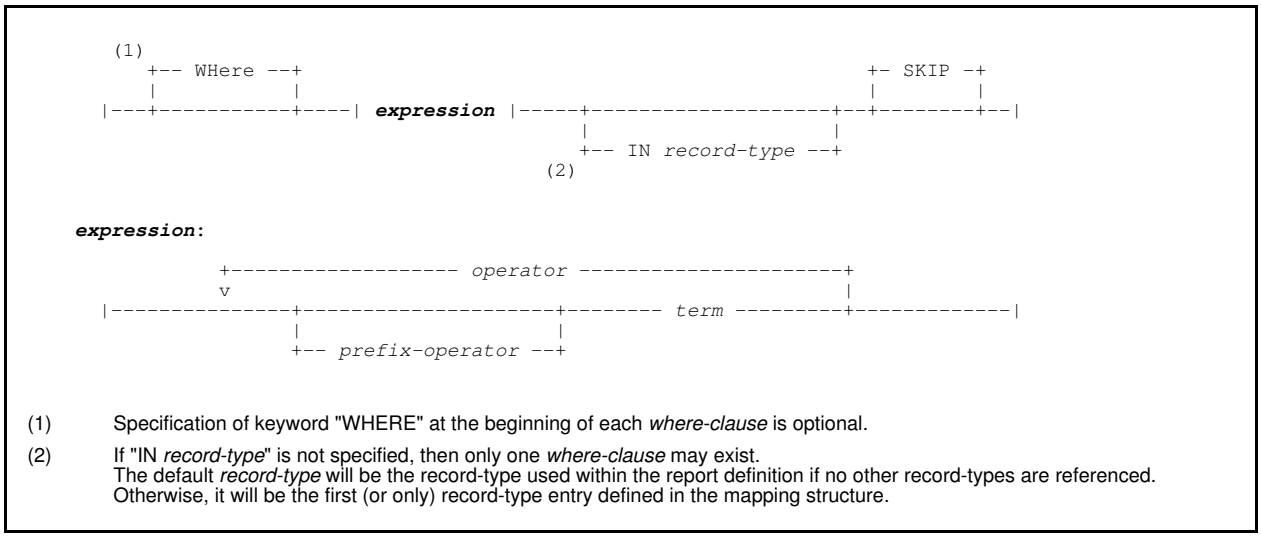
Multiple secondary record segments, mapped by record-type "SMF099#06_Period_Data_Section" may exist within the SMF record. The WHERE section contains a single expression/record-type entry which will cause all occurrences of this

secondary segment to be bypassed if the zGOALVAL field has a value which is not greater than zero. All record segments which have a zGOALVAL field value greater than zero, or are not mapped by this record-type will be processed.

Syntax:



***where-clause*:**



(1) Specification of keyword "WHERE" at the beginning of each *where-clause* is optional.

(2) If "IN *record-type*" is not specified, then only one *where-clause* may exist.
The default *record-type* will be the record-type used within the report definition if no other record-types are referenced. Otherwise, it will be the first (or only) record-type entry defined in the mapping structure.

Parameters:

where-clause
A where clause will include input records (or record segments) that are mapped by a particular record-type that matches a provided expression.

IN *record-type*
Specifies the *record-type* for the where-clause *expression*.

If the input record or record segment is mapped by *record-type*, then its data is tested against the WHERE *expression*. If the input record data satisfies the criteria specified by *expression*, then the record/record segment passes the filter criteria and is included in REPORT processing.

[WHERE] *expression*
The WHERE keyword is optional and indicates the start of the record/record segment selection *expression*.

If WHERE is not specified, the start of the *expression* is the start of the *where-clause*.

An *expression* is an SDE (Data Editor) expression that returns a Boolean value (1="true", 0="false") or a numerical value. A numerical result is treated as being Boolean in nature so that a value of zero (0) is a "false" condition and any non-zero value is a "true" condition. If "true", the input record/record segment passes the selection criteria.

For example, the following *expression* will return "1" (a true condition) if the field "FORENAME" (length 16) contains "John".

```
STRIP(FORENAME) = c'John'
```

ZZSD061E is returned if *expression* contains the name of an *input-field* not defined in the *record-type* used to map the input record.

An *expression* consists of one or more *terms* (with or without a *prefix-operator*) and zero or more *operators*.

- A **term** may be a literal string, numerical value, input field specification (*input-field*), function call or another *expression*. REPORT utility *compute-field* or *built-in-field* definitions are not supported as terms in a where clause *expression*.

- A **prefix-operator** is optional and applies to the term that follows. It may be a unary plus ("+"), unary minus ("-") or a logical NOT ("¬") symbol.
- An **operator** acts on the pair of terms between which it is positioned.

An expression and individual terms within an expression are evaluated from left to right. However, the order in which operators are actioned depends on their defined level of precedence and the presence of parentheses.

An operator with a higher precedence level will be actioned before operators with a lower precedence level. This process is repeated until the entire expression is evaluated. e.g. In the following expression, where operator2 has a higher precedence level than operator1:

```
term1  operator1  term2  operator2  term3
```

The sub-expression `term2 operator2 term3` will be evaluated first.

When parentheses are encountered, the entire sub-expression between the parentheses is evaluated immediately when the term is required. In this way parentheses may be used to force the action of an operator with a lower precedence level before that with a higher level. e.g. Logical NOT has a higher precedence level than logical AND, therefore `¬1&0` evaluates to 0, however `¬(1&0)` evaluates to 1.

The order of operator precedence is as follows (highest level at the top):

Prefix operators	`¬ (Logical NOT)  Unary +  Unary -`
Arithmetic Power	`**`
Arithmetic Multiply and Divide	`*  /  %  //`
Arithmetic Add and Subtract	`+  -`
Relational operators	`=  \=  >  <  >=  <=  <<  \<<  >>  \>>`
Logical AND	`&`
Logical OR	`|`

See publication *"FileKit Data Editor (SDE)"* for a detailed description of expressions including operator descriptions and available functions.

`SKIP`

Applicable to segmented record input, the keyword SKIP may be specified at the end of a *where-clause* to force processing to skip all remaining secondary segments in the current record when the WHERE *expression* returns a false condition.

Each input record segment is assigned a record-type and, if a where-clause exists for that record-type, the segment data is tested against the where-clause expression. If the segment fails the test and SKIP has been specified on the where-clause definition, then the current segment and all segments within the record that follow, are skipped. i.e. Processing continues with the base segment of the next input record.

Appendix A. Built-in Fields

The REPORT utility program maintains a number of "built-in" fields. The value assigned to a built-in field is initialised when the REPORT utility is started and may be updated over the course of the program execution.

