Compute (Bridgend) Ltd

FILEKIT

z/OS Productivity Toolkit

Data Editor

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

This document may be downloaded from www.cbl.com/documentation.php

Contents

Contents

SDE Concepts

Contents

SDE Menus and Popup Windows

Contents

Command Line (Primary) Commands

Contents

Command Line (Primary) Commands

Contents

SET/QUERY/EXTRACT

Documentation Notes

Fifth Edition, April 2024

Information in this document details general features and functionality of the Data Editor application which is part of the **CBL Product Suite** component, **FileKit**.

CBL Product Suite for z/OS, z/VM (CMS) and z/VSE operating systems, which includes SELCOPY, 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.

Summary of Changes

First Edition (October 2012)

This section is a summary of significant new features provided in SELCOPYi Release 3.20.

Regular Expressions

Regular expressions may be used as arguments to commands that involve a search on character data for complex pattern matching. These commands are FIND, CHANGE, EXCLUDE and ONLY.

For details, see:

◊ Regular Expressions in CBLe text editer documentation.
◊ CHANGE
◊ EXCLUDE
◊ FIND
◊ ONLY

XML Generation

In a SDE data edit display of formatted data, support primary command XMLGEN with no input DSN to generate XML source using the names of the selected columns as XML tags.

For details, see:

◊ XMLGEN

Enhancements to SDE Expressions

SDE expressions, used to identify and select records, have been enhanced to support named COBOL level-88 condition-name VALUE clause definitions.

For details, see:

◊ *"Expression Terms"*

Text Edit <-> Structured Data Edit/Browse

Easily switch display of the loaded data between the text edit and SDE data edit utilities.

For details, see:

◊ GO

COBOL an PL1 Picture String Enhancement

Correctly interpret and display a COBOL copy book formatted field that is defined by a PICTURE string which includes the "P" character to represent an assumed decimal scaling position. Specifically, the "P" character is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item.

Correctly interpret and display a PL1 copy book formatted field that is defined by a PICTURE string which includes one of the picture string characters "T", "I" or "R" in the junior byte position so representing an overpunch digit and sign (i.e. fixed zoned decimal value).

Escaped Quotes

Character literal strings used in EXCLUDE, CHANGE, FIND, ONLY and any SDE expression now support the escaping of the apostrophe (') and quote (") characters when the same character is used to delimit the string.

For details, see:

◊ *"Expression Terms"*
◊ CHANGE
◊ EXCLUDE
◊ FIND
◊ ONLY

Second Edition (March 2015)

This section is a summary of significant new features provided in SELCOPYi Release 3.30.

XML Generation Panel

Support XML interactive panel and accompanying SDE primary command XMLGEN, invoked to process formatted data in the focus SDE data edit display.

For details, see:

◊ SDE XML Generation Panel
◊ XMLGEN

CSV Generation Panel

Support CSV interactive panel and SDE primary command CSVGEN to generate comma separated variable output for formatted data in the focus SDE data edit display.

For details, see:

◊ SDE CSV Generation Panel
◊ CSVGEN

Print File Panel

Introduce the PRINT interactive panel to accompany primary command PRINT and so print the current view of records in the focus SDE data edit diaplay.

For details, see:

◊ SDE PRINT File Panel

Select Columns File Panel

Introduce the SELECT columns interactive panel to accompany primary command SELECT. The panel panel may be used to select, deselect, re-order and re-size the display of columns belonging to the focus record type.

For details, see:

◊ SDE SELECT Columns Panel

Save/Restore Focus View Settings

For use in SDE Rexx Edit macros, PUSH and POP commands introduced to temporarily save and restore the values of particular SET options for the current view.

For details, see:

◊ POP
◊ PUSH

DB2 Editing Overview

Expanded description of DB2 table editing concepts..

For details, see:

◊ DB2 Table Browse and Edit

DB2 Related Table Edit

For edited DB2 tables with enforced referential (parent/foreign key) constraints, related table edit is introduced to protect against constraint violation and to edit related parant or dependent DB2 tables.

For details, see:

◊ REDIT
◊ DB2 Save SQL Error Panel

DB2 Row Selection Panel

DB2 table row selection (SQL WHERE clause) panel introduced for use on SDE edit/browse of a DB2 table.

For details, see:

◊ DB2 Row Selection Panel
◊ BROWSE
◊ EDIT

DB2 Variable Length and Nullable Fields

DB2 table edit options introduced for display and manipulation of text in variable length and nullable fields.

For details, see:

◊ DB2 Edit Settings
◊ NULLCHAR
◊ NULLIFBLANK
◊ VENDCHAR
◊ VSHOWEND
◊ VSTRIP

DB2 Table Information

The DSI (Dataset Information utility) has been extended to display HTML report output for DB2 tables and views. SDE primary command INFORMATION is a synonym.

For details, see:

◊ INFORMATION

Assembler DSECT and DFSORT SYMNAMES Structures

Support introduced for SDE structures generated from Assembler (HLASM) programming language DSECTs and DFSORT symbolic names (SYMNAMES) structures.

For details, see:

◊ CREATE STRUCTURE
◊ BROWSE
◊ EDIT

SDEAMAIN Batch Rexx Macros

SDEAMAIN program (SELCOPYi batch execution) support included for execution of edit (Rexx) macros.

Third Edition (September 2017)

This section is a summary of significant new features provided in SELCOPYi Release 3.40.

JSON Generation Panel

Support JSON interactive panel and accompanying SDE primary command JSONGEN, to generate JavaScript Object Notation output for formatted data in the focus SDE data edit display.

For details, see:

◊ SDE JSON Generation Panel
◊ JSONGEN

LOCATE Records Panel

Support LOCATE Records panel to generate and/or execute a LOCATE primary command using an SDE expression specification.

For details, see:

◊ SDE LOCATE Records Panel
◊ LOCATE

FILTER Records Panel

Support FILTER Records panel and accompanying SDE primary command FILTER, to generate and/or execute a WHERE, MORE or LESS primary command that uses an SDE expression specification.

For details, see:

◊ SDE FILTER Records Panel
◊ FILTER

Structure (SDO) Creation

Creation of SELCOPYi structures to map formatted data has been enhanced to support the following new features:

COBOL Copybook Parsing
Parse COBOL copybook Data Division data description entries in order to build record-type definitions in an SDO. This method is an alternative to performing a COBOL compile of the copybook source in order to generate SYSADATA output.

COBOL Copybook 2nd Level REDEFINES
Structured Data Edit Settings panel (=0.4) option **Auto Rec-Types** introduced to determine whether or not, when creating a new SDO, SELCOPYi creates a separate record-type definition for each 2nd level entry in a COBOL copybook when the following is true:

1. No more than one 01 level entry exits in the copybook. If no 01 level entry exists, one will be generated by SELCOPYi.

2. All 2nd level entries belonging to the 01 group entry begin mapping data from the same offset within the group. i.e. All 2nd level entries are a union (REDEFINES) of the other 2nd level entries.

COBOL Copybook REPLACING options
CREATE STRUCTURE syntax support for specific COBOL REPLACING commands and override of SELCOPYi system defaults.

Subset Record-Type Definition
CREATE STRUCTURE syntax supports creation of a record-type definition which is a subset of fields belonging to another record-type definition within the same SDO structure.

Start Field Group Selection
CREATE STRUCTURE syntax supports creation of a record-type definition with an implied offset value based on a named field (STARTFIELD) value. The record-type definition will be for the group in which the start field is defined with an allocate offset value equal to the offset within the group at which the start field is located.

PL1 Compiler Options Override
Data Edit PL/1 Compiler Options panel (=0.4.2) field **PL/1 Compiler Additional Options** introduced to override installation default PL1 compiler options when creating SDO record-type definitions from a PL1 copybook source.

VARHEX and HEXVAR Datatypes
CREATE STRUCTURE Direct definition syntax supports data types VARHEX and HEXVAR.

BROWSE and EDIT Options

CREATE STRUCTURE syntax support for browse and edit display features:

- Column/row colour highlighting.
- DB2 column display width.
- Set Data Editor options.
- Execute Data Editor commands.

For details, see:

◊ CREATE STRUCTURE
◊ COPYBOOKPROC - SET/QUERY/EXTRACT Option
◊ Structured Data Edit Settings
◊ PL/1 Compiler Options

DB2 XML Column Data Processing

For DB2 XML columns, specify column display width, open separate Text Editor window for edit/view and allow import/export between DB2 tables and z/OS datasets.

For details, see:

◊ BROWSE
◊ EDIT
◊ GETXML
◊ PUTXML
◊ XMLBROWSE
◊ XMLEDIT
◊ XMLLENGTH
◊ XMLVIEW

DB2 Scrollable Cursors

If activated by a SELCOPYi administrator, support use of DB2 scrollable cursors to restrict the amount of user storage allocated to edit and browse table rows.

For details, see:

◊ DB2 Table Browse and Edit
◊ BROWSE
◊ EDIT

Aggregate Operations for Column Values

Report or extract the average, maximum, minimum and/or sum total of values in specific columns displayed within the Data Editor view.

For details, see:

◊ AVERAGE
◊ MAXIMUM
◊ MINIMUM
◊ SUM
◊ TOTALS

Common Text and Data Editor Commands/Options

Support for Text Editor commands and options within a Data Editor window view.

For details, see:

◊ LIST
◊ RUNSELCOPY
◊ RUNSLC
◊ FIDCHANGED - SET/QUERY/EXTRACT Option

LOCATE/RFIND Repetition Factor

LOCATE and RFIND commands now support a repetition factor to identify the numbered occurrence of the search string. This is particularly useful when redisplaying excluded lines that contain the first (or last) number of occurrences of the search string.

For details, see:

◊ LOCATE
◊ RFIND

Displayed Variable Length Column Width

Global option COLWIDTHAUTO introduced to determine the diplayed width of variable length column data.

For details, see:

◊ COLWIDTHAUTO - SET/QUERY Option

Fourth Edition (July 2020)

This section is a summary of significant new features provided in SELCOPYi Release 3.50.

PDSE V2 Member Generations

General support for processing member generations in PDSE version 2 libraries allocated with a MAXGENS value. Utilities that support member generations include Text Editor VIEW and EDIT, Data Editor BROWSE and EDIT, File Copy, File Search & Update, File Compare.

For details, see:

◊ z/OS PDSE Library Member Generations in SELCOPYi reference documentation.
◊ GEN in CBLe text editer documentation.
◊ GENCOMP in CBLe text editer documentation.
◊ GENORPH in CBLe text editer documentation.

SELCOPYi SDO structure Data Types

New field data types for IPADDRESS, TIME and TIMESTAMP introduced for SELCOPYi SDO structures created using CREATE STRUCTURE **Direct Definition** syntax.

For details, see:

◊ CREATE STRUCTURE

SELCOPYi SDO structure Compressed Data

EXPAND Clause introduced in structures created using CREATE STRUCTURE **Direct Definition** syntax to conditionally expand areas of records that are CSRCE compressed.

For details, see:

◊ CREATE STRUCTURE

BROWSE DD=*ddname*

BROWSE and other SELCOPYi utilities now support data set input via a previously allocated DD name.

For details, see:

◊ BROWSE

Cataloged Tape Dataset

BROWSE and other SELCOPYi utilities now support TAPE data set input via a cataloged DSN.

For details, see:

◊ BROWSE

String Search Scope

Operations that accept a search value, can now search records of different record-type mapping definitions. Previously, a search was performed only on records assigned the default (focus record) record-type.

For details, see:

◊ RTSCOPE
◊ CHANGE
◊ EXCLUDE

◊ FIND
◊ ONLY

Display ENUM values

Open a SELCOPYi List window to display the enum definitions for each record-type mapping in a specified SDO structure. If no structure name is specified, the structure for the current SDE window is used.

For details, see:

◊ ENUMS

Batch Execution & Macros

A number of features have been introduced to support SDE macro development and execution of SDE utilities in a batch environment.

For details, see:

◊ ENUMS
◊ FILEIO
◊ PRINTS
◊ TASK
◊ CLIPBOARD
◊ FVALUE/FVALUEQF/FVALUEQP - EXTRACT

Fifth Edition (April 2024)

This section is a summary of significant new features provided in FileKit Release 3.60.

Product Rebrand

Starting at release 3.60 (and release 3.50 with PTF RS35003 applied), "**SELCOPYi**" is rebranded as "**FileKit**".

Product materials now have name and/or aliases that reflect the product name change. For example, the FileKit batch executable (ZZSSMAIN) has both alias names **SDEAMAIN** and **FILEKITB**.

Introduction to SDE

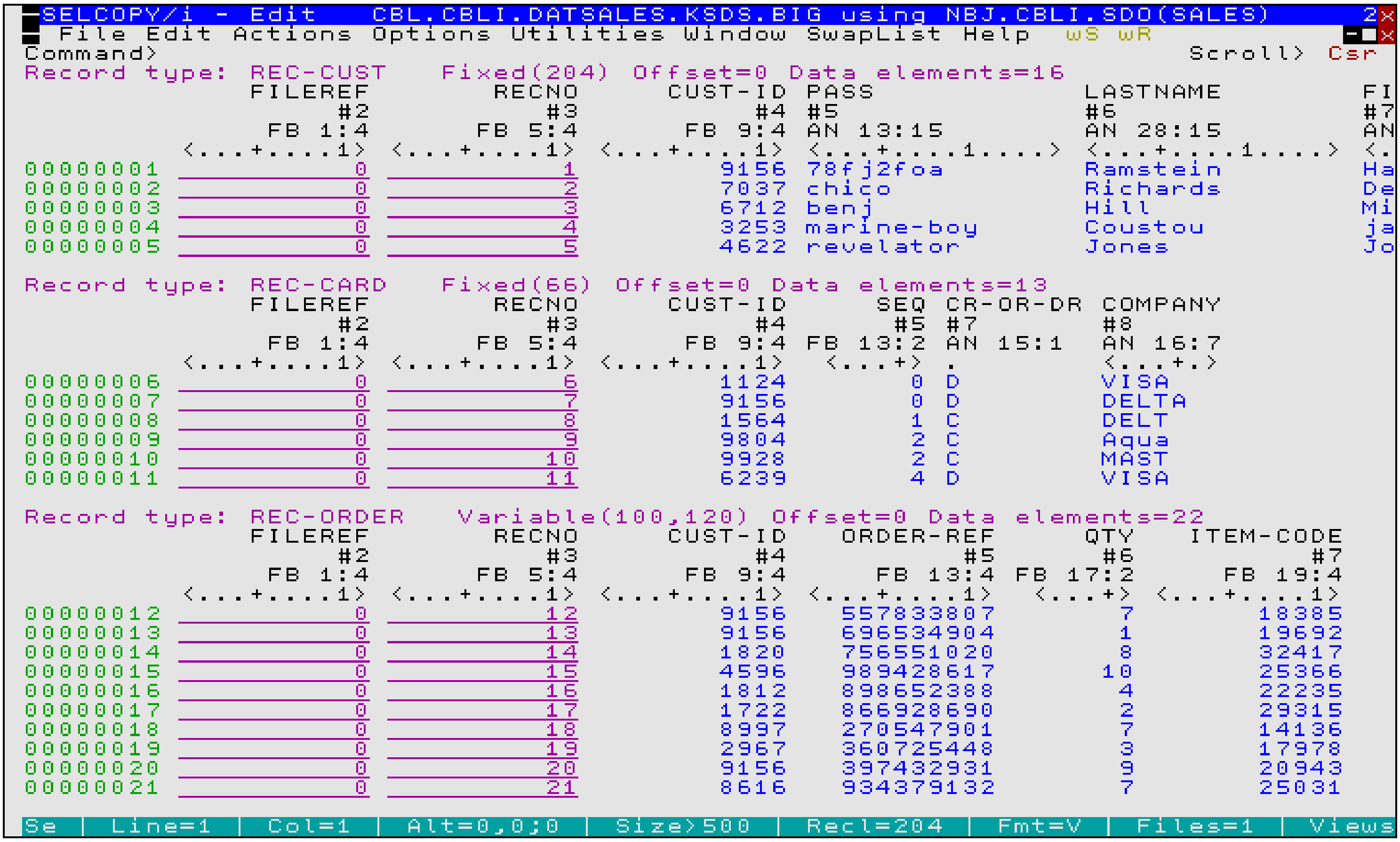
Data set records may have an associated file structure that maps field information (position, length and data type) for all data within each data set record. These structures often exist as a PL/1 or COBOL copybook.

The FileKit Structured Data Environment (SDE) allows users to display and process structured data sets using a pre-defined SDE structure so that record data is formatted and arranged in field columns. An SDE structure may be generated from a copybook or using SDE's Create Structure internal syntax, and can contain a number of mappings, one for each different type of data set record.

SDE runs as a sub-component of the CBLe Text Editor so that window views displaying structured file data may be opened as child windows of a CBLe MDI Frame window.

FileKit SDE enables users to do the following:

- Display, copy, edit and update structured data.
- Concurrently display records of more than one record type within the same view.
- Work with data in multiple or single record views.
- Select and order the fields displayed for each record type.
- Define new record structures using FileKit SDE syntax.



```
 SELCOPY/i - Edit   CBL.CBLI.DATSALES.KSDS.BIG using NBJ.CBLI.SDO(SALES)     2
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                             Scroll> Csr
Record type: REC-CUST   Fixed(204) Offset=0 Data elements=16
             FILEREF       RECNO     CUST-ID PASS            LASTNAME        FI
                  #2          #3          #4 #5              #6              #7
              FB 1:4      FB 5:4      FB 9:4 AN 13:15        AN 28:15        AN
         <...+....1> <...+....1> <...+....1> <...+....1....> <...+....1....> <.
00000001           0           1        9156 78fj2foa        Ramstein        Ha
00000002           0           2        7037 chico           Richards        De
00000003           0           3        6712 benj            Hill            Mi
00000004           0           4        3253 marine-boy      Coustou         ja
00000005           0           5        4622 revelator       Jones           Jo

Record type: REC-CARD   Fixed(66) Offset=0 Data elements=13
             FILEREF       RECNO     CUST-ID     SEQ CR-OR-DR COMPANY
                  #2          #3          #4      #5 #7       #8
              FB 1:4      FB 5:4      FB 9:4 FB 13:2 AN 15:1  AN 16:7
         <...+....1> <...+....1> <...+....1>  <...+>   .      <...+.>
00000006           0           6        1124       0 D        VISA
00000007           0           7        9156       0 D        DELTA
00000008           0           8        1564       1 C        DELT
00000009           0           9        9804       2 C        Aqua
00000010           0          10        9928       2 C        MAST
00000011           0          11        6239       4 D        VISA

Record type: REC-ORDER   Variable(100,120) Offset=0 Data elements=22
             FILEREF       RECNO     CUST-ID   ORDER-REF     QTY   ITEM-CODE
                  #2          #3          #4          #5      #6          #7
              FB 1:4      FB 5:4      FB 9:4     FB 13:4 FB 17:2     FB 19:4
         <...+....1> <...+....1> <...+....1> <...+....1>  <...+>  <...+....1>
00000012           0          12        9156   557833807       7       18385
00000013           0          13        9156   696534904       1       19692
00000014           0          14        1820   756551020       8       32417
00000015           0          15        4596   989428617      10       25366
00000016           0          16        1812   898652388       4       22235
00000017           0          17        1722   866928690       2       29315
00000018           0          18        8997   270547901       7       14136
00000019           0          19        2967   360725448       3       17978
00000020           0          20        9156   397432931       9       20943
00000021           0          21        8616   934379132       7       25031

Se | Line=1 | Col=1 | Alt=0,0;0 | Size>500 | Recl=204 | Fmt=V | Files=1 | Views
```

Figure 1. Structured Data Environment View.

Terminology

The following terminology is used throughout FileKit documentation in relation to record data processing in the FileKit SDE environment.

ADATA File

SYSADATA (Associated Data) file output generated by the Enterprise COBOL compiler when compile time option ADATA is specified, or by the Enterprise PL1 compiler when compile time option MSG, sub-option XINFO is specified.

FileKit SDE uses an ADATA file to generate an SDO.

COBOL Copybook

A data set or PDS/PDSE library member that contains one or more Enterprise COBOL group definitions (not necessarily 01-level) and is usually included by COBOL source programs to provide a file record layout.

Note that FileKit supports Enterprise COBOL only. Earlier releases of COBOL for z/OS do not support generation of ADATA output.

Expanded Record/Segment

A formatted record/segment that contains variable length fields or repeating groups.

In the process of formatting record data, Variable length fields are formatted as fields of length equal to the maximum length of the variable field. Repeating groups of fields are formatted so that a field group exists for the defined maximium number of occurrences of the repeating group.

All other types of record/segment are **unexpanded**.

Focus line or row

The focus line or row is the file line or table row which contains the cursor. If the cursor is not in a file line or table row then the focus line or row is the top line or row in the window.

Many commands make implicit use of the contents of the focus line or row as a command argument.

Formatted Record/Segment

A typed record/segment for which the record data has been arranged into discrete fields as defined by the assigned RTO . An **unformatted** record/segment is one that is **untyped** or one that is **typed** with data arranged as a single character field of length equal to the record length.

Mapped Record/Segment

An alternative name for a formatted record/segment.
An **unmapped** record/segment is an alternative name for an untyped record/segment.

PL1 Copybook

A data set or PDS/PDSE library member that contains one or more Enterprise PL1 structure definitions and is usually included by PL1 source programs to provide a file record layout.

Record Structure

The definition of one or more fields that correspond to areas of a file record or DB2 table row or column. A record structure may define the complete or partial layout of fields within a file record or DB2 column.

Record-type

The name assigned to a single RTO definition in a FileKit SDO structure. Record-type is analogous with RTO and is often used to refer to an RTO definition.

RTO

A Record Type Object is a single record structure within an SDO. An RTO is referenced by a unique record-type name.

RTO Data Length

The sum of all field lengths defined by the RTO. The RTO data length may encompass a range of values if the RTO includes variable length fields or a variable number of repeating groups (e.g. array elements.)

SDO

A Structure Definition Object is a FileKit SDE generated structure, comprises one or more RTO and is used by FileKit SDE to map record data. If a COBOL/PL1 copybook or ADATA file is specified for record mapping, then FileKit will automatically generate an SDO from the copybook/ADATA source.

For this reason and unless otherwise stated, an SDO is often referenced simply as a structure. An SDO is referenced by its structure name which, if non-temporary, is the fileid of the structure definition disk file to which it is saved.

Segment or **Record Segment**

An area of record data treated as an individual record for Segmented browse/edit. Mapped segment boundaries are determined by the RTO data length and any offset defined for that RTO. An unmapped segment comprises all remaining data in the record that follows the last mapped segment.

Segmented Browse/Edit

A type of SDE file edit or browse that treats consecutive, non-overlapping areas (segments) of record data as individual records. Segmented browse/edit invoked when an SDO containing one or more RTO of type "SEC" is used to format record data.

Structure

A named object that contains one or more record structure. Unless otherwise stated, structure refers explicitly to a FileKit SDE structured data object (SDO).

Where stated, a structure may also apply to a COBOL or PL1 copybook or a COBOL or PL1 ADATA (Associated Data) compiler output file.

Typed Record/Segment

A record/segment that has an assigned record-type (RTO).
An **untyped** record/segment is one that has **not** been assigned an RTO .

SDE Concepts

Although FileKit SDE runs within the CBLe text edit frame window, it is a much more sophisticated method of editing data.

Presentation and edit of record data within SDE must adhere strictly to precedents defined by the associated file structure. In order to do this, SDE introduces certain concepts and terminology that are used in its interpretation of data and are referenced throughout this documentation.

It is recommended that users familiarise themselves with the terms detailed in this section prior to performing any advanced edit operations.

Record Structure Definition

In order to display a file's data records correctly within FileKit SDE, an appropriate data definition must exist that accurately defines fields within the data records.

For FileKit SDE, a data definition is called the **structure** and must exist in an internal SDE format generated by the CREATE STRUCTURE command. Any existing COBOL, PL1, HLASM or DFSORT source code that is used to map the file's data records may be also be used to generate the SDE structure.

Structure Definition Object

The Structure Definition Object (SDO) is an in-storage copy of an SDE structure which comprises all the objects required to map, display and select data. These include:

- A structure identification including a 64-character title and/or 512-character description.
- One or more record-type objects used to format data records or DB2 table rows.
- Record-type selection criteria for data comprising records or record segments mapped by different record formatting.
- DB2 commit, isolation level, locked row, audit and primary key options.
- Record-type colouring based on WHERE expressions.

Creating an SDO Structure

Structures may be created explicitly (via the CREATE STRUCTURE primary command or Create Structure panels) or implicitly when the Data Editor or an SDE utility (e.g. FSU, COMPFILE) requires use of a structure to format data.

A structure may be generated from one or more copybook library members containing the following:

- COBOL group item(s).
- PL1 declared structure(s).
- HLASM (Assembler) DSECT control section(s).
- DFSORT SYMNAMES statements.
- FileKit structure definition (SDO).

If an SDO structure is to contain record type mapping entries generated from a PL1 declared structure of HLASM control section, then FileKit calls the Enterprise PL1 compiler or HLASM assembler as appropriate to generate SYSADATA output from the copybook source. This SYSADATA output is then processed to generate the data mapping structures.

When an SDO structure is to contain record type mapping entries generated from a COBOL source, FileKit may either interpret the COBOL Data Division field definition entries directly (internal processing) or execute the Enterprise COBOL compiler to generate the SYSADATA output (compiler proessing). The method used by FileKit is controlled by the prevailing value of the COPYBOOKPROC option (default INTERNAL) or by the COPYBOOKPROC parameter keyword specified on a **CREATE STRUCTURE** operation. The internal processing method has the following advantages:

1. The Enterprise COBOL Compiler need not be installed.

2. Processing is quicker. Execution of a COBOL compile naturally takes longer and requires a large TSO region size for successful operation (e.g. 128M for Enterprise COBOL 5.2).

Loading an SDO Structure

When a structure is created explicitly (via the CREATE STRUCTURE primary command or Create Structure panels) or when the Data Editor or an SDE utility (e.g. FSU, COMPFILE) requires use of a structure to format data, an SDO is loaded into local storage as follows:

1. If the structure is an SDO structure name, the SDO is loaded from an existing library member or sequential data set (structure definition file).

2. If the structure is for a DB2 result table or is specified as an existing COBOL, PL1, Assembler or DFSORT source copybook member, the SDO is generated automatically.

Once in storage, an SDO remains there until dropped either explicitly, via the DROP command or a panel recompile specification, or implicitly when the FileKit session is ended.

SDO Structure Name

If an SDO is created explicitly using CREATE STRUCTURE with parameter TEMPORARY or is generated automatically by the Data Editor or an SDE utility, the SDO structure is **temporary** and is not automatically saved to a structure definition file. Whether temporary or non-temporary, an SDO is referenced by its structure name which is determined as follows:

1. The structure name (*structname*) parameter specified by execution of the CREATE STRUCTURE command or Create Structure panels.

2. The sequential data set name or library and member name of the structure definition file from which the SDO was loaded.

3. For a temporary SDO generated from COBOL, PL1, Assembler or DFSORT source or from an ADATA file, the assigned structure name is the data set (and member) name of the source copybook, DSECT, SYMNAMES or ADATA file.

4. For a temporary SDO generated by specific DB2 SQL query syntax, the assigned structure name is of the format TBA*nnnnn.sqlid*.SQLVIEW*m*, where *nnnnn* and *m* are numeric sequences and *sqlid* is the current SQLID.

5. For a temporary SDO generated for a specific cataloged DB2 table or view name, the assigned structure name is of the format TBA*nnnnn.schema.name*, where *nnnnn* is a numeric sequence, *schema* and *name* are the schema and name values for the DB2 table or view as defined in the SYSIBM.SYSTABLES catalog table.

Saving SDO Structures

Both temporary and non-temporary structures may be saved explicitly, using the SAVESTRUCTURE primary command, or implicitly by making a permanent update to the SDO. e.g. Record-type selection criteria (USE), column display selection and sequencing (SELECT), column width overrides (SET COLWIDTH), and record-type colouring (RCOLOUR).

If the structure is non-temporary, the SDO is saved to the structure definition file with DSN matching the structure name (or new structure name if saved using SAVESTRUCTURE). If the structure is temporary, the "Specify new Structured Data Object (SDO) name" panel is opened prompting for a structure definition file data set and optional library member name.

Record Type Object

A Record Type Object (RTO) is a single record, record segment or DB2 table row structure mapping definition that exists within an SDO. It contains the following information:

- Field offset, length and data format of all fields within a data record.
- Record offset at which the record-type object will begin mapping the record data. (This may be a positive or negative value).
- Column selection, column sequencing and column width definitions.
- DB2 row selection (WHERE clause) and row sequencing (ORDER BY clause). Note that, for SDE applications that use non-DB2 table SDOs, record selection (filtering) based on WHERE expressions may be achieved using a Filter file.

For structures containing more than one RTO, selection criteria must be defined which identifies the RTO to be used when formatting individual records or record segments belonging to the file to which the structure is applied.

Each RTO is referenced by a name which is unique within the SDO. This name is also referred to as the **record-type** and is allocated to an RTO when the SDO is created. The RTO record-type may be used as a generic description of those data records that are assigned that specific RTO. (See *"Record Type Assignment"* .) e.g. Client1 data records are considered to be records that are assigned an RTO with the name "Client1".

RTO record-type selection criteria may be updated after an SDO has been created using the USE *record-type* WHEN primary command.

Record Type Object Data Length

The RTO data length is the sum of all field lengths defined by the RTO.

If all RTO fields are of fixed length, the RTO data length is a fixed value.

If, however, the RTO includes variable length field definitions or a vriable number of repeating field groups, the RTO data length will encompass a range of possible values. In this case, the RTO data length is variable and has a minimum and maximum value.

The RTO data length is used in record-type assignment processing.

Structure Definition File

A Structure Definition File (SDF) is a DASD file (sequential data set or PDS/PDSE library member) containing a saved (i.e. non-temporary) copy of a Structure Definition Object.

An SDF data set must be of variable record format (RECFM=V/VB) with a minimum LRECL value of 16378 (16K-8). However, an SDF data set does not need to be pre-allocated before attempting to save an SDO structure.

When creating a non-temporary SDO structure using CREATE STRUCTURE or the SAVESTRUCTURE primary commands, the SDO is immediately saved to the sequential data set or library member identified by the structure name. If the data set or library does not already exist, the Allocate Non-VSAM dialog window is opened automatically with RECFM and LRECL input fields auto-filled with the minimum values required. After optionally updating other allocation fields and selecting "OK" to continue, the SDF will be allocated and the SDO saved accordingly.

On starting the Data Editor or an SDE utility that references an SDO structure name then, if the SDO is not already loaded in storage, it gets loaded from the SDF.

Record Type Assignment

Where a structure has been specified for SDE processing of data records (e.g. SDE EDIT, BROWSE, File Compare, etc.) then, on initial load of the data records or following execution of a USE WHEN/ALWAYS/NEVER command, record-type assignment processing is performed.

The record-type (RTO) assigned to an individual record is determined using the following order of precedence. Note that, unless specified by a USE ALWAYS ON condition, an RTO for which the **USE NEVER** condition has been set **ON** will be excluded from any RTO assignment test.

1. Use the RTO identified by a USE *record_type* ALWAYS ON condition. If the USE NEVER condition is ON for this RTO, it will be set OFF.
2. Use the first RTO in the structure definition (SDO) for which a USE WHEN expression tests "true".
3. Use the first RTO in the SDO defined as being DEFAULT.
4. Use the first RTO defined in the SDO for which the unexpanded record length matches the RTO data length. For an RTO of variable data length, the unexpanded record length is considered a match if it falls within the minimum and maximum RTO data length values.
   Note that, having found a match, no other RTOs in the SDO are tested.
5. Use the first RTO in the SDO that is of a **fixed** RTO data length.
6. Use the first RTO in the SDO.
7. Use the default record type "UnMapped" which consists of a single field "UnMapped" of maximum length equal to the data set's LRECL value, and with a data type AN (alpha-numeric.)

Specification of USE record-type criteria allows assignment of that RTO to a record having a length that does not necessarily match the RTO data length. If this is the case, only areas of the record that are formatted by the assigned RTO will be displayed.

Any unformatted record data that follows the formatted data of the same record is not displayed but is preserved if changes are made to the formatted data. The unformatted data may be displayed and changed at any time using a different display format. e.g. FORMAT CHAR or FORMAT HEX.

Where unformatted data exists at the end of a record, the flag "=LGTH>" is displayed in the prefix area.

Structured Data Fields

Records that have been assigned a record type (RTO) are displayed with the record data presented in individual formatted field columns.

The names assigned to each field are those defined by the SDO, COBOL copybook or PL1 include file. Similarly, the assigned field reference number reflects the order in which the field occurs in the record structure hierarchy. Both are displayed in the record type headers (see *SDE Window Views*.)

SDE commands and expressions may reference a field via its field name or its field reference number.

Field Name
If unique, the field may be identified simply by specifying its defined field name.

However, if the field name is non-unique, occuring in one or more sub-structures (COBOL data description groups) that exist in the record type definition, then the field name must be fully or partially qualified by the the structure names of each level of structure in the hierarchy to which the field belongs. The default qualifier separator symbol is '.' (dot/period) but this may be tailored using the SET QSEPARATOR option. e.g. "structA.structN.fieldX".

The number of preceding structure name qualifiers specified before the field name need only be enough to uniquely identify the required field. e.g. "structN.fieldX" is enough to destinguish it from "structM.fieldX".

If desired, the field name may contain qualifiers of every level in the hierarchy including the record type name, which is itself a structure definition. e.g. "OrderRec.structA.structN.fieldX".

Note that, unless parameter CASE(RESPECT) was used to create the structure (SDO), the field name and any of its qualifier structure names, may be typed in any character case.

Field Reference

The field reference number is unique in the record type definition.

The required field may be identified by specifying its field reference number prefixed by a "#" (hash) symbol.

Field Subscripts

Fields that constitute elements of an array (COBOL OCCURS clause) may be individually identified using a parenthesised, comma separated list of array vectors as a subscript to the array field name or field reference.

One array vector value must be specified for each dimension defined to the array. Each vector is an integer value identifying the field's position in that dimension of the array. e.g. #13(3) is element 3 of a single dimension field array, RoomArea(6,2) is the x,y coordinate of an element in a two dimensional field array.

Structure to Dataset Associations

In most cases, the structure used to map data in dataset records never changes and so FileKit provides a convenient facility to define and save an association between a data file name mask and a structure file. i.e. a FileKit SDO structure or a COBOL, PL1, Assembler, ADATA source member.

Once an association has been saved, subsequent data edit/browse of any dataset (or HFS file) that matches the data file mask, will automatically use the associated structure to format the file's data records. When this facility is implemented, a structure need never be referenced directly in the Structured Data Edit/Browse panel or via the USING parameter of the EDIT/BROWSE primary commands.

This structure to dataset association facility is implemented via the Manage Copybook Associations panel (=0.4.6) or the data editor AUTOSTRUCTURE option.

By default, this facility is set on at a level so that any specification of a structure on data edit or browse will automatically be saved as an association definition for the specific data set name. Likewise, any data edit or browse without a structure reference will automatically interrogate the associations table and use the structure associated with a matching DSN mask. Option AUTOSTRUCTURE controls whether or not association definitions are saved, applied or both.