The name of a built-in field may be referenced in the REPORT definition input. When an output report line is generated, any reference to a built-in field will reflect the prevailing value assigned to the field.

Built-in field names always begin with a hash symbol ("#"). The following table provides a brief description of each built-in field maintained by the FileKit REPORT utility.

Built-in Field	Description
#DATAFILE	The data set name or DB2 table name containing the input records or DB2 table rows.
#DAYNAME	Name of the current day of the week in mixed case. (e.g. "Friday").
#DD	The 2-digit day of month number at which program execution started. (e.g. "27".)
#HHMMSS	The normal (24-hour clock) time at which program execution started. (Format: 'hh:mm:ss'. e.g. "23:59:09")
#ITEMS	The number of items (detail lines) in the last control group reported following a control break.
#KEY	For a KSDS input data set only, the key value in the current input record.
#MM	The 2-digit month of year number at which program execution started. (e.g. "01" corresponds to January.)
#PAGE **#PAGENUM**	The current printed report output page number.
#RECLEN	The length of the current input record or DB2 table row.
#RECNUM	The sequence number of the current input record or DB2 table row.
#RECTYPE	The name of the record type mapping (*record-type*) used to map the record or record segment currently being processed.
#SEQUENCE	The report detail line sequence number. This is equivalent to the total number of items (detail lines) that have been printed to the report so far. (It is the running total that will be reported as the #GRAND break #ITEMS value.)
#TIME	The civil (12-hour clock) time at which program execution started. (Format: 'hh:mm xx' where 'xx' is 'am' or 'pm'. e.g. "11:59 pm")
#TIME24	The 24-hour clock time (hours and minutes only) at which program execution started. (Format: 'hh:mm'. e.g. "23:59")
#TIMESTMP	The date and time at which program execution started. This is equivalent to "#TODAY #TIME24" (Format: 'YYYY/MM/DD hh:mm'. e.g. "2020/01/27 23:59")
#TODAY	The ISO date at which program execution started. (Format: 'YYYY/MM/DD'. e.g. "2020/01/27")
#YYYY	The 4-digit year number at which program execution started. (e.g. "2020".)

Built-in Field Descriptions

#DATAFILE

A variable length character field containing the name of the data set or DB2 table (format "*owner.name*") from which input data records or table rows are read. The value of this built-in field remains constant throughout the program execution. The default length of the #DATAFILE built-in field is 44-bytes.

Sample values:

```
CBL.SMF.GDG.G8856V00
DSN8910.EMP
```

#DAYNAME

A variable length character field containing the mixed case day name on which program execution started. The maximum length is 9 characters and the value of this built-in field remains constant throughout the program execution.

Sample values:

```
Wednesday
Sunday
```

#DD

A 2-byte character field containing the day-of-month number on which program execution started. The value of this built-in field will contain a leading zero if necessary and will remain constant throughout the program execution.

#HHMMSS

An 8-byte character field containing the 24-hour clock time-of-day (hours, minutes and seconds since midnight) at which program execution started. The hours, minutes and seconds values are each 2 characters in length with a leading zero if necessary and are separated by a colon (":") symbol. The value of this built-in field remains constant throughout the program execution.

Sample values:

```
03:46:06
15:01:55
```

#ITEMS

For use in BREAK section control statements, the #ITEMS built-in field contains a numeric value equal to the number of report line items (detail lines) reported in the last control group.

The default length of the #ITEMS value is 9-bytes with non-significant zeroes displayed as blank characters. The value is updated prior to output of the report control break lines that appear after the control group.

#KEY

Applicable only if input records are read from a VSAM KSDS data set, the #KEY built-in field contains the key value of the last record read. The value has a length equal to the length of the KSDS cluster key field. The value of this built-in field is updated following the read of and input record.

#MM

A 2-byte character field containing the month-of-year number in which program execution started. The value of this built-in field will contain a leading zero if necessary and will remain constant throughout the program execution.

#PAGE
#PAGENUM

A numeric field value equal to the current report page number.

The default length of the #PAGE or #PAGENUM value is 9-bytes with non-significant zeroes displayed as blank characters. The value is incremented at the start of each new page of the printed report.

#RECLEN

A numeric field value equal to the length of the last input data record or DB2 table row read.

The default length of the #RECLEN value is 9-bytes with non-significant zeroes displayed as blank characters. The value is updated following input of each new data record.

#RECNUM

A numeric field value equal to the sequence number of the last data record or DB2 table row read. For input data sets, this value is the actual data set record number, not the input sequence number following record filtering.

The default length of the #RECNUM value is 9-bytes with non-significant zeroes displayed as blank characters.

#RECTYPE

A variable length character field containing the mixed case name of the record type mapping (*record-type*) used to map the record or record segment currently being processed. The default length of the #RECTYPE built-in field is 30-bytes.

#SEQUENCE

A numeric field value equal to the output sequence number of the current report detail line being written.

The default length of the #SEQUENCE value is 9-bytes with non-significant zeroes displayed as blank characters. The value is incremented prior to output of a report detail line.

#TIME

An 8-byte character field containing the civil, 12-hour clock time-of-day (hours and minutes) at which program execution started. The hours value is either 1 or 2 characters in length, ranging from 1 to 12 and the minutes value is 2 characters in length with a leading zero if necessary. The 2 values are separated by a colon (":") symbol. The time-of-day value is followed by a blank character and eith "am" or "pm" to distinguish times in the morning (midnight 12:00 am through 11:59 am) from times in the afternoon (noon 12:00 pm through 11:59 pm). The value of this built-in field remains constant throughout the program execution.

Sample values:

```
 3:46 am
10:07 pm
```

#TIME24

A 5-byte character field containing the time-of-day (hours and minutes) since midnight at which program execution started. The hours and minutes values are each 2 characters in length with a leading zero if necessary and are separated by a colon (":") symbol. The value of this built-in field remains constant throughout the program execution.

Sample values:

```
03:46
15:01
```

#TIMESTMP

A 16-byte character field containing timestamp on which program execution started.

The #TIMESTMP value is comprised of the the date, as described for built-in field #TODAY, followed by a single blank and the time of day, as described for built-in field #TIME24. The value of this built-in field will remain constant throughout the program execution.

Built-in field #TIMESTMP is functionally equivalent to:

```
#TODAY #TIME24
```

#TODAY

A 10-byte character field containing the date (year number, month-of-year number and day-of-month number) on which program execution started. The 3 values are each 2 characters in length with a leading zero if necessary and are separated by a slash ("/") symbol. The value of this built-in field will remain constant throughout the program execution.