Edit Associations Table

Association definitions are saved in the user's own copy of the FileKit table library member, ZZSDSUSE, which gets automatically generated the first time an association is defined.

This table of association definitions may be viewed and modified by typing the **EDIT (E)** primary command from the **Manage Copybook Associations** panel or executing the Data Editor option STRUCTURE with no parameters (i.e. from outside the data editor, execute **SD STRUCT**). This table is presented as a structured data edit view which supports all the usual data editor primary and line-commands such as FIND, CHANGE, SELECT, WHERE, etc.

Each table row defines a relationship between a structure (**MappingFile**) and one or more data files (**DataFileMask**). Furthermore, if the structure refers directly to a structure which is COBOL, PL1, Assembler or ADATA source, then this must be indicated in the **Lang** column by COBOL, PL1, HLASM or ADATA respectively. For SDO structures, Lang may be specified as SDO or left as blank.

Standard pattern matching wildcards may be specified in the data file mask as follow.

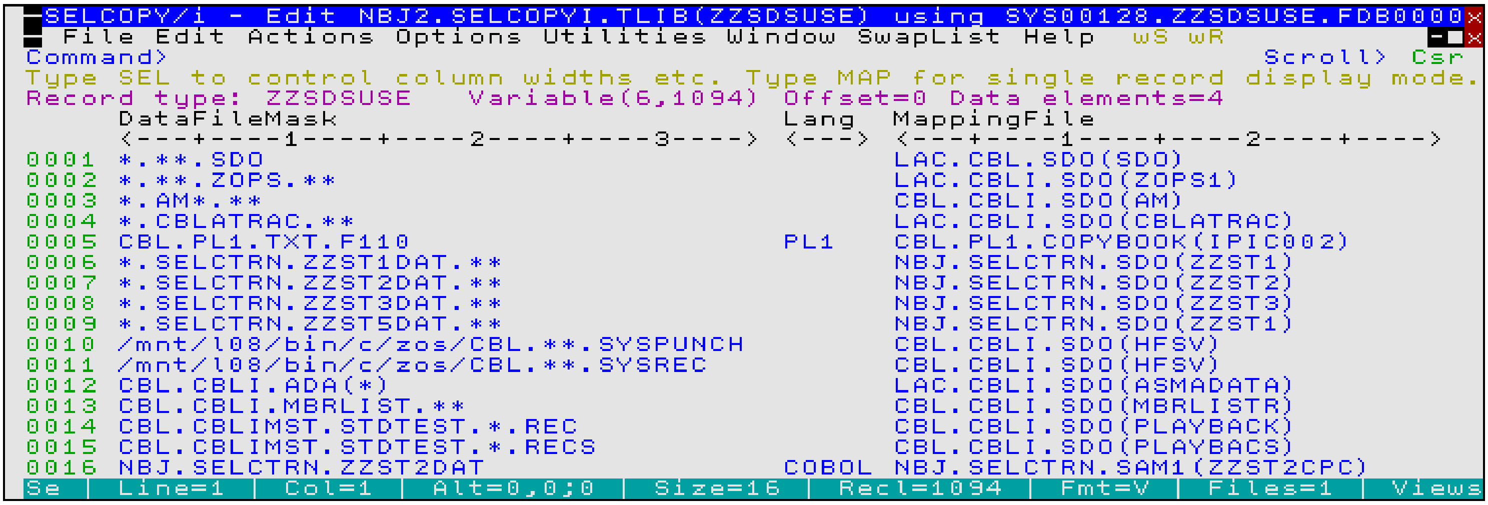
`*` A single asterisk indicates that either a qualifier or one or more characters within a qualifier can occupy that position. An asterisk can precede or follow a set of characters.

`**` A double asterisk indicates that zero or more qualifiers can occupy that position. A double asterisk cannot precede or follow any characters; it must be preceded or followed by either a dot or a blank.

`%` A single percent sign indicates that exactly one character can occupy that position. (Up to 8 percent signs can be specified in each qualifier.)

An optional library member name mask may also be specified in "( )" (parentheses) following a library DSN mask. This supports wildcards as follow.

*	A single asterisk represents a member name or zero or more characters within a member name mask.
%	A single percent sign represents exactly one character within a member name mask. Up to 8 percent signs can be specified in each member name mask.



```
SELCOPY/i - Edit NBJ2.SELCOPYI.TLIB(ZZSDSUSE) using SYS00128.ZZSDSUSE.FDB0000
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                          Scroll> Csr
Type SEL to control column widths etc. Type MAP for single record display mode.
Record type: ZZSDSUSE   Variable(6,1094) Offset=0 Data elements=4
     DataFileMask                         Lang  MappingFile
     <---+----1----+----2----+----3----> <---> <---+----1----+----2----+---->
0001 *.**.SDO                                  LAC.CBL.SDO(SDO)
0002 *.**.ZOPS.**                              LAC.CBLI.SDO(ZOPS1)
0003 *.AM*.**                                  CBL.CBLI.SDO(AM)
0004 *.CBLATRAC.**                             LAC.CBLI.SDO(CBLATRAC)
0005 CBL.PL1.TXT.F110                     PL1   CBL.PL1.COPYBOOK(IPIC002)
0006 *.SELCTRN.ZZST1DAT.**                     NBJ.SELCTRN.SDO(ZZST1)
0007 *.SELCTRN.ZZST2DAT.**                     NBJ.SELCTRN.SDO(ZZST2)
0008 *.SELCTRN.ZZST3DAT.**                     NBJ.SELCTRN.SDO(ZZST3)
0009 *.SELCTRN.ZZST5DAT.**                     NBJ.SELCTRN.SDO(ZZST1)
0010 /mnt/l08/bin/c/zos/CBL.**.SYSPUNCH        CBL.CBLI.SDO(HFSV)
0011 /mnt/l08/bin/c/zos/CBL.**.SYSREC          CBL.CBLI.SDO(HFSV)
0012 CBL.CBLI.ADA(*)                           LAC.CBLI.SDO(ASMADATA)
0013 CBL.CBLI.MBRLIST.**                       CBL.CBLI.SDO(MBRLISTR)
0014 CBL.CBLIMST.STDTEST.*.REC                 CBL.CBLI.SDO(PLAYBACK)
0015 CBL.CBLIMST.STDTEST.*.RECS                CBL.CBLI.SDO(PLAYBACS)
0016 NBJ.SELCTRN.ZZST2DAT                 COBOL NBJ.SELCTRN.SAM1(ZZST2CPC)
Se | Line=1 | Col=1 | Alt=0,0;0 | Size=16 | Recl=1094 | Fmt=V | Files=1 | Views
```

Figure 2. SDE: Structure to Dataset Associations Table.

Data File Mask Matching

In identifying an association for a given data file name, a search of the associations table is performed in the following order of precedence, stopping when a match is found.

1. Search data file masks that contain no wild card characters.
2. For library datasets with member name specification, search data file masks that contain a parenthesised member name mask. Both the library DSN mask and member name mask can contain wildcard characters.
3. For library datasets with member name specification, search data file masks that contain no wild card characters in the library DSN and no parenthesised member name mask.
4. Search all remaining data file masks that contain wild card characters.

Note that, within the scope of the above search order, if the data file matches more than one data file name mask then the first matching association definition within the table will be selected.

SDE Window Views

An SDE window is an MDI child window (view) that may be opened from within the CBLe Text Editor or SELCOPY Interactive application parent window.

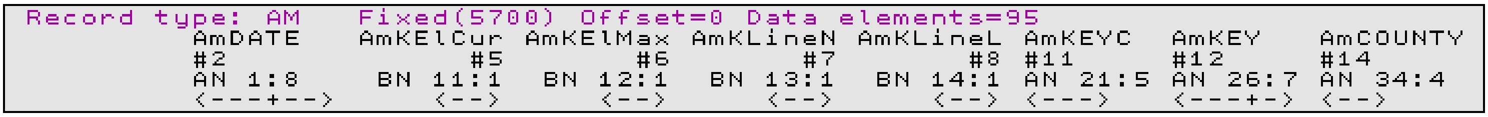
Data within an SDE window is presented as either a multi record or single record view.

Multi Record View

This is an SDE MDI child window containing multiple records displayed horizontally so that each record occupies a single line of the display.

Field Column-Heading Lines

By default, a group of 5 field column-heading lines are displayed within the window display area immediately above the first data record in view. Unless records associated with a different record type are visible (see Record Data Display below), no further groups of column-heading lines are displayed within the current display area view.



```
Record type: AM   Fixed(5700) Offset=0 Data elements=95
         AmDATE   AmKElCur AmKElMax AmKLineN AmKLineL AmKEYC  AmKEY   AmCOUNTY
         #2             #5       #6       #7       #8 #11     #12     #14
         AN 1:8    BN 11:1  BN 12:1  BN 13:1  BN 14:1 AN 21:5 AN 26:7 AN 34:4
         <---+-->     <-->     <-->     <-->     <--> <--->   <---+-> <-->
```

Figure 3. SDE Multi-Record Display Header Lines.

Header Line 1: Record Type Information

For SDE display of file records, this header identifies:

1. The record-type assigned to the records (or record segments) that follow.
2. The record-type format (Fixed or Variable) followed by the RTO Data Length in "( )" (parentheses). Note that, for variable record-types, the valid range of RTO data lengths is displayed as a minimum and maximum RTO data length value.
3. The offset into the record at which record type begins to format the data. This may be a positive or negative value. (Offset=*n*)
4. The number of data elements (field references) within the record-type definition. This number includes the record-type STRUCTURE field, and any other STRUCTURE fields in the RTO definition. Therefore, the data elements value may be greater than the number of fields in view.

   Note that a repeating group (array) contains a fixed number of data elements which is reflected in the total data element count. Each array element posesses a one or more dimensional vectors, with each vector having been defined maximum number of entries. Array vector entries are not reflected in the data element count.

For SDE DB2 table row display, this header identifies:

1. The record type (**Row name**) assigned to the table rows - default is the table owner ID name.
2. The record type format (Fixed or Variable) followed by the record type length (i.e. the sum of the record type field lengths) in "( )" (parentheses). Note that, for variable record types a maximum,minimum length pair is displayed.
3. The number of columns defined by the DB2 results table.

Header Line 2: Field Name

Where present, the names of fields selected for display, are used as the column header names and are located in the second line of the column header.
This header line may not be switched off.

Header Line 3: Field Reference

The third column header line displays the field number (reference) within the record. By default, when table format is specified, the fields displayed will be the lowest level of a nested group of fields. Therefore, the names of fields that define a group of fields, will not be displayed and so their corresponding field numbers will also be missing from the sequence of displayed fields.
This header line may be switched on/off using the SET REFERENCE command.

Header Line 4: Field Type

For DB2 table edit, the fourth column header line displays the field data type as specified in the table column definition.

For data files, the fourth column header line displays the field structure (type) as a 2-character data type code followed by the field's start position (byte number within the record-type definition) and the field length. For fields whose lengths are defined in number of bits, the field start position includes an offset into the byte following a "." (decimal point) and the unit for field length is bits. The start position and length displays are separated by a ":" (colon).

Recognised file editing data type codes are as follow:

AN	CHARACTER, VARCHAR, XVARCHAR, STRUCTURE or UNION
BI	BINTEGER
BN	INTEGER
BT	BIT
FB	FIXEDHEX, FIXEDBIN
FP	FLOAT
PD	DECIMAL
X	HEXADECIMAL
ZD	ZONED

See *"SDE Data Types"* for descriptions of each of the supported data types and their representation in the Field Type header line.
This header line may be switched on and off using the SET TYPE command.

Header Line 5: Scale

The fifth column header line displays a scale for each field in the display. The width of the scale is equivalent to the width of the field display area which is wide enough to display all the field data or, for numeric fields, the largest possible numeric value.

Where the display field is wider than 2 bytes, a scale begins with "<" (less than) as position 1 and ends with ">" (greater than) as the last position of the field.
The scale characters displayed and the the ability to switch this header line on and off is controlled by the SET SCALE command.

Prefix Area

By default, the record prefix is set on and so the prefix area is displayed containing the record sequence numbers. The record prefix area may tailored or set on/off using the SET PREFIX command.

Data Records

By default, all records of the file, regardless of assigned record type, are selected for display.
Records that have been selected for display are identified as **data records**.

Using the VIEW command, records of different record types may be displayed or suppressed within a multi record view.

Where multiple record types are visible, each record in the window display area that is of a different record type to the previous record, will be immediately preceded by its column headers. Therefore, when viewing a mixture of records of different record types, the display area may contain multiple groups of column header lines. It is also possible that only part of a group of column header lines is displayed at the bottom of the display area.

Within formatted data records, the display of character (AN, CH or VARCHAR) fields that contain non-printable characters differs from the rest of the displayed fields in both multi record and single record views. Printable characters in these fields are displayed in a different colour with underscore highlighting so that non-printable characters appear as immoveable gaps, effectively fixing the length of printable character data before them. Only printable characters in these fields may be overtyped. Non-printable characters may only be updated using the CHANGE command or by setting HEX ON, so allowing overtype of the hex display of the data.

Scrolling left and right will scroll the fields within all records of the default record type only.
Scrolling up and down will scroll through all records of the file.

Shadow Lines

A record or group of consecutive records that are not displayed because they are suppressed, specifically excluded by the user or have no associated RTO (not selected) are, by default, replaced by a **shadow line**.

A shadow line specifies the number of consecutive records excluded from the display, the reason for their exclusion ("suppressed", "excluded" or "not selected") and, if applicable, the associated record type.
Shadow lines may be set on/off using the SET SHADOW command.

Single Record View

An SDE MDI child window containing data from a single record displayed vertically so that each field of the record occupies a single line of the display.

Records that are suppressed, excluded or not selected cannot be viewed within a single record view.

Scrolling left and right will display the previous and next records respectively that have not been suppressed.
Scrolling up and down will display the previous and next fields within the record respectively.

A group of 3 field column-heading lines are displayed within the window display area immediately above the first field in view.

Header Line 1: Record Type Information
Identical to **Header line 1** in a multi-record view.

Header line 2: Record Information
Contains the record (or record segment) sequence number within the file, any assigned record flags (as displayed by the RECINFO option) and the record length.

Header line 3: Field Data Headers
Contains the column headers for the record data that follows. The headers displayed in this line and so the format of the data displayed beneath, may be altered using the SHOW command.

Column headers and data displayed for each occur left to right as follows: and may be changed by setting, the

Ref
If SHOW NUMBER is in effect, the **Ref** header is visible so that field reference (data element) numbers, assigned by FileKit to each field, are displayed.

Field
If SHOW LEVEL is **not** in effect, the **Field** header is displayed to the right of the **Ref** header. For each field entry that follows, only the field name is displayed and, for array field elements, the vector entry subscript is included in parentheses.

Alternatively, if SHOW LEVEL is in effect, then, by default, **Field** header occupies the first two column headers, replacing the **Ref** header, and for each field entry that follows, a hierarchical level number is displayed to the left of the field name. Indentation occurs for each successive level number to highlight the record structure. In this way the Field column entries resemble a COBOL copy book definition.
Note that the **Ref** header may be redisplayed, in addition to this hierarchical format of the **Field** data, by setting option REFERENCE on.

Type

If SHOW TYPE is in effect, the **Type** header is displayed to the right of the **Field** header. For each field entry that follows, the field's 2-character data type code, its position within the record-type definition and its maximum width is displayed as for **Header Line 4: Field Type** in a multi-record view (code pos:length).

Format

If SHOW FORMAT is in effect, the **Format** header is displayed to the right of the **Field** header. For each field entry that follows, the field's maximum width and data type is displayed as width/type.

Picture

If SHOW PICTURE is in effect, the **Picture** header is displayed to the right of the **Field** header. For each field entry that follows, the COBOL or PL/1 field definition PICTURE string. If the field was defined without a PICTURE string, entries are displayed using the **Type** column data format.

Offset | **Position** | **HexOff**

If SHOW OFFSET is in effect, one of the following headers is displayed to the right of the **Field** header depending on the current value of the OFFSET option.

◊ **Offset**
Displayed when OFFSET RELATIVE is in effect, for each field entry that follows, this column displays the decimal offset of the field within the record-type definition. If the data displayed for a character data field wraps onto the next line, the offset within the record of the first character to be wrapped onto the new line is displayed in this column in a different colour.

◊ **HexOff**
Displayed when OFFSET HEX is in effect, this column is identical to **Offset** except offsets are displayed in hex.

◊ **Position**
Displayed when OFFSET COLUMNS is in effect, this column is similar to **Offset** except that the field position (i.e. field offset+1) within the record-type definition is displayed instead.

Scale Header

To the right of all other column headers, the scale header is simply a counting guide for the field data displayed beneath. The scale line, and so the width of field data displayed, always contains a multiple of 10 character columns which increases or decreases accordingly as the width of the non-maximised edit view window is stretched or compressed.

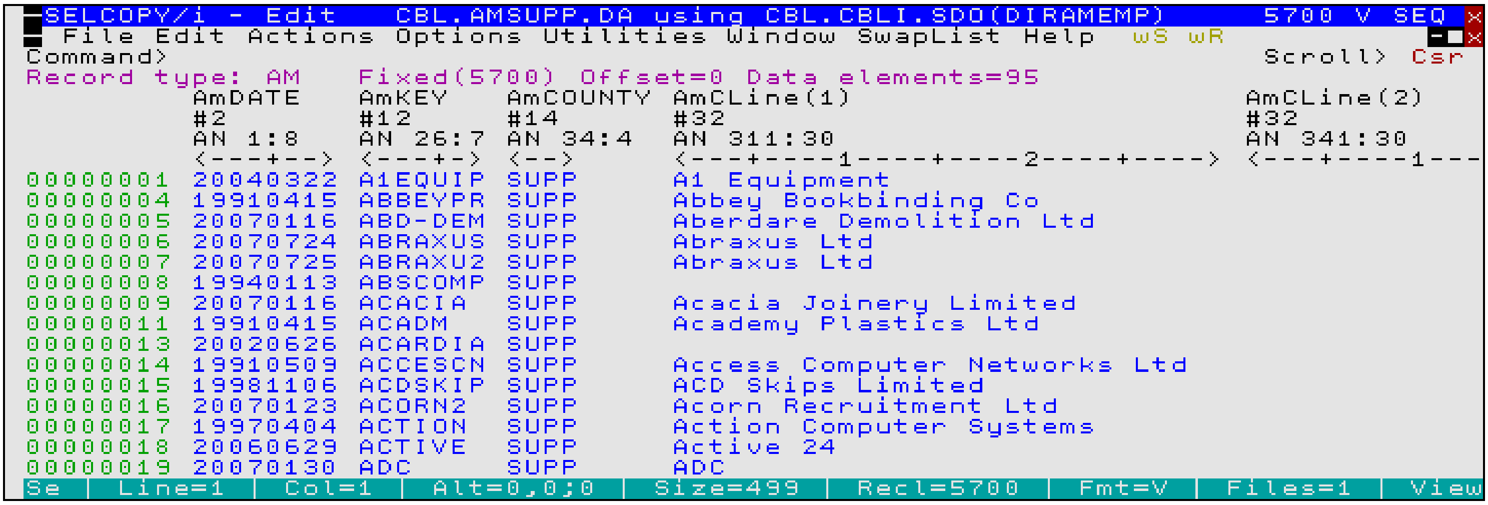
Character data fields that have a width greater than that of the scale will wrap onto the next line beneath the scale header. In this case, the start and end positions of characters within the data field that occur on the wrapped line are displayed in a different colour in the **Type**, **Format** or **Picture** columns. If the **Offset**, **HexOff** or **Position** columns are visible, then only the offset or position of the first character on that line is displayed.

SDE Data Formats

Data within an SDE window view is displayed in one of the following data formats.

Table Format

Forces a multi record window view. If table format is selected on EDIT or BROWSE, then records are assigned an appropriate record type (RTO) and formatted so that record data is split into its component fields as defined by the RTO. The data is processed so that all fields are displayed as printable character, numeric data fields having first been converted to decimal.

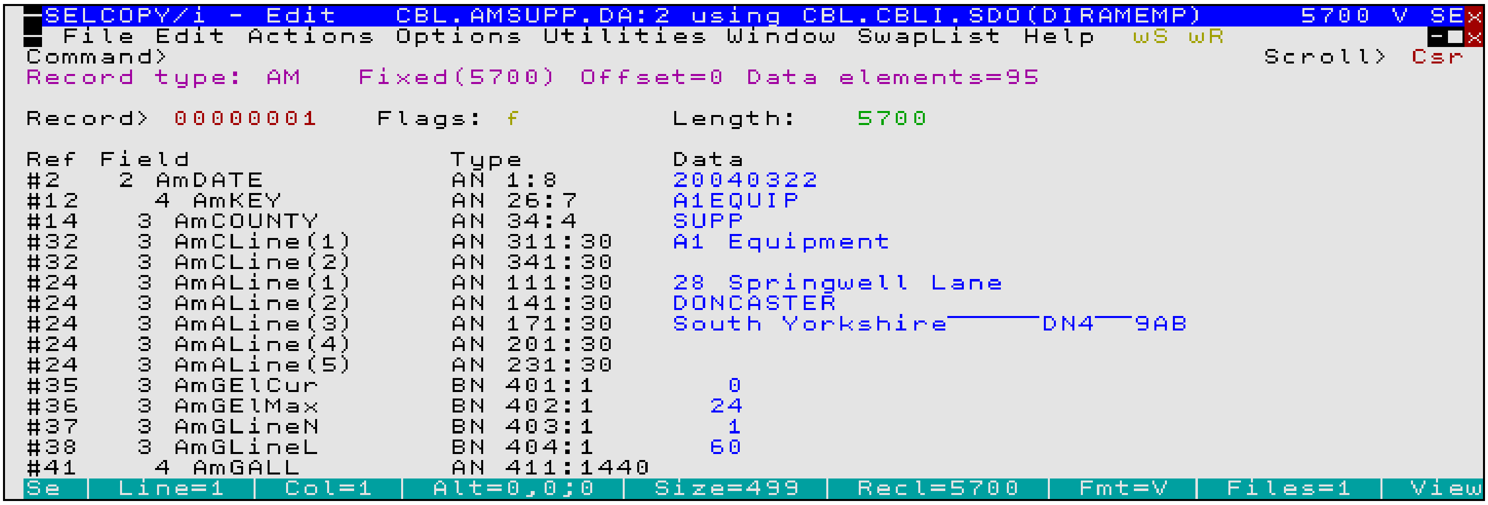


```
SELCOPY/i - Edit    CBL.AMSUPP.DA using CBL.CBLI.SDO(DIRAMEMP)     5700 V SEQ
 File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                           Scroll> Csr
Record type: AM    Fixed(5700) Offset=0 Data elements=95
         AmDATE    AmKEY   AmCOUNTY AmCLine(1)                     AmCLine(2)
         #2        #12     #14      #32                            #32
         AN 1:8    AN 26:7 AN 34:4  AN 311:30                      AN 341:30
         <---+---> <---+-> <-->     <---+----1----+----2----+----> <---+----1---
00000001 20040322  A1EQUIP SUPP     A1 Equipment
00000004 19910415  ABBEYPR SUPP     Abbey Bookbinding Co
00000005 20070116  ABD-DEM SUPP     Aberdare Demolition Ltd
00000006 20070724  ABRAXUS SUPP     Abraxus Ltd
00000007 20070725  ABRAXU2 SUPP     Abraxus Ltd
00000008 19940113  ABSCOMP SUPP
00000009 20070116  ACACIA  SUPP     Acacia Joinery Limited
00000011 19910415  ACADM   SUPP     Academy Plastics Ltd
00000013 20020626  ACARDIA SUPP
00000014 19910509  ACCESCN SUPP     Access Computer Networks Ltd
00000015 19981106  ACDSKIP SUPP     ACD Skips Limited
00000016 20070123  ACORN2  SUPP     Acorn Recruitment Ltd
00000017 19970404  ACTION  SUPP     Action Computer Systems
00000018 20060629  ACTIVE  SUPP     Active 24
00000019 20070130  ADC     SUPP     ADC
Se |  Line=1  |  Col=1  |  Alt=0,0;0  |  Size=499  |  Recl=5700  |  Fmt=V  |  Files=1  |  View
```

Figure 4. SDE Table Format.

Single Format

Record data is processed as for a table view except that the display becomes a single record window view of the default record.

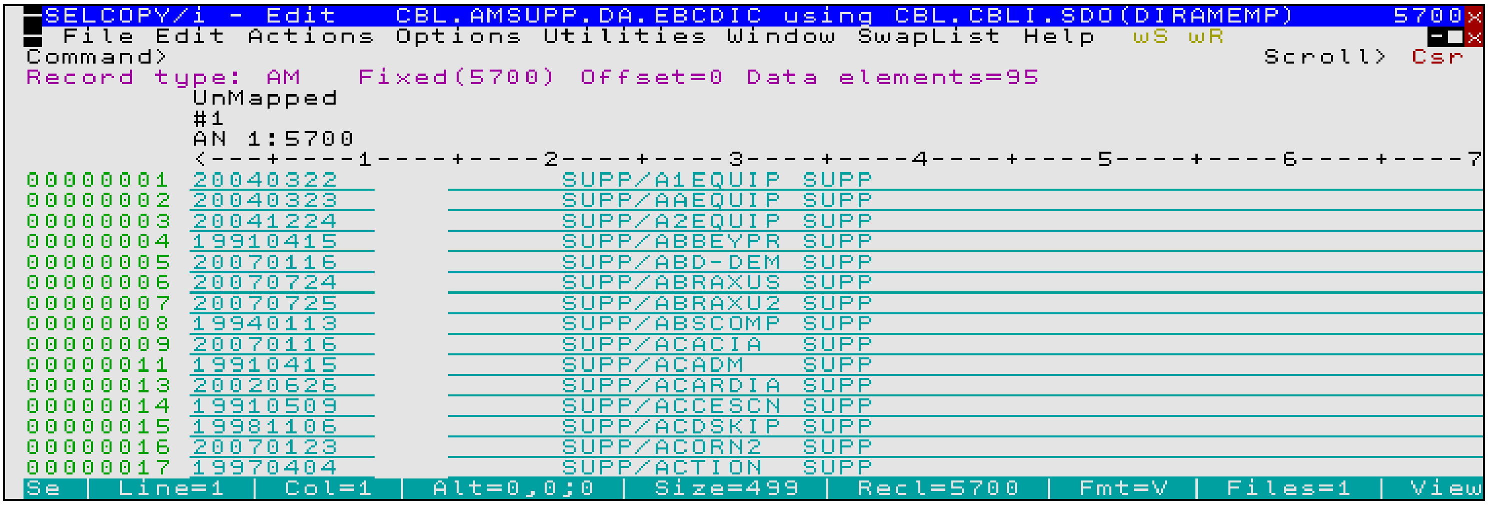


```
 SELCOPY/i - Edit    CBL.AMSUPP.DA:2 using CBL.CBLI.SDO(DIRAMEMP)     5700 V SE
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                             Scroll> Csr
Record type: AM   Fixed(5700) Offset=0 Data elements=95

Record> 00000001   Flags: f         Length:   5700

Ref  Field                  Type        Data
#2    2 AmDATE              AN 1:8      20040322
#12     4 AmKEY             AN 26:7     A1EQUIP
#14    3 AmCOUNTY           AN 34:4     SUPP
#32    3 AmCLine(1)         AN 311:30   A1 Equipment
#32    3 AmCLine(2)         AN 341:30
#24    3 AmALine(1)         AN 111:30   28 Springwell Lane
#24    3 AmALine(2)         AN 141:30   DONCASTER
#24    3 AmALine(3)         AN 171:30   South Yorkshire─────DN4──9AB
#24    3 AmALine(4)         AN 201:30
#24    3 AmALine(5)         AN 231:30
#35    3 AmGElCur           BN 401:1       0
#36    3 AmGElMax           BN 402:1      24
#37    3 AmGLineN           BN 403:1       1
#38    3 AmGLineL           BN 404:1      60
#41     4 AmGALL            AN 411:1440
Se | Line=1 | Col=1 | Alt=0,0;0 | Size=499 | Recl=5700 | Fmt=V | Files=1 | View
```

Figure 5. SDE Single Format.

Character Format

Record data in the current multi or single record view is mapped as a single, variable length character field with field name "UnMapped" or, for record segments, "UnMappedSeg". No conversion of numeric or unprintable data is performed and the window view (either multi or single record) remains unchanged.



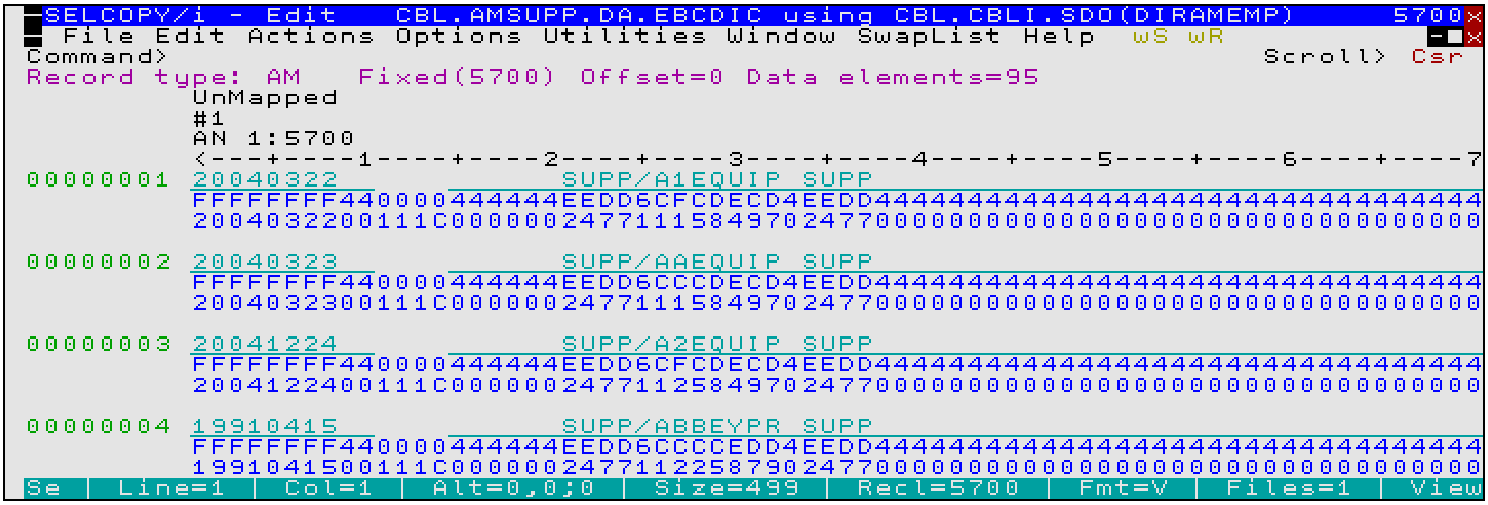
```
 SELCOPY/i - Edit    CBL.AMSUPP.DA.EBCDIC using CBL.CBLI.SDO(DIRAMEMP)     5700
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                             Scroll> Csr
Record type: AM   Fixed(5700) Offset=0 Data elements=95
         UnMapped
         #1
         AN 1:5700
         <----+----1----+----2----+----3----+----4----+----5----+----6----+----7
00000001 20040322            SUPP/A1EQUIP SUPP
00000002 20040323            SUPP/AAEQUIP SUPP
00000003 20041224            SUPP/A2EQUIP SUPP
00000004 19910415            SUPP/ABBEYPR SUPP
00000005 20070116            SUPP/ABD-DEM SUPP
00000006 20070724            SUPP/ABRAXUS SUPP
00000007 20070725            SUPP/ABRAXU2 SUPP
00000008 19940113            SUPP/ABSCOMP SUPP
00000009 20070116            SUPP/ACACIA  SUPP
00000011 19910415            SUPP/ACADM   SUPP
00000013 20020626            SUPP/ACARDIA SUPP
00000014 19910509            SUPP/ACCESCN SUPP
00000015 19981106            SUPP/ACDSKIP SUPP
00000016 20070123            SUPP/ACORN2  SUPP
00000017 19970404            SUPP/ACTION  SUPP
Se | Line=1 | Col=1 | Alt=0,0;0 | Size=499 | Recl=5700 | Fmt=V | Files=1 | View
```

Figure 6. SDE Character Format.

Hexadecimal Format

Record data in the current multi or single record view is displayed as for character format with the addition that the hexadecimal representation of the data is displayed below the character data.

Note that the Hex representation occupies an additional 2 lines of the display for each record and consists of characters 0-9, A-F (indicating 4-bit binary values, B'0000'-B'1111'). The first line contains the high order 4 bits of the byte and the second line contains the low order 4 bits.



```
SELCOPY/i - Edit   CBL.AMSUPP.DA.EBCDIC using CBL.CBLI.SDO(DIRAMEMP)     5700
 File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                        Scroll> Csr
Record type: AM   Fixed(5700) Offset=0 Data elements=95
         UnMapped
         #1
         AN 1:5700
         <---+----1----+----2----+----3----+----4----+----5----+----6----+----7
00000001 20040322            SUPP/A1EQUIP SUPP
         FFFFFFFF440000444444EEDD6CFCDECD4EEDD444444444444444444444444444444444
         2004032200111C00000024771115849702477000000000000000000000000000000000

00000002 20040323            SUPP/AAEQUIP SUPP
         FFFFFFFF440000444444EEDD6CCCDECD4EEDD444444444444444444444444444444444
         2004032300111C00000024771115849702477000000000000000000000000000000000

00000003 20041224            SUPP/A2EQUIP SUPP
         FFFFFFFF440000444444EEDD6CFCDECD4EEDD444444444444444444444444444444444
         2004122400111C00000024771125849702477000000000000000000000000000000000

00000004 19910415            SUPP/ABBEYPR SUPP
         FFFFFFFF440000444444EEDD6CCCCEDD4EEDD444444444444444444444444444444444
         1991041500111C00000024771122587902477000000000000000000000000000000000
Se | Line=1 | Col=1 | Alt=0,0;0 | Size=499 | Recl=5700 | Fmt=V | Files=1 | View
```

Figure 7. SDE Hexadecimal Format.

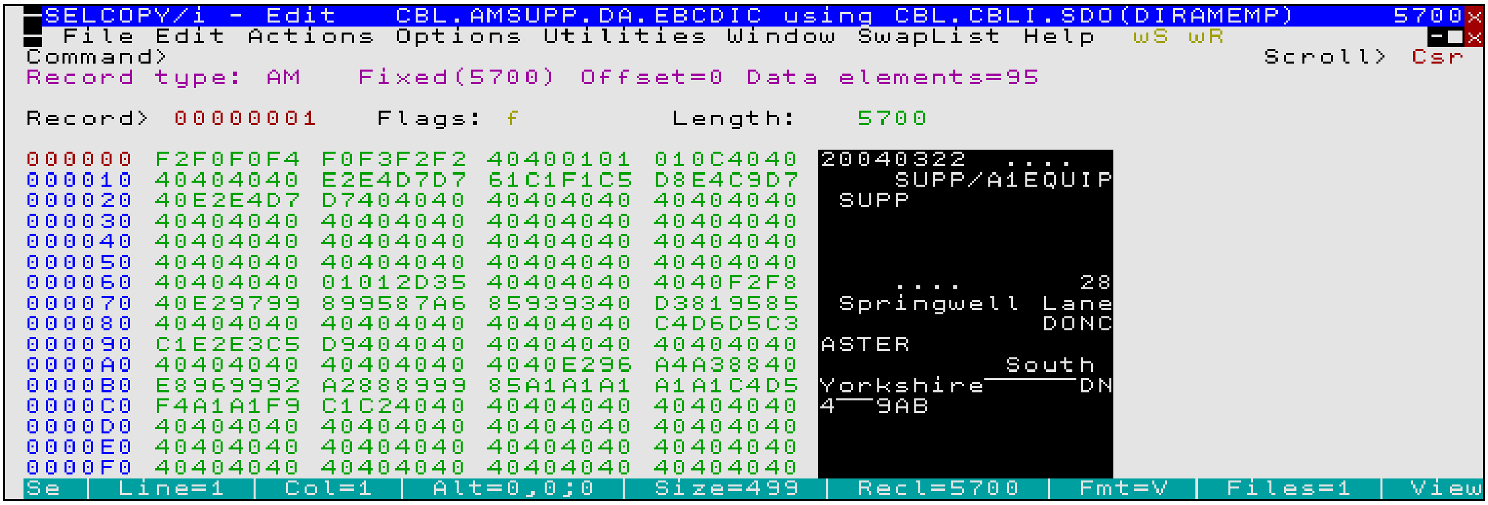
Hexadecimal Dump Format

Record data is displayed without any record-type formatting, in a single record window hex dump view of the default record.

On the left of the display is a field containing the position of the data within the record (in hex or decimal representation.)

Following this field, the record data is represented by a number of blank delimited fullwords (groups of 4 bytes) in long hex format, followed by the equivalent number of bytes in character format.

Long hex format is where the hexadecimal representation of a character occupies two characters of the display. e.g. C'A' is represented as C'C1' in long hex format.



```
SELCOPY/i - Edit   CBL.AMSUPP.DA.EBCDIC using CBL.CBLI.SDO(DIRAMEMP)     5700
 File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                        Scroll> Csr
Record type: AM   Fixed(5700) Offset=0 Data elements=95

Record> 00000001   Flags: f        Length:   5700

000000 F2F0F0F4 F0F3F2F2 40400101 010C4040 20040322  ....  
000010 40404040 E2E4D7D7 61C1F1C5 D8E4C9D7     SUPP/A1EQUIP
000020 40E2E4D7 D7404040 40404040 40404040  SUPP           
000030 40404040 40404040 40404040 40404040                 
000040 40404040 40404040 40404040 40404040                 
000050 40404040 40404040 40404040 40404040                 
000060 40404040 01012D35 40404040 4040F2F8     ....      28
000070 40E29799 899587A6 85939340 D3819585  Springwell Lane
000080 40404040 40404040 40404040 C4D6D5C3             DONC
000090 C1E2E3C5 D9404040 40404040 40404040 ASTER           
0000A0 40404040 40404040 4040E296 A4A38840           South 
0000B0 E8969992 A2888999 85A1A1A1 A1A1C4D5 Yorkshire‾‾‾‾‾DN
0000C0 F4A1A1F9 C1C24040 40404040 40404040 4‾‾9AB          
0000D0 40404040 40404040 40404040 40404040                 
0000E0 40404040 40404040 40404040 40404040                 
0000F0 40404040 40404040 40404040 40404040                 
Se | Line=1 | Col=1 | Alt=0,0;0 | Size=499 | Recl=5700 | Fmt=V | Files=1 | View
```

Figure 8. SDE Hexadecimal Dump Format.

SDE Data Types

Structured records may be mapped by a record type object (RTO) whereby fields may be interpreted as being of any of the the following supported field types:

Bit Flags

A Binary field of length specified in number of bits where each bit is interpreted as a flag switch which is either ON (1) or OFF (0). Fields of this data type have a data type record header of **"BT *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bits).

Binary Data Fixed

A **DB2** Binary data field (built-in data type BINARY) of fixed length whose contents are interpreted as printable ASCII or EBCDIC character data. Fields of this data type have a data type record header of **"BINARY(*nn*)**" where *nn* is the length of the field (in bytes).

Binary Data Variable

A **DB2** Binary data field (built-in data type VARBINARY) of variable length contents are interpreted as printable ASCII or EBCDIC character data. Fields of this data type have a data type record header of **"VARBIN(*nn*)**" where *nn* is the maximum length of the field (in bytes).

Character Fixed

A Character field of fixed length specified in number of bytes whose contents are interpreted as printable ASCII or EBCDIC character data. Fields of this data type have a data type record header of **"AN *ppp*:*nn***" or, for **DB2** table edit/browse, "**CH(*nn*)**" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Character Variable (Field of Varying Length with Nominated Length Field)

A Character field of variable length specified in number of bytes by a nominated field within the record data. The nominated field may be of any numerical data type and the character field contents are interpreted as printable ASCII or EBCDIC character data. Fields of this data type have a data type record header of **"AN *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Character Variable (Field of Varying Length with Preceding Length Field)

A Character field of variable length specified in number of bytes by an Integer Binary (Byte) field located immediately before the character data. The Integer Binary field is of a predefined length which may be defined as being included with or excluded from the character field length. The character field contents are interpreted as printable ASCII or EBCDIC character data. Fields of this data type have a data type record header of **"AN *ppp*:*nn***" or, for **DB2** table edit/browse, "**VARCHAR(*nn*)**" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Character Variable (Field of Static Length with Preceding Length Field)

A Character field of variable length padded with blanks to occupy a field of a fixed, predefined length. The variable length of the character data is determined by a 2-byte Integer Binary (Byte) field located immediately before the character data. The 2-byte Integer Binary field length is not included within the stored length. This data type is equivalent to PL/1 fields declared as CHARACTER VARYING. The character field contents are interpreted as printable ASCII or EBCDIC character data. Fields of this data type have a data type record header of **"AN *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes). The value *nn* includes the 2-byte length field.

Character Variable (Field of Static Length with Null Termination)

A Character field of variable length occupying a field of a fixed, predefined length. The end of the variable length character data is determined by at least one null (x'00') termination character. Therefore, in order to support a character strings of the maximum field length, the predefined fixed field length is always 1 byte longer than the maximum length requested. This data type is equivalent to PL/1 fields declared as CHARACTER VARYINGZ. The character field contents are interpreted as printable ASCII or EBCDIC character data. Fields of this data type have a data type record header of **"AN *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes). This value includes the additional 1-byte.

Date

A Date field displayed in ISO format with separator character "/" (yyyy/mm/dd) or, for **DB2** table edit/browse, in the locally defined DB2 date string format. The internal format of the date string for non-DB2 edit/browse may be that returned by the TIME Assembler macro for packed or binary data, or the internal date format of ICF catalog records or VTOC format 1/3 records. Fields of this data type have a data type record header of **"DT"** or, for DB2 table edit/browse, "**DATE**".

Fixed Point Binary

A Binary field defined as having a precision and scale specified in number of decimal digits, and whose value is displayed as a fixed point decimal number. The length of the binary field is either 2-bytes (Halfword), 4-bytes (Word) or 8-bytes (Doubleword) as implied by the precision which defines the total number of decimal digits. i.e. precision <5 implies halfword, precision >=5 and <10 implies word, precision >=10 implies doubleword. The scale defines the number of digits to the right of the decimal point. Fields of this data type have a data type record header of **"FB *ppp*:*nn*"** where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Fixed Point Packed Decimal

A Packed Decimal field defined as having a precision and scale specified in number of decimal digits, and whose value is displayed as a fixed point decimal number. The length of the packed decimal field is implied by the precision which defines the total number of decimal digits. The scale defines the number of digits to the right of the decimal point. Fields of this data type have a data type record header of **"PD *ppp*:*nn*"** or, for **DB2** table edit/browse, "**DEC(*pr*,*sc*)**" where *ppp* is the position of the field within the expanded record (in bytes), *nn* is the length of the field (in bytes), *pr* is the defined precision and *sc* the defined scale.

Fixed Point Zoned Decimal

A Zoned Decimal Character field defined as having a precision and scale specified in number of bytes, and whose value is displayed as a fixed point decimal number. The length of the zoned decimal field is implied by the precision which defines the total number of decimal digits. The scale defines the number of digits to the right of the decimal point. Fields of this data type have a data type record header of **"ZD *ppp*:*nn*"** where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Floating Point Hexadecimal

A Hexadecimal Floating Point field of length 4 bytes (short) or 8 bytes (long) whose value is displayed as a decimal number consisting of a normalised mantissa and exponent. Fields of this data type have a data type record header of **"FP *ppp*:*nn*"** or, for **DB2** table edit/browse, "**REAL**" for short fields and "**FLOAT**" for long fields where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Floating Point Binary

A Binary Floating Point (IEEE 754) field of length 4 bytes (short) or 8 bytes (long) whose value is displayed as a decimal number consisting of a normalised mantissa and exponent. Fields of this data type have a data type record header of **"FP *ppp*:*nn*"** where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Floating Point Decimal

A Decimal Floating Point (IEEE 754r) field of length 8 bytes or 16 bytes whose value is displayed as a decimal number consisting of a normalised mantissa and exponent. Fields of this data type have a data type record header of **"FP *ppp*:*nn*"** or, for **DB2** table edit/browse, "**DECFLOAT(*pr*)**" where *ppp* is the position of the field within the expanded record (in bytes), *nn* is the length of the field (in bytes) and *pr* is the defined precision (16 or 34).

Hexadecimal

A Hexadecimal field of a fixed length specified in number of bytes whose contents are displayed as a printable hexadecimal character string. Fields of this data type have a data type record header of **"X *ppp*:*nn*"** or, for **DB2** table edit/browse, "**BINARY(*nn*)**" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Hexadecimal Variable (Field of Varying Length with Preceding Length Field)

A Hexadecimal field of variable length specified in number of bytes by an Integer Binary (Byte) field located immediately before the printable hexadecimal data. The Integer Binary field is of a predefined length which may be defined as being included with or excluded from the character field length. Fields of this data type have a data type record header of **"X *ppp*:*nn*"** or, for **DB2** table edit/browse, "**VARBIN(*nn*)**" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Hexadecimal Variable (Field of Static Length with Preceding Length Field)

A Hexadecimal field of variable length padded with blanks to occupy a field of a fixed, predefined length. The variable length of the printable hexadecimal data is determined by a 2-byte Integer Binary (Byte) field located immediately before the data. The 2-byte Integer Binary field length is not included within the stored length. Fields of this data type have a data type record header of **"X *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes). The value *nn* includes the 2-byte length field.

Integer Binary (Bit)

A Binary field of length specified in number of bits, the contents of which are interpreted as being numeric and whose value is displayed as an integer decimal number. Fields of this data type have a data type record header of **"BI *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bits). Compare with Integer Binary (Byte) data type which is a binary integer field with length expressed in number of bytes.

Integer Binary (Byte)

A Binary field of length specified in number of bytes (maximum 8-bytes), the contents of which are interpreted as being numeric and whose value is displayed as an integer decimal number. Fields of this data type have a data type record header of **"BN *ppp*:*nn***" or, for **DB2** table edit/browse, "**SMINT**" (halfword), "**INT**" (word) or "**BIGINT**" (doubleword) where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes). Compare with Binary Integer (Bit) data type which is a binary integer field with length expressed in number of bits.

PL/1 Picture String (Character)

A PL/1 style PICTURE string representing a character data item (i.e. no numerical interpretation) of length determined by the specified picture string. The picture string may contain any valid PL/1 picture character for character data. Fields of this data type have a data type record header of **"AN *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

PL/1 Picture String (Fixed Point Numerical)

A PL/1 style PICTURE string representing a FIXED numerical character data item of length determined by the specified picture string. The picture string may contain any valid PL/1 picture character for numeric character data except for exponent characters "E" and "K". Fields of this data type have a data type record header of **"AN *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

PL/1 Picture String (Floating Point Numerical)

Not Yet Supported
A PL/1 style PICTURE string representing a FLOAT numerical character data item of length determined by the specified picture string. The picture string may contain any valid PL/1 picture character for numeric character. Fields of this data type have a data type record header of **"AN *ppp*:*nn***" where *ppp* is the position of the field within the expanded record (in bytes) and *nn* is the length of the field (in bytes).

Structure

A Structure consisting of one or more fields each defined as being any of the supported data types. The structure field length is defined as being the sum of all field lengths within the structure. A field within a structure may itself be of the structure data type, so defining a structure nested within a structure.

Time

A Time field displayed with separator character ":" (HH:MM:SS) or, for **DB2** table edit/browse, in the locally defined DB2 time string format. The internal format of the time string for non-DB2 edit/browse may be that returned by the TIME Assembler macro for packed or binary data. Fields of this data type have a data type record header of **"TI**" or, for DB2 table edit/browse, "**TIME**".

Timestamp

A Timestamp (Date and Time) field displayed in ISO format with date separator character "/" (yyyy/mm/dd) and time separator character ":" (HH:MM:SS), or, for **DB2** table edit/browse, in the locally defined DB2 timestamp string format. The internal format of the date and time string for non-DB2 edit/browse may be that returned by the TIME Assembler macro for packed or binary data, the system TOD clock (returned by the STCK operation) or an HFS file directory entry. Fields of this data type have a data type record header of **"TS**" or, for DB2 table edit/browse, "**TIMESTAMP**".

Union

A Union of one or more fields of any of the supported data types. Each field in the union occupies the same start position in the data record. The union field length is defined as being the length of the longest field in the union. Unions allow the same data to be interpreted as more than one data-type.

SDE Edit Techniques

SDE data edit employs sophisticated techniques for in-storage record data management. These techniques allow for more control over the data being edited and also allows the user to edit data sets that would not otherwise fit in storage.

Prior to edit, FileKit SDE establishes the user supplied edit options, the data set organisation and size, and the amount of storage available. Based on these, SDE will perform the appropriate editing technique as follows:

Full Edit

Supported for sequential data sets, PDS(E) members, DB2 tables and VSAM RRDS and ESDS files defined with option REUSE, all records belonging to the data set are loaded into storage prior to display in an SDE window view. Full edit supports change, insert, delete, copy and move of data records. Furthermore, records may be updated to alter the record length if supported by the data set organisation or structure (SDO).

This edit technique will be used where conditions for Update-in-place and KSDS edit are not satisfied. Before a data set is opened for Full Edit, SDE first establishes whether enough storage is available to load the entire data set. For non-DB2 table edit, if insufficient storage is available or AUXILIARY is specified on the EDIT command, AUXILIARY edit will be used instead. For DB2 table edit, insufficient storage will result in a storage error.

Although Full Edit can result in a delay as the data set is loaded into storage, subsequent edit operations that involve scrolling or data scanning will operate faster. When the data set is saved, the entire data set is re-written back to disk using the in-storage copy of the records.

If Full Edit is in effect, "Edit" is displayed in the SDE window title bar to the left of the DSN.

Read Only Edit

Supported for sequential data sets, PDS(E) members, DB2 tables and VSAM RRDS and ESDS files, all records belonging to the data set are loaded into storage prior to display in an SDE window view.

Read Only edit supports all functionality provided by Full Edit except that changes cannot be saved to disk unless the fileid is first updated (using SET DSN, FILEID, FMODE, FNAME, FPATH or FTYPE) to be a non-existing fileid. Having done this, saving the data will open a dialog window to allocate or IDCAMS DEFINE the data set. This allows the user to effectively edit and create sample data before copying it to a new data set, whilst being secure in the knowledge that the orginal data set will remain intact.

Read Only edit is invoked by specifying READONLY on the SDE EDIT CLI command. Like Full Edit, AUXILIARY edit will be used if insufficient storage is available or AUXILIARY is specified on the EDIT command.

If Read Only Edit is in effect, "Edit(RO)" is displayed in the SDE window title bar to the left of the DSN. If AUXILIARY edit is used for read only editing, "Edit(AUXRO)" is displayed instead.

Update-in-place Edit

Not supported for DB2 table edit, this edit technique allows in-place record update only. Records cannot be inserted, deleted, moved or copied and record lengths cannot be altered. The exception to this is RRDS update-in-place edit where empty slots may be created by record delete and made enterable by record insert.

Update-in-place edit is invoked or implied by the following:

1. Specification of parameter UPDATE on the Data Editor EDIT command.
2. Parameter FROM and/or FOR (indicating start record and number of records) is specified on the EDIT command.
3. Data set is a VSAM ESDS or RRDS file with IDCAMS DEFINE CLUSTER option NOREUSE.
4. Data set is not VSAM KSDS and load of the data set into storage has been interrupted by the user following receipt of the Load Threshold break-in modal window.

Before a data set is opened for Update-in-place Edit, SDE first establishes whether enough storage is available to load the entire data set. If insufficient storage is available, SDE will only load a single page of records for display in the SDE window view. Subsequently, only records that have been altered and records displayed in the current SDE window view are held in storage.

When the data set is saved, any record that has alterations is updated in place (replaced) by the modified, in-storage copy of the record.

If Update-in-place Edit is in effect, "Edit(UP)" is displayed in the SDE window title bar to the left of the DSN.

KSDS Edit

KSDS Edit is used for Read Only, Full Edit or Update-in-place Edit of VSAM KSDS data sets.

For KSDS Edit, only records that have been altered, inserted or deleted and records displayed in the current SDE window view are held in storage. It is possible to do this, even for Full Edit, as KSDS records may be deleted and inserted out of sequence.

The capabilities and restrictions of Read ONly Edit, Full Edit and Update-in-place Edit still apply and "Edit(RO)", "Edit" or "Edit(UP)" is displayed in the title bar as appropriate. Note that Read Only Edit or Update-in-place Edit of KSDS data sets is invoked using the READONLY or UPDATE parameter of the SDE EDIT command respectively, otherwise Full Edit is default.

AUXILIARY Edit

Auxiliary edit first makes a copy of the edit data set to a second, auxiliary file on disk.

Only records that have been altered, inserted or deleted and records in the current SDE window view are held in storage. For Auxiliary Edit, a data set can only be saved when it is closed during exit of the SDE window view. This is becuase the data set is completely re-written using a combination of records obtained from the auxiliary data set and records held in storage. Note that SAVE is not supported using this edit technique.

Auxiliary edit is used to provide full edit capabilities for sequential data sets, PDS(E) members and VSAM RRDS and ESDS files defined with option REUSE, when one of the following is true:

1. The data set is too large to be loaded into available storage.
2. Parameter AUXILIARY is specified on the SDE EDIT CLI command.

If Auxiliary Edit is in effect, "Edit(AUX)" is displayed in the SDE window title bar to the left of the DSN.

The auxiliary data set is allocated as new by SDE and subsequently deleted after the edited file is closed. The data set name has HLQ prefix, unit and/or System Managed Storage (SMS) classes as defined in the Data Edit Auxiliary Dataset Settings panel (=0.4.4). The default data set name HLQ is **%USER%.FILEKIT.SDEAUX** whereas the default unit and SMS classes are the same as the data set being edited. The remainder of the data set name is of the format Dyyyyjjj.Thhmmsst representing the current date and time.

If Auxiliary Edit is to be performed on a sequential data set defined with DSNTYPE=LARGE, then the auxiliary data set will also be allocated as DSNTYPE=LARGE.

Changing Record Lengths

The SDE is a structured data editor and so, in order to preserve the structure of record data, the lengths of updated records are unchanged when the data is saved. This occurs whether or not a structure (SDO) or copybook has been applied to the record data and has the effect of preserving any trailing blanks that exist in the records.

However, the length of a variable length record may be altered in the following circumstances:

1. Execution of the CHANGE command results in a change to the record length. This may be a change to a numerical field value that represents the current length of a variable length field, or a change to the length of character data in a variable length field.

   In particular, use of CHANGE with parameter DATA and different length search and replace strings will result in a field length change.
   Note that, for variable length records, record type "UnMapped" is used for unformatted data and comprises a single field (field name "UnMapped") which is of variable length.

2. When the record information Length field is displayed (RECLENGTH ON or SET RECINFO ON LENGTH), then the length of a record may be altered by simply overtyping the Length value with the new record length.

   Altering the record length in this way will force a re-assessment of the record type assigned to the record data. This is done using standard record type (RTO) assignment rules.

3. When a structure (SDO) or copybook has been applied to the record data and the assigned record type contains length or array vector fields, then overtyping the values in any of these fields will also update the record length accordingly.

Attempting to alter the record length will fail if the new length is greater than the maximum LRECL defined for the file.

SDE Current Window

The current SDE window is the SDE window view on which focus was last placed. This may be the focus window or, if the focus window is not an SDE window view, the last SDE window view visited.

The concept of a current SDE window view is important when executing SDE commands from a CBLe text edit view via the SDATA command which directs the SDE command to the current SDE window view.

Using SDATA is particularly useful for issuing stored SDE commands from within a CMX file.

Default Record Type

SDE windows are capable of displaying records of more than one record type simultaneously within the same display area.

Because many SDE commands and functions operate on records of a single, specific record type, the concept of a default record type exists for use when a record type has not been specifically identified via a command parameter.

The following identifies the order in which the default record type is determined by FileKit SDE:

1. The record type of the focus line if the focus line is a visible record or an EXCLUDED shadow line.
2. The prevailing setting of DRECTYPE.

The value of DRECTYPE is assigned when the SDE window is opened and is established in the following order:

1. The record type that is the first, non-generic argument specified on the VIEW parameter of the EDIT or BROWSE command.
2. The record type of the RTO in the SDO defined as being the DEFAULT.
3. The record type of the first visible record of the edited or browsed data.
4. The record type of the first RTO in the SDO.

The DRECTYPE value may be updated by any of the following:

- SET DRECTYPE explicitly sets the DRECTYPE value.
- Execution of the VIEW command will update DRECTYPE to be the first, non-generic record type argument.
- Execution of any command that supports a *record_type* argument will update DRECTYPE to be the record type selected for the command. This is true whether the record type has been specified explicitly in the command syntax or implied by the record type of focus line. (e.g. WHERE, LOCATE, etc.)

The current value DRECTYPE may be interrogated using the QUERY or EXTRACT DRECTYPE commands.

SDE support for DB2 data

Included as part of FileKit DB2 functionality, the SDE data editor has support for viewing and changing DB2 table data.

Since the DB2 support is integrated into the SDE editor, all SDE commands and operations, such as FIND, CHANGE, WHERE and LOCATE, apply in the same way to the display and edit of DB2 table data as for other types of operating system datasets.

- DB2 Table Browse and Edit
- DB2 Table Display

DB2 Table Browse and Edit

The FileKit structured data editor (SDE) supports BROWSE and EDIT of DB2 tables and views.

Full reference information for structured edit is in FileKit Structured Data Editor (SDE) Contents and there is also an SDE section in the comprehensive FileKit Quick Reference document.

The following topics provide introductory information about working with DB2 data in structured edit:

- Starting a DB2 table edit or browse session.
- The "PkUIFCND" Column Attributes display.
- How DB2 subsystem connections are managed.
- Limiting the number of rows loaded.
    - ♦ Using non-scrollable cursors.
    - ♦ Using scrollable cursors.
    - ♦ Scrollable cursor example.
- Defining a structure to use with DB2 tables.
- How DB2 rows are saved.
- Dealing with save errors.
- Auditing edit sessions.
- Editing or browsing related tables
- Useful commands.
- Comparing DB2 table edit with dataset edit.

Starting a DB2 table edit or browse session

The EDIT and BROWSE primary commands with the DB2 subsystem parameter start a DB2 table edit or browse session. For example the command:

```
browse db2(cbla) sysibm.systables
```

will open a structured browse view and load into storage all the rows of the table **SYSIBM.SYSTABLES** for subsystem **CBLA**. A temporary structure will be generated using the DB2 catalog to define the columns and their data types.

These primary commands have many optional parameters and it is not, in general, practical to type the whole command you need on the command line. Alternative methods are:

- The DB2 edit panel. This is option 4 of the DB2 primary option menu. This panel will generate an EDIT command from fields entered on the panel and either execute it or place it in a dataset as a command to be executed by the ACTION key.
- The DB2 browse panel. This is option 5 of the DB2 primary option menu. This panel will generate a BROWSE command from fields entered on the panel and either execute it or place it in a dataset as a command to be executed by the ACTION key.
- Using the ACTION key to execute an edit or browse primary command stored in a text (command) file.

The **WHERE** and **ORDER BY** parameters of the edit or browse command are used to generate an **SQL SELECT** statement which is used to initially load the rows of the table.

The "PkUIFCND" Column Attributes display

Whilst browsing or editing a DB2 table you may view a row in "single-record" mode either by typing the MAP command, or by pressing the "ZoomW" key (Shift-F5).

In this mode, to the left of each value, along with each column's name and data-type you will see the column headed **PkUIFCND** which displays column attributes reflecting information obtained from the DB2 catalog table SYSIBM.SYSCOLUMNS as follow.

More detailed information is available by entering the INFO primary command.

Attribute	Description
Pk	A 2 byte field displaying the numeric position of the column within the **Primary Key**, otherwise blank.
U	Displays **"u"** which indicates that this column is part of a **unique key**, otherwise blank.
I	Displays **"d"** which indicates that this column is part of a **non-unique (duplicate) key index**, otherwise blank.
F	Displays **"f"** which indicates that this column is part of a **referential constraint** foreign key, otherwise blank.
C	Displays **"c"** which indicates that this column is involved in a **check constraint**, otherwise blank.
N	Displays **"n"** which indicates that this column supports a **null value**, otherwise blank.
D	Displays one of the following single character **DEFAULT value indicators**. See description of the DEFAULT column of table SYSIBM.SYSCOLUMNS in the *"IBM DB2 SQL Reference"* manual.

How DB2 subsystem connections are managed

In order to access DB2 data FileKit must make a connection to a DB2 subsystem. The architecture of DB2 for z/OS is such that an operating system task can have at most one connection to DB2.

Since FileKit wants to support multiple concurrent edits of DB2 tables in multiple subsystems, it uses multiple connection subtasks so that each window accessing DB2 data can potentially have its own connection. FileKit uses these connection subtasks as follows:

- For browse and listing the DB2 catalog there is one task connecting to each active subsystem. This connection can be reused for any number of lists or browses. It is created when the first request to browse or list for a subsystem is processed and destroyed when the last such request terminates.
- For edit there is one dedicated connection task for each active session. This is required so that committing and rolling back changes to the data can be isolated from other edit sessions. This connection is created and destroyed with the edit session.

Limiting the number of rows loaded

FileKit support for DB2 edit and browse has two distinct methods for managing the loading of rows into virtual storage:

- Using a non-scrollable cursor. With this method all eligible rows are loaded into virtual storage at the start of the edit or browse session, so for large tables some method of restricting the eligible rows must be used.
- Using a scrollable cursor.

Non-scrollable cursors

With a non-scrollable cursor there are two methods of limiting the number of rows loaded which can be used separately or together:

1. Using the **FROM** and **FOR** parameters. For example:

   ```
   browse db2(cbla) sysibm.systables from 21 for 100
   ```

   will open a structured browse view and load 100 rows of the table **SYSIBM.SYSTABLES** starting at the 21st row.
2. Using the **WHERE** parameter which has a standard SQL **WHERE** clause in parentheses as an argument. For example:

   ```
    browse db2(cbla) sysibm.systables where(createdby='NBJ')
   ```

   will open a structured browse view and load all rows of the table **SYSIBM.SYSTABLES** for which the **CREATEDBY** column is equal to the string **'NBJ'.**

   Note that the argument to the **WHERE** parameter can by given as **?** which will invoke the DB2 row selection dialog which can be used to generate a **WHERE** clause without having to type the column names. So for example:

   ```
    browse db2(cbla) sysibm.systables where(?)
   ```

   will first run the row selection dialog so that a **WHERE** clause can be generated to limit the rows loaded before starting the browse.

Scrollable cursors

The use of a scrollabe cursor is triggered by the **SCROLL** keyword parameter of the EDIT or BROWSE command.

Note that the use of DB2 scrollable cursors may be disabled by the installer of FileKit. In order to use them the system INI file must contain the **DB2.SCROLL=YES** variable.

FileKit uses a scrollabe **SENSITIVE STATIC** cursor for EDIT and a scrollable **INSENSITIVE** cursor for BROWSE. With either of these types of cursor the results table has a fixed size determined when the cursor is opened. The results table is materialised (a temporary copy is made) during open and this means that for large tables opening the cursor may take a long time and consume much resource.

Once the cursor has been opened the results table functions rather like a fixed sized VSAM RRDS. It contains a fixed number of rows each of which may be directly referenced by row number. The edit mode is update in place, so rows cannot be inserted or moved. Deletes are allowed, although the deleted row is not removed from the display, becoming instead a form of shadow line.

With this form of table browse and edit only a relatively small number of rows is kept in storage at any time. The fact that rows can be retrieved directly by row number means that direct access to the middle or end of the table is relatively fast because intermediate rows do not have to be read.

The size of the results table is not known until the last row has been read and the table size in the status bar is prefixed by a greater than sign to indicate this. Scrolling to the end of the table (with command **DOWN MAX** for example) will establish the actual table size.

As for non-scrollable cursors, a **WHERE** clause can be used to limit the size of the results table, but the **FOR** and **FROM** keywords cannot be used with the **SCROLL** keyword.

For browse the use of an **INSENSITIVE** cursor means that the results table does not change even if other processes are deleting or updating rows in the base table.

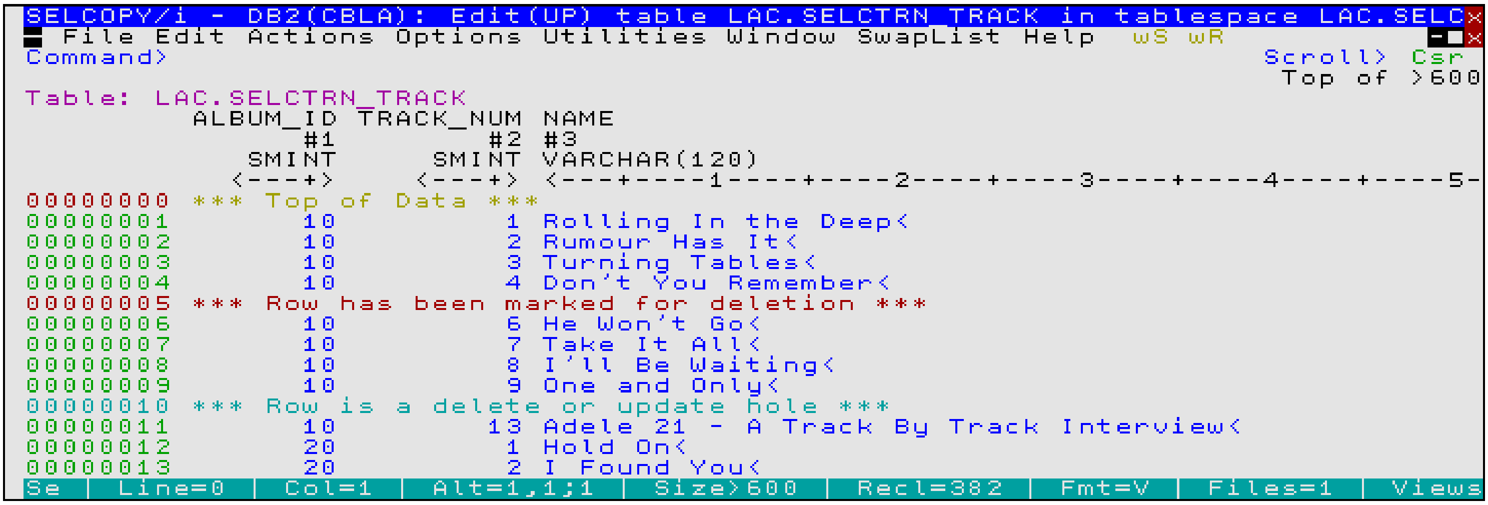
For edit the use of a **SENSITIVE STATIC** cursor means that when another process either deletes a row or updates a row so that it no longer satisfies the **WHERE** clause then such a row becomes a **cursor hole** and can no longer be fetched. Cursor holes are represented by a form of shadow line.

Scrollable cursor example

In the following example table **LAC.SELCTRN_TRACK** is being edited with a scrollable sensitive static cursor.

- **Edit(UP)** in the panel title bar indicates that the edit mode is **update in place** so that rows cannot be moved or inserted.
- **Top of >600** at the top right of the panel indicates that 600 rows of the table have been fetched but the end of the table has not yet been reached.
- Row 5 has been deleted (using the **D** prefix command) and the *** **Row has been marked for deletion** *** message indicates the change has not yet been saved.
- Row 10 has been deleted or updated by another process and having been re-fetched was found to be a **cursor hole** as indicated by the *** **Row is a delete or update hole** *** message.

  The row was re-fetched because only a subset of table rows are kept in storage and the user has scrolled away from the top of the table and then scrolled back causing the first part of the table to have been removed from storage and then reloaded.



```
SELCOPY/i - DB2(CBLA): Edit(UP) table LAC.SELCTRN_TRACK in tablespace LAC.SELC
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                           Scroll> Csr
                                                                    Top of >600
Table: LAC.SELCTRN_TRACK
         ALBUM_ID TRACK_NUM NAME
               #1        #2 #3
            SMINT     SMINT VARCHAR(120)
           <---+>    <---+> <---+----1----+----2----+----3----+----4----+----5-
00000000 *** Top of Data ***
00000001       10         1 Rolling In the Deep<
00000002       10         2 Rumour Has It<
00000003       10         3 Turning Tables<
00000004       10         4 Don't You Remember<
00000005 *** Row has been marked for deletion ***
00000006       10         6 He Won't Go<
00000007       10         7 Take It All<
00000008       10         8 I'll Be Waiting<
00000009       10         9 One and Only<
00000010 *** Row is a delete or update hole ***
00000011       10        13 Adele 21 - A Track By Track Interview<
00000012       20         1 Hold On<
00000013       20         2 I Found You<
Se | Line=0 | Col=1 | Alt=1,1;1 | Size>600 | Recl=382 | Fmt=V | Files=1 | Views
```

Figure 9. DB2: Scrollable cursor edit.

Defining a structure to use with DB2 tables

FileKit uses Structure Definition Objects (SDOs) to map the data displayed in structured data edit or browse windows. An SDO contains a description of the attributes of the data to be formatted when using structured edit or browse.

There is no need to predefine an SDO for use with DB2 table edit and browse. The DB2 catalog contains the names and attributes of the table columns and FileKit uses this information to automatically generate a temporary SDO if no predefined SDO is given as an argument of the **USING** parameter of the edit or browse command.

However if when you want to look at a particular table you often require the same **WHERE** clause, limiting the rows retrieved; the same **ORDER BY** clause, defining the order in which you want to view the rows; the same **SELECT** command syntax, defining a subset of columns you want to display in the initial view, and other concurrency, locking and commit options, you may want to use the CREATE STRUCTURE command to generate an SDO containing all these options and then just specify that SDO in your edit or browse command.

How DB2 rows are saved

When the SAVE command is issued (either explictly or implicitly when closing the edit session) each row that has been deleted, inserted or updated since the last time that data was saved (or since the start of the edit session) is processed by generating and executing an appropriate dynamic SQL statement to achieve the required change.