Built-in field #TODAY is functionally equivalent to:

```
#YYYY 0 '/' 0 #MM 0 '/' 0 #DD
```

#YYYY

A 4-byte character field containing the year number (including the century) in which program execution started. The value of this built-in field will remain constant throughout the program execution.

Appendix B. Built-in Functions

The COMPUTE section contains REXX program statements that are executed prior to output of a report detail line. The purpose of the REXX routine is to update the values assigned to computed fields.

Because the routine is passed to the TSO/E REXX language processor, the computational expressions may include any function supported by the TSO/E REXX language (see IBM publication "*TSO/E REXX Reference*"). Input fields, built-in fields and computed fields may be used as arguments to REXX functions.

In addition to the standard TSO/E REXX functions, the REPORT utility supports a number of useful built-in REXX functions that may be used in the report definition REXX routines.

The following table provides a brief description of each built-in function provided by the FileKit REPORT utility.

Built-in Funtion	Description
ADDTIME	Add 2 elapsed time values expressed as a number of hours, minutes and seconds.
BYPASS	Skip reporting on the record or record segment currently being processed.
COUNTCHAR	Return the number of occurrances of a specific character within a character string.
DATEINC	Increment (or decrement) a date value by a number of days, months or years.
EOF	Force end of input to skip reporting on the current record or record segment, and all records that follow.
MONTHBEG	Return the ISO format date for the first day in the month of the current or specified date.
MONTHEND	Return the ISO format date for the last day in the month of the current or specified date.
SECS2TIME	Convert a number of seconds value to a number of of hours, minutes and seconds.
TIME2SECS	Convert a date, time or timestamp (date and time) value to a number of seconds.
TIMEINC	Increment (or decrement) a time or timestamp (date & time) value by a number of hours, minutes or seconds.

Built-in Function Descriptions

ADDTIME(*time1*,*time2*)

The elapsed time values *time1* and *time2* are added together to return a total elapsed time value.

An elapsed time value may be expressed as one of the following formats:

```
hours:minutes:seconds.fraction
hours:minutes:seconds

minutes:seconds.fraction
minutes:seconds

seconds.fraction
seconds
```

Where *hours* is a number of elapsed hours, *minutes* is a number of elapsed minutes, *seconds* is a number of elapsed seconds and *fraction* is an elapsed fraction of a second expressed as a decimal. Each of these elapsed time units is a positive integer value that may be specified with or without leading, non-significant zeroes. Furthermore, the units may be non-normalised so that the value of *seconds* and *minutes* may be greater than 59.

The format of *time1* does not have to match that of *time2*.

The returned elapsed time value will be normalised in the format:

```
hours:minutes:seconds.fraction
```

Where:
hours will be a value 00 upwards.
minutes will be a value 00 through 59.
seconds will be a value 00 through 59.
fraction will be a value containing a number of digits equal to the larger number of fractional digits specified by *time1* and *time2*. If no fractional digits have been specified for *time1* and *time2*, then the returned elapsed time will also have no decimal point (".") or *fraction* value. *fraction* value.

Examples:

```
Total = ADDTIME("22:15","1:48:08")          /* Returns:  "02:10:23"      */
Total = ADDTIME("28:35:52.5","3600.23")     /* Returns:  "29:35:52.73"   */
Total = ADDTIME("8:45.8","2.453")           /* Returns:  "00:08:48.253"  */
```

BYPASS()

BYPASS has no parameters and returns a value 0. It indicates to the REPORT utility that the record (or record segment) currently being processed should be bypassed and not included in the report output.

Before the COMPUTE section REXX statements are executed, the REPORT utility will assign values from fields in the current record/segment to REXX variables of the same name. The BYPASS function allows for the record or segment to be bypassed based on its field values.

Example:

```
if  TRACKLEN / 60 > 5   then  rc=BYPASS()     /* Bypass this record if track is longer than 5 minutes. */
```

COUNTCHAR(*char*,*string*[,ESC])

Counts the number of occurrences of the character, *char*, in the text string, *string*. The optional third argument, ESC, indicates that *char* may be escaped so that two consecutive occurrences of *char* will be treated as a single occurrence.

The function returns an integer value equal to the number of occurrences of *char*.

Examples:

```
NChar = COUNTCHAR("#","## Highlight ###")                  /* Returns:  5  */
NChar = COUNTCHAR("a","Have as many cakes as you want.")   /* Returns:  6  */
NChar = COUNTCHAR("&","Jane && Jill && Jacqui.",ESC)       /* Returns:  2  */
```

DATEINC([*date*][,[*n*][,[*unit*][,*datefmt*]]])

Adds *n*, a whole number of Days, Weeks, Months or Years, to a supplied ISO, US or UK/European format date (*date*). The returned date value will be normalised with 4-byte *year*, 2-byte *month* and 2-byte *day* values arranged in the same date format and using the same year, month, day separater character as the input date.

The DATEINC function input parameters are as follows:

1. The origin date (*date*) or "TODAY". (Default is "TODAY")
2. The increment value (*n*) which must be a positive or negative whole number. If negative, then *n* is subtracted from the date. (Default is +1)
3. The increment unit (*unit*) specified as "DAYS", "WEEKS", "MONTHS" or "YEARS". (Default is "DAYS")
4. The date format (*datefmt*) specified as "ISO", "UK" or "EUROPEAN" or "USA". (Default is "ISO")

Except for date format "UK", values for *unit* and *datefmt* have a minimum abbreviation of one character. For example, date format "USA" may be specified as "US" or "U", and unit "WEEKS" may be specified as "WEEK", "WEE", "WE" or "W".

A *date* value is expressed using a year number (*year*), month of year (*month*) and day of month (*day*) values. The order of these values is based on the value of *datefmt* as follows:

year, *month*, *day*	for **ISO** format date (default).
month, *day*, *year*	for **USA** format date.
day, *month*, *year*	for **UK** or **EUROPEAN** format date.

Any single, non-numeric, non-blank character may be used to separate the *year*, *month* and *day* values. Alternatively, *date* may be specified without separators, in which case it must comprise a 2 or 4-digit *year*, a 2-digit *month* and 2-digit *day* number in the order dictated by the date format. For example, a US format date "04031981" is recognised as 3rd April 1981.

If a 2-byte *year* is used, then a 100-year sliding window, ranging between -50 and +49 years from the current year, is used to determine the 4-byte representation of *year*. For example, if the current year is "2020", a 2-byte year "70" will be treated as "1970", but the 2-byte year "69" will be treated as "2069".

If *date* is omitted or specified as "TODAY" then the current date is used with slash ("/") year, month, day separator character. The default format for *date* is ISO and, if not specified, the default increment is +1 day. Therefore, DATEINC() with no parameters will return tomorrow's date in a 10-byte ISO format with a slash "/" separator between the *year*, *month* and *day* values.

Examples:

```
NDate = DATEINC("2007/04/27",+630)             /* Returns:  2009/01/16   */
NDate = DATEINC("04 - 27 - 07",+308,,"US")     /* Returns:  02-29-2008   */
NDate = DATEINC("27#03#07",-1,"WEEK","UK")     /* Returns:  20#03#2007   */
NDate = DATEINC("2007/04/27",,"year")          /* Returns:  2008/04/27   */
NDate = DATEINC("2008/02/29",3,"y")            /* Returns:  2011/03/01   */
NDate = DATEINC("2008/02/29",4,"y")            /* Returns:  2012/02/29   */
```

EOF()

EOF has no parameters and returns a value 0. It indicates to the REPORT utility that End-of-File (end of input object) is to be flagged so that no further processing of input data occurs.

If end of input is triggered, the REPORT utility will start its end of report processing so that the current record (or record segment) being processed and all subsequent records/segments are excluded from the report output.

Before the COMPUTE section REXX statements are executed, the REPORT utility will assign values from fields in the current record/segment to REXX variables of the same name. The EOF function allows for early end of input based on field values in the current record/segment.

Example:

```
                                        /* Records sorted in ascending order of ARTIST name. */
if  left(ARTIST,1) > 'F'   then  rc=EOF()   /* End of input processing for ARTIST name > 'F'.    */
```

MONTHBEG([*date*][,*datefmt*])

Returns the date of the first day in the month for a supplied ISO, US or UK/European format date (*date*). The returned date value will be normalised with 4-byte *year*, 2-byte *month* and 2-byte *day* values arranged in the same date format and using the same year, month, day separater character as the input date.

The MONTHBEG function input parameters are as follows:

1. The origin date (*date*) or "TODAY". (Default is "TODAY")
2. The date format (*datefmt*) specified as "ISO", "UK" or "EUROPEAN" or "USA". (Default is "ISO")

Except for date format "UK", values for *datefmt* have a minimum abbreviation of one character. For example, date format "USA" may be specified as "US" or "U".

A *date* value is expressed using a year number (*year*), month of year (*month*) and day of month (*day*) values. The order of these values is based on the value of *datefmt* as follows:

year, *month*, *day*	for **ISO** format date (default).
month, *day*, *year*	for **USA** format date.
day, *month*, *year*	for **UK** or **EUROPEAN** format date.

Any single, non-numeric, non-blank character may be used to separate the *year*, *month* and *day* values. Alternatively, *date* may be specified without separators, in which case it must comprise a 2 or 4-digit *year*, a 2-digit *month* and 2-digit *day* number in the order dictated by the date format. For example, a US format date "04031981" is recognised as 3rd April 1981.

If a 2-byte *year* is used, then a 100-year sliding window, ranging between -50 and +49 years from the current year, is used to determine the 4-byte representation of *year*. For example, if the current year is "2020", a 2-byte year "70" will be treated as "1970", but the 2-byte year "69" will be treated as "2069".

If *date* is omitted or specified as "TODAY" then the current date is used with slash ("/") year, month, day separator character. The default format for *date* is ISO. Therefore, MONTHBEG() with no parameters will return the date of the first day of the current month in a 10-byte ISO format with a slash "/" separator between the *year*, *month* and *day* values.

Examples:

```
MDate = MONTHBEG("2012/02/27")                  /* Returns:  2012/02/01   */
MDate = MONTHBEG("12-4-18","US")                /* Returns:  12-01-2018   */
MDate = MONTHBEG("120498","UK")                 /* Returns:  01041998     */
MDate = MONTHBEG(DATEINC("2007/04/27",+630))    /* Returns:  2009/01/01   */
```

MONTHEND([*date*][,*datefmt*])

Returns the date of the last day in the month for a supplied ISO, US or UK/European format date (*date*). The returned date value will be normalised with 4-byte *year*, 2-byte *month* and 2-byte *day* values arranged in the same date format and using the same year, month, day separater character as the input date.

The MONTHEND function input parameters are as follows:

1. The origin date (*date*) or "TODAY". (Default is "TODAY")
2. The date format (*datefmt*) specified as "ISO", "UK" or "EUROPEAN" or "USA". (Default is "ISO")

Except for date format "UK", values for *datefmt* have a minimum abbreviation of one character. For example, date format "USA" may be specified as "US" or "U".

A *date* value is expressed using a year number (*year*), month of year (*month*) and day of month (*day*) values. The order of these values is based on the value of *datefmt* as follows:

year, *month*, *day* for **ISO** format date (default).
month, *day*, *year* for **USA** format date.
day, *month*, *year* for **UK** or **EUROPEAN** format date.

Any single, non-numeric, non-blank character may be used to separate the *year*, *month* and *day* values. Alternatively, *date* may be specified without separators, in which case it must comprise a 2 or 4-digit *year*, a 2-digit *month* and 2-digit *day* number in the order dictated by the date format. For example, a US format date "04031981" is recognised as 3rd April 1981.

If a 2-byte *year* is used, then a 100-year sliding window, ranging between -50 and +49 years from the current year, is used to determine the 4-byte representation of *year*. For example, if the current year is "2020", a 2-byte year "70" will be treated as "1970", but the 2-byte year "69" will be treated as "2069".

If *date* is omitted or specified as "TODAY" then the current date is used with slash ("/") year, month, day separator character. The default format for *date* is ISO. Therefore, MONTHEND() with no parameters will return the date of the last day of the current month in a 10-byte ISO format with a slash "/" separator between the *year*, *month* and *day* values.

Examples:

```
MDate = MONTHEND("2012/02/27")                 /* Returns:  2012/02/29   */
MDate = MONTHEND("12-4-18","US")               /* Returns:  12-31-2018   */
MDate = MONTHEND("120498","UK")                /* Returns:  30041998     */
MDate = MONTHEND(DATEINC("2007/04/27",+630))   /* Returns:  2009/01/31   */
```

SECS2TIME(*nsecs*[,*scale*])

Converts a number of seconds to an elapsed time format. The number of seconds value, *nsecs*, may be expressed as one of the following formats:

```
seconds.fraction
seconds
```

Where *seconds* is a number of seconds and *fraction* is a fraction of a second expressed as a decimal.

The optional second argument, *scale*, is an integer value indicating the number of significant fractional digits to appear after the decimal point in the returned elapsed time value. If specified, then the fractional digit in the *scale* position will be rounded up or down based on the value of the fractional digit in the position following.