If a non-zero SQLCODE is returned as a result of executing the generated statement, the SQL communication Area (SQLCA) describing the error condition is saved with the row.

When generating the SQL statement for delete or update the target row must be identified in some way.

If the table has a unique index or a **ROWID** column then these columns can be used to uniquely identify the target row and a searched update or delete can be used of the form:

```
UPDATE table-name SET ... WHERE search-condition
DELETE FROM table-name    WHERE search-condition
```

where the **search-condition** is generated from the unique index or rowid.

If the table does not have a unique index or a **ROWID** column then there is no way of uniquely identifying the row to be deleted or updated. In this case all the columns of the target row are used to generate the **WHERE** clause of a **SELECT** statement. A cursor is defined by this statement, it is opened and the first row fetched is used as the target and a positioned update or delete is used of the form:

```
UPDATE table-name SET ... WHERE CURRENT OF cursor-name
DELETE FROM table-name    WHERE CURRENT OF cursor-name
```

If a delete fails, then the row in error, which has already been removed from the displayed list of rows, must be re-inserted so that the problem can be analysed. Any such rows are re-inserted at the top of the current view.

Finally, depending on the seting of the COMMIT option, the changes may or may not be committed.

Dealing with save errors

Any non-zero SQLCODEs returned as a result of saving changes are stored with the affected row in a copy of the SQL communication Area (SQLCA).

In table view the SQLCODE is displayed in the prefix area or, if the SET RECINFO ON SQLCODE option is in effect, in the SQLCODE column of the record information area.

In single view the SQLCODE is displayed in its own field at the top of the view.

You can use the E prefix command on the row in error or the ERROR primary command when the focus row is the row in error to display detailed information about the SQLCODE in the DB2 Save SQL Error Panel.

Auditing edit sessions

You can keep a record of your DB2 table edit sessions by including the **AUDIT** parameter in your EDIT command.

With this option an audit log file is allocated and all the edit actions taken in the session are recorded.

From the Audit Trail Functions menu you can specify the characteristics of the audit datasets using the Audit Log Dataset Options panel, and print an audit log dataset using the Print Audit Report panel.

Editing or browsing related tables

FileKit has specific support for editing tables which have relationships defined by referential integrity (RI) constraints.

An RI constraint establishes a parent-dependent relationship between two tables by means of a foreign key. A foreign key is a set of columns in the dependent table which correspond to a unique key in the parent table.

When an RI constraint exists certain types of change to the tables involved are not permitted. Breaking these rules leads to an **RI error** of one of the following types:

- Missing parent key **SQLCODE -530.** This happens when an insert or update of a row in a dependent table has a foreign key which does not represent an exising row in the parent table. In other words you cannot make orphans by inserting or updating foreign keys in the dependent table.
- Parent key update error **SQLCODE -531.** This happens when an update of a row in a parent table changes a key which is a foreign key in a dependent table and has dependent rows in that table. In other words you cannot make orphans by changing parent keys.
- RI delete rule violation **SQLCODE -532.** When the RI constaint was defined with a delete rule of **RESTRICT** or **NO ACTION** this error will happen when deleting a row in the parent table which has dependent rows in the dependent table. In other words for this type of RI constraint deletes must be done from the bottom up.

The RE prefix command and REDIT primary command can be used in a number of ways with tables related by RI constraints:

- List all parent and dependent relationships for the current table.
- Edit or browse the parent row or dependent rows of the focus row for any RI constraint.
- After an RI error when saving changes, edit the parent row or dependent rows of the row which had the error.

Useful commands

Some commands are specifically for DB2 table edit. The following overview is not exhaustive but highlights some of the features:

Command	Description
E prefix command ERROR	Display a panel explaining the meaning of a non-zero SQLCODE returned by DB2 when saving changes.
INFORMATION (INFO)	Generate and display an HTML format report of the attributes of the current edit or browse table. This report lists all the columns, indexes, contraints etc of the current table.
RE prefix command REDIT (RE)	List, edit or browse tables related to the current table by referential constraints.
SET/QUERY/EXTRACT COMMIT	Specify when saved changes to the table are committed.
SET/QUERY/EXTRACT COLATTRIBUTES	Control display of Column Attributes information (heading=**"PkUIFCND"**) in single-record (MAP) format diplay. Attributes include information on Prime, Unique and Non-Unique keys, Referential and Check Constraints, Null and Default values.

SET/QUERY/EXTRACT EPK
SET/QUERY/EXTRACT EDITPRIMEKEY
	Specify whether the table prime key columns should be enterable or not.

SET/QUERY/EXTRACT NULLCHAR	Specify special characters which indicate whether a column has the null value (for output) or is to be set to the null value (on input).

SET/QUERY/EXTRACT NULLIFBLANK	Specify whether a column which is nullable is to be set to the null value if it is input as blank.
SET/QUERY/EXTRACT VENDCHAR	Specify special characters which indicate the end of variable length character data on input and output.
SET/QUERY/EXTRACT VSTRIP	Specify whether variable length character columns are to be stripped of trailing blanks.

Comparing DB2 table edit with dataset edit

A browse or edit session of a DB2 table looks very like that of an operating system dataset with table rows corresponding to dataset records. The edited data is manipulated in the same way and most of the edit commands such as FIND and CHANGE etc. work as expected.

However, because of the structural differences between DB2 maintained data and operating system datasets, some commands and options are not relevant and others only apply to DB2 data.

Some of the more obvious differences between editing DB2 tables and operating system datasets are summarised in the following table:

	DB2	**VSAM Sequential PDS(E) HFS**
I/O	I/O is column based using a dynamic SQL QUERY (SELECT statement) to fetch the data and dynamic SQL DELETE, INSERT and UPDATE statements to change the data.	I/O is record or byte stream based using operating system access methods.
Data map	There is always an implicit structure mapping the data defined by the selected columns and the DB2 catalog. The unmapped FORMAT CHARACTER is not relevant and not supported.	Any data mapping structure has to be explicitly defined (for example using COBOL or PL/1 copybooks) and explicitly applied to the dataset. Datasets may be edited or browsed in FORMAT CHARACTER with no mapping structure applied.
Record type	There is always exactly one record type defined by the selected columns.	A dataset may have many record types.
Group fields	There is only one level of data. There are no group fields.	The structure applied to the dataset may have many levels of group field.
Row order	DB2 tables are conceptually sets with no inherent row order. Order is imposed only at QUERY time with the ORDER BY clause of the SELECT statement.	Datasets have an inherent record order defined by the access method.
Saving changes	The SAVE command deals with each deleted, inserted or updated row by constructing and executing a dynamic SQL statement for the change to that row. Individual SQLCODEs are saved for each row and are visible when the command completes either in the row prefix area or, if the option RECINFO ON SQLCODE is in effect, in the record information SQL code area.	The SAVE command stops at the first error.
Commit and roll back	Changes made to DB2 data with the SAVE command must be committed before they become permanent changes to the database. When this happens is controlled by the COMMIT option. The CANCEL command may be used to roll back any changes to the previous commit point and end the edit session.	Changes made to a dataset with the SAVE command are permanent. Datasets have no built in capability for committing or rolling back changes.

DB2 Table Display

Differences between the SDE window display of structured data set records and that of DB2 tables, are detailed below.

Header Lines

The Record Type header line displays the table name.

The Field Type header line displays the the DB2 built-in data type as specified in the results table column definition. i.e. no field position is displayed and any applicable field length or field precision and scale are represented in "( )" (parentheses).

See *"SDE Data Types"* for descriptions of each of the supported data types and their representation in the Field Type header line.

Window View Format

DB2 table SDE views support multi record and single record views as displayed for parameters TABLE and SINGLE of the FORMAT command. Display of table data in unformatted character or hex (FORMAT CHAR/HEX) is not supported. However, HEX ON/OFF may be used to display, not only the interpreted printable character representation of the formatted field, but also its content in hexadecimal.

Record Information Columns

Record information (RECINFO) field ID is not applicable to DB2 table rows and so is always suppressed. However, record information field **SQLCODE** may be displayed to report the last SQL code received following an SQL INSERT, DELETE or UPDATE performed on the table row as a result of a SAVE operation.

In single view for edit only, there is a separate SQLCODE field in which is displayed any non-zero SQL code returned by the last save command.

DB2 Column Attributes

Single record view of a table row includes an additional column within the display which flags attributes defined to the individual table columns. These column attributes are obtained from the DB2 catalog table SYSIBM.SYSCOLUMNS and flags whether a column is part of the primary key, a unique key, foreign key, a duplicate key index, has a check constraint, a default value and/or supports a null value.

Display of the column attributes information may be suppressed using the COLATTRIBUTES ON/OFF option.

Varible Length Fields

Data in variable length character fields (VARCHAR) that is not of the maximum length is suffixed with a "<" (less than) character. The "<" itself does not form part of the data but is displayed to indicate that the character to the left is the last character of the field data. This indication is particularly necessary where the variable length field contains trailing blank characters which are to be preserved.

In editing the field data the "<" indicator may be overtyped or shifted right as appropriate. If overtyped or deleted, and the field data is not of the maximum length, then FileKit will reinsert the "<" indicator following the last **non-blank** character. (i.e. any trailing blanks will not be preserved.)

The "<" indicator may also be manually deleted and re-entered in blank data following the last non-blank character of the field. This technique may be used to insert trailing blanks in the field. Beware that, when using this technique, if the original "<" indicator is not deleted first then it will become part of the field data.

Pop-up Utility Menu

Pressing <F16> in an SDE DB2 table edit view now opens the DB2 Browse/Edit Utilities Menu for DB2 specific table editing options instead of the SDE Browse/Edit Utilities Menu for structured file editing options.

Segmented Records

Records may be organised in such a way that they are split into a number of record segments, each segment being mapped by a unique structure (COBOL group or PL1 major/minor structure).

Segmented records begin with a single primary (base) segment immediately followed by any number of non-overlapping, secondary segments. A secondary segment may have the same or different segment record-type (RTO) mapping as other secondary segments in the record.

```
Record: 1
 +-----------+--------------+--------------+--------------+--------------+
 | Primary_1 | Secondary_1  | Secondary_1  | Secondary_1  | Secondary_1  |
 +-----------+--------------+--------------+--------------+--------------+

Record: 2
 +-----------+--------------+-----------------+-----------------+
 | Primary_1 | Secondary_1  | Secondary_2     | Secondary_2     |
 +-----------+--------------+-----------------+-----------------+

Record: 3
 +----------------+--------------+------------------------------------------+
 | Primary_2      | Secondary_1  | Secondary_4                              |
 +----------------+--------------+------------------------------------------+

Record: 4
 +-----------+--------------+-----------------+-----------------------+
 | Primary_1 | Secondary_1  | Secondary_2     | Secondary_3           |
 +-----------+--------------+-----------------+-----------------------+
```

The record data must contain ID fields that identify which segment mapping is to be used to format individual segments of the record.

ID fields that identify a primary segment mapping must exist within the primary segment data. ID fields that identify a secondary segment mapping may exist within the secondary segment data, within the data of any previously mapped segment belonging to the same record, or, specifically, within the primary segment data.

Structure Definition and Record-Type Assignment

A FileKit SDE structure may be created using the CREATE STRUCTURE command or Create Structure (SDO) Menu panels (=9) to format segmented records so that a record type (RTO) mapping definition exists for each record segment in the file.

Each RTO is defined as being either a primary (PRI) or secondary (SEC) segment map, or the default (primary) segment map (DEF). Except for the default segment RTO, which must **not** be defined with segment identification criteria, all primary and secondary segment RTOs must be defined with (USE WHEN) segment identification criteria.

A primary segment RTO is assigned to primary (base) segment of the record. However, if ID fields do not satisfy the segment identification criteria for any of the primary segment RTOs, the default segment RTO is used. See *"Record Type Assignment"* which defines the precedence for assigning primary segment RTOs.

Unprocessed record data that follows the primary segment is assigned a secondary segment RTO based on associated identification criteria. The assignment of secondary segment RTOs is repeated for all remaining record data until the end of record is reached or the remaining data does not satisfy any segment identification criteria belonging to the secondary segment RTOs. In the latter case, the remaining record data is displayed as unformatted character data with the record type "UnmappedSeg".

Specifying USE WHEN Segment Identification Criteria

USE WHEN segment identification criteria is mandatory for non-default primary and secondary segment RTO definitions.

Segment identification criteria associated with each RTO definition is based on an SDE expression. This identifies all ID fields within the record data and also values against which those ID fields will be tested in order to determine whether the RTO segment mapping is suitable to be assigned to an individual record segment.

Within any RTO segment mapping USE WHEN expression, ID fields may be referenced using the following methods:

Unformatted data position and length

Built-in function SEGPOS() may be used in the USE WHEN *expression* record type identification criteria to reference a data field in the current segment or the segment immediately preceding the current segment. Similarly, BASPOS() may be used to reference a data field in the primary (base) segment only.

e.g. The following identification criterion is based on an ID field of length 3 characters located 10 bytes before the end of the segment immediately preceding the current segment:

```
USE WHEN  SegPos(-10,3) = C'X12'
```

If the ID field is a Packed Decimal value then LEFTDEC() may be used to obtain the numeric value with or without a scale. e.g. The following ID field is a packed decimal field located at position 12 of the primary segment:

```
USE WHEN  LeftDec(BasPos(12)) > 31
```

Formatted data field

A formatted ID field references a field in the current segment or within a previous segment of a specific structure (record type) name. If referencing a field in a previous segment, the record type name which maps that segment must be specified as a high level qualifier to the field name (i.e. *record-type.field_name*). If more than one preceding segment is mapped using *record-type*, then the the one closest to the current segment is tested.

Because formatted fields have a specific data type, the USE WHEN expression may involve operators and terms based on the data type of the ID field. e.g. The following identification criterion is based on a floating point ID field XVAL in a preceding segment of record type MAPBASE1:

```
USE WHEN  MAPBASE1.XVAL > 27.83
```

Segmented Records SDE Browse/Edit View

As for display of non-segmented records, files containing segmented records may be displayed in multi-record (FORMAT TABLE/CHAR/HEX) or single-record (FORMAT SINGLE) view.

Primary commands CHAR, MAP (FMT), VFMT and UNFMT are supported to display the current segment in multi-record unformatted character view, single-record formatted view, multi-record formatted view and single-record unformatted character view respectively.

Data in record segments is displayed in exactly the same way as non-segmented records. Prefix area, record type headers and record information is also displayed as for non-segmented records with the following exceptions:

1. Segmented record header display differs from that of non-segmented records only in that, instead of "**Record Type:**", header line 1 of each group of matching segment types displays either "**Base:**", for primary segments; "**Base(D):**", for default primary segments; or " **; Seg:**", for secondary segments. The leading blanks for on the heading for secondary segments causes them to appear visually indented underneathe their parent primary segment when displayed in multi-record table format.

   In order to easily distinguish between primary and secondary segments, the header line 1 of secondary segments is coloured yellow by default. See SET/QUERY/EXTRACT option COLOUR (or COLOR) SEGMENT.

2. For secondary segments only, the record information id column, displayed using the SET/QUERY/EXTRACT option RECINFO, displays the offset into the record of the start of the segment.

   Primary segments display the TTR/Offset or RBA value of the start of the record as for non-segmented records.

3. Prefix area numbers in a multi-record view correspond to either record or segment numbers in the file as determined by the SET PREFIX option. In single-record view, PREFIX PHYSICAL will display the record number and the current segment number within the record, whereas PREFIX LOGICAL diplays only the current record segment number within the file.

   Using command LOCATE to navigate the file data, is sensitive to the current setting (PHYSICAL or LOGICAL) of the PREFIX option. Therefore, if PREFIX LOGICAL is in effect, "LOCATE 35" will scroll to the 35th (primary or secondary) segment within the file, **not** the 35th record, of the file.

Navigating Segmented Record Display

In both multi-record and single-record views, the display of segments may be navigated using standard SDE CLI (primary) commands FIND, LOCATE, UP, DOWN, LEFT, RIGHT, TOP, BOTTOM, NEXT and PREV.

Note that, in single-record view, LEFT and RIGHT will scroll backwards and forwards respectively through every segment in the file regardless of the type of segment (primary or secondary) or record to which a segment belongs.

Commands NEXT and PREV may be used to restict scrolling to only secondary segments belonging to the current record. This is the NEXT and PREV default, however, parameters are supported to scroll to the next and previous primary or unmapped (UnmappedSeg) segments.

Segmented Record Edit

FileKit SDE segmented record edit is supported for all file organisations other than VSAM KSDS.

Full Edit, Auxiliary Edit and Update-in-Place Edit SDE edit techniques are all supported for segmented record edit.

Data Updates that affect Record Segment Mapping

The RTO mapping assigned to a segment is determined by values in ID fields which may occur in the same segment or within the primary and/or previous secondary segments. Therefore, any change to a segment may affect the RTO mapping identification criteria for that segment or any subsequent segment in the record.

FileKit SDE is able to detect changes to ID fields which have been identified via a USE WHEN expression that includes only fields referenced explicity by name or field reference number. Changes to ID field name "Unmapped" (reference number #1) are detected only when record-type formatting is disabled. Where an ID field change is detected, or where the length of the segment data is altered via CHANGE or RECLEN updates, the ID flag is set on for the changed segment.

The ID flag, represented by "m" in the record information flag display and ==ID> in multi-record view prefix area, is an indication that the record to which this segment belongs may require a remap (re-assignment of segment RTOs) based on new ID field values. This flag may be ignored but may be of importance if the user wants to first verify that integrity between the required segment mapping and its identification criteria is maintained prior to saving the data and ending the edit session.

Remap of a record and reset of the ID flag is achieved using the primary command IDENTIFY, or prefix (line) commands **ID<*n*>** or **IDD**. Note that the remap is **not** performed automatically in order to allow the user the opportunity to make further changes to the data before executing IDENTIFY. Note that, if ID field values do not satisfy any secondary segment mapping identification critera, the remainder of that record is displayed as unformatted data (UnmappedSeg).

Remap of a record may also be required where an update has occurred for an ID field identified in a USE WHEN expression by its unformatted record position and length, or when segments have been inserted, deleted, duplicated, copied or moved. For this reason, the default action of the IDENTIFY command is to not only remap records containing segments with the ID flag, but also records flagged as having changed segments. (See option IDSCOPE to force IDENTIFY to remap **only** those records containing a segment with ID flag set on).

Full Edit Functionality

Update-in-place edit only supports changing data in existing segments without changing the the length of a segment. In contrast, **full function** edit supports changing data, changing the length of a segment and also the addition and removal of record segments.

When segments are displayed in multi-record view, individual segments (or blocks of consecutive segments) may be replicated, deleted, inserted, moved or copied using standard SDE edit primary and line commands.

In single record view, segments may only be deleted, duplicated or inserted using primary commands DELETE, DUPLICATE and INSERT respectively.

Edit actions that affect the location of a primary segment within the file directly affects the location of the record to which it belongs. In contrast, secondary segments may be deleted, moved, copied, duplicated and inserted without affecting the location of a record within the file. For secondary segments, these edit operations may only increase or decrease the length of existing records.

Record Truncation

Beware that full edit operations involving primary segments can potentially split and join together records in the file.

If as a result of adding secondary segments to a record, the length of that record exceeds the maximum defined logical record length of the file (LRECL), then remapping the record or executing a save operation will truncate the excess data. For variable record format files, if data at the end of the truncated record represents an incomplete record segment map, then that data is also truncated.

Following an execution of IDENTIFY in which record truncation occurs, the truncated segments are removed from the display, the truncation error flag (**=TRNC>**) is set on in the primary segment of each truncated record and the following message is returned:

```
  ZZSD447I IDENTIFY process caused 1 physical records to be truncated
```

The UNDO command may be executed to undo the remap and recover the truncated segments. The user may then correct the cause of the truncation.

Following execution of SAVE, FILE or END in which record truncation occurs, the truncated segments are **not** removed from the display, the truncation error flag (**=TRNC>**) is set on in the primary segment of every truncated record and the following message is returned:

```
  ZZSD406E The physical record associated with a base segment is too long
        (greater than nnn). Truncated to length mmm.
```

Where *nnn* is the maximum allowable record length and *mmm* is the length at which the record was truncated. If truncation occurs on execution of FILE or END, the close of the SDE edit view is temporarily disabled in order to allow the user the opportunity to correct the truncation error.

Although the file has been saved to disk with truncated records, the in-storage copy of the records remains unchanged. Therefore, the user may correct and re-save the file data before closing the edit view.

Full Edit Operations

The effects of full edit operations that add, remove and/or reposition record segments are summarised below:

DELETE

Delete individual segments using command DELETE or prefix (line) command **D<*n*>** (where *n* is the number of segments to be deleted). Alternatively, delete a block of segments using prefix command **DD** on the first and last segments to be deleted.

On deleting a primary (base) segment, the record to which the primary segment belongs is deleted and its secondary segments are appended to segments belonging to the previous record.

Deleting a secondary segment simply removes that segment from the record, hence reducing the record length. All subsequent segments in the record are shifted left to fill the gap left by the deleted segment.

INSERT

Insert individual segments following the focus segment using command INSERT or prefix (line) command **I<*n*>** (where *n* is the number of segments of the focus segment type to be inserted).

Inserting a new primary segment inserts a new record in the file. If the inserted primary segment is inserted before a secondary segment (i.e. in the middle of an existing record), then the existing record is split so that all secondary segments following the inserted primary segment now belong to the new record.

Inserting a new secondary segment simply adds the segment to the focus record.

COPY and DUPLICATE

Copy individual segments using prefix (line) command **C<*n*>** (where *n* is the number of segments to be copied). Alternatively, copy a block of segments using prefix command **CC** on the first and last segments to be copied. Prefix commands **A** or **B** must be used to specify whether segments are to be copied After or Before the selected segment.

Duplicate a segment using command DUPLICATE or prefix (line) command **R<*n*>** (where *n* is the number times the segment is to be duplicated). Alternatively, duplicate a block of segments using prefix command **RR<*n*>** on the first and last segments to be duplicated.

The effect of a copy or duplicate operation on existing primary and secondary segments is the same as for insert.

MOVE

Move individual segments using prefix (line) command **M<*n*>** (where *n* is the number of segments to be moved). Alternatively, move a block of segments using prefix command **MM** on the first and last segments to be moved. Prefix commands **A** or **B** must be used to specify whether segments are to be moved After or Before the selected segment.

The effect of a move operation on existing primary and secondary segments located at the destination of the move is the same as for insert.

The effect of a move operation on primary and secondary segments immediately preceding and following the original location of the moved segments is the same as for delete.

Expressions

A selection of SDE commands support an expression as parameter input. e.g. The Edit and Browse FILTER parameter, LOCATE, WHERE and USE.

An expression consists of one or more terms (literal strings, numerical values, field values, function calls or sub-expressions) and zero or more operators. e.g.

```
term1

term1  operator1  term2  operator2  term3

operator1  term1
```

An expression is evaluated from left to right with modifications imposed by the presence of sub-expressions and the defined precedence of operators. Each term within an expression is evaluated as appropriate.

The result of an expression evaluation is either a numerical value or a character string. If numerical, then the result is considered to have a Boolean value whereby a zero (0) result has the value "false" and any non-zero result has the value "true".

Expression Terms

Individual terms within an expression are evaluated as they are encountered parsing from left to right. Following evaluation of a term, the resultant value is assigned a data type which is stored internally by the program.

A user need not know the internal data type of an evaluated term, simply that it is either character or numerical. For details on specific data types used by SDE, see SDE Data Types.

A term may be one of the following:

Literal string

A literal string stored internally as a character data type and is represented as a delimited string, character string or a hexadecimal string. A literal string may be null (length zero). e.g. C''.

When the character string is one of a pair of terms being acted upon by an operator, then the character strings may be padded with blank characters in order to match the value length of the other term involved in the evaluation process. e.g. 'AB' < 'ABC' would return a true condition since 'AB' is padded with 1 blank ('AB ') which is less than 'ABC'. Blank (EBCDIC x'40') is less than 'C' (EBCDIC x'C3').

Delimited string

A string of character text, possibly including blanks and special characters, enclosed in quotation marks (") or apostrophes ('). Alphabetic characters in a delimited string are treated as being both upper and lower case. Therefore, the following are considered equal:

```
'TERM XX' = 'term xx' = 'Term xx' = "tErm Xx"
```

Delimited strings may contain quotation marks or apostrophes that are to be interpreted as string data but are the same as the delimiting characters. To achieve this, each string data character that matches the delimiting characters must be escaped using the escape character. The escape character is the same as the string delimiter. e.g.

```
'He said "It''s my brother''s car."'
```

Character string

A string of character text in quotation marks (") or apostrophes (') and prefixed with "C" or "c". The case of alphabetic characters in a delimited string are respected. e.g. C'TERMXX' is not equal to C'Termxx'.

Escaped quotation mark or apostrophe characters that match the delimiting character are supported as for delimited strings.

Hexadecimal string

A string of even length consisting of valid hexadecimal characters (0-9, A-F, a-f) in quotation marks (") or apostrophes (') and prefixed with "X" or "x". e.g. X'E3859994F1' is equal to EBCDIC C'Term1'.

Numerical value

A numeric value is an unquoted signed integer or rational number. The decimal point in a rational number is represented by a period (.) and non-significant zeroes are also permitted. The sign of a numerical value is represented by one of the prefix operators, unary plus or unary minus. e.g. 12, +19, +0027.00, 67.353, -0.30

A numerical value is assigned a specific numeric data type based on its value. This data type may be converted during the evaluation of the expression in order to satisfy arithmetic and relational operations.

Value Range

Valid only as a term following a BT (Between) or NB (Not Between) relational operator. A value range is specified as a pair or literal string or numerical values separated by a comma (",") character and/or blank(s) and enclosed in parentheses "()". e.g. (23,55)

The pair of values define the limits of a range of values. e.g. `ELAPSED NB (2.50,2.999)` where ELAPSED is a field value containing number or elapsed seconds (ssss.ttt) will return a true value (1) if the elapsed value is less than 2.5 or greater than 2.999.

Similarly, for character field USERNAME of length 8, `USERNAME BT (c'J', c'M9999999')` will return a true value (1) for all user names beginning 'J', 'K', 'L' or 'M'.

Value List

Valid only as a term following the IN (contained in) relational operator. A value list is specified as any number or literal string or numerical values each separated by a comma (",") character and/or blank(s) and enclosed in parentheses "()". e.g. (6,14,25,26,27,28)

Valid only in value lists, the special operator **THRU** (or its synonym ":" or "><") may be used to denote a range of values. e.g. (4,32,81:87,90 THRU 93) is a list of values 4, 32 and any value in the inclusive ranges between 81 and 87, 90 and 93. The range includes values that have a fractional component (83.5, 86.666, etc.)

A value list may also be a set of literal strings (potentially padded with blanks). e.g. `USERNAME IN (c'A01', c'NBJ', c'J' >< c'M9999999')` will return a true value (1) if USERNAME is 'A01', 'NBJ' or begins 'J', 'K', 'L' or 'M'.

Field value

A field value is either a hash (#) prefixed number or an unquoted character string that matches a formatted field column reference number or field column name belonging to the named (or focus) record type.

A field name may be fully qualified, partially qualified or unqualified. An unqualified field name is simply the field's designated name (e.g. ADDR), a partially qualified field name includes the names of any nested groups in which the field is defined (e.g. MAST.ADDR or INVOICE.ADDR) and a fully qualified field name includes the record type name (e.g. LOCATION-REC.MAST.ADDR).

Note that the default qualifier separator character is "." (dot) but may be altered using the QSEPARATOR option. Furthermore, if the ABBREVIATION option is set on, then any group or field item designator within the file name may be abbreviated, starting with the first letter of the designator, to a length that identifies it as a unique group name or field item.

The data type of the field value matches that of the field being referenced so that a field which is of numerical data type will evaluate to be a numerical value and a field which is of character data type will evaluate to be a literal string with character case preserved.

A field column reference/name may be immediately followed by a subscript in a left and right parenthesis. This references the value of an individual entry within a field array. If the field array contains multiple dimensions, then a subscript must exist for each dimension of the array in order to identify a single field value. e.g. #12(5), RoomSize(6,2,4).

The nominated field is evaluated to be that field's value within the next data record assigned the identified record type. The data type assigned to the evaluated value is that of the nominated field.

If record data is unformatted, the record data belongs to a single field of field name "UnMapped" or, for unformatted record segments, "UnMappedSeg" and field reference number #1.

To reference record data in its unformatted state (whether or not a record type mapping is assigned), the buit-in function RECORD() may be used.

Condition-name VALUE clause

A condition-name VALUE clause is an unquoted character string that matches the name of a COBOL level-88 data item defined in the named (or focus) record type.

This type of expression term applies only to FileKit SDO structures that have been generated from a COBOL copy book that contains condition-name values. (See "*Enterprise COBOL for z/OS V4.2 Language Reference*" description of the "*VALUE clause - Format 2*".)

The level-88 condition-name VALUE clause associates a condition-name with an associated value or range of values that may be assigned to the conditional variable that preceeds it. i.e. It identifies potential values for the data mapped by the conditional variable field name.

Specification of a condition-name VALUE clause within an expression tests the associated field entry for the value or range of values associated with the clause. Therefore, no expression operator should follow the condition-name VALUE clause name. e.g. A COBOL copy book, processed by FileKit to format data, may contain the following:

```
03 RECORD-TYP                               .
  05 RT                          PIC  X(001).
    88 RT-ARTIST                 VALUE '1'.
    88 RT-ALBUM                  VALUE '2'.
    88 RT-TRACK                  VALUE '3'.
    88 RT-ANY                    VALUE '1' THRU '3'.
    88 RT-NONE                   VALUE '4' THRU '9'.
```

Data formatted using this copy book may then use the condition names specified by the 88 levels to test the contents of the preceeding, lower level RT field. e.g. To display only records which heve RT = '2', the following primary command may be used:

```
WHERE  RT-ALBUM
```

Function Call

A function call is a call to one of FileKit's internal routines which return a single result string or numerical value. The function name is a an unquoted character string immediately followed by zero or more, comma separated function arguments bracketed by a left and right parenthesis.

The data type of the value returned from a function call will be that defined by the function. See Function Calls for all available functions supported by the FileKit SDE environment and their return values.

Sub-expression

A sub-expression is any expression which is bracketed by a left and right parenthesis.

Keyword Value

Applicable to DB2 table display only, the keyword **NULL** may be used as the term of an expression to represent the null value. NULL is used to test a table column which has been defined with a null indicator.

Expression Operators

An operator may act on the pair of terms between which it is positioned or, in the case of unary plus, minus and logical NOT which are prefix operators, act on the single term immediately to the right of the operator.

Operators may be divided into the following 3 categories: Arithmetic, Relational and Logical.

Arithmetic Operators

Multiple numerical values, numerical field values and function calls that yield a numerical value, may be combined using arithmetic operators so resulting in another numerical value when evaluated.

The data type of an evaluated term acted upon by the arithmetic operator, must be numerical otherwise error ZZSD068E is returned.

In order to perform the arithmetic operation, the numerical data type one of the terms involved may be temporarily converted to match the data type of the other term so as to preserve numerical precision and scale. This numerical data type is then assigned to the resultant value.

Supported arithmetic operators are:

Operator	Description
`+`	Add.
`-`	Subtract.
`*`	Multiply.
`/`	Divide.
`%`	Integer Divide. (Divide and return the integer part of the result.)
`//`	Remainder. (Divide and return the remainder.)
`**`	Power. (Raise a number to a whole-number power.)
`Prefix +`	Unary plus. Equivalent to: "0+number".
`Prefix -`	Unary minus. Equivalent to: "0-number".

Relational Operators

The value returned on evaluating a relational operator is either "1" if the result is "true", or "0" if the value is "false".

If the terms acted upon by a relational operator are numerical, then the comparison will be arithmetic and signed. In this case the numerical data type of one or both of the terms may be temporarily converted before performing the evaluation.

If one of the terms has a character data type, then the comparison will be logical so that a simple character-by-character compare occurs. Except for relational operators ">>", "<<", "\>>" and "\<<", the shorter string is padded on the right with blanks.

Supported relational operators are:

Operator	Description
=	Equal.
\= ¬= != <> NOT=	Not equal, less than or greater than.
>	Greater than.
\> ¬> !> <= NOT>	Not greater than, less than or equal.
<	Less than.
\< ¬< !< >= NOT<	Not less than, greater than or equal.
>>	Beginning. (For logical compare only.)
\>> ¬>> !>> NOT>>	Not beginning. (For logical compare only.)
<<	Contains. (For logical compare only.)
\<< ¬<< !<< NOT<<	Does not contain. (For logical compare only.)
BT	Between. The term that follows BT must be a value range. The term preceding must fall within this range of values to return a true value (1).
NB	Not Between. The term that follows NB must be a value range. The term preceding must fall outside this range of values to return a true value (1).
IN	Contained in. The term that follows IN must be a value list. The term preceding must match a value in the list of values to return a true value (1).

Logical Operators

Logical operators are Boolean and so, like relational operators, return a value of "1" for "true" and "0" for "false".

The terms acted upon by a logical operator must evaluate to numerical values otherwise error ZZSD068E is returned. Any non-zero value is treated as being "1", true.

Supported logical operators are:

Operator	Description
& AND	And. Returns 1 if both terms are true.
| ¦ OR	Inclusive Or. Returns 1 if either term is true.
Prefix ¬ Prefix \ Prefix NOT	Logical Not. Negates the term so that 0 is returned if the term is true, 1 if the term is false.

Parentheses and Operator Precedence

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 precedence and the presence of parentheses.

An operator with a higher precedence will be actioned before operators with a lower precedence. This process is repeated until the entire expression is evaluated. e.g. In the following expression, where operator2 has a higher precedence 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 before that with a higher precedence. e.g. Logical Not has a higher precedence than logical And, therefore `¬1&0` evaluates to 0, but `¬(1&0)` evaluates to 1.

The order of operator precedence is as follows (highest at the top):

Prefix operators	¬ (Logical NOT) Unary + Unary -
Arithmetic Power	**
Arithmetic Multiply and Divide	* / % //
Arithmetic Add and Subtract	+ -
Relational operators	= \= > < >= <= << \<< >> \>> BT NB IN
Logical operator And	&
Logical operator Or	|

Function Calls

Functions are internal routines that return a single result string or value, and may be included as a term within an expression.

A function call is referenced by the name of the function immediately followed by zero or more, comma separated argument expressions enclosed in parentheses. Each argument expression may itself contain function calls.

If the function has zero arguments, then the parentheses may be omitted. e.g. function call LRECL() may be expressed simply as LRECL. However, specification of parentheses will distinguish the function from a non-subscripted record field with the same name. If such a field exists in the focus record type and no parentheses are specified following the function name, then the field name is used for evaluation.

Note that, when parentheses are specified, there can be no intervening blanks between the function name and the opening parenthesis.

Subscripted Field Name Conflict

Subscripted field names are used to reference array elements within a field belonging to the focus record type. e.g. RoomDimensions(5,1,2)

Because function calls are expressed in the same way as subscripted fields, a conflict may arise when attempting to reference a function call which has the same name as a field containing array elements.

If the function name matches the name of a subscripted field, then the function is not called and the field is evaluated instead. To force use of the function name, the function's internal name should be used. This is the name of the function prefixed with "BIF". e.g. The internal name of the FIND() function is BIFFIND().