The returned elapsed time value will be normalised in the format:

```
hours:minutes:seconds.fraction
```

Where:
hours will be a value 00 upwards.
minutes will be a value 00 through 59.
seconds will be a value 00 through 59.
fraction will be a value containing a number of digits equal to the *scale* value or otherwise the number of fractional digits specified by the *nsecs* value. If no *scale* is specified and no fractional digits have been specified for *nsecs* then the returned elapsed time will also have no decimal point (".") or *fraction* value.

Examples:

```
ETime = SECS2TIME("64354.23")            /* Returns:  17:52:34.23   */
ETime = SECS2TIME("76475.3758",3)        /* Returns:  21:14:35.376  */
```

TIME2SECS(*source*[,[*scale*][,*datefmt*]])

Converts a *source* value to a number of seconds with or without a fraction of a second value with *scale* number of decimal places.

The TIME2SECS function input parameters are as follows:

1. The *source* value specified as either a calendar date (*date*), an elapsed time (*etime*) or both (*etimestamp*).
2. A whole number of decimal places (*scale*) representing a fraction of a second. The least significant fractional digit in the *scale* position will be rounded up or down based on the value of the fractional digit in the position following. If *scale* is 0 (zero), then the returned number of seconds will be an integer value with no decimal point (".") or fraction value. Default is the number of fractional digits specified by the *etime* or *etimestamp* value, otherwise 0 (zero).
3. Applicable to *date* and *etimestamp* source values only, the date format (*datefmt*) specified as "ISO", "UK" or "EUROPEAN" or "USA". Except for "UK", *datefmt* values have a minimum abbreviation of one character. For example, date format "USA" may be specified as "US" or "U". (Default is "ISO")

The format of *source* value is as follows:

date	By definition, a calendar *date* value is the number of days elapsed since "0001/01/01" expressed using a year number (*year*), month of year (*month*) and day of month (*day*) values. The order of these values is based on the value of *datefmt* as follows: *year*, *month*, *day* for **ISO** format date (default). *month*, *day*, *year* for **USA** format date. *day*, *month*, *year* for **UK** or **EUROPEAN** format date. Any single non-numeric, non-blank character may be used to separate the *year*, *month* and *day* values. Alternatively, *date* may be specified without separators, in which case it must comprise a 2 or 4-digit *year*, a 2-digit *month* and 2-digit *day* number in the order dictated by the date format. For example, a UK format date "03062021" is recognised as 3rd June 2021. If a 2-byte *year* is used, then a 100-year sliding window, ranging between -50 and +49 years from the current year, is used to determine the 4-byte representation of *year*. For example, if the current year is "2020", a 2-byte year "70" will be treated as "1970", but the 2-byte year "69" will be treated as "2069".
etime	An *etime* value is an elapsed number seconds expressed in hours, minutes, seconds and/or fractions of a second as follows: *hours*:*minutes*:*seconds*.*fraction* *hours*:*minutes*:*seconds* *minutes*:*seconds*.*fraction* *minutes*:*seconds* *seconds*.*fraction* *seconds* Where: ***hours*** is a number of elapsed hours. ***minutes*** is a number of elapsed minutes. ***seconds*** is a number of elapsed seconds. ***fraction*** is an elapsed fraction of a second expressed as a decimal. A colon (":") symbol must be used to separate *hours*, *minutes* and *seconds* values, and a dot/period (".") used as the decimal point symbol before the *fraction* value. Each of these elapsed time units is a positive integer value that may be specified with or without leading, non-significant zeroes. Furthermore, an *etime* value is not bound by the usual limits of a time-of-day value. i.e. The value for hours may be greater than 23, and the value for *seconds* and *minutes* may each be greater than 59.
etimestamp	An *etimestamp* value may be expressed as one of the following formats: *date etime* Where: ***date*** is an elapsed date value as described for *date* above. ***etime*** is an elapsed time value as described for *etime* above.

Examples:

```
NSecs = TIME2SECS("1:53:08")                      /* Returns:  6788           */
NSecs = TIME2SECS("123:32:16")                    /* Returns:  444736         */
NSecs = TIME2SECS("88:42.39742",2)                /* Returns:  5322.40        */
NSecs = TIME2SECS("2020/01/13 13:22:05.165",1)    /* Returns:  63714518525.2  */
```

TIME2SECS may be used to calculate the difference (number of seconds elapsed) between two two date, timestamp or elapsed time values. The SECS2TIME function may then be used to convert this value back to an elapsed time value.

```
Duration = SECS2TIME( TIME2SECS("2020/01/13 13:22:05") - TIME2SECS("2020/01/12 21:44:34") )
                                                        /* Duration value is:  15:37:31   */
```

TIMEINC([*origin*][,[*n*][,[*unit*][,*datefmt*]]])

Adds *n*, a whole number of Seconds, Minutes or Hours, to a supplied *origin* date and/or time. The returned date and/or time value will be normalised. A returned date will comprise 4-byte *year*, 2-byte *month* and 2-byte *day* values arranged in the same date format and using the same year, month, day separater character Similarly, a returned time value will comprise 2-byte hour, minutes and seconds values using the same hour, minutes, seconds separator as the *origin* time. A fraction of seconds value will also be included if one exists in the *origin* time.

The TIMEINC function input parameters are as follows:

1. The original date and/or time value (*origin*) or "NOW". (Default is "NOW")
2. The increment value (*n*) which must be a positive or negative whole number. If negative, then *n* is subtracted from *origin*. (Default is +1)
3. The increment unit (*unit*) specified as "SECONDS", "MINUTES" or "HOURS". (Default is "SECONDS")
4. Applicable only to *origin* values containing a date, the date format (*datefmt*) specified as "ISO", "UK" or "EUROPEAN" or "USA".

Except for date format "UK", values for *unit* and *datefmt* have a minimum abbreviation of one character. For example, date format "USA" may be specified as "US" or "U", and unit "HOURS" may be specified as "HOUR", "HOU", "HO" or "H".