ABBREV()

```
>>-- ABBREV( --- string1 -- , --- string2 ---+-----------------+-- ) --------><
                                             |                 |
                                             +-- , -- length --+
```

ABBREV() or BIFABBREV() returns "1" if *string2* is equal to the leading characters of *string1* and *string2* has a length that is greater than or equal to *length*, otherwise "0" is returned.

If not specified, *length* defaults to the length of *string2*.

Note that the data type of both *string1* and *string2* must be character and the data type of *length* must be positive integer otherwise ZZSD386E is returned.

Examples:

```
  ABBREV('input','inp')      =  1
  ABBREV(c'input','inp')     =  0
  ABBREV('input','inp',4)    =  1
  ABBREV('input','inp',2)    =  0
  ABBREV('input','int')      =  0
```

BASPOS()

```
>>-- BASPOS( --- start -+-------------------+- ) ----------------------------><
                        |                   |
                        +- , --- length ----+
```

BASPOS() or BIFBASPOS() returns a character string which is a sub-string of the current record's primary (base) segment starting at position *start* for length *length*. If the current structure is not segmented, then the current record is used as the primary segment.

BASPOS() is primarily used for mapping segmented records where a record-type definition (RTO) is assigned to secondary segments based on field values in the primary segment.

If *start* is greater than the length of the primary segment, then a blank character string is returned. If *length* is greater than the length of the primary segment, the returned character string will include data from the start of the segment that follows the primary segment.

If not specified, *length* defaults to be the length of the primary segment from position *start* to the end of the primary segment.

Note that the data type of the primary segment is character and *start* and *length* must be positive integer values otherwise ZZSD386E is returned.

Examples:

In the following, the primary segment contains a 2 character country code `c'UK'` at position 15.

```
  BASPOS(15,2)                   =  'UK'
```

CHANGED()

```
>>-- CHANGED() -------------------------------------------------------------><
```

CHANGED() or BIFCHANGED() has no arguments. It returns "1" if the focus record, segment or DB2 table row has been changed in the current edit session and "0" otherwise.

Because CHANGED() has no arguments, it may be expressed simply as CHANGED.

CHANGEERROR()

```
>>-- CHANGEERROR() ---------------------------------------------------------><
```

CHANGEERROR() or BIFCHANGEERROR() has no arguments. It returns "1" if a CHANGE command error (**==ERR>**) has been flagged on the focus record, segment or DB2 table row in the current edit session and "0" otherwise.

Because CHANGEERROR() has no arguments, it may be expressed simply as CHANGEERROR.

CHANGEOK()

```
>>-- CHANGEOK() ------------------------------------------------------------><
```

CHANGEOK() or BIFCHANGEOK() has no arguments. It returns "1" if a CHANGE command executed successfully (**==CHG>**) on the focus record, segment or DB2 table row in the current edit session and "0" otherwise.

Because CHANGEOK() has no arguments, it may be expressed simply as CHANGEOK.

DATATYPEERROR()

```
>>-- DATATYPEERROR() -------------------------------------------------------><
```

DATATYPEERROR() or BIFDATATYPEERROR() has no arguments. It returns "1" if the focus record or segment is assigned a non-default record type and one or more of its fields contains data invalid for the field's data type (e.g. invalid packed decimal data in a DECIMAL field) and "0" otherwise.

Because DATATYPEERROR() has no arguments, it may be expressed simply as DATATYPEERROR.

DUPLICATEKEY()

```
>>-- DUPLICATEKEY() ---------------------------------------------------------><
```

DUPLICATEKEY() or BIFDUPLICATEKEY() has no arguments. It applies to KSDS edit mode only and returns "1" when either of the following is true, otherwise "0" is returned:

1. Following a SAVE operation, the focus record was flagged as having a duplicate key with another record which has been changed or inserted in the current edit session (**=DUPE>**).
2. Following a SAVE operation, the focus record was flagged as having a duplicate key with another unchanged record whithin the file (**=DUPF>**).

Because DUPLICATEKEY() has no arguments, it may be expressed simply as DUPLICATEKEY.

EMPTYSLOT()

```
>>-- EMPTYSLOT() ------------------------------------------------------------><
```

EMPTYSLOT() or BIFEMPTYSLOT() has no arguments. It applies to edit of VSAM RRDS and VRDS only and returns "1" if, on load, the focus record was flagged as being an empty slot and "0" otherwise.

Because EMPTYSLOT() has no arguments, it may be expressed simply as EMPTYSLOT.

EXCLUDED()

```
>>-- EXCLUDED() -------------------------------------------------------------><
```

EXCLUDED() or BIFEXCLUDED() has no arguments. It returns "1" if the focus record, segment or DB2 table row belongs to a group of one or more excluded lines in the current edit session and "0" otherwise.

Because EXCLUDED() has no arguments, it may be expressed simply as EXCLUDED.

FIND()

```
>>-- FIND( -- find_syntax ----- ) -------------------------------------------><
```

FIND() or BIFFIND() supports a single argument *find_syntax* which is a literal or charcter string comprising a structured data edit FIND command parameter string. Note that the parameter string will be upper cased unless specified as a character string (i.e. specified in quotation marks (") or apostrophes (') and prefixed with "c" or "C".

If successful, FIND() returns the position of the found string within the first data field that contains a match for the search string, otherwise zero is returned. If no structure is applied to the data, this position will be a position in the unexpanded record.

If the search string is a numerical value which is found in a field of numerical data type, then the returned value is "1".

Examples:

```
  FIND( c" c'Needle' 21 50 " )
  FIND( " 'hit' last prefix (#2:#4) " )
  FIND( " 22  (#8,#12)" )
```

HASPOINT()

```
>>-- HASPOINT() --------------------------------------------------------------><
```

HASPOINT() or BIFHASPOINT() has no arguments. It returns "1" if the focus record, segment or DB2 table row has an active SET POINT label name and "0" otherwise.

Because HASPOINT() has no arguments, it may be expressed simply as HASPOINT.

HASPREFIXCMD()

```
>>-- HASPREFIXCMD() ----------------------------------------------------------><
```

HASPREFIXCMD() or BIFHASPREFIXCMD() has no arguments. It returns "1" if the focus record, segment or DB2 table row has a prefix command pending on the line and "0" otherwise.

Because HASPREFIXCMD() has no arguments, it may be expressed simply as HASPREFIXCMD.

IDREQUIRED()

```
>>-- IDREQUIRED() ------------------------------------------------------------><
```

IDREQUIRED() or BIFIDREQUIRED() has no arguments. It returns "1" if an ID field (identified by USE WHEN criteria) belonging to the focus record or segment, has been changed and so flagged as possibly requiring execution of the IDENTIFY command (**==ID>**) to remap the record. Otherwise "0" is returned.

Because IDREQUIRED() has no arguments, it may be expressed simply as IDREQUIRED.

INSERTED()

```
>>-- INSERTED() --------------------------------------------------------------><
```

INSERTED() or BIFINSERTED() has no arguments. It returns "1" if the focus record, segment or DB2 table row has been inserted in the current edit session and "0" otherwise.

Because INSERTED() has no arguments, it may be expressed simply as INSERTED.

KEYCHANGED()

```
>>-- KEYCHANGED() ------------------------------------------------------------><
```

KEYCHANGED() or BIFKEYCHANGED() has no arguments. It applies to KSDS edit mode only and returns "1" if the focus record is flagged as having had a change to its key during the current edit session and "0" otherwise.

Because KEYCHANGED() has no arguments, it may be expressed simply as KEYCHANGED.

LASTPOS()

```
>>-- LASTPOS( --- string1 -- , --- string2 ---+-----------------+-- ) -------><
                                              |                 |
                                              +-- , -- start ---+
```

LASTPOS() or BIFLASTPOS() returns the position of the last occurrence of *string1* in *string2* starting the backwards search from position *start* in *string2*. A search is successful only if *string1* exists entirely between position 1 of *string2* and the *start* position. If *string1* is not found, "0" is returned.

If not specified or a value greater than the length of *string2*, *start* defaults to be the last position (i.e. length) of *string2*.

Note that the data type of both *string1* and *string2* must be character and the data type of *start* must be a positive integer otherwise ZZSD386E is returned.

Examples:

In the following, field name SourceText exists in the focus record type and contains character data `c'To be or not to be'`.

```
LASTPOS('be',SourceText)        =  0
LASTPOS(c'be',SourceText)       =  17
LASTPOS(c'be',SourceText,17)    =  4
```

LEFT()

```
>>-- LEFT( --- string -- , --- length ---+---------------+-- ) --------------><
                                         |               |
                                         +-- , -- pad ---+
```

LEFT() or BIFLEFT() returns a character string equal to the left-most *length* characters of *string*. If *length* is greater than the length of *string*, the returned character string is padded on the right with the *pad* character.

If not specified, *pad* defaults to be character blank.

Note that the data type of *string1* must be character, *length* must be a positive integer and *pad* must be a character string of length 1 otherwise ZZSD386E is returned.

Examples:

```
LEFT('Hello World',5)        =  'HELLO'
LEFT(c'Hello World',15)      =  'Hello World    '
LEFT(c'Hello World',13,'#')  =  'Hello World##'
```

LENGTH()

```
>>-- LENGTH( --- string --- ) -----------------------------------------------><
```

LENGTH() or BIFLENGTH() returns the length of *string*.

Note that the data type of *string* must be character otherwise ZZSD386E is returned.

Examples:

In the following, field reference #13 contains character data `c'James Joyce'`.

```
LENGTH('Hello')    =  5
LENGTH(#13)        =  11
```

LENGTHERROR()

```
>>-- LENGTHERROR() -----------------------------------------------------------><
```

LENGTHERROR() or BIFLENGTHERROR() has no arguments. It returns "1" if the focus record or segment is flagged as having been assigned a record-type whereby the length of the record/segment falls outside the limits of possible lengths for the record type (**=LGTH>**). Otherwise "0" is returned.

Because LENGTHERROR() has no arguments, it may be expressed simply as LENGTHERROR.

LOWER()

```
>>-- LOWER( --- string --- ) -------------------------------------------------><
```

LOWER() or BIFLOWER() returns the *string* with all upper case alpha characters lower cased and all other characters unchanged.

Note that the data type of *string* must be character otherwise ZZSD386E is returned.

Examples:

In the following, field array element Email(2) contains character data `c'Mark123@CBL.COM'`.

```
LOWER('HELLO')    =  'hello'
LOWER(email(2))   =  'mark123@cbl.com'
```

LRECL()

```
>>-- LRECL() -----------------------------------------------------------------><
```

LRECL() or BIFLRECL() has no arguments. It returns the length of the focus record, segment or DB2 table row.

Because LRECL() has no arguments, it may be expressed simply as LRECL.

NOEOL()

```
>>-- NOEOL() -----------------------------------------------------------------><
```

NOEOL() or BIFNOEOL() has no arguments and applies to HFS files containing record EOL characters. This function returns "1" if the focus record or segment is flagged as having been split due to no EOL delimiter being found within the file's declared LRECL length (**==EOL>**). i.e. record length is graeter than LRECL. Otherwise the function returns "0".

Because NOEOL() has no arguments, it may be expressed simply as NOEOL.

POS()

```
>>-- POS( --- string1 -- , --- string2 ---+-----------------+-- ) ----------><
                                          |                 |
                                          +-- , -- start ---+
```

POS() or BIFPOS() returns the position of the first occurrence of *string1* in *string2* starting the search from position *start* in *string2*. A search is successful only if *string1* exists entirely between the *start* position and the last position of *string2*. If *string1* is not found, "0" is returned.

If not specified, *start* defaults to be the first position of *string2*.

Note that the data type of both *string1* and *string2* must be character and the data type of *start* must be a positive integer otherwise ZZSD386E is returned.

Examples:

In the following, field name SourceText exists in the focus record type and contains character data `c'To be or not to be'`.

```
POS('be',SourceText)       =  0
POS(c'be',SourceText)      =  4
POS(c'be',SourceText,5)    =  17
POS(c' ',SourceText,10)    =  13
```

RECORD()

```
>>-- RECORD( ---+---------------------+-- ) --------------------------------><
                |                     |
                +- relative_record# --+
```

RECORD() or BIFRECORD() returns a character string which is the unformatted contents of the record identified by *relative_record#*, a record number relative to the focus record.

If not specified, *relative_record#* defaults to be 0 (zero) identifying the focus record. A negative *relative_record#* value identifies a record occurring before the focus record, a positive value identifies a record occurring after the focus record. If the specified record does not exist the null string is returned.

Note that *relative_record#* must be zero, positive or negative integer value otherwise ZZSD386E is returned.

If RECORD() is specified with no arguments, it may be expressed simply as RECORD.

Examples:

```
RECORD()     * Contents of the focus record.
RECORD(-1)   * Contents of the record immediately before the focus record.
RECORD(2)    * Contents of the 2nd record following the focus record.
```

RIGHT()

```
>>-- RIGHT( --- string -- , --- length ---+---------------+-- ) -------------><
                                          |               |
                                          +-- , -- pad ---+
```

RIGHT() or BIFRIGHT() returns a character string equal to the right-most *length* characters of *string*. If *length* is greater than the length of *string*, the returned character string is padded on the left with the *pad* character.

If not specified, *pad* defaults to be character blank.

Note that the data type of *string1* must be character, *length* must be a positive integer and *pad* must be a character string of length 1 otherwise ZZSD386E is returned.

Examples:

In the following, field name XLen exists in the focus record type and contains binary integer value `8`.

```
RIGHT('Hello World',5)       =  'WORLD'
RIGHT(c'Hello World',15)     =  '    Hello World'
RIGHT(c'Hello World',13,'#') =  '##Hello World'
RIGHT(c'Hello World',XLen)   =  'lo World'
```

SAVEREQUIRED()

```
>>-- SAVEREQUIRED() ---------------------------------------------------------><
```

SAVEREQUIRED() or BIFSAVEREQUIRED() has no arguments. It returns "1" if the focus record, segment or DB2 table row has been changed or inserted and has not yet been saved, and "0" otherwise.

Because SAVEREQUIRED() has no arguments, it may be expressed simply as SAVEREQUIRED.

SEGPOS()

```
>>-- SEGPOS( --- start -+-------------------+- ) ----------------------------><
                        |                   |
                        +- , --- length ----+
```

SEGPOS() or BIFSEGPOS() returns a character string which is a sub-string of the current or previous segment within the current record starting at position *start* for length *length*.

SEGPOS() is primarily used for mapping segmented records where a record-type definition (RTO) is assigned to segments based on field values in the current or previous segment.

If *start* is a negative value, then it refers to a position that many characters from the end of in the previous segment.

If *start* is greater than the length of the record, then a blank character string is returned. If *length* is greater than the length of the record, the returned character string is padded on the left with the blank character.

If not specified, *length* defaults to be the length of data from position *start* to the end of the segment in which *start* is located.

Note that the data type of the segment is character, *start* must be a positive or negative integer value and *length* must be a positive integer value otherwise ZZSD386E is returned.

Examples:

In the following, the segment previous to the segment currently being mapped contains the 11 character record-type name of the next segment starting 16 characters before the end of that segment (c'REC-CUST-01'). Similarly, the current segment data contains this name at position 31 of the segment.

```
  SEGPOS(-16,11)                 =  'REC-CUST-01'
  SEGPOS(31,8)                   =  'REC-CUST'
```

SQLERROR()

```
>>-- SQLERROR() -------------------------------------------------------------><
```

SQLERROR() or BIFSQLERROR() has no arguments and applies to DB2 table rows only. If the formatted DB2 table row experienced an SQL error on SAVE, this function returns the SQL error code, otherwise zero is returned.

Because SQLERROR() has no arguments, it may be expressed simply as SQLERROR.

STRIP()

```
>>-- STRIP( --- string -- , --- option ---+----------------+-- ) ------------><
                                          |                |
                                          +-- , -- char ---+
```

STRIP() or BIFSTRIP() returns character string *string* with leading and/or trailing *char* characters removed.

The *option* argument may be specified as character string "L", "T" or "B" (upper or lower case) to indicate respectively that Leading, Trailing or Both leading and trailing characters are to be stripped. If not specified, *option* defaults to "B".

The *char* argument nominates the character to be stripped and, if not specified, defaults to be character blank. Each consecutive occurrence of *char* at the start and/or end of *string* will be stripped until a character other than *char* is encountered at which point the strip algorithm stops.

Note that the data type of *string1*, *option* and *char* must be character with *option* and *char* being of length 1, otherwise ZZSD386E is returned.

Examples:

In the following, field reference #2 in the focus record type contains character data `c' Time to go. '`.

```
  STRIP('  Hello   World ')            =  'HELLO   WORLD'
  STRIP(c'##CPU Seconds###','L','#')   =  'CPU Seconds###'
  STRIP(c'##CPU Seconds###','t','#')   =  '##CPU Seconds'
  STRIP(#2,'B')                        =  'Time  to  go.'
```

SUBSTR()

```
>>-- SUBSTR( -- string -- , -- start -+--------------------------------+- ) -><
                                      |                                |
                                      +- , --+----------+-+------------+
                                             |          | |            |
                                             +- length -+ +- , -- pad -+
```

SUBSTR() or BIFSUBSTR() returns a character string which is a sub-string of *string* starting at position *start* within *string* for length *length*.

If *start* is greater than the length of *string*, then a blank character string is returned. If *length* is greater than the length of *string*, the returned character string is padded on the left with the *pad* character.

If not specified, *length* defaults to be the length of *string* from position *start* to the end of *string* and *pad* defaults to be character blank.

Note that the data type of *string* must be character, *start* and *length* must be a positive integer and *pad* must be a character string of length 1 otherwise ZZSD386E is returned.

Examples:

In the following, field names XText and XLen exist in the focus record type and contain character data `c'The quality of mercy is not strained'` and binary integer value `16` respectively.

```
  SUBSTR(XText,29)             =  'strained'
  SUBSTR(XText,5,XLen)         =  'quality of mercy'
  SUBSTR(XText,29,15,c'x')     =  'strainedxxxxxxx'
  SUBSTR(XText,29,15,'x')      =  'strainedXXXXXXX'
```

TRANSLATE()

```
>>-- TRANSLATE( string -+---------------------------------------------+- ) --><
                        |                                             |
                        +- , -+--------+-+----------------------------+
                              |        | |                            |
                              +- tabo -+ +- , -+--------+-+-----------+
                                               |        | |           |
                                               +- tabi -+ +-, -- pad -+
```

TRANSLATE() or BIFTRANSLATE() returns the character string *string* with characters at offsets in the input translate table *tabi* translated to characters at equivalent offsets in the output translate table *tabo*. Characters in *string* that are not included in *tabi* are not translated.

If both *tabo* and *tabi* are specified then:

- If *tabo* has a shorter length than *tabi*, then *tabo* is padded on the right to the length of *tabi* using the *pad* character.
- If *tabo* has a longer length than *tabi*, then *tabo* is truncated to the length of *tabi*.

If *tabo* is specified but *tabi* is not, *tabi* defaults to be all 255 single byte characters in the range X'00' to X'FF'.

If *tabi* is specified but *tabo* is not, *tabo* defaults to be a string equal to the length of *tabi* containing only *pad* characters.

If neither *tabo* nor *tabi* are specified, then *string* is simply upper cased as for the UPPER() function call.

If not specified, *pad* defaults to be character blank.

Note that the data type of *string*, *tabo* and *tabi* must be character, and *pad* must be a character string of length 1 otherwise ZZSD386E is returned.

Usage Note:

It is recommended that, when expressed as a literal string, translate tables *tabi* or *tabo* be prefixed with "C" or "c" to avoid unintentional upper casing of lower case translate table alpha characters.

Examples:

In the following, field reference #6 in the focus record type contains character data `c'Ill met by moonlight, proud Titania'`.

```
  TRANSLATE(#6)                   =  'ILL MET BY MOONLIGHT, PROUD TITANIA'
  TRANSLATE(#6,c'SsXx',c'TtOo')   =  'Ill met by mxxnlighs, prxud Sisania'
  TRANSLATE(#6,,c'LlAa',c'#')     =  'I## met by moon#ight, proud Tit#ni#'
  TRANSLATE(c'321',c'abc',c'123') =  'cba'
```

TRUNCATED()

```
>>-- TRUNCATED() ------------------------------------------------------------><
```

TRUNCATED() or BIFTRUNCATED() has no arguments and applies to edit of segmented records only. It returns "1" if the focus segment is a primary (base) segment which has been flagged as belonging to a record that has been truncated by an IDENTIFY or a SAVE, FILE or END operation (**=TRNC>**). Otherwise "0" is returned.

Because TRUNCATED() has no arguments, it may be expressed simply as TRUNCATED.

UPPER()

```
>>-- UPPER( --- string --- ) ------------------------------------------------><
```

UPPER() or BIFUPPER() returns the *string* with all lower case alpha characters upper cased and all other characters unchanged.

Note that the data type of *string* must be character otherwise ZZSD386E is returned.

Examples:

In the following, field reference #2 contains character data `c'Mark Williams'`.

```
  UPPER('hello')    =  'HELLO'
  UPPER(#2)         =  'MARK WILLIAMS'
```

VALUEERROR()

```
>>-- VALUEERROR() -----------------------------------------------------------><
```

VALUEERROR() or BIFVALUEERROR() has no arguments. It returns "1" if the formatted focus record, segment or DB2 table is flagged as including at least one field value which, while valid for the field's data type, is outside the range of valid values defined for the field (**=ERRV>**). Otherwise "0" is returned.

The VALUEERROR flag may be set on when any of the following conditions are true:

1. A field used to define the length of a variable length field has a value greater than the maximum length defined for the variable length field.
2. A field used to define the size of a variable array is outside the range of values specified for the array size.
3. An enum field contains a value not listed as a defined enum value.

Because VALUEERROR() has no arguments, it may be expressed simply as VALUEERROR.

WORD()

```
>>-- WORD( --- string -- , -- #word --- ) -----------------------------------><
```

WORD() or BIFWORD() returns a character string which is equal to word number *#word* in *string*. A word is considered to be a blank delimited token in a character string.

If *#word* is greater than the number of words in *string*, then a null string is returned.

Note that the data type of *string* must be character and *#word* must be a positive integer otherwise ZZSD386E is returned.

Examples:

In the following, field reference #3 in the focus record type contains character data `c'But, soft! what light through yonder window breaks?'`.

```
WORD('  Hello   World ',2)     =  'WORLD'
WORD(#3,2)                     =  'soft!'
WORD(#3,6)                     =  'yonder'
WORD(#3,9)                     =  ''
```

WORDS()

```
>>-- WORDS( --- string --- ) ------------------------------------------------><
```

WORDS() or BIFWORDS() returns the number of words in *string*. A word is considered to be a blank delimited token in a character string.

Note that the data type of *string* must be character otherwise ZZSD386E is returned.

Examples:

In the following, field name Quote_01 exists in the focus record type and contains character data `c'So foul and fair a day I have not seen'`.

```
WORDS(Quote_01)  =  10
WORDS('  ')      =  0
```

XRANGE()

```
>>-- XRANGE( --+-- start --+--+------------+---------------------------------><
               |           |  |            |
               +-----------+  +-, -- end --+
```

XRANGE() or BIFXRANGE() returns a string of single byte character representations between and including *start* and *end*.

If not specified, *start* defaults to X'00' and *end* defaults to X'FF'. If *start* is greater *end*, then the returned string will contain characters from *end* to X'FF' and X'00' to *start*.

Note that the data type of *start* and *end* must be character length 1 otherwise ZZSD386E is returned.

Examples:

```
XRANGE(c'A',c'J')     =  'ABCDEFGHIJ'
XRANGE(X'C1',X'C6')   =  X'C1C2C3C4C5C6'
XRANGE(X'FC',X'03')   =  X'FCFDFEFF010203'
XRANGE(,X'08')        =  X'000102030405060708'
XRANGE(X'FC')         =  X'FCFDFEFF'
```

REXX Macros

As in the CBLe text edit environment, user macros may be written to perform functions within SDE sessions using the REXX procedure language.

The name associated with the Structured Data Environment is CBLSDATA and should be specified on the REXX instruction ADDRESS if commands are to be directed to the SDE environment. However, if a macro is executed from within an SDE window, CBLSDATA is automatically set as the default environment.

The current environment may be identified using the REXX built-in function ADDRESS().

Macro Path

An SDE REXX macro is executed by executing the MACRO command followed by the full fileid of the macro.

Alternatively, if the macro exists in a library within the macro path, simply specify the macro name. Libraries in the macro path are searched in order until a file name that matches the macro name is found.

The macro path used by SDE is the same as that defined for the CBLe text edit environment. i.e. The macro path is a list of directories assigned to the **Edit.MacroPath** variable set via the System or User INI file, or via the CBLe SET MACROPATH command.

For MVS systems, the macro path usually consists of three PDS/PDSE libraries as follow:

1. A user specific macro library.
2. A site-wide macro library common to all users.
3. An installation macro library common to all users and containing useful macros written and owned by CBL and distributed to all SELCOPY Product Suite customers.

   Users should not update or modify macros in this library as they may be subject to change at the discretion of CBL. A detailed description on the use of each of an individual macro is documented within the macro executable file itself. (See CBLe REXX Macros for a list of CBL supplied macros.)

SDE Profile (SDEPROF) Macro

When an SDE window view is opened, the macro path libraries are searched for the first occurrence of the SDE profile macro, **SDEPROF**.

The SDEPROF macro may be used to define the user's SDE environment. Any System or User INI file options that correspond to SDE SET command options, may be overridden for an SDE session by including the SET command in the SDEPROF macro.

A default SDEPROF macro is distributed as part of the SELCOPY Product Suite package and exists within the installation macro library.

SDE Menus and Popup Windows

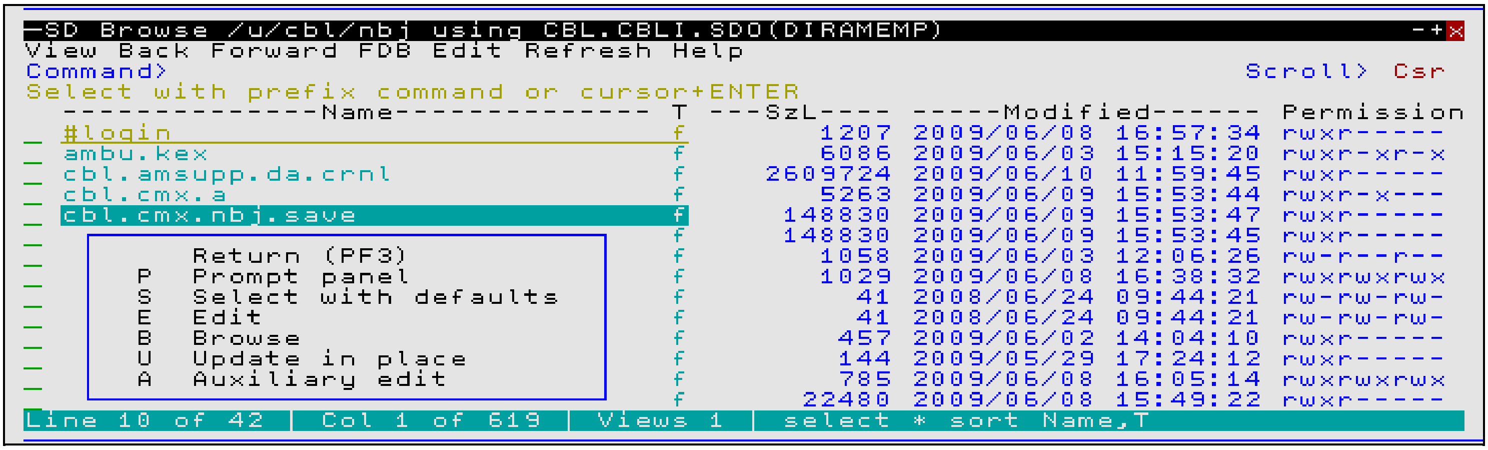
To assist the user with general SDE editing, some helpful menus and dialog panels exist in addition to the startup CBLe SDE Edit Dialog Window

Library/Directory Member Selection

The SD Edit/Browse Library/Directory Member Selection list window is opened when a PDS(E) library DSN with no member name or HFS directory path with no HFS file name, is specified on the SDE EDIT or BROWSE command.

Select the required entry (position the cursor and hit <Enter>) for Edit/Browse, as specified on the invoking command, or enter select one of the prefix commands.

The window remains open after a selection has been made in order to allow selection of further entries.



```
-SD Browse /u/cbl/nbj using CBL.CBLI.SDO(DIRAMEMP)                          -+x
View Back Forward FDB Edit Refresh Help
Command>                                                          Scroll> Csr
Select with prefix command or cursor+ENTER
   --------------Name--------------  T ---SzL---- -----Modified------ Permission
_  #login                            f      1207 2009/06/08 16:57:34 rwxr-----
_  ambu.kex                          f      6086 2009/06/03 15:15:20 rwxr-xr-x
_  cbl.amsupp.da.crnl                f   2609724 2009/06/10 11:59:45 rwxr-----
_  cbl.cmx.a                         f      5263 2009/06/09 15:53:44 rwxr-x---
_  cbl.cmx.nbj.save                  f    148830 2009/06/09 15:53:47 rwxr-----
_                                    f    148830 2009/06/09 15:53:45 rwxr-----
_          Return (PF3)              f      1058 2009/06/03 12:06:26 rw-r--r--
_       P  Prompt panel              f      1029 2009/06/08 16:38:32 rwxrwxrwx
_       S  Select with defaults      f        41 2008/06/24 09:44:21 rw-rw-rw-
_       E  Edit                      f        41 2008/06/24 09:44:21 rw-rw-rw-
_       B  Browse                    f       457 2009/06/02 14:04:10 rwxr-----
_       U  Update in place           f       144 2009/05/29 17:24:12 rwxr-----
_       A  Auxiliary edit            f       785 2009/06/08 16:05:14 rwxrwxrwx
_                                    f     22480 2009/06/08 15:49:22 rwxr-----
Line 10 of 42 | Col 1 of 619 | Views 1 | select * sort Name,T
```

Figure 10. SD Edit/Browse Library/Directory Member Selection List Window.

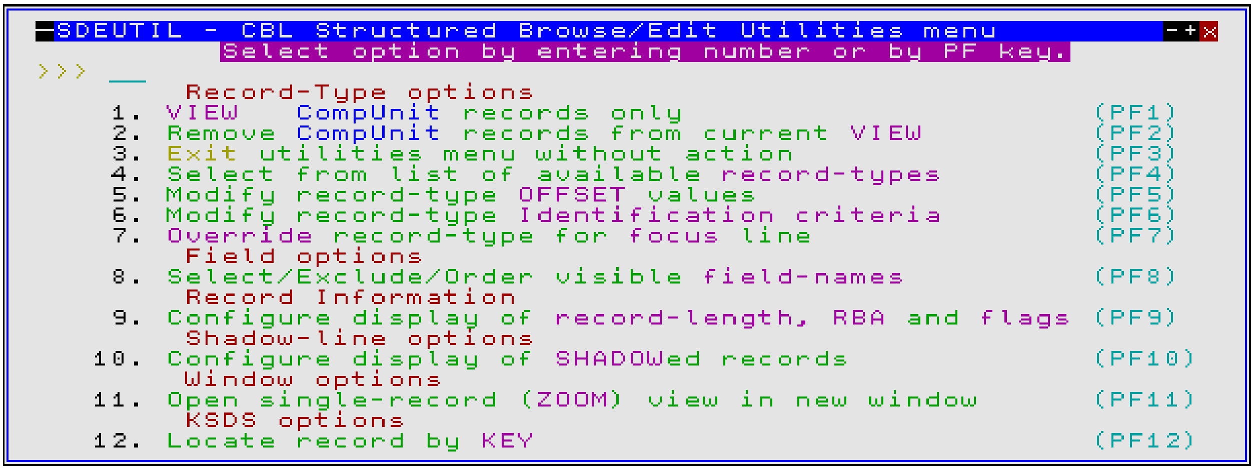
Prefix Line Commands

The following prefix line commands are available:

Command	Description
(blank)	Prefix line command S. (Hit the <Enter> key with the cursor on the line containing the required entry).
A	Edit the entry using Auxiliary Edit.
B	Browse the entry.
E	Edit the entry using Full Edit.
P	Open the CBLe SDE Edit Dialog Window.
S	Select the entry for Edit or Browse as specified by the invoking command.
U	Edit the entry using Update-in-place Edit.
/	Open a drop down menu containing valid prefix command functions for the list window entry. Position the cursor on the required function and hit <Enter> to action the command. Assigned to F16 by default.
>	Open a new window containing a zoomed vertical display of the entry's fields. Particularly useful for list windows that have a large number of displayed columns. Assigned to PF17 (Shift-F5) by default.

SDE Browse/Edit Utilities Menu

The SDE Browse/Edit Utilities Menu may be opened from any SDE file edit or browse window view by executing the SDEUTIL macro which is assigned to <F16> by default.



```
 SDEUTIL - CBL Structured Browse/Edit Utilities menu                       -+x
                Select option by entering number or by PF key.
 >>> ___
          Record-Type options
      1. VIEW    CompUnit records only                                 (PF1)
      2. Remove CompUnit records from current VIEW                     (PF2)
      3. Exit utilities menu without action                            (PF3)
      4. Select from list of available record-types                    (PF4)
      5. Modify record-type OFFSET values                              (PF5)
      6. Modify record-type Identification criteria                    (PF6)
      7. Override record-type for focus line                           (PF7)
          Field options
      8. Select/Exclude/Order visible field-names                      (PF8)
          Record Information
      9. Configure display of record-length, RBA and flags             (PF9)
          Shadow-line options
     10. Configure display of SHADOWed records                         (PF10)
          Window options
     11. Open single-record (ZOOM) view in new window                  (PF11)
          KSDS options
     12. Locate record by KEY                                          (PF12)
```

Figure 11. SDE Browse/Edit Utilities Window.

This menu provides a user friendly method of executing commonly used SDE commands that alter the display of record data within the SDE window view.

Options are selected by entering the required option number at the prompt or by pressing the appropriate PFKey.

Note that items in the menu are sensitive to the prevailing default record type.

Record-Type Options

`VIEW` *`record_type`* `records only`

Executes the following SDE VIEW command to make visible only records that are associated with the specified record type (RTO).

```
VIEW  record_type
```

`Add/Remove` *`record_type`* `records to/from current VIEW`

If the focus line is a shadow line representing a suppressed record group, then this option is "Add" records to the display, otherwise this option is "Remove" records from the display.

The following SDE VIEW command is executed to add to (+) or remove from (-) the display of visible records all records that are associated with the specified record type (RTO).

```
VIEW  (+/-) record_type
```

`Select from list of available record-types`

Generates a temporary CMX command centre file containing a number of SDE VIEW commands that enables the user to easily add or remove records assigned individual record types from the current display of visible records.

The temporary CMX file is displayed in a CBLe text edit window which allows the user to execute the VIEW commands simply by placing the cursor on the required command (line beginning with "<") and pressing <F16<.

The CMX file contains the following VIEW commands:

`sd view + *`

View all records of any record type.

`sd view - *`

Suppress all records of all record types. Following execution of this command, execute one or more of the "VIEW + *record_type*" commands that follow to display only records of the selected record types.

`sd view Record`

View only records not assigned a record type. (i.e. NOT SELECTED records.)

`sd view + _record_type`

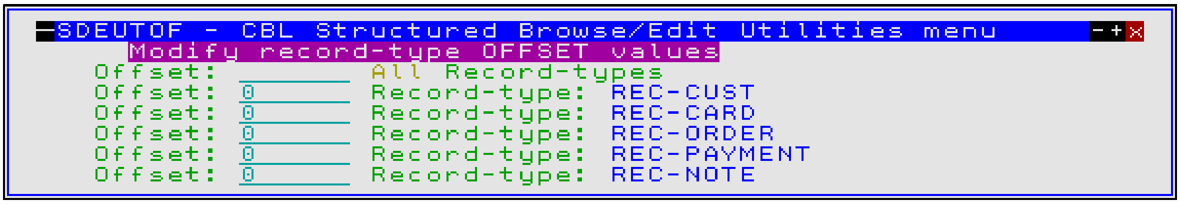
Add all records of record type *record_type* to the existing view of records. One of these VIEW commands is generated for every record type (RTO) defined in the structure (SDO).

Execution of any of these VIEW commands will immediately update the display of the records in the current SDE window view. The text edit window containing the CMX file remains open to allow execution of other VIEW commands, until closed (using <PF3<).

`Modify record-type OFFSET values`

Opens the SDEUTOF sub-window which prompts the user to enter a positive or negative numeric offset against any (or all) of the record types (RTOs) defined in the current SDO.

Where an offset value is assigned to a record type, record mapping starts at that offset into (or before) the record data. (See the USEOFFSET SET/QUERY/EXTRACT option for details.)



```
 SDEUTOF - CBL Structured Browse/Edit Utilities menu                -+x
        Modify record-type OFFSET values
      Offset: ______   All Record-types
      Offset: 0_____   Record-type: REC-CUST
      Offset: 0_____   Record-type: REC-CARD
      Offset: 0_____   Record-type: REC-ORDER
      Offset: 0_____   Record-type: REC-PAYMENT
      Offset: 0_____   Record-type: REC-NOTE
```

Figure 12. SDE Browse/Edit Modify Offsets Window.

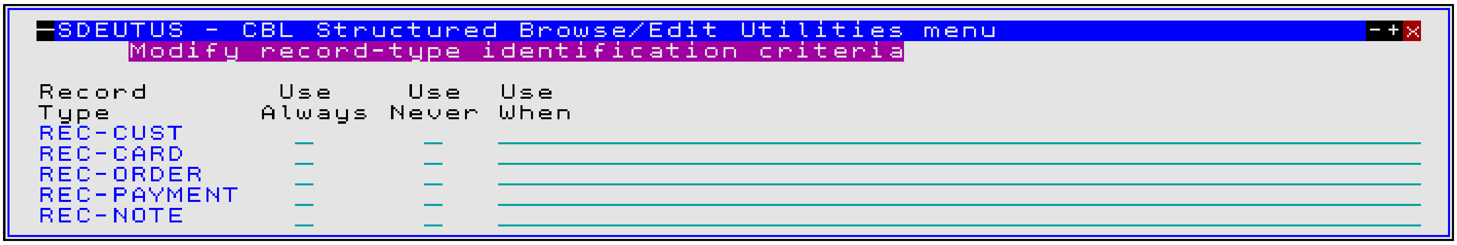
<Enter> updates the affected RTO definitions in storage and actions the offset changes immediately. Record type assignment processing is performed for records within the file following a change to any record type's offset value.

<PF3> closes the SDEUTOF window and, if offset changes have occurred, the SDEUTSS sub-window is displayed prompting the user to save the changed structure, overwriting the structure data file on disk, before returning to the utilities menu. If not saved, the user is given another opportunity to save changes to updated structures when the FileKit session is ended normally.

`Modify record-type Identification criteria`

Opens the SDEUTUS sub-window which prompts the user to enter record type (RTO) selection criteria against any or all record types defined in the SDO.

The Use Always option may set for one record type only. If specified for multiple record types, the first selected will be used. See command USE for descriptions on use of the WHEN *expression*, ALWAYS and NEVER parameters.



```
 SDEUTUS - CBL Structured Browse/Edit Utilities menu                -+x
        Modify record-type identification criteria

 Record          Use     Use    Use
 Type           Always   Never  When
 REC-CUST         _       _     __________________________________________
 REC-CARD         _       _     __________________________________________
 REC-ORDER        _       _     __________________________________________
 REC-PAYMENT      _       _     __________________________________________
 REC-NOTE         _       _     __________________________________________
```

Figure 13. SDE Browse/Edit Record Type Identification Window.

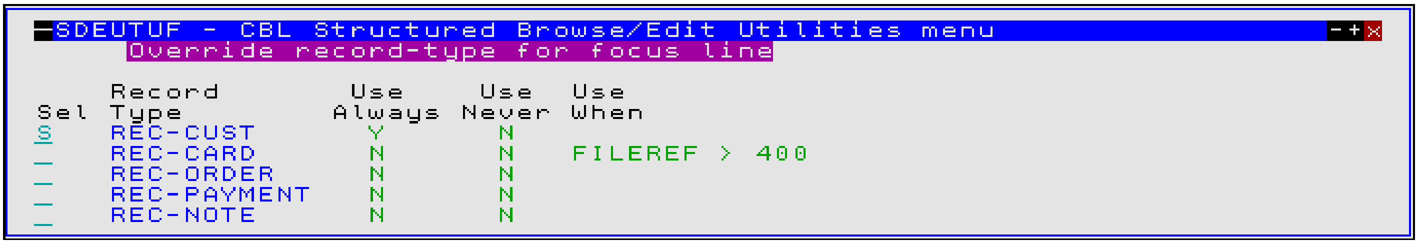
<Enter> updates the affected RTO definitions in storage and immediately actions record type assignment processing for the records within the SDE window view.

<PF3> closes the SDEUTUS window and, if offset changes have occurred, the SDEUTSS sub-window is displayed prompting the user to save the changed structure, overwriting the structure data file on disk, before returning to the utilities menu. If not saved, the user is given another opportunity to save changes to updated structures when the FileKit session is ended normally.

`Override record-type for focus line`

Opens the SDEUTUF sub-window which prompts the user to select the record type (RTO) to be assigned to the record occupying the focus line. The status of ALWAYS and NEVER flags and any WHEN expressions are displayed in this window for information purposes only.

Only one record type may be selected. If multiple record types are selected, then the last record type in the list to be selected is used. See command USE for descriptions on use of the FOR FOCUS parameter.



```
 SDEUTUF - CBL Structured Browse/Edit Utilities menu                -+x
      Override record-type for focus line

     Record          Use      Use    Use
 Sel Type           Always   Never   When
 S   REC-CUST          Y        N
 _   REC-CARD          N        N     FILEREF > 400
 _   REC-ORDER         N        N
 _   REC-PAYMENT       N        N
 _   REC-NOTE          N        N
```

Figure 14. SDE Browse/Edit Focus Line Record Type Identification.

<Enter> applies the record type to the record in the focus line immediatley. <PF3> closes the SDEUTUS window and returns to the utilities menu.

Field Options

`Select/Exclude/Order visible field-names`

Executes SDESEL to generate a temporary CMX command centre file containing an SDE SELECT command that, by default, selects all columns for display in ascending order of field reference number.

The temporary CMX file is displayed in a CBLe text edit window which allows the user to employ standard text editing techniques to re-order and/or exclude the fields from the SELECT command and so select and re-order the display of table row fields.

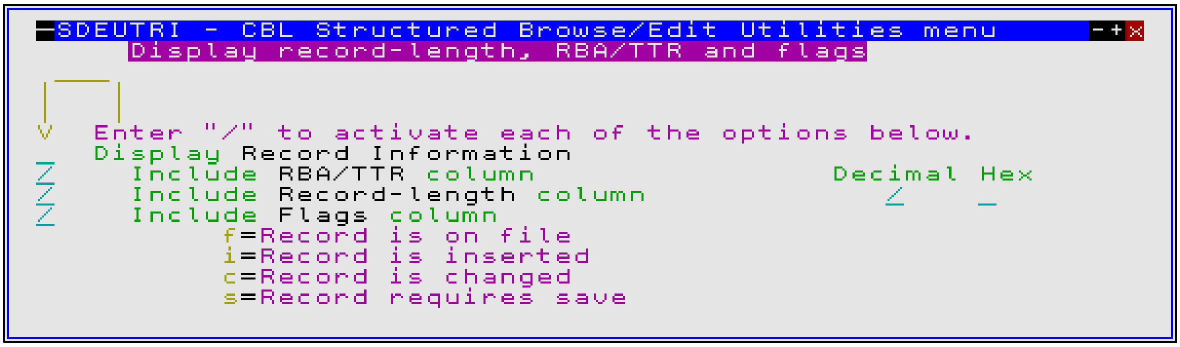
The SELECT operation may be executed by placing the cursor on the first line of the SELECT command (beginning with "<") and pressing <F16<.

Record Information

`Configure display of record-length, RBA and flags`

Opens the SDEUTRI sub-window which prompts the user to select the record information to be displayed at the start of all rows in the current SDE window view.

The "Display Record Information" option must be selected before any of the selected record information fields will be displayed.



```
SDEUTRI - CBL Structured Browse/Edit Utilities menu                -+x
      Display record-length, RBA/TTR and flags

V   Enter "/" to activate each of the options below.
    Display Record Information
/     Include RBA/TTR column                   Decimal Hex
/     Include Record-length column                /     _
/     Include Flags column
           f=Record is on file
           i=Record is inserted
           c=Record is changed
           s=Record requires save
```

Figure 15. SDE Browse/Edit Record Info Window.

<Enter> actions the information display in the SDE edit view but keeps the SDEUTRI window open. <PF3> closes the window and returns focus to the Utilities Menu.

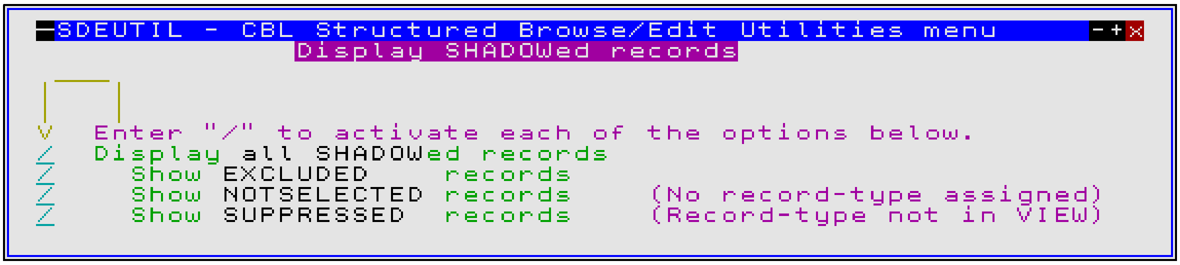
See the RECINFO SET/QUERY/EXTRACT option for details of the individual record information fields.

Shadow-line Options

`Configure display of SHADOWed records`

Opens the SDEUTSH sub-window which has option fields indicating whether shadow lines are displayed for record groups that are EXCLUDED, NOTSELECTED or SUPPRESSED.

The "Display all SHADOWed records" option must be selected before any of the selected shadow lines will be displayed.



```
SDEUTIL - CBL Structured Browse/Edit Utilities menu                -+x
               Display SHADOWed records

V   Enter "/" to activate each of the options below.
/   Display all SHADOWed records
/     Show EXCLUDED    records
/     Show NOTSELECTED records    (No record-type assigned)
/     Show SUPPRESSED  records    (Record-type not in VIEW)
```

Figure 16. SDE Browse/Edit SHADOW Lines Window.

<Enter> actions the selection for the current SDE window view but keeps the SDEUTSH window open. <PF3> closes the window and returns focus to the Utilities Menu.

See the SHADOW SET/QUERY/EXTRACT option for details of shadow lines.

Window Options

`Open single-record (ZOOM) view in new window`

Executes SDEZOOMW to open another SDE window view containing a single record view of the record occupying the focus line. If the current SDE window view is already in SINGLE format then data in the new view will be a multi-record view of the prevailing format (TABLE, CHAR or HEX.)

See the SDE command ZOOM.

Locate Options

`Locate record by KEY/RBA/RECORD Number`

Locate record by KEY is displayed for KSDS data sets and opens the SDEUTKY sub-window which has a single input field prompting the user for a complete or partial KSDS key identifying the new current record.
The first record with a key which is equal to or greater than this value will become the current record.

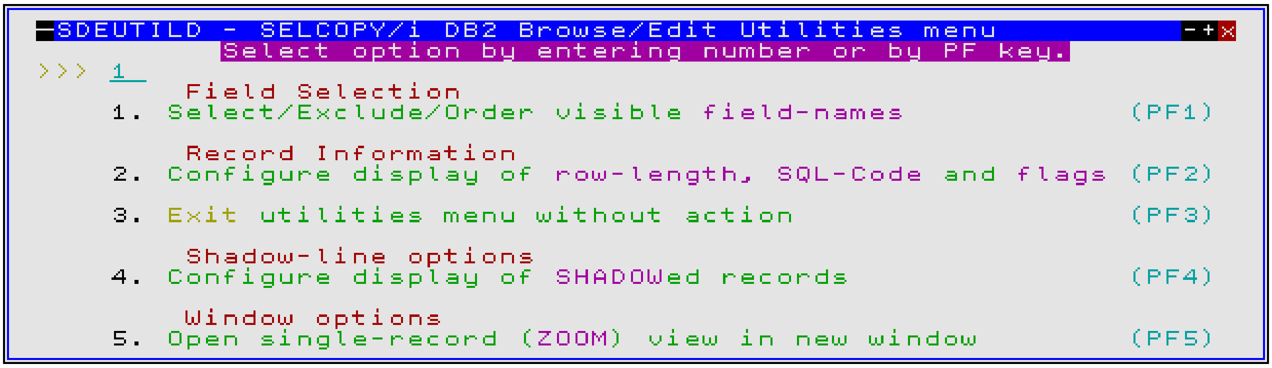
Locate record by RBA is displayed for ESDS data sets and opens the SDEUTRB sub-window which has a single input field prompting the user for the relative byte address identifying the new current record.
The first record with an RBA which is equal to or greater than this value will become the current record.

Locate record by RECORD Number is displayed for RRDS and VRDS data sets and opens the SDEUTRR sub-window which has a single input field prompting the user for the record number identifying the new current record.

See the SDE command LOCATE for details on locating records by Key, RBA and Record number.

DB2 Browse/Edit Utilities Menu

The DB2 Browse/Edit Utilities Menu may be opened from any SDE DB2 table edit or browse view by executing the SDEUTILD macro which is assigned to <F16> by default.



```
SDEUTILD - SELCOPY/i DB2 Browse/Edit Utilities menu                   -+x
               Select option by entering number or by PF key.
 >>> 1
              Field Selection
         1.  Select/Exclude/Order visible field-names                 (PF1)

              Record Information
         2.  Configure display of row-length, SQL-Code and flags      (PF2)

         3.  Exit utilities menu without action                       (PF3)

              Shadow-line options
         4.  Configure display of SHADOWed records                    (PF4)

              Window options
         5.  Open single-record (ZOOM) view in new window             (PF5)
```

Figure 17. DB2 Browse/Edit Utilities Menu.

This menu provides a user friendly method of executing commonly used SDE commands that alter the display of DB2 table data within the SDE window view.

Options are selected by entering the required option number at the prompt or by pressing the appropriate PFKey.

Field Selection

`Select/Exclude/Order visible field-names`

Executes SDESEL to generate a temporary CMX command centre file containing an SDE SELECT command that, by default, selects all columns for display in ascending order of field reference number.

The temporary CMX file is displayed in a CBLe text edit window which allows the user to employ standard text editing techniques to re-order and/or exclude the fields from the SELECT command and so select and re-order the display of table row fields.

The SELECT operation may be executed by placing the cursor on the first line of the SELECT command (beginning with "<") and pressing <F16<.

Record Information

`Configure display of row-length, SQL-Code and flags`

Opens the SDEUTRI sub-window which prompts the user to select the record information to be displayed at the start of all rows in the current SDE window view.

The "Display Record Information" option must be selected before any of the selected record information fields will be displayed. <Enter> actions the information display in the SDE edit view but keeps the SDEUTRI window open. <PF3> closes the window and returns focus to the Utilities Menu.

See the RECINFO SET/QUERY/EXTRACT option for details of the individual record information fields.

Shadow-line Options

`Configure display of SHADOWed records`

Opens the SDEUTSH sub-window which has a single option field indicating whether shadow lines are displayed for groups of EXCLUDED table rows view.

<Enter> actions the selection for the current SDE window view but keeps the SDEUTSH window open. <PF3> closes the window and returns focus to the Utilities Menu.

See the SHADOW SET/QUERY/EXTRACT option for details of shadow lines.

Field Options

`Configure display of Var-Length and Nullable fields`

Opens the ZZSGSETV panel containing several options to control the way Variable Length and Nullable fields are handled by the DB2 Table Editor.

`Delete focus field (<fieldname>) value`

Deletes the current value from the field at the cursor location.

This option is handy for columns containing very long values (i.e. that span multiple screens in table view, or multiple lines in single-row view) where you would otherwise have to use the <Erase-EOL> key many times to remove the current value.

Selecting this option executes the command:

```
REPLACELINE (<fieldname>) VALUE("")
```

Window Options

`Open single-record (ZOOM) view in new window`

Executes SDEZOOMW to open another SDE window view containing a single record view of the table row occupying the focus line. If the current SDE window view is already in SINGLE format then data in the new view will be a multi-record TABLE format view.

See the SDE command ZOOM.

DB2 Edit Settings

The DB2 Edit Settings panel (ZZSGSETV) is an interactive panel window, opened on selection of option 5. from the DB2 Browse/Edit Utilities popup.

This panel specifies options relating to Variable-Length and Nullable fields in the FileKit DB2 editor.

Panel Input Fields

`Varying Length Columns:`

`Remove trailing blanks>`
When set ON, any trailing blanks in the column value are stripped and the length of the column data adjusted accordingly.

When set OFF, there is no special treatment of trailing blanks.

This option corresponds to the SET/QUERY/EXTRACT option VSTRIP.

`Show end of string>`
When set ON, the end of the variable length column data is marked with the String DelimiterDisplay character.

When set OFF, the end of the variable length column data is not marked.

This option corresponds to the SET/QUERY/EXTRACT option VSHOWEND.

`String Delimiters - Input>`
The character which if present in the input determines the end of the variable length column data. The default is **#** (the hash sign).

This option corresponds to the SET/QUERY/EXTRACT option VENDCHAR (1).

`String Delimiters - Output>`
The character which is used on output to indicate the end of the data in a variable length column. The default is **<** (the less than sign).

This option corresponds to the SET/QUERY/EXTRACT option VENDCHAR (2).

`Columns that allow Null values:`

`Convert space to null>`
When set ON, setting a nullable column input field to blanks in the edit view will cause the column value (of whatever data type) to be set to NULL.

When set OFF, there is no special effect when setting a column to blanks. Note that non-character fields may give an invalid data type message in this case.

This option corresponds to the SET/QUERY/EXTRACT option NULLIFBLANK.

`Null Column Indicators - Input>`
The character which if specified sets the column value to NULL.

For character based fields, this value must be specified as the first and only non-blank value in the input area.

For non-character based fields, this value may specified anywhere in the input area.

The default is @ (the commercial at sign).

This option corresponds to the SET/QUERY/EXTRACT option NULLCHAR (1).

`Null Column Indicators - Output>`
The character which is used on output to indicate that a column has a NULL value.

The default is _ (the underscore).

This option corresponds to the SET/QUERY/EXTRACT option NULLCHAR (2).

DB2 Row Selection Panel

The DB2 Row Selection Panel is an interactive panel window used to generate a basic SQL WHERE clause for use in a DB2 table edit or browse operation, or in generating a FileKit SDO structure.

Starting the DB2 Row Selection Panel

The panel is started by any of the following:

- Specify "?" (question mark) for parameter *where-clause* in the DB2 options of the SDE EDIT, BROWSE or CREATE STRUCTURE primary commands.
- Specify "?" in the Where> input field of the Edit Object or Browse Object panels.
- Specify MODEL in the Action> input field of the DB2 Save SQL Error panel. This option is available only if the DB2 table row in error has a missing parent key in a referential constraint relationship. (SQL error -530)

Panel Description

This panel contains an embedded table of rows each of which represent 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 belonging to the DB2 table or view 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 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.

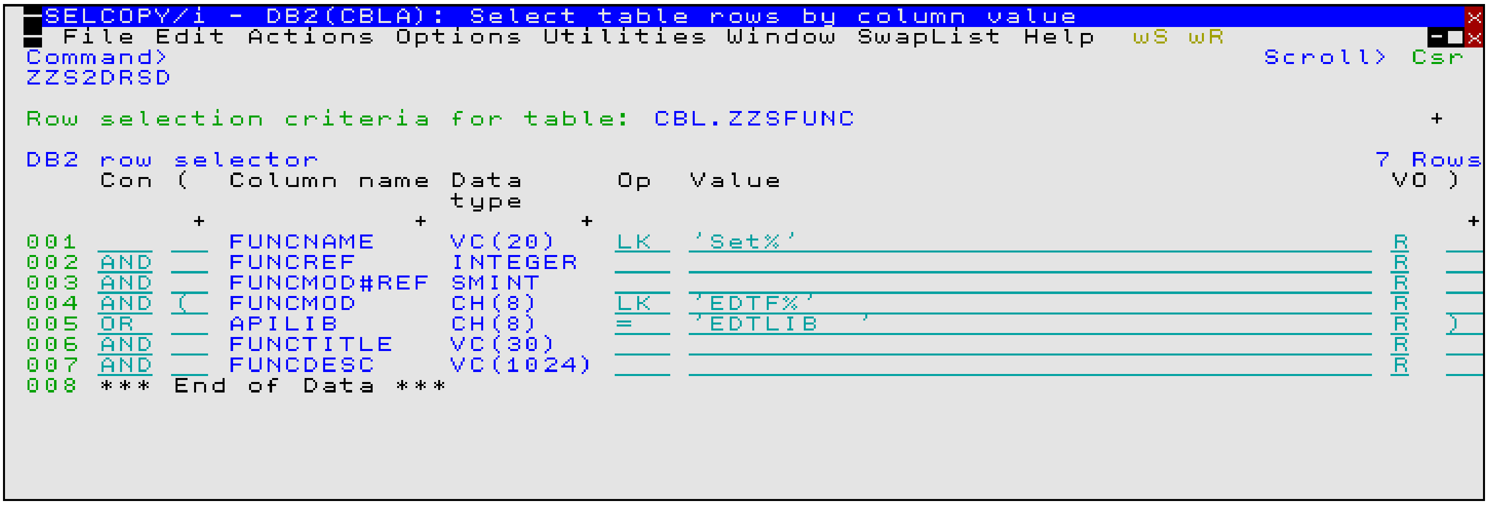
A predicate is selected for inclusion in the final WHERE search condition by the presence of a valid operator. A blank or null value in the operator column will deselect the predicate. The order in which selected predicate entries occur within the table dictates their location within the generated search 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 supports 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 the Value or parentheses fields in the embedded table view is not sufficient to enter the required input value, then the table row should be zoomed (default <F17>) and, if necessary, the field expanded (default <F14>) to accomodate the input 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. This is immediately passed to the originating EDIT, BROWSE or CREATE STRUCTURE operation for immediate, continued execution.



```
SELCOPY/i - DB2(CBLA): Select table rows by column value
 File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                       Scroll> Csr
ZZS2DRSD

Row selection criteria for table: CBL.ZZSFUNC                              +

DB2 row selector                                                        7 Rows
    Con (   Column name Data      Op  Value                              VO )
                        type
           +           +        +                                            +
001     ___ FUNCNAME    VC(20)    LK  'Set%'____________________________ R  ___
002 AND ___ FUNCREF     INTEGER   ___ __________________________________ R  ___
003 AND ___ FUNCMOD#REF SMINT     ___ __________________________________ R  ___
004 AND (   FUNCMOD     CH(8)     LK  'EDTF%'___________________________ R  ___
005 OR  ___ APILIB      CH(8)     =   'EDTLIB  '________________________ R  )
006 AND ___ FUNCTITLE   VC(30)    ___ __________________________________ R  ___
007 AND ___ FUNCDESC    VC(1024)  ___ __________________________________ R  ___
008 *** End of Data ***
```

Figure 18. DB2: Row Selection Panel.

Panel 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.

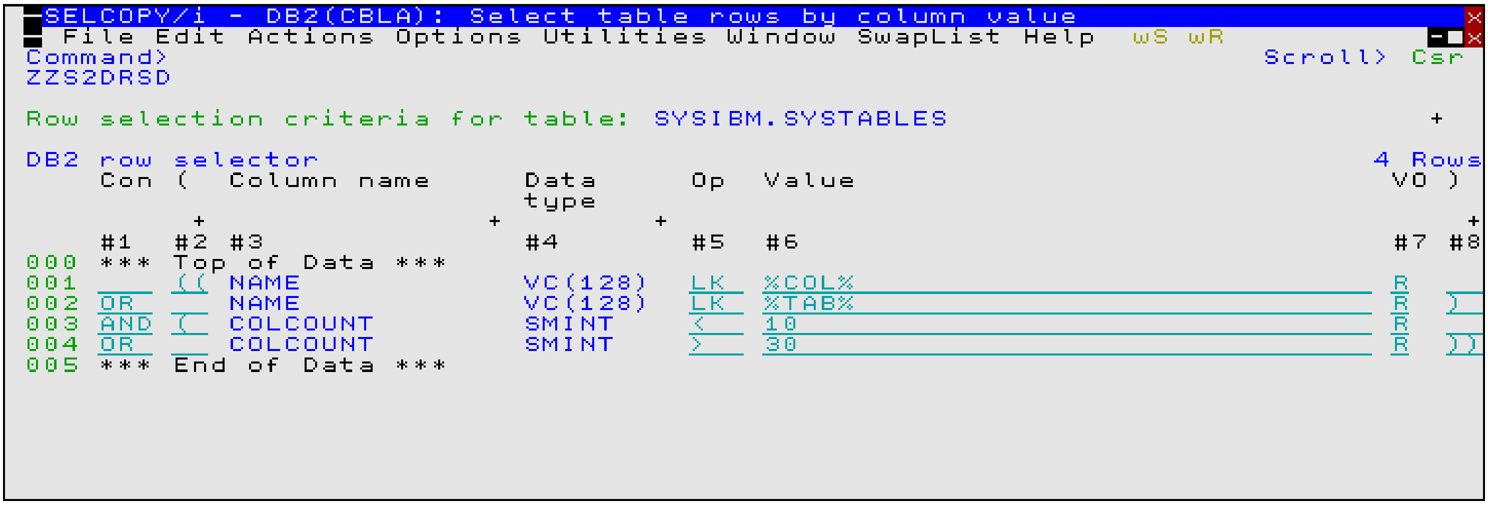
Any Case

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 "(" .

Example



```
 SELCOPY/i - DB2(CBLA): Select table rows by column value
  File Edit Actions Options Utilities Window SwapList Help   wS wR
Command>                                                           Scroll> Csr
ZZS2DRSD

Row selection criteria for table: SYSIBM.SYSTABLES                          +

DB2 row selector                                                         4 Rows
    Con (  Column name     Data     Op  Value                             VO )
                           type
         +               +        +                                           +
    #1  #2 #3              #4       #5  #6                                #7 #8
000 *** Top of Data ***
001     (( NAME            VC(128)  LK  %COL%                             R
002 OR     NAME            VC(128)  LK  %TAB%                             R  )
003 AND (  COLCOUNT        SMINT    <   10                                R
004 OR     COLCOUNT        SMINT    >   30                                R  ))
005 *** End of Data ***
```

Figure 19. DB2: Row Selection Panel Sample Use.

In this example we want to select all rows in the SYSIBM.SYSTABLES table which have **NAME** containing either **COL** or **TAB** as a substring and have **COLCOUNT** either less than **10** or greater than **30.**

We have replicated the **NAME** and **COLCOUNT** rows using the table prefix command **R** so that we can have multiple predicates for the same column.

We have also deleted all other column rows, apart from the ones we are interested in, using table prefix command **D** (delete single row) **DD** (delete a block of rows) or **D*** (delete the row and all rows following). This is not required but makes things clearer.

The outer enclosing parentheses are not actually required in this case but illustrate that multiple levels of perenthesis are supported. Note that the width of the left and right parenthesis columns is set to 2 by default. Up to 8 levels can be specified. To widen the parenthesis columns:

1. Show the column reference numbers using the command:

   ```
   SET REF ON
   ```

   (this has already been done in the above example)

2. Widen the parenthesis columns using the commands:

   ```
   SET COLWIDTH #2 8
   SET COLWIDTH #8 8
   ```

The SQL WHERE clause generated by this panel is:

```
WHERE ((NAME LIKE '%COL%' OR NAME LIKE '%TAB%')
       AND
       (COLCOUNT < 10 OR COLCOUNT > 30 ))
```

DB2 Save SQL Error Panel

The DB2 Save SQL Error panel is a modal, interactive panel window which displays details of a failure of a SAVE command issued while editing a DB2 table.

SQL DELETE, INSERT or UPDATE statements are generated and executed as appropriate for each changed DB2 row and the SQLCODE is saved. On completion of the SAVE command, all rows for which the generated SQL statement had a non-zero SQLCODE are flagged as being in error.

In table view the SQLCODE is displayed in the prefix area or, if the SET RECINFO ON SQLCODE option is in effect, in the SQLCODE column of the record information area.

In single view the SQLCODE is displayed in its own field at the top of the view.

The DB2 Save SQL Error panel is opened either by entering the E prefix command in the prefix area of the row in error, or by using the ERROR primary command with the focus row being the row in error.

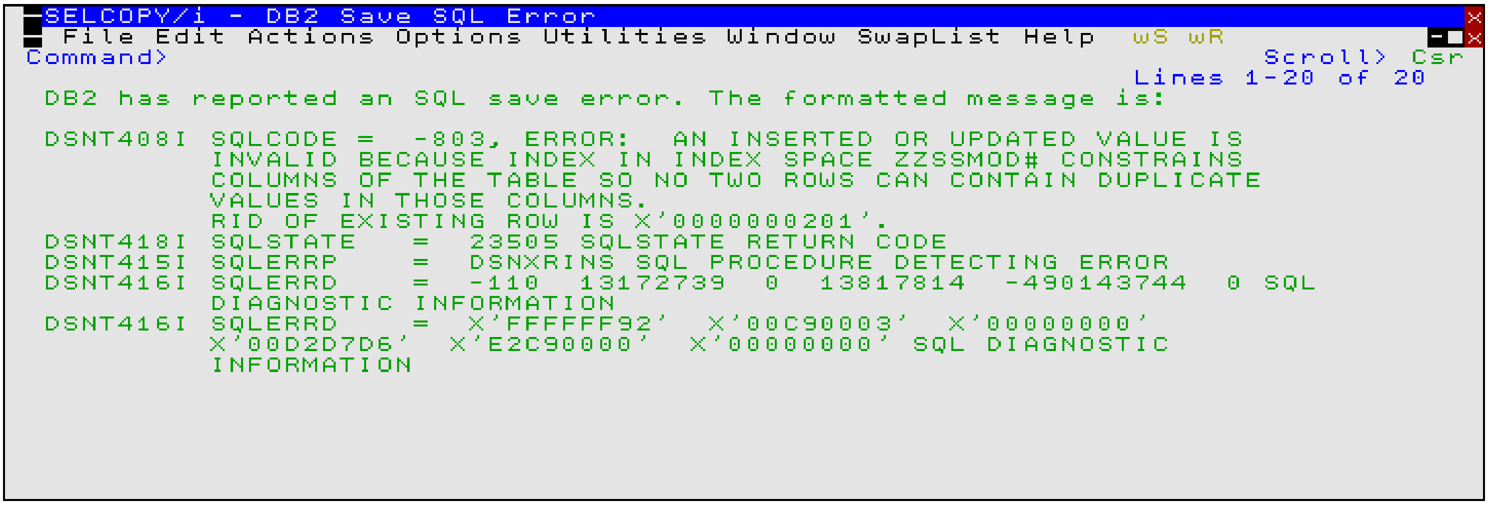
The SQL error panel has a number of formats depending on the value of the SQLCODE.

SQLCODE	Error format
All but those listed below	The default format see Format 1.
+100	The row not found format see Format 2.
-530	The missing parent key format see Format 3.
-531	Foreign key update error format see Format 4.
-532	Delete rule violation error format see Format 5.

Format 1. The default SQL error panel.

In most cases the SQL error panel displays the formatted DB2 error message corresponding to the SQLCODE as described in the **DB2 Codes** manual. The displayed SQLCODE is prefixed by "E".

For example, if an attempt was made to insert a row with a duplicate prime key then SQLCODE **-803** is returned and the following SQL error panel is displayed:


SELCOPY/i - DB2 Save SQL Error
File Edit Actions Options Utilities Window SwapList Help wS wR
Command>
Scroll> Csr
Lines 1-20 of 20

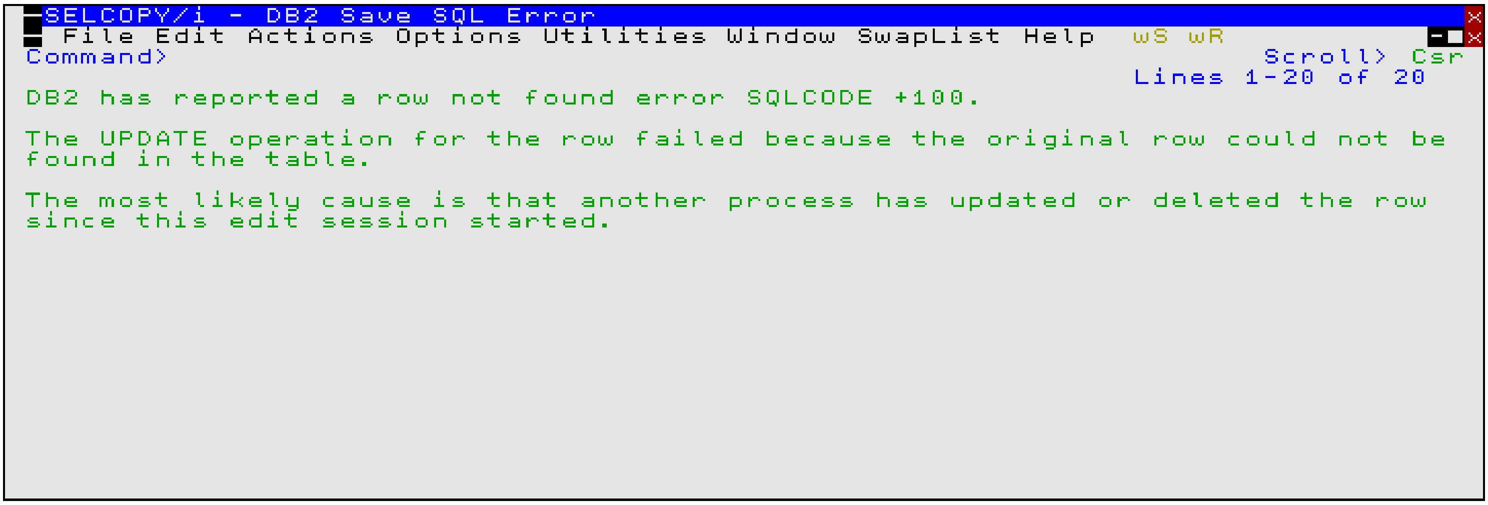
```
 DB2 has reported an SQL save error. The formatted message is:

 DSNT408I SQLCODE =  -803, ERROR:  AN INSERTED OR UPDATED VALUE IS
          INVALID BECAUSE INDEX IN INDEX SPACE ZZSSMOD# CONSTRAINS
          COLUMNS OF THE TABLE SO NO TWO ROWS CAN CONTAIN DUPLICATE
          VALUES IN THOSE COLUMNS.
          RID OF EXISTING ROW IS X'0000000201'.
 DSNT418I SQLSTATE   =  23505 SQLSTATE RETURN CODE
 DSNT415I SQLERRP    =  DSNXRINS SQL PROCEDURE DETECTING ERROR
 DSNT416I SQLERRD    =  -110  13172739  0  13817814  -490143744  0 SQL
          DIAGNOSTIC INFORMATION
 DSNT416I SQLERRD    =  X'FFFFFF92'  X'00C90003'  X'00000000'
          X'00D2D7D6'  X'E2C90000'  X'00000000' SQL DIAGNOSTIC
          INFORMATION
```

Figure 20. DB2: Save SQL Error - General.