The *origin* value is specified as either a calendar date (*date*), a time-of-day (*time*) or both (*timestamp*) as follows:

date	A *date* value is expressed using a year number (*year*), month of year (*month*) and day of month (*day*) values. The order of these values is based on the value of *datefmt* as follows: *year*, *month*, *day* for **ISO** format date (default). *month*, *day*, *year* for **USA** format date. *day*, *month*, *year* for **UK** or **EUROPEAN** format date. Any single, non-numeric, non-blank character may be used to separate the *year*, *month* and *day* values. Alternatively, *date* may be specified without separators, in which case it must comprise a 2 or 4-digit *year*, a 2-digit *month* and 2-digit *day* number in the order dictated by the date format. For example, a UK format date "03062021" is recognised as 3rd June 2021. If a 2-byte *year* is used, then a 100-year sliding window, ranging between -50 and +49 years from the current year, is used to determine the 4-byte representation of *year*. For example, if the current year is "2020", a 2-byte year "70" will be treated as "1970", but the 2-byte year "69" will be treated as "2069". When *date* is specified, the default *time* value is "00:00:00".
time	The time of day value (*time*) may be expressed as one of the following formats: *hh*:*mm*:*ss.fraction* *hh*:*mm*:*ss* *hh*:*mm* *mm*:*ss.fraction* Where: ***hh*** is a valid hour-of-day (24-hour clock) value in the range 0-23 (default 0). ***mm*** is a valid minute-of-hour value in the range 0-59. ***ss*** is a valid second-of-minute value in the range 0-59 (default 0). ***fraction*** is a fraction of a second expressed as a decimal. Any single, non-numeric, non-blank character may be used to separate *hh*, *mm* and *ss* values, and a different non-numeric, non-blank character may be used as the decimal point before the *fraction* value. For example, "21:05:15.035", "21-05-15/035" and "21.05.15,035" are all valid and represent the same time of day value. Each of these time units is a positive integer value that may be specified with or without leading, non-significant zeroes. For example, "3:24" is equivalent to "03:24:00" and "7:1.0" is equivalent to "00:07:01.0". When *time* alone is specified, then the incremented value may be earlier than the original time if it wraps over a midnight boundary (i.e. a value greater than "23:59:59.999999"). For example, "22:15:00" incremented by 4 hours would return a value "02:15:00".
timestamp	A *timestamp* value is expressed as a date and time: *date time* Where: ***date*** is a date value as described for *date* above. ***time*** is a time value as described for *time* above.

If *date* or *time* is specified, then the REPORT utility will verify the entry as being either a valid date or time. If the entry is a valid time value and a valid date value, then it will be treated as a *time* value unless slash ("/") is used as the only separator character, in which case it will be treated as a *date* value. For example, "20-05-11" will be treated

as a time value, but "20/05/11" will be treated as a date value.

If *date*, *time* or *timestamp* is omitted or specified as "NOW", then the current ISO date and time is used with slash ("/") year, month, day separator character, colon (":") hours, minutes, seconds separator and a dot/period (".") decimal point before the fraction of second value. The default *datefmt* is ISO and, if not specified, the default increment is +1 second. Therefore, TIMEINC() with no parameters will return the ISO date and time 1 second on from the current date and time.

Examples:

```
TS=TIMEINC("2007/04/27 10:33:21.222",+12,"s","iso")  /* Returns:  2007/04/27 10:33:33.222 */
TS=TIMEINC("3-16-2021 9.22.18",+2000,"MIN","US")     /* Returns:  03-17-2021 18.42.18     */
TS=TIMEINC("2021/09/28",+60*60*24*3)                 /* Returns:  2021/10/01 00:00:00     */
TS=TIMEINC("10.58.59,222",-18,"HOURS")               /* Returns:  16.58.59,222            */
```

Appendix C. Sample Data

The following details the sample data and associated structures used in examples in this manual.

Formula 1 Drivers

Structure: COBOL Copy Book **ZZS.ZZSSAM1(ZZSCF1DR)**

```
 01 F1-Driver.
   05 NUMBER                    PIC  99  COMP-4.
   05 NAME                      PIC  X(20).
   05 COUNTRY                   PIC  X(20).
   05 BIRTH-PLACE               PIC  X(20).
   05 DATE-OF-BIRTH             PIC  9999/99/99.
   05 FIRST-RACE                PIC  9999/99/99.
   05 FIRST-RACE-CIRCUIT        PIC  X(3).
```

Formula 1 2019 Race Venues (Circuits)

Structure: FileKit SDO Structure **ZZS.ZZSDIST.SDO(ZZSSF1VE)**

This structure is created using the following FileKit primary command:

```
<sdata create structure       ZZS.ZZSDIST.SDO(ZZSSF1VE)           \
      (                                                           \
       F1-Venue            structure                              \
           (                                                      \
            CODE                        character( 3)             \
           ,COUNTRY                     character(10)             \
           ,TRACK                       character(30)             \
           ,TYPE                        character(10)             \
           ,LAPS                          integer( 2)  unsigned   \
           ,LAP-LENGTH-KM                   fixed(4,3) unsigned   \
           ,TOTAL-DISTANCE-KM               fixed(6,3) unsigned   \
           ,TURNS                         integer( 2)  unsigned   \
           ,RACE-LAP-RECORD                  time(stck,22)        \
           ,RACE-LAP-RECORD-DATE             date(dec)            \
           ,RACE-LAP-RECORD-HOLDER      character(20)             \
           ,RACE-LAP-RECORD-TEAM        character(10)             \
           )                                                      \
     )    names(cobol)
```

Formula 1 2019 Race Events

Structure: FileKit SDO Structure **ZZS.ZZSDIST.SDO(ZZSSF1EV)**

This structure is created using the following FileKit primary command:

```
<sdata create structure       ZZS.ZZSDIST.SDO(ZZSSF1EV)           \
      (                                                           \
       F1-2019-Event       structure                              \
           (                                                      \
            EVENTID                     character( 7)             \
           ,VENUE                       character( 3)             \
           ,EVENT-DATE                       date(dec)            \
           ,LOCAL-TIME                       time(dec)            \
           ,UTC-OFFSET                  integer( 2)    signed     \
           ,TRACK-CONDITION             character( 3)             \
           ,TRACK-TEMPERATURE-CELCIUS     integer( 2)    signed   \
           ,HUMIDITY-PERCENTAGE           integer( 2)  unsigned   \
           )                                                      \
     )    names(cobol)
```

Formula 1 2019 Results

Structure: FileKit SDO Structure **ZZS.ZZSDIST.SDO(ZZSSF1RE)**

This structure is created using the following FileKit primary command:

```
<sdata create structure       ZZS.ZZSDIST.SDO(ZZSSF1RE)           \
      (                                                           \
       F1-2019-Result      structure                              \
          (                                                       \
            EVENT                 character( 7)                   \
           ,TRACK                 character( 3)                   \
           ,POSITION                integer( 2)                   \
           ,DRIVER-NUMBER           integer( 2)                   \
           ,DRIVER                character(20)                   \
           ,DRIVER-NATIONALITY    character(20)                   \
           ,DRIVER-TEAM           character(20)                   \
           ,FINISH-TIME                time(stck,22)              \
           ,LAPS-COMPLETED          integer( 2)       unsigned    \
           ,GRID-POSITION           integer( 2)       unsigned    \
           ,POINTS                  integer( 2)       unsigned    \
           ,NOTES                 character(20)                   \
          )                                                       \
       )    names(cobol)
```

ALBUM Tracks

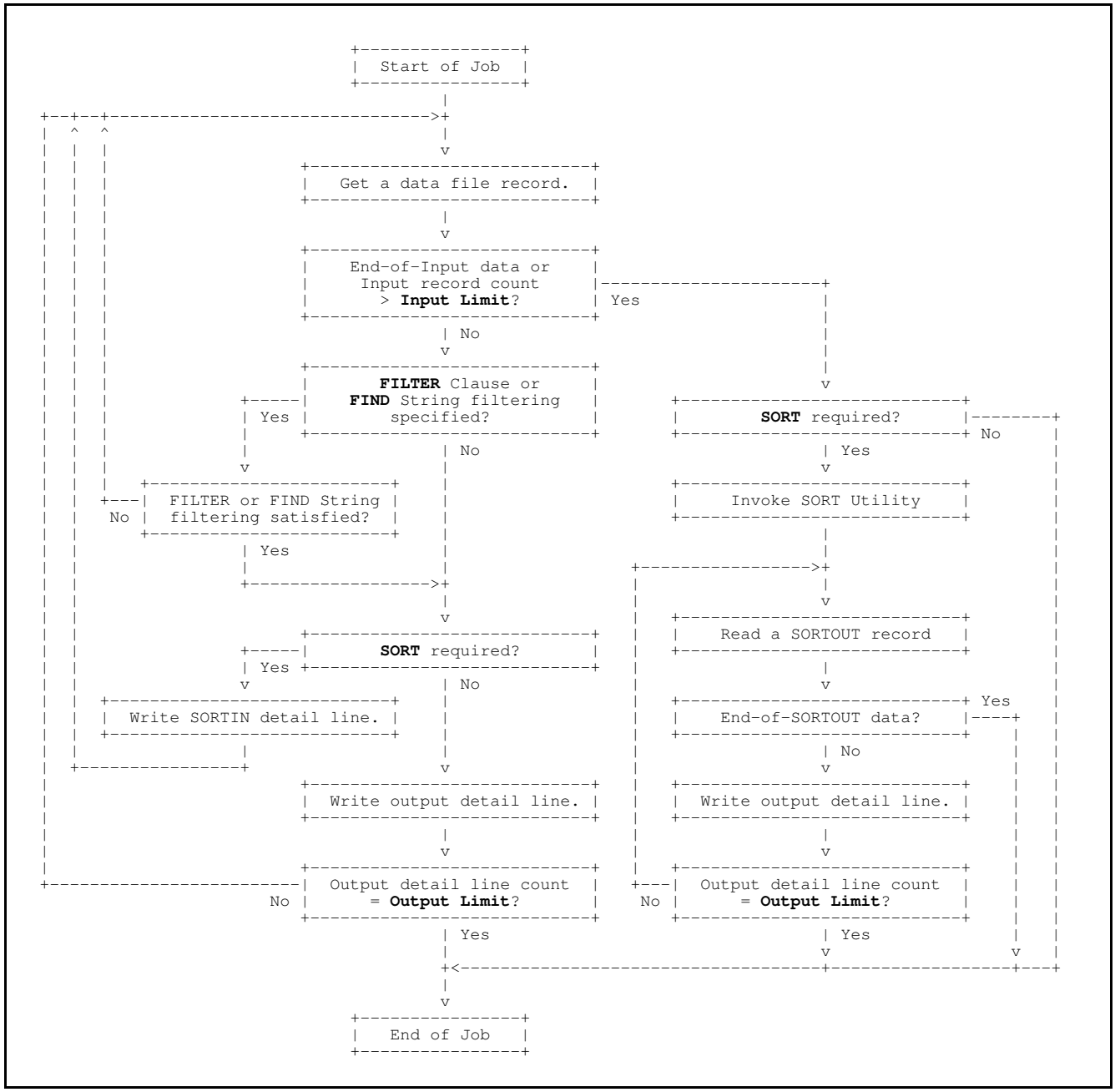
Structure: COBOL Copy Book **ZZS.ZZSSAM1(ZZST1CPC)**

```
  01 TRACK                                .
    05 PERSISTENT-ID             PIC  X(016).
    05 TRACK-NUM                 PIC  9(003).
    05 TRACK-ID                  PIC  9(004).
    05 NAME                      PIC  X(120).
    05 ARTIST                    PIC  X(070).
    05 ALBUM                     PIC  X(070).
    05 TOTAL-TIME                PIC  9(007) BINARY.
    05 FILE-SIZE                 PIC  9(009) BINARY.
    05 BIT-RATE                  PIC  9(004) BINARY.
    05 SAMPLE-RATE               PIC  9(005) PACKED-DECIMAL.
    05 YEAR                      PIC  9(004).
    05 NORMALIZATION             PIC S9(005) PACKED-DECIMAL.
    05 DISC-NUMBER               PIC  9(003).
    05 ALBUM-ARTIST              PIC  X(041).
    05 RELEASE-DATE              PIC  X(020).
    05 DATE-ADDED                PIC  X(020).
    05 DATE-MODIFIED             PIC  X(020).
```