Format 2. The Row not found panel for SQLCODE +100.

The SQLCODE **+100** is returned when the row requested is not found. The displayed SQLCODE is prefixed by "W".

If an attempt is made to update a row which has been changed or deleted by another process since the edit session started the following SQL error panel is displayed:


SELCOPY/i - DB2 Save SQL Error
File Edit Actions Options Utilities Window SwapList Help wS wR
Command>
Scroll> Csr
Lines 1-20 of 20

```
DB2 has reported a row not found error SQLCODE +100.

The UPDATE operation for the row failed because the original row could not be
found in the table.

The most likely cause is that another process has updated or deleted the row
since this edit session started.
```

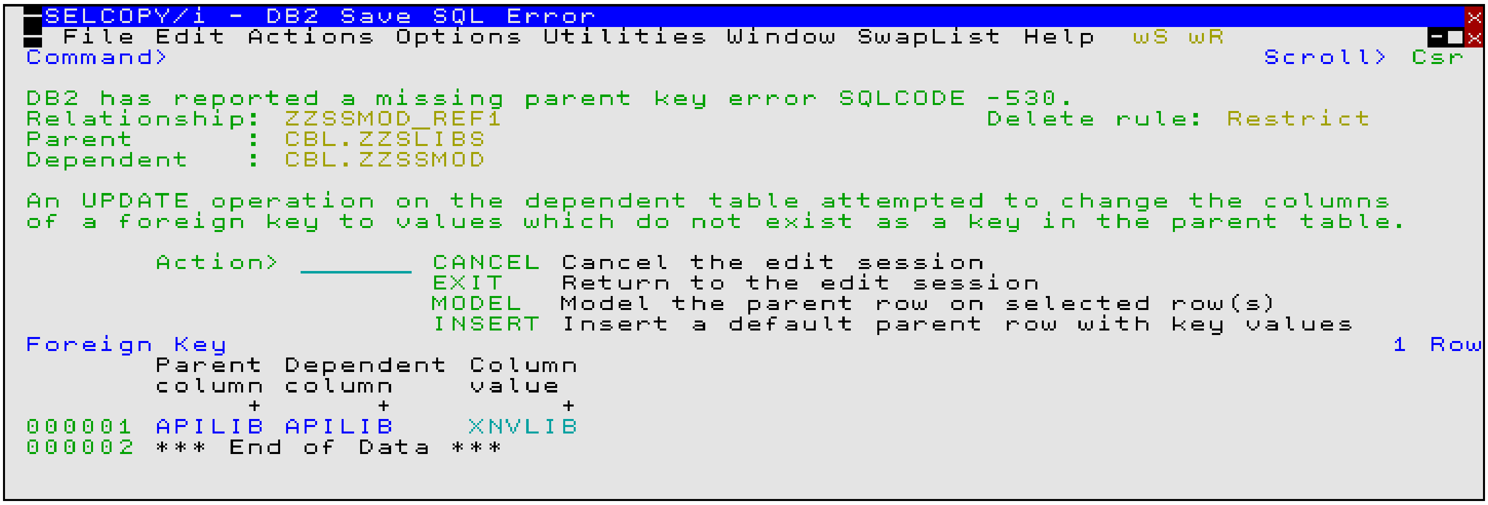
Figure 21. DB2: Save SQL Error +100 - Missing Table Row.

Format 3. Missing parent key panel for SQLCODE -530

The SQLCODE **-530** is returned when a row is inserted or updated with the values of the columns of a foreign key (the parent key) not representing an existing key in the parent table. The displayed SQLCODE is prefixed by "R" to indicate a referential constraint error. The following SQL error panel is displayed:

As well as describing the foreign key in error, the **MODEL** or **INSERT** action commands can be used to start a dependent edit session to insert the missing parent row.

See Panel Fields for a description of the panel contents.



```
 SELCOPY/i - DB2 Save SQL Error
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                       Scroll> Csr

DB2 has reported a missing parent key error SQLCODE -530.
Relationship: ZZSSMOD_REF1                          Delete rule: Restrict
Parent      : CBL.ZZSLIBS
Dependent   : CBL.ZZSSMOD

An UPDATE operation on the dependent table attempted to change the columns
of a foreign key to values which do not exist as a key in the parent table.

       Action> ______ CANCEL Cancel the edit session
                      EXIT   Return to the edit session
                      MODEL  Model the parent row on selected row(s)
                      INSERT Insert a default parent row with key values
Foreign Key                                                          1 Row
       Parent Dependent Column
       column column    value
            +       +          +
000001 APILIB APILIB    XNVLIB
000002 *** End of Data ***
```

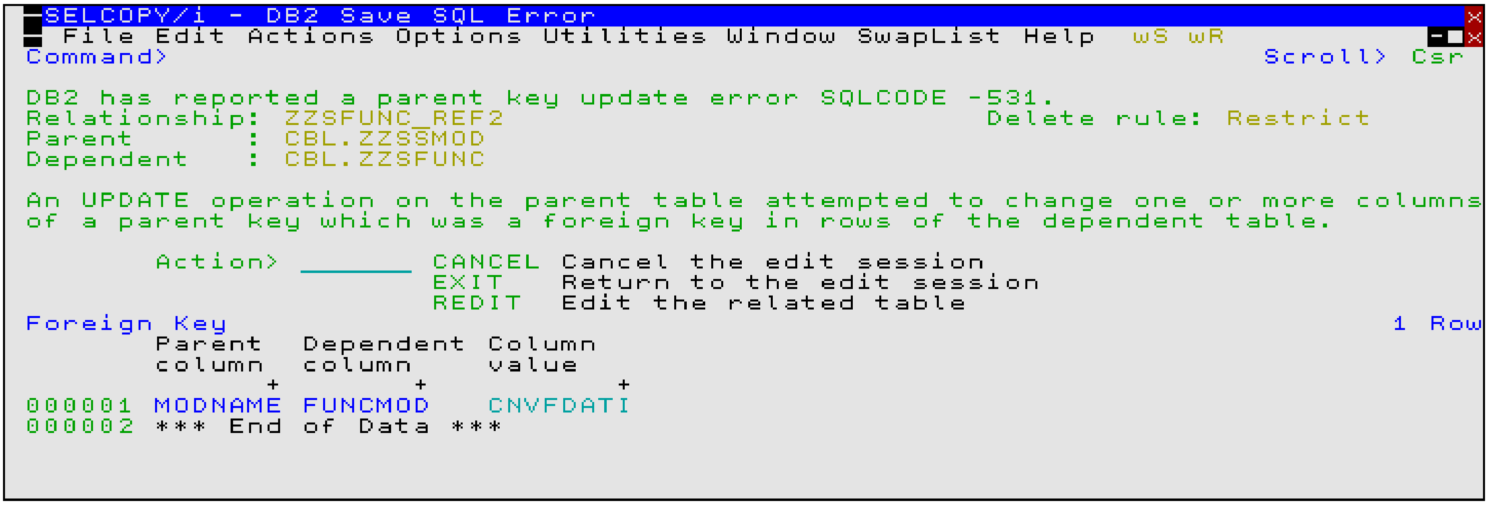
Figure 22. DB2: Save SQL Error -530 - Missing Parent Key.

Format 4. Parent key update error panel for SQLCODE -531

The SQLCODE **-531** is returned when a row is updated in such a way that changes the values of columns which represent the foreign key of an existing dependent (child) table row. The displayed SQLCODE is prefixed by "R" to indicate a referential constraint error. The following SQL error panel is displayed:

As well as describing the foreign key in error, the **REDIT** action command can be used to start a dependent edit session to edit the dependent rows.

See Panel Fields for a description of the panel contents.



```
 SELCOPY/i - DB2 Save SQL Error
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                       Scroll> Csr

DB2 has reported a parent key update error SQLCODE -531.
Relationship: ZZSFUNC_REF2                          Delete rule: Restrict
Parent      : CBL.ZZSSMOD
Dependent   : CBL.ZZSFUNC

An UPDATE operation on the parent table attempted to change one or more columns
of a parent key which was a foreign key in rows of the dependent table.

       Action> ______ CANCEL Cancel the edit session
                      EXIT   Return to the edit session
                      REDIT  Edit the related table
Foreign Key                                                          1 Row
       Parent  Dependent Column
       column  column    value
             +        +          +
000001 MODNAME FUNCMOD   CNVFDATI
000002 *** End of Data ***
```

Figure 23. DB2: Save SQL Error -531 - Parent Key Update.

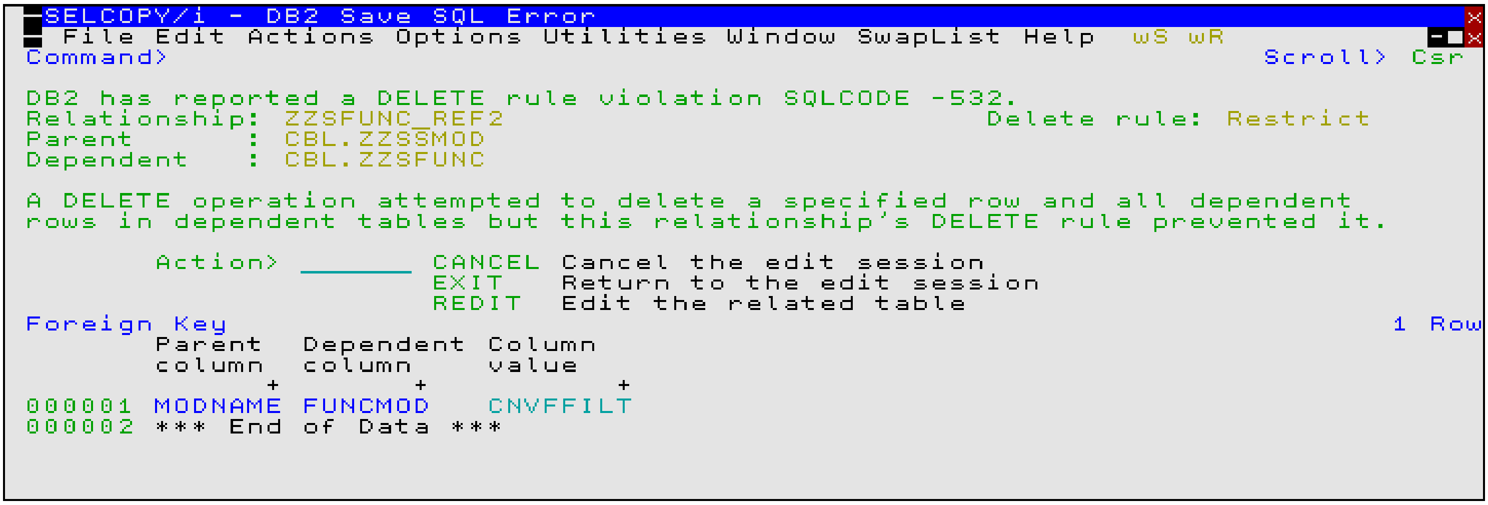
Format 5. Delete rule violation error panel for SQLCODE -532

The SQLCODE **-532** is returned when a row is deleted which has dependent (child) rows for which the referential constraint delete rule is **RESTRICT** or **NO ACTION.** The displayed SQLCODE is prefixed by "R" to indicate a referential constraint error. The following SQL error panel is displayed:

As well as describing the foreign key in error, the **REDIT** action command can be used to start a dependent edit session to edit the dependent rows.

See Panel Fields for a description of the panel contents.

Note that when a deleted row causes this error the deleted row is re-inserted at the top of the current view. Having corrected the relationship that caused the error, the row must be re-deleted from the edit view in order to action the SQL DELETE of the row on a subsequent save operation.



```
 SELCOPY/i - DB2 Save SQL Error
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                           Scroll> Csr

DB2 has reported a DELETE rule violation SQLCODE -532.
Relationship: ZZSFUNC_REF2                          Delete rule: Restrict
Parent      : CBL.ZZSSMOD
Dependent   : CBL.ZZSFUNC

A DELETE operation attempted to delete a specified row and all dependent
rows in dependent tables but this relationship's DELETE rule prevented it.

       Action> ______ CANCEL Cancel the edit session
                      EXIT   Return to the edit session
                      REDIT  Edit the related table
Foreign Key                                                               1 Row
       Parent  Dependent Column
       column  column    value
             +       +          +
000001 MODNAME FUNCMOD   CNVFFILT
000002 *** End of Data ***
```

Figure 24. DB2: Save SQL Error -532 - Delete Rule Violation.

Panel Fields

Relationship:

A non-enterable field displaying the name of the referential constraint for which the constraint violation occurred.

Delete rule:

A non-enterable field displaying the delete rule assigned to the referential constraint.

The delete rule determines the action performed when a row of the parent DB2 table is deleted. Possible values are: **No Action**, **Cascade**, **Set null** and **Restrict**.

Parent:

A non-enterable field displaying the name of the DB2 parent table in the relationship.

Dependent:

A non-enterable field displaying the name of the DB2 dependent (foreign key) table in the relationship.

Action>

Enter one of the valid actions to be performed for the row in error. A minimum abbreviation of 1 character may be used to specify any of the following valid actions:

CANCEL

Cancel the edit of the DB2 table. The data edit view is closed and all row changes for which an SQL error has been flagged are discarded.

EXIT

Exit the DB2 Save SQL Error panel and return to the DB2 table edit view.

MODEL

Applicable only to SQL error -530 (missing parent key), this action closes the DB2 save SQL Error panel and opens the DB2 Row Selection Panel so that rows of the parent table that match a specific search condition may be selected and displayed in an edit related table view.

In addition to the selected parent table rows, the edit view will contain a new row in which the parent key columns contain the missing foreign key value. Values within the selected table rows may be used to model values for other columns in the newly inserted row before it is saved.

Having inserted and saved the new parent table row, the edit related table view may be closed, returning focus to the original DB2 table edit view in which the error was detected.

INSERT

Applicable only to SQL error -530 (missing parent key), this action closes the DB2 save SQL Error panel and opens an edit related table view containing only a newly inserted row in which the parent key columns contain the missing foreign key value. Values in other columns of the row are initialised to their defaults.

Having inserted and saved the new parent table row, the edit related table view may be closed, returning focus to the original DB2 table edit view in which the error was detected.

REDIT

Applicable only to SQL errors -531 (parent key update) and -532 (delete rule), this action closes the DB2 save SQL Error panel and opens an edit related table view containing only those rows of the dependent table for which the foreign key is in error.

Foreign Key Table

A table displaying the value of the dependent DB2 table foreign key for which the relationship would be broken. Rows of the Foreign Key table represent corresponding columns of the parent and foreign keys in the order in which they occur within those keys.

Parent column

Displays the name of the parent key columns in the order in which they occur within the parent key.

Dependent column

Displays the name of the dependent table foreign key columns in the order in which they occur within the foreign key.

Column value

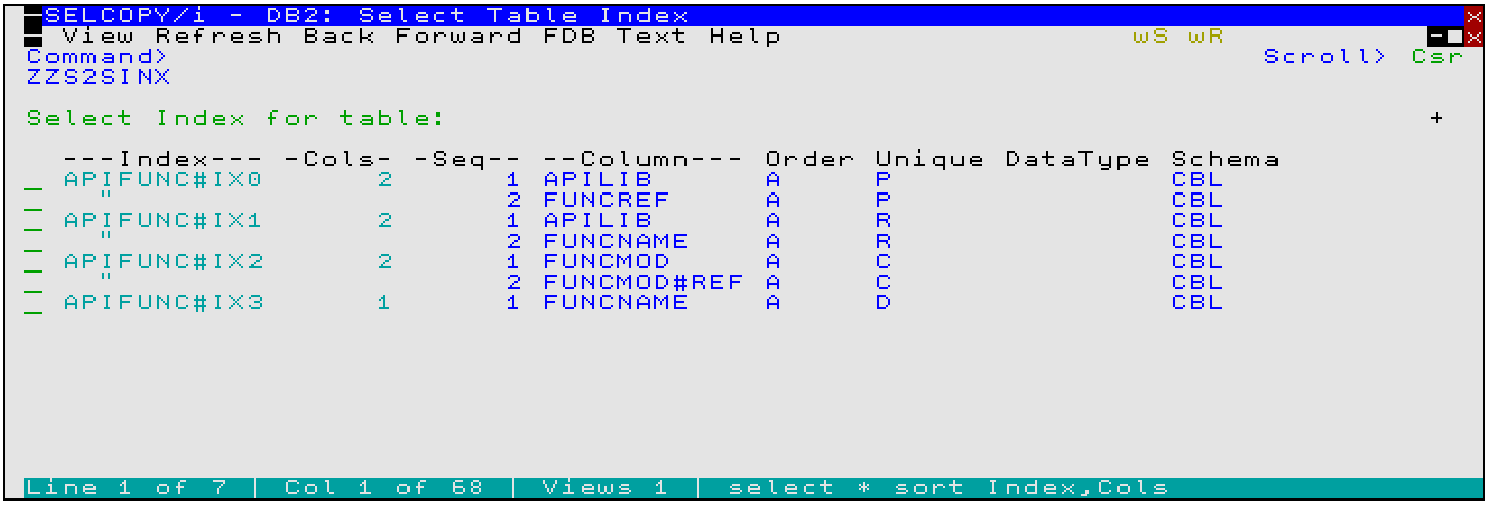
Displays the values in the dependent table foreign key columns for the foreign key violation caused by the SQL INSERT, DELETE or UPDATE operation.

DB2 Select Table Index

The DB2 Select Table Index panel (ZZS2SINX) is a modal, interactive panel window which displays a selectable list of indexes that have been defined on the DB2 table.

The panel is displayed when the SORTINDEX primary command is executed with no parameters from an existing DB2 browse/edit session.

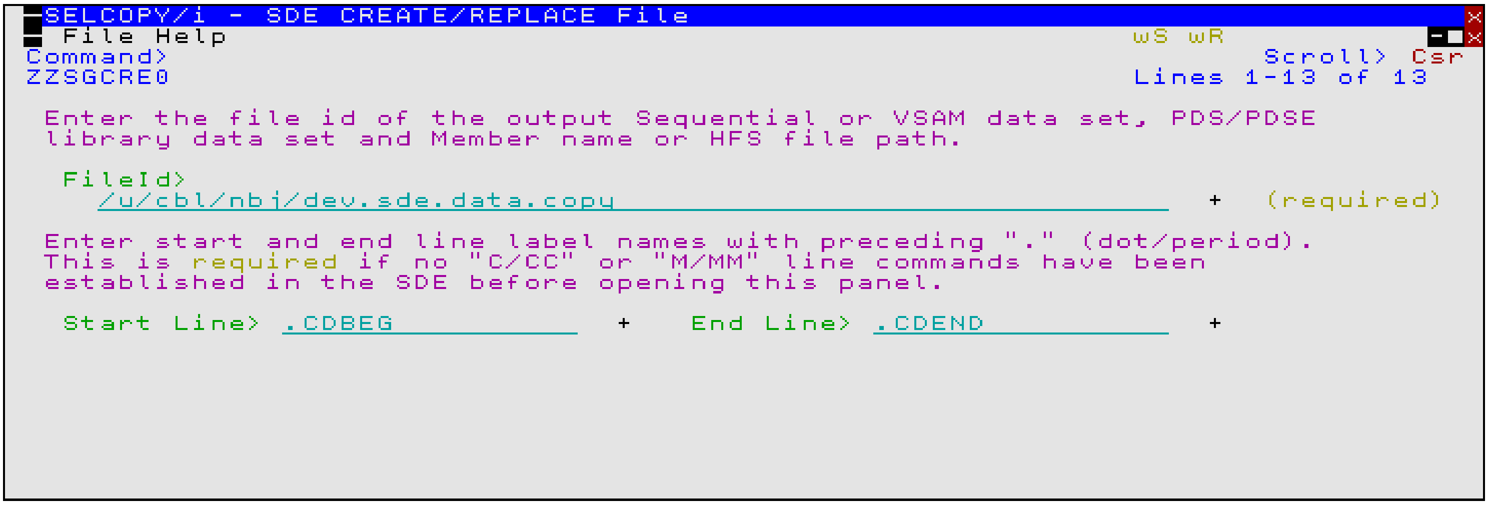
If more than one key column is defined for an index, then repeated list rows exist for that index with one row for each index key column. Selection of any of these rows will select the equivalent index and sort the rows by that index's key columns.



```
 SELCOPY/i - DB2: Select Table Index
  View Refresh Back Forward FDB Text Help                           wS wR
Command>                                                         Scroll> Csr
ZZS2SINX

Select Index for table:                                                    +

  ---Index--- -Cols- -Seq-- --Column--- Order Unique DataType Schema
_ APIFUNC#IX0      2      1 APILIB      A     P               CBL
_    "                    2 FUNCREF     A     P               CBL
_ APIFUNC#IX1      2      1 APILIB      A     R               CBL
_    "                    2 FUNCNAME    A     R               CBL
_ APIFUNC#IX2      2      1 FUNCMOD     A     C               CBL
_    "                    2 FUNCMOD#REF A     C               CBL
_ APIFUNC#IX3      1      1 FUNCNAME    A     D               CBL








Line 1 of 7 | Col 1 of 68 | Views 1 | select * sort Index,Cols
```

Figure 25. DB2 Select Table Index Panel.

SDE CREATE/REPLACE File Panel



```
SELCOPY/i - SDE CREATE/REPLACE File
 File Help                                                   wS wR
Command>                                                             Scroll> Csr
ZZSGCRE0                                                          Lines 1-13 of 13

 Enter the file id of the output Sequential or VSAM data set, PDS/PDSE
 library data set and Member name or HFS file path.

  FileId>
    /u/cbl/nbj/dev.sde.data.copy______________________________  +   (required)

 Enter start and end line label names with preceding "." (dot/period).
 This is required if no "C/CC" or "M/MM" line commands have been
 established in the SDE before opening this panel.

  Start Line> .CDBEG__________    +    End Line> .CDEND__________    +
```

Figure 26. SDE Create/Replace File Panel.

The SDE Create/Replace File panel (ZZSGCRE0) is an interactive panel window, opened on executing primary commands CREATE or REPLACE without any parameters.

This panel is used to create new or replace existing sequential or VSAM data sets, PDS/PDSE library members or HFS files.

Panel Input Fields

FileId>

Specifies the fileid of the target file to be created or replaced.

If the specified fileid is less than 8 characters in length and is a valid member name, the target file will be a member name belonging to the same PDS/PDSE library referenced in the focus SDE view. If the focus SDE view does not display a library member, then the fileid will be treated as an HFS file within the current HFS working directory.

For a REPLACE operation only, if the fileid includes a volume id, then the target file may be a cataloged or uncataloged data set or PDS/PDSE library which exists in that volume's VTOC. e.g. VOLWKA:DEV.UNCATLG.FILE.

Start Line>

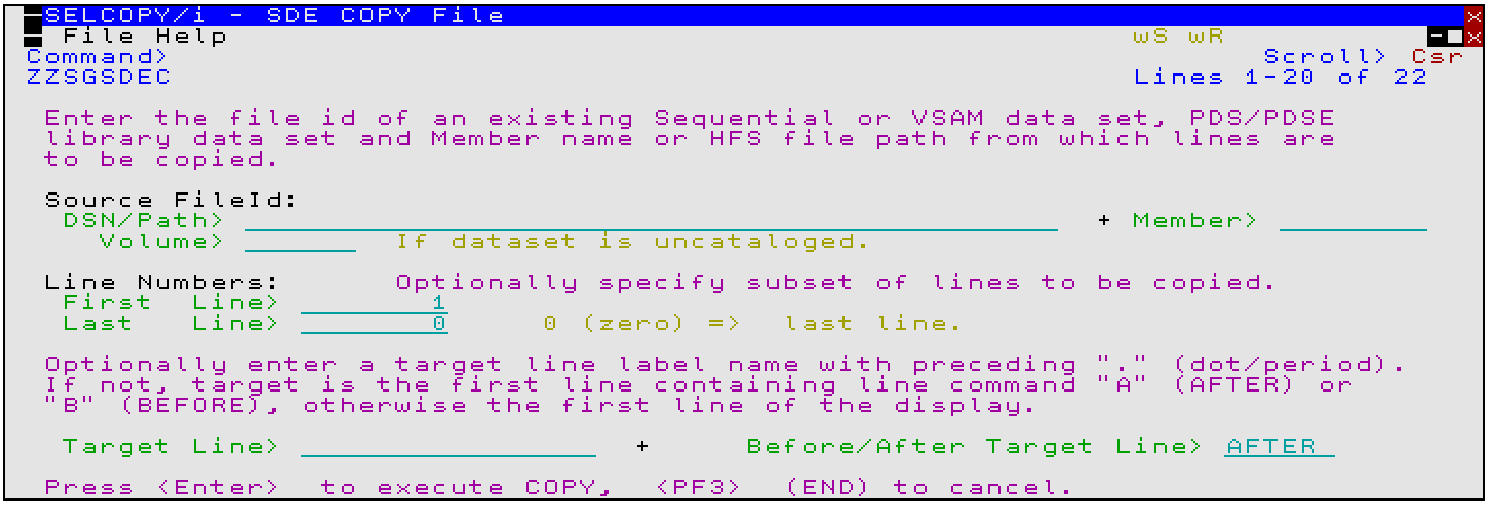
A label name, which includes the preceding "." (dot/period), identifying the first line of the group of lines to be copied to the target file.

If not specified, then the group of lines must have been marked using "C" or "M" line (prefix area) commands.

End Line>

A label name, which includes the preceding "." (dot/period), identifying the last line of the group of lines to be copied to the target file. If a start line label name has been specified, a valid entry in theis field is mandatory.

SDE COPY File Panel



```
 SELCOPY/i - SDE COPY File
  File Help                                                  wS wR
Command>                                                            Scroll> Csr
ZZSGSDEC                                                      Lines 1-20 of 22

 Enter the file id of an existing Sequential or VSAM data set, PDS/PDSE
 library data set and Member name or HFS file path from which lines are
 to be copied.

 Source FileId:
  DSN/Path> ____________________________________________  + Member> ________
    Volume> ______   If dataset is uncataloged.

 Line Numbers:       Optionally specify subset of lines to be copied.
  First  Line> _______1
  Last   Line> _______0      0 (zero) =>  last line.

 Optionally enter a target line label name with preceding "." (dot/period).
 If not, target is the first line containing line command "A" (AFTER) or
 "B" (BEFORE), otherwise the first line of the display.

  Target Line> ________________   +     Before/After Target Line> AFTER

 Press <Enter>  to execute COPY,  <PF3>  (END) to cancel.
```

Figure 27. SDE Copy File Panel.

The SDE Copy File panel (ZZSGSDEC) is an interactive panel window, opened on executing primary command COPY without any parameters.

This panel is used to copy lines from an existing sequential or VSAM data set, PDS/PDSE library member or HFS file into data being edited in the focus SDE edit view.

Panel Input Fields

Source FileId:
Fields that identify the source file from which lines are to be copied.

DSN/Path>
An absolute or relative HFS Path name or the fully qualified name of a sequential data set or PDS/PDSE library. Quotes are permitted but unnecessary.

A selectable list of data set names or HFS files will be displayed as appropriate if either wild card character "%" (percent) or "*" (asterisk), both representing zero or more characters, is specified. If a volume id exists in the Volume field, then a list of selectable data sets will be restricted to those contained in that volume's VTOC.

Member>
If DSN/Path field is the DSN of a PDS/PDSE library, specifies the source file member name.

A selectable list of members will be presented if wildcard character "*" or "%" occurs anywhere within the specified member name or the member name is left blank.

Volume>
Specifies a volume serial id mask for an uncataloged source file. (Not applicable to HFS files.)

Line Numbers:
Fields that optionally specify the start and end lines of a group of lines to be copied from the source file.

First Line>
Line number of the first line of a subset group of lines to be copied from the the source file.

End Line>
Line number of the last line of a subset group of lines to be copied from the the source file.

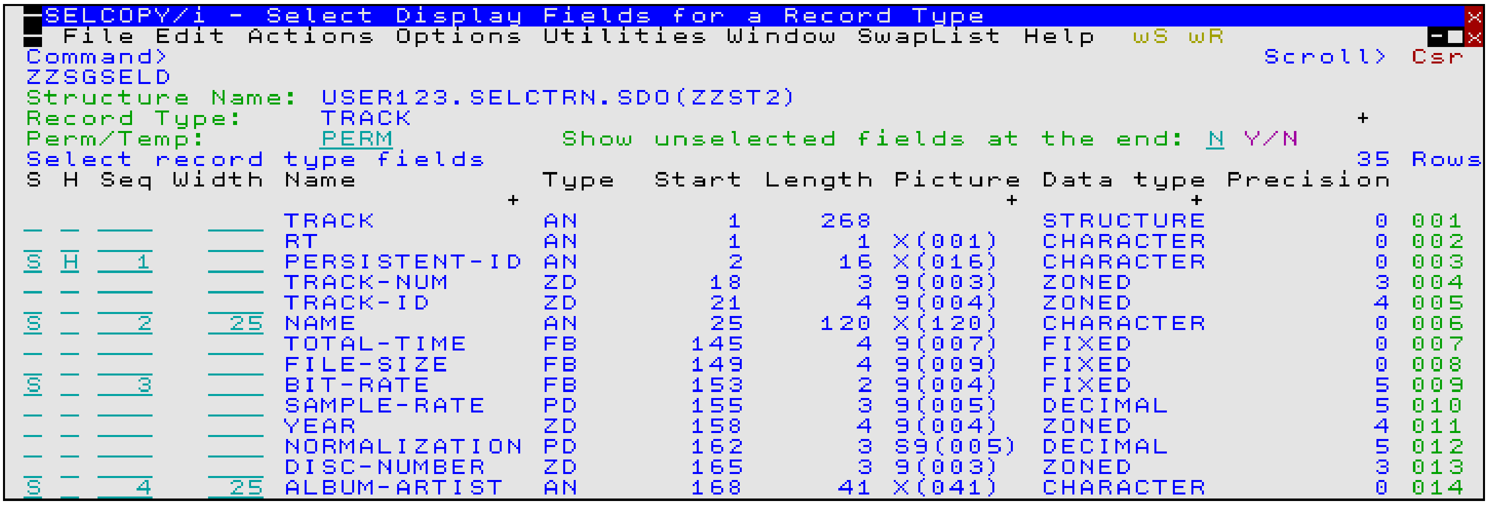
If 0 (zero), the default is the last line of the source file.

Target Line>
A label name, which includes the preceding "." (dot/period), identifying the target line within the focus SDE edit view where lines are to be copied.

If not specified, then the target line is the first line in the SDE view containing line (prefix area) command "A" or "B", otherwise the first line of the SDE view.

Before/After Target Line>
Indicates whether the copied lines are to be copied before or after the line assigned the specified line label.

SDE SELECT Columns Panel



```
SELCOPY/i - Select Display Fields for a Record Type
 File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                        Scroll> Csr
ZZSGSELD
Structure Name: USER123.SELCTRN.SDO(ZZST2)
Record Type:    TRACK                                              +
Perm/Temp:      PERM       Show unselected fields at the end: N Y/N
Select record type fields                                        35 Rows
S H Seq Width Name          Type  Start Length Picture  Data type Precision
                           +                         +          +
_ _ ___  ___  TRACK         AN        1    268          STRUCTURE      0 001
_ _ ___  ___  RT            AN        1      1 X(001)   CHARACTER      0 002
S H __1  ___  PERSISTENT-ID AN        2     16 X(016)   CHARACTER      0 003
_ _ ___  ___  TRACK-NUM     ZD       18      3 9(003)   ZONED          3 004
_ _ ___  ___  TRACK-ID      ZD       21      4 9(004)   ZONED          4 005
S _ __2  _25  NAME          AN       25    120 X(120)   CHARACTER      0 006
_ _ ___  ___  TOTAL-TIME    FB      145      4 9(007)   FIXED          0 007
_ _ ___  ___  FILE-SIZE     FB      149      4 9(009)   FIXED          0 008
S _ __3  ___  BIT-RATE      FB      153      2 9(004)   FIXED          5 009
_ _ ___  ___  SAMPLE-RATE   PD      155      3 9(005)   DECIMAL        5 010
_ _ ___  ___  YEAR          ZD      158      4 9(004)   ZONED          4 011
_ _ ___  ___  NORMALIZATION PD      162      3 S9(005)  DECIMAL        5 012
_ _ ___  ___  DISC-NUMBER   ZD      165      3 9(003)   ZONED          3 013
S _ __4  _25  ALBUM-ARTIST  AN      168     41 X(041)   CHARACTER      0 014
```

Figure 28. SDE Select Columns Panel.

The SDE Select Columns panel (ZZSGSELD) is an interactive panel window, opened on executing either of the following:

1. Primary command SELECT with no parameters. In this case the default record type is used.
2. Line (prefix) command *SEL* in the prefix area of a record of the required record type.

This panel is used to update column width, column selection and column display order for a particular record type data mapping in the focus SDE view. The changes made in this panel may be temporary or saved as a permanent update to the structure (SDO).

Note that the dialog is a *modal* panel. While it is open no other window can be made the focus. The dialog must be terminated (with the **END** or **CANCEL** command) before any other window can become active.

If opened for a record type which has array (OCCURS DEPENDING) entries, then the individual array element entries, represented as field names suffixed with a parenthesised element number subscript, are excluded by default. These entries may be displayed and excluded using primary commands +ELEMENT (+E) and -ELEMENT (-E) respectively.

Panel Input Fields

The dialog panel contains header fields naming the structure and record type and a table of all data elements for the record type.

Header fields:

`Structure Name:`
The name of the structure being used in the current edit or browse session. This field is set automatically and is not enterable.

`Record Type:`
The name of the record type for which the data element selection is to be made. It is the focus record type when the dialog was envoked. This field is set automatically and is not enterable.