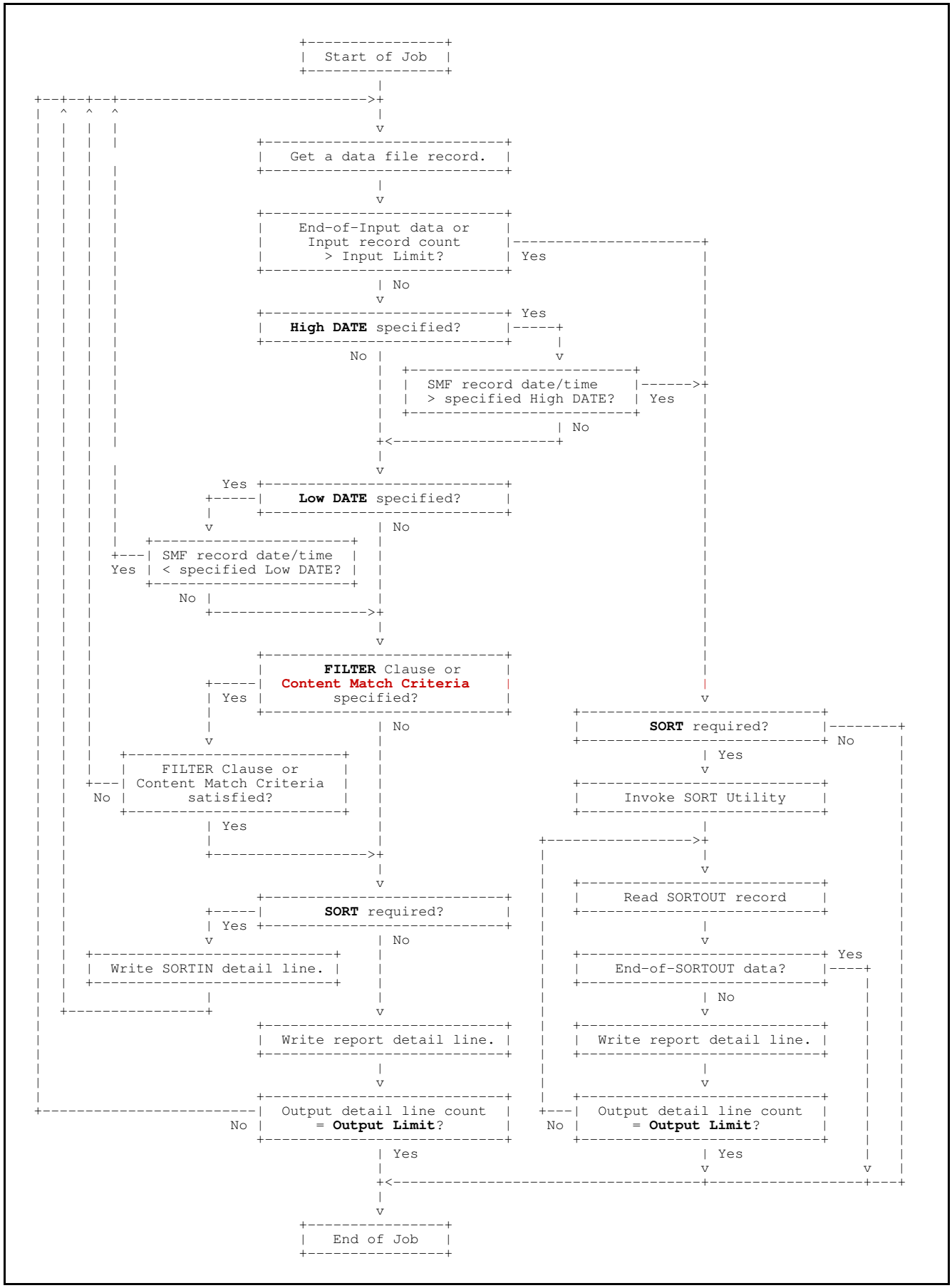
Appendix D. REPORT Logic Flow

The following logic flow diagrams illustrate REPORT processing stages for the three different types of data input (SDE, SMF and DB2).

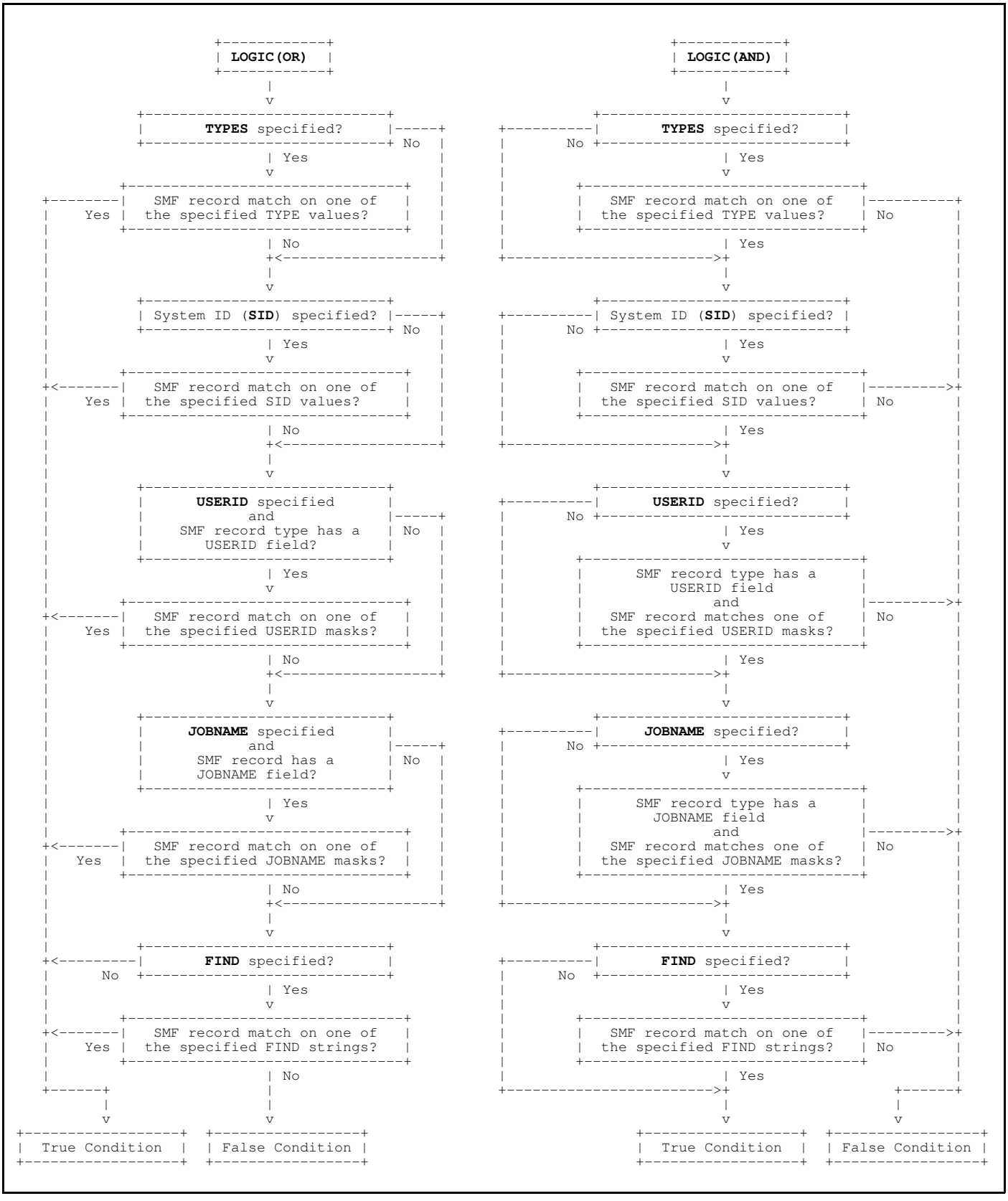
SDE Dataset Processing



SMF Records Dataset Processing



Content match criteria:



DB2 Result Table Processing