`Field Name:`
Enter a field/column name mask to filter the embedded table rows by the contents of the **Name** field. All fields with names that do not match the field name mask are excluded from the table display.

If a name mask is specified, the embedded table command `ALL NAME LIKE "`*`name_mask`*`"` is generated and executed when <Enter> is pressed.

The name mask is **not** case sensitive and may include wildcard characters "*" (asterisk), representing zero or more characters, and "%" (percent), representing a single character. If no wildcard characters are used, the specified name mask is treated as having an implied wildcard character "*" at both the start and end of the name mask. e.g. A name mask of *File* is equivalent to _*File*_. Use of a field name mask with wildcards may be particularly useful in selecting only fields belonging to a particular level. e.g. _*3 *_ will select only rows with level 3 field names.

Beware that rows that have been selected using the selection column of the embedded table but then excluded from the table display by the field name filter, are **not** included in the final selection.

Perm/Temp:

The permanent or temporary indicator. This field determines whether the selection is temporary (i.e. just for the current edit or browse session) or permanent (i.e. the selection will be saved in the structure and therefore automatically invoked in the next edit or browse session using this structure).

If PERM is selected but the active SDO is a temporary structure (e.g. one created dynamically for a DB2 table or from a COBOL, PL1 or Assembler copybook or ADATA file), then the Specify new Structured Data Object (SDO) name panel is opened. This panel prompts for the name of a Structure Definition File (SDF), a data set or library member to which the SDO will be saved and so made permanent.

Show unselected fields at the end:

If set to **Y**, this field indicates that any fields that are not explcitly selected should be added to the end in their default order.

Table fields:

The dialog panel contains a scrollable table of all the data elements in the record type.

A selection is made by entering the selection, hold and sequence fields in the table rows. A data element is selected only if its selection field contains *S* (entering the hold or sequence fields does not select the field).

Note that excluded rows are not selected even if their selection field is set to *S.*

If a selection has already been made for the record type then the selection, hold and sequence fields will be automatically set on entry to the dialog to reflect the previous selection.

S

The selection column. Enter a non-blank character to select the data element. Any non-blank character entered causes this field to be set to *S* indicating that the data element in that row is selected.

H

The hold column. Enter a non-blank character to hold the selected data element when scrolling horizontally. Any non-blank character entered causes this field to be set to *H* indicating that the data element in that row is held if it is selected.

Seq

The selection sequence column. Enter a sequence number to control the order in which the selected data elements will appear in the display. If no sequence numbers are entered the selection sequence is the same as the order of the selected data elements in the table. If some but not all selected data elements are given sequence numbers then those with sequence number 0 will be selected *after* all those with sequence number >0, in table order.

Note that the order of data elements in the table can be changed using the *M* (move) or *MM* (block move) prefix commands.

Width

This option controls the number of characters that will be displayed for the selected data element. Column width cannot be increased beyond the column's defined maximum width.

The *"prefix"* area

Each table row has a prefix area (actually displayed on the right) which contains the data element sequence number. This field is enterable and may be used for table prefix commands (for example *X* to exclude rows, *M* to move rows).

All other table fields are not enterable and contain attributes of the record type data elements:

Name

The level and name of the data element.

Type

The abbreviated data type of the data element.

Start

The start position of the data element. If the record type describes a record which has variable length fields (and therefore potentially variable positions) this represents the start position in the expanded form of the record where all fields are expanded to maximum length.

Length

The length of the data element. If the data element is of variable length this is the maximum possible length.

Picture

The PICTURE string used to define the data element.

Data type

The internal data type of the data element.

Precision

The precision (number of digits) of *DECIMAL* data elements.

Scale

The scale (number of decimal fraction digits) of *DECIMAL* data elements.

RefNo
The data element reference number (its defined sequence in the record type).

Dimension
The number of array dimensions of an array data element (0 if not an array).

Max length
The maximum length of the data element.

Min length
The minimum length of the data element.

Array Type
Array status of the data element. NotArray, Array or ArrayElement.

Remarks
The content of any remarks string defined for the data element in the structure.

Panel Primary Commands

Commands with a specific effect on the SELECT dialog are:

END
End the dialog and generate and execute any selection made by the dialog.
Set on Function Key **F3** by default.

CANCEL (CAN)
End the dialog without making a selection.

RESET (RES)
Set all selection, hold, sequence and width fields to their default values. To reset any of the fields individually add the keyword options **S**, **HOLD (H)**, **SEQ** and **WIDTH (W)** respectively.

ARRAYELEMENT (ARRAYE)
Exclude or Display array (OCCURS DEPENDING) elements. To display elements, add the keyword option **SHOW (S)** or **+**. To exclude elements, add the keyword option **EXCLUDE (EX)**, **X** or **-**.

+ELEMENT (+E)
Equivalent to ARRAYELEMENT SHOW.

-ELEMENT (-E)
Equivalent to ARRAYELEMENT EXCLUDE.

EXECUTE (EX)
Generate and execute the selection without leaving the dialog.
Set on Function Key **F16** by default.

This command is espeecially handy when operating in "windowed" mode where the invoking structured browse/edit window is simultaneously visible. Please note that the SELECT dialog window itself may be moved, shrunk/stretched in the usual way in order to reveal the invoking browse/edit window.

SELALL (SELA)
Set the selection field to 'S' for all data elements.
Set on Function Key **F5** by default.

DESELALL (DES)
Reset the selection field to ' ' for all data elements.
Set on Function Key **F6** by default.

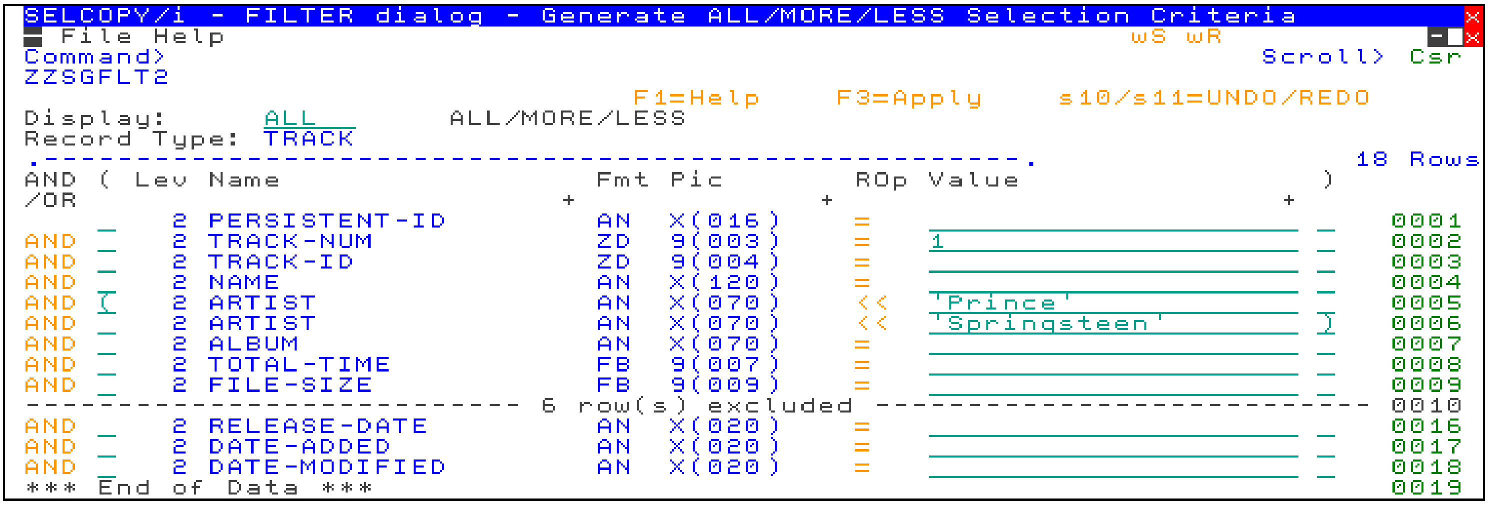
General table edit commands are also available. For example:

UNDO
Undo the last change to the table.
Set on Function Key **F22** by default.

REDO
Redo the last change to the table that was undone.
Set on Function Key **F23** by default.

UP | DOWN | LEFT | RIGHT | TOP | BOTTOM
Table scrolling commands.

SDE FILTER Records Panel



```
SELCOPY/i - FILTER dialog - Generate ALL/MORE/LESS Selection Criteria
 File Help                                                     wS wR
Command>                                                                   Scroll> Csr
ZZSGFLT2
                                     F1=Help     F3=Apply     s10/s11=UNDO/REDO
Display:      ALL          ALL/MORE/LESS
Record Type:  TRACK
.-------------------------------------------------------.                  18 Rows
AND  (  Lev  Name                    Fmt  Pic         ROp  Value                    )
/OR                               +                 +                          +
     _   2  PERSISTENT-ID           AN   X(016)      =                          _    0001
AND  _   2  TRACK-NUM               ZD   9(003)      =    1                     _    0002
AND  _   2  TRACK-ID                ZD   9(004)      =                          _    0003
AND  _   2  NAME                    AN   X(120)      =                          _    0004
AND  (   2  ARTIST                  AN   X(070)      <<   'Prince'              _    0005
AND  _   2  ARTIST                  AN   X(070)      <<   'Springsteen'         )    0006
AND  _   2  ALBUM                   AN   X(070)      =                          _    0007
AND  _   2  TOTAL-TIME              FB   9(007)      =                          _    0008
AND  _   2  FILE-SIZE               FB   9(009)      =                          _    0009
--------------------------- 6 row(s) excluded ---------------------------       0010
AND  _   2  RELEASE-DATE            AN   X(020)      =                          _    0016
AND  _   2  DATE-ADDED              AN   X(020)      =                          _    0017
AND  _   2  DATE-MODIFIED           AN   X(020)      =                          _    0018
*** End of Data ***                                                                  0019
```

Figure 29. SDE Filter Records Panel.

The SDE Filter Records panel (ZZSGFLT2) is an interactive panel window, opened on executing either of the following:

1. Primary command FILTER (FILT) with no parameters. In this case the default record type is used.
2. Line (prefix) command *FLT* in the prefix area of a record of the required record type.

This panel is used to filter the display of records by executing a WHERE (ALL), MORE or LESS command that is generated using a panel in which the users enters selection criteria against a table of fields that comprise the focus record-type.

The functionality is equivalent to use of the FIND/ EXCLUDE features, but with the added advantage of filtering records based on possibly multiple, complex (parenthesised) selection criteria.

ALL
Display only matching record(s)

MORE
Additionally display matching record(s)

LESS
Exclude matching record(s)

Panel Input Fields

The dialog panel contains header fields to:

- Define the function (ALL/MORE/LESS)
- Display the name of the record type

Where formatting of data using a structure/copybook overlay is in effect, the header fields are followed by a table of all data elements for the applied record type.

If formatting is not applied then the header fields are followed by a table of sample conditions which may be usefully applied to the "UnMapped" record.

These include conditions such as:

```
Record                  << "My Value"
```

which selects records containing (**<<**) a given value.

Other samples for unformatted data include testing a sub-string of the record e.g.

```
substr (Record,101,5)   =  c'MyVal'
```

The final sample allows the user to select records based on their record length e.g.

```
LRECL                   >  256
```

The samples for unformatted record are provided as a template only and the condition's left-hand term (**Name** field) should be modified by the user to suit their requirements as well as the right-hand term (**Value** field)

The panel contains a table where each row represents a sub-expression of the single ALL/MORE/LESS (WHERE expression) command generated. Each generated sub-expression is based on values referenced by one or more field names defined in the record structure (or functions applied to the UnMapped record).

By default, the table contains one row for every level of nested field defined within the record structure. These entries may be duplicated, copied or re-ordered before generating the sub-expression. Note that table entries that do not have a test value will **not** be included as part of the generated sub-expression.

The sub-expressions are separated by logical operator AND or OR, where AND is higher in the order of operator precedence than OR. i.e. A AND B OR C is equivalent to (A AND B) OR C. To filter on fields A AND (B OR C), then two rows should exist for the same record structure in the **FILTER (formatted) - INCLUDE**/**EXCLUDE record-types** table so that one tests fields A AND B and the other tests fields A AND C.

Standard FileKit table editing techniques should be used to update, move, copy and exclude table row entries for each filter sub-expression.

Each table row contains input fields for logical operator (AND/OR), optional opening parentheses "(", relational operator (ROp), the value against which the field will be tested and optional closing parentheses ")". Null or invalid enties may be entered in the **AND**/**OR** and **ROp** fields to display and select permitted entries for these fields.

All other fields are output (non-updatable) fields providing useful information about the field to be tested (i.e. the field's level of nesting, its name, data type (format) and, if defined, its picture definition.)

Since logical operators AND and OR are dyadic, the **AND**/**OR** field of the first table row (the first term of the expression) is always blank and cannot be updated.

<PF5> and <PF6> may be used to alternate between display of only selected rows and all rows in the table respectively. This has no effect on the generated filter, but is useful as a visual aid.

The user can display the "form" view of any individual row using the ZOOM command (assigned to <PF2> by default.) As well as providing additional helpful comment information, the single record view allows the user to use the panel EXPAND feature to enter a Value field entry that is longer than the provided input area.

Pressing <PF3> to exit the zoomed view of the panel, will update the filter sub-expression table row and return to the multi-record view of the table.

Press <PF3> again to execute the ALL/MORE/LESS command.

Alternatively, type the **CMX** command to display the generated command in a temporary Text-Edit window, ready for execution using the ACTION feature.

Header fields:

`Display: ALL/MORE/LESS`

`ALL`

Display only matching record(s)

`MORE`

Additionally display matching record(s)

`LESS`

Exclude matching record(s)

Enter blank for a selectable list.

`Record Type:`

The name of the record type for which the data element selection is to be made. It is the focus record type when the dialog was envoked. This field is set automatically and is not enterable.

Table fields:

The dialog panel contains a scrollable table of all the data elements in the record type.

`Logical Operator> AND|OR`

Logical operator that identifies the relationship of this field test sub-expression with the previously specified sub-expression. If this is the first, then this field is ignored. Permissible operators are **AND** or **OR**.

`Opening Bracket(s)>`

One or more leading "(" that the user may provide in order to specify parenthesised expressions.

Space is provided for only one "(" in the table view of the selection criteria panel. To specify more than one "(", first use the **"ZOOM"** key **(Shift-F4)** to display a form panel corresponding to the focus table row.

The form panel provides larger entry fields for many of the table columns.

`Field Details:`

Informational output fields that describe the formatted field.

`Level:`

Level of nesting below the first level (GROUP or STRUCTURE) field.

`Name:`

Field name.

`Data Type:`

The field's defined data type. See *"SDE Data Types"* for details of supported data types and their abbreviated names.

`Picture:`

If the source field is defined with a COBOL picture string, its representation is displayed in this field.

`Relational Operator>`

Identifies the type of test to be performed on the field data. Enter blank in this field to display a selectable list of supported relational operator symbols and a brief description of each.

`Value>`

Specifies the literal string term against which the field data will be tested.

Values may be specified using standard Expression Terms.

To specify values longer than the available entry field length, first use the **"ZOOM"** key **(Shift-F4)** to display a form panel corresponding to the focus table row.

The form panel provides a larger entry field for the Value. If this is still inadequate, you may press the "EXPAND" key (Shift-F2) with your cursor in the form entry field in order to open a Text-Edit style window providing for values of up to 256 bytes.

`Closing Bracket(s)>`

One or more leading ")" that the user may provide in order to specify parenthesised expressions.

Space is provided for only one ")" in the table view of the selection criteria panel. To specify more than one ")", first use the **"ZOOM"** key **(Shift-F4)** to display a form panel corresponding to the focus table row.

The form panel provides larger entry fields for many of the table columns.

Panel Primary Commands

Commands with a specific effect on the FILTER dialog are:

`END`

End the dialog and generate and execute any record selection made by the dialog.
Set on Function Key **F3** by default.

`CANCEL (CAN)`

End the dialog without making any record selection.

`CMX`

End the dialog then display the generated command in a temporary Text-Edit window, ready for execution using the ACTION feature.

General table edit commands are also available. For example:

`UNDO`

Undo the last change to the table.
Set on Function Key **F22** by default.

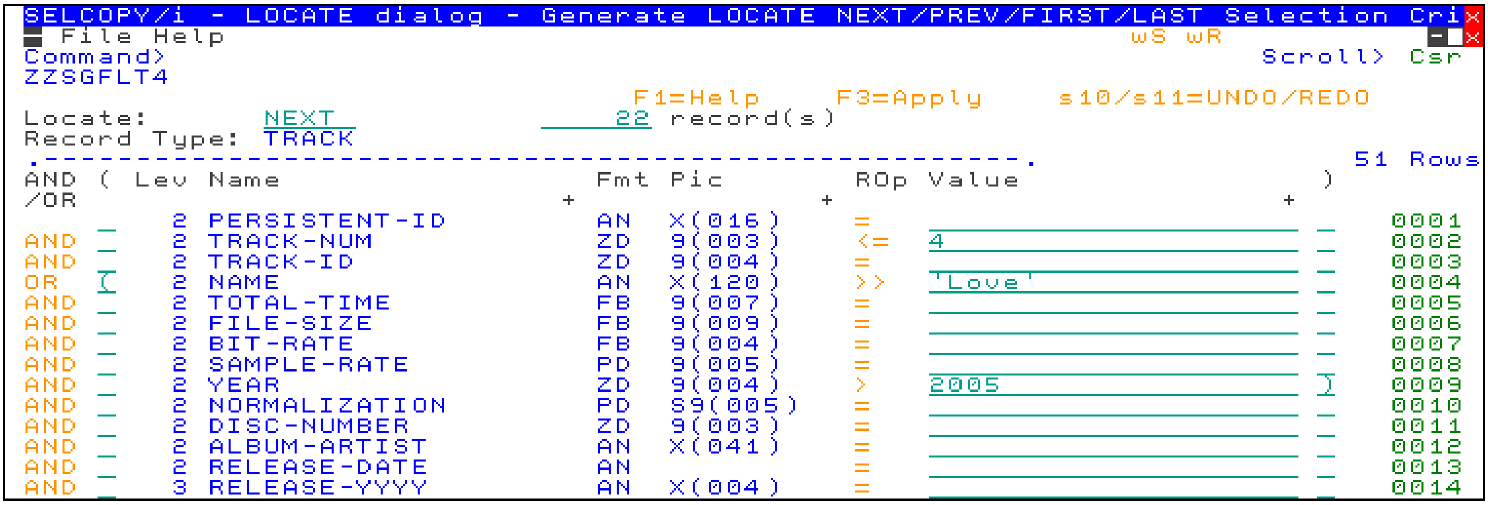
`REDO`

Redo the last change to the table that was undone.
Set on Function Key **F23** by default.

`UP | DOWN | LEFT | RIGHT | TOP | BOTTOM`

Table scrolling commands.

SDE LOCATE Records Panel



```
SELCOPY/i - LOCATE dialog - Generate LOCATE NEXT/PREV/FIRST/LAST Selection Cri
  File Help                                                  wS wR
Command>                                                         Scroll> Csr
ZZSGFLT4
                                    F1=Help     F3=Apply     s10/s11=UNDO/REDO
Locate:        NEXT           22 record(s)
Record Type:   TRACK
.-----------------------------------------------------------.           51 Rows
AND  (  Lev Name                  Fmt  Pic        ROp Value                 )
/OR                             +               +                       +
     _   2  PERSISTENT-ID         AN   X(016)     =                        _  0001
AND  _   2  TRACK-NUM             ZD   9(003)     <=  4                    _  0002
AND  _   2  TRACK-ID              ZD   9(004)     =                        _  0003
OR   (   2  NAME                  AN   X(120)     >>  'Love'               _  0004
AND  _   2  TOTAL-TIME            FB   9(007)     =                        _  0005
AND  _   2  FILE-SIZE             FB   9(009)     =                        _  0006
AND  _   2  BIT-RATE              FB   9(004)     =                        _  0007
AND  _   2  SAMPLE-RATE           PD   9(005)     =                        _  0008
AND  _   2  YEAR                  ZD   9(004)     >   2005                 )  0009
AND  _   2  NORMALIZATION         PD   S9(005)    =                        _  0010
AND  _   2  DISC-NUMBER           ZD   9(003)     =                        _  0011
AND  _   2  ALBUM-ARTIST          AN   X(041)     =                        _  0012
AND  _   2  RELEASE-DATE          AN              =                        _  0013
AND  _   3  RELEASE-YYYY          AN   X(004)     =                        _  0014
```

Figure 30. SDE Locate Records Panel.

The SDE Locate Records panel (ZZSGFLT4) is an interactive panel window, opened on primary command LOCATE with no parameters. In this case the default record type is used.

This panel is used to locate records by executing a LOCATE command that is generated using a panel in which the user enters selection criteria against a table of fields that comprise the focus record-type.

The functionality is equivalent to use of the FIND feature but with the added advantage of locating records based on possibly multiple, complex (parenthesised) selection criteria.

Panel Input Fields

The dialog panel contains header fields to:

- Define the function (LOCATE NEXT/PREV/FIRST/LAST)
- Define the number of records to locate
- Display the name of the record type

Where formatting of data using a structure/copybook overlay is in effect, the header fields are followed by a table of all data elements for the applied record type.

If formatting is not applied then the header fields are followed by a table of sample conditions which may be usefully applied to the "UnMapped" record.

These include conditions such as:

```
Record                  << "My Value"
```

which selects records containing (**<<**) a given value.

Other samples for unformatted data include testing a sub-string of the record e.g.

```
substr (Record,101,5)   =  c'MyVal'
```

The final sample allows the user to select records based on their record length e.g.

```
LRECL                   >  256
```

The samples for unformatted record are provided as a template only and the condition's left-hand term (**Name** field) should be modified by the user to suit their requirements as well as the right-hand term (**Value** field)

The panel contains a table where each row represents a sub-expression of the single LOCATE (WHERE expression) command generated. Each generated sub-expression is based on values referenced by one or more field names defined in the record structure (or functions applied to the UnMapped record).

By default, the table contains one row for every level of nested field defined within the record structure. These entries may be duplicated, copied or re-ordered before generating the sub-expression. Note that table entries that do not have a test value will **not** be included as part of the generated sub-expression.

The sub-expressions are separated by logical operator AND or OR, where AND is higher in the order of operator precedence than OR. i.e. A AND B OR C is equivalent to (A AND B) OR C. To filter on fields A AND (B OR C), then two rows should exist for the same record structure in the **LOCATE (formatted) - INCLUDE**/**EXCLUDE record-types** table so that one tests fields A AND B and the other tests fields A AND C.

Standard FileKit table editing techniques should be used to update, move, copy and exclude table row entries for each filter sub-expression.

Each table row contains input fields for logical operator (AND/OR), optional opening parentheses "(", relational operator (ROp), the value against which the field will be tested and optional closing parentheses ")". Null or invalid enties may be entered in the **AND**/**OR** and **ROp** fields to display and select permitted entries for these fields.

All other fields are output (non-updatable) fields providing useful information about the field to be tested (i.e. the field's level of nesting, its name, data type (format) and, if defined, its picture definition.)

Since logical operators AND and OR are dyadic, the **AND**/**OR** field of the first table row (the first term of the expression) is always blank and cannot be updated.

<PF5> and <PF6> may be used to alternate between display of only selected rows and all rows in the table respectively. This has no effect on the generated filter, but is useful as a visual aid.

The user can display the "form" view of any individual row using the ZOOM command (assigned to <PF2> by default.) As well as providing additional helpful comment information, the single record view allows the user to use the panel EXPAND feature to enter a Value field entry that is longer than the provided input area.

Pressing <PF3> to exit the zoomed view of the panel, will update the filter sub-expression table row and return to the multi-record view of the table.

Press <PF3> again to execute the LOCATE command.

Alternatively, type the **CMX** command to display the generated command in a temporary Text-Edit window, ready for execution using the ACTION feature.

Header fields:

`Locate: NEXT/PREV/FIRST/LAST`

`NEXT`
Locate next 'nn' matching record(s)

`PREV`
Locate previous 'nn' matching record(s)

`FIRST`
Locate first 'nn' matching record(s)

`LAST`
Locate last 'nn' matching record(s)

Enter blank for a selectable list.

`nnnn record(s)`
The number of records to locate.

Since "located" records become "unexcluded", you may wish to start this dialog following a "X ALL" command to first exclude all records, then specify the number of records (e.g. 10) to see a selection of hits, without processing the whole file (as would occur using the FILTER dialog).

`Record Type:`
The name of the record type for which the data element selection is to be made. It is the focus record type when the dialog was envoked. This field is set automatically and is not enterable.

Table fields:

The dialog panel contains a scrollable table of all the data elements in the record type.

`Logical Operator> AND|OR`
Logical operator that identifies the relationship of this field test sub-expression with the previously specified sub-expression. If this is the first, then this field is ignored. Permissible operators are **AND** or **OR**.

`Opening Bracket(s)>`
One or more leading "(" that the user may provide in order to specify parenthesised expressions.

Space is provided for only one "(" in the table view of the selection criteria panel. To specify more than one "(", first use the **"ZOOM"** key **(Shift-F4)** to display a form panel corresponding to the focus table row.

The form panel provides larger entry fields for many of the table columns.

`Field Details:`
Informational output fields that describe the formatted field.

`Level:`
Level of nesting below the first level (GROUP or STRUCTURE) field.

`Name:`
Field name.

`Data Type:`
The field's defined data type. See *"SDE Data Types"* for details of supported data types and their abbreviated names.

`Picture:`
If the source field is defined with a COBOL picture string, its representation is displayed in this field.

`Relational Operator>`
Identifies the type of test to be performed on the field data. Enter blank in this field to display a selectable list of supported relational operator symbols and a brief description of each.

`Value>`
Specifies the literal string term against which the field data will be tested.

Values may be specified using standard Expression Terms.

To specify values longer than the available entry field length, first use the **"ZOOM"** key **(Shift-F4)** to display a form panel corresponding to the focus table row.

The form panel provides a larger entry field for the Value. If this is still inadequate, you may press the "EXPAND" key (Shift-F2) with your cursor in the form entry field in order to open a Text-Edit style window providing for values of up to 256 bytes.

`Closing Bracket(s)>`
One or more leading ")" that the user may provide in order to specify parenthesised expressions.

Space is provided for only one ")" in the table view of the selection criteria panel. To specify more than one ")", first use the **"ZOOM"** key **(Shift-F4)** to display a form panel corresponding to the focus table row.

The form panel provides larger entry fields for many of the table columns.

Panel Primary Commands

Commands with a specific effect on the LOCATE dialog are:

`END`
End the dialog and generate and execute the LOCATE command.
Set on Function Key **F3** by default.

`CANCEL (CAN)`
End the dialog without executing a LOCATE command.

`CMX`
End the dialog then display the generated command in a temporary Text-Edit window, ready for execution using the ACTION feature.

General table edit commands are also available. For example:

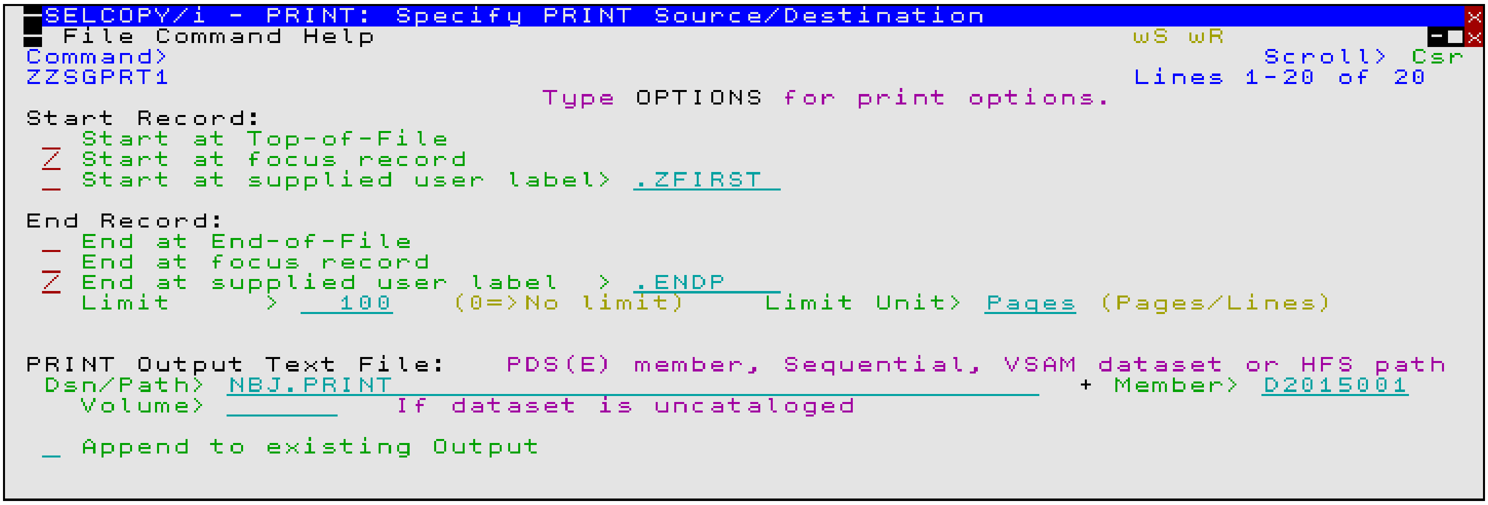
`UNDO`
Undo the last change to the table.
Set on Function Key **F22** by default.

`REDO`
Redo the last change to the table that was undone.
Set on Function Key **F23** by default.

`UP | DOWN | LEFT | RIGHT | TOP | BOTTOM`
Table scrolling commands.

SDE PRINT File Panel

The **PRINT** panel may be displayed by typing the **PRINT** primary command without parameters from an existing Structured Data Browse/Edit session.



```
 SELCOPY/i - PRINT: Specify PRINT Source/Destination
  File  Command  Help                                          wS wR
Command>                                                            Scroll> Csr
ZZSGPRT1                                                        Lines 1-20 of 20
                               Type OPTIONS for print options.
Start Record:
    Start at Top-of-File
  Z Start at focus record
  _ Start at supplied user label> .ZFIRST

End Record:
  _ End at End-of-File
    End at focus record
  Z End at supplied user label  > .ENDP
    Limit       >   100    (0=>No limit)    Limit Unit> Pages  (Pages/Lines)


PRINT Output Text File:    PDS(E) member, Sequential, VSAM dataset or HFS path
 Dsn/Path> NBJ.PRINT                                      + Member> D2015001
   Volume>              If dataset is uncataloged

  _ Append to existing Output
```

Figure 31. SDE Print File Panel.

The PRINT process will honour any SELECT issued prior to entry in order to restrict the fields generated and will operate only on un-excluded records that are currently in VIEW.

The PRINT process is also sensitive to the current state of the following **SET** options.

ASCII	HEX	REFERENCE	TYPE
COLATTRIBUTES	MAPPING	RESERVED	UNNAMED
COLWIDTH	PREFIX	SCALE	VIEW
FORMAT	RECINFO	SHADOW	

The PRINT process will be executed in the foreground.

Menu Bar Items

File

The File drop-down menu contains the single item "Exit" which simply closes the panel window. Note that, unlike CANCEL, CLOSE will save field values entered in the panel so that they may be redisplayed the next time the panel is opened.

Command

Generate the PRINT command line syntax and display it in a temporary CMX file text edit view. This command may be executed using ACTION point-and-shoot execution <F16> or copied into the user's HOME file and saved for future execution.

The user has the opportunity to edit the command prior to its execution and/or copying it to the home (CMX) command centre for future reference and re-execution.

Help

Display help for this panel view.

Panel Input Fields

Start Record:

Start at Top-of-File:

XML generation will start from the top of file.

Start at focus record

XML generation will start from the focus (cursor) record.

Start at supplied user label>

XML generation will start from the record specified by the supplied user label.

End Record:

End at End-of-File:
XML generation will continue until end of file.

End at focus record
XML generation will continue up to and including the focus (cursor) record.

End at supplied user label>
XML generation will continue up to and including the record specified by the supplied user label.

Limit>
Specifies the number of pages/lines of print output to be generated before the process is to be automatically terminated.

Limit Unit>

Pages
Limit specifies a number of pages.

Lines
Limit specifies a number of lines.

PRINT Output Text File:
Input fields which together identify a single output sequential, VSAM or PDS/PDSE library data set, HFS file or PDS/PDSE library member.

DSN/Path>
Identifies the fully qualified data set name or an absolute or relative HFS file path.

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

If a DSN is specified for a data set that does not already exist, a prompt data set dialog will be opened to allocate the new output file.

Member>
If the DSN/Path> field contains the DSN of a PDS/PDSE library, then this field may specify the name of a new or existing member within that library.

A selectable list of members will be presented if the entered value contains wildcards characters "*" (asterisk) or "%" (percent), or is blanked out.

Volume>
Specifies the name of the output data set volume. This is required only if output is to an uncataloged data set.

Append to existing Output
Select this option if the generated records are to be appended to existing records in the output data set.

PRINT Source and Options

Table-Mode (VFMT/CHAR) Print Options:

Order>
Where printed lines are longer than the output page width, then this field value determines the action taken on the overflowing line data.

Unless T (Truncate) is selected, overflow line data is printed on as many continuation pages as is required to accomodate the longest print line in the current set of print lines. Note that a set of print lines is the number of SDE view lines that may be displayed for the specified page depth.

A - Across Then Down
Indicates that continuation pages for the current set of lines are to be printed before scrolling down to the next set of print lines. i.e. Print all pages scrolling across to the right then scroll down to the first page of the next set of lines and repeat the process.

This option corresponds to parameter ACROSSTHENDOWN of the FileKit primary command PRINT.

Because only as many pages are printed to accomodate the longest line in the current set of print lines, the number of pages printed scrolling across may be different for each set of print lines.

D - Down Then Across
Indicates that the first page of all sets of print lines are to be printed before printing the next (continuation) page of all sets of print lines and repeating this process until all contuation pages have been printed. i.e. Print pages scrolling down, scroll back to the first set of print lines, scroll across to the right once and repeat the process.

This option corresponds to parameter DOWNTHENACROSS of the FileKit primary command PRINT and is suitable only when formatted records of a single record-type are selected for display. (See "Select Record-types to Print" if the SDO used contains multiple record-type definitions.)

If this option is selected and records of different record-types are displayed concurrently, then the print will fail with the following message:

```
ZZSD472E The DOWNTHENACROSS format of the PRINT command requires
         that only one record type is visible.
```

Because only one record-type is used, the number of pages printed scrolling across will be the same for each set of print lines.

T - Truncate

Indicates that no continuation pages are to be printed so truncating the print lines. Only the first page of all sets of print lines will be printed.

This option corresponds to parameter TRUNC of the FileKit primary command PRINT.

Page Geometry:

Page Width>

Set the print output page width (number of columns).

A page width value of 0 (zero) will use the default page width of 133 for SYSOUT and HFS file output, and the maximum record length (LRECL) for an output data set or library member.

Note that, for FMT and UNFMT printed output, page width is restricted to a maximum of 255 print columns.

Page Depth>

Set the print output page depth (number of lines).

The page depth value includes the 5 Print header lines so that the number of lines of data printed will be 5 less than the page depth value.

Primary Commands

The following primary commands are supported.

OPTIONS (OPT)	Display PRINT processing options panel.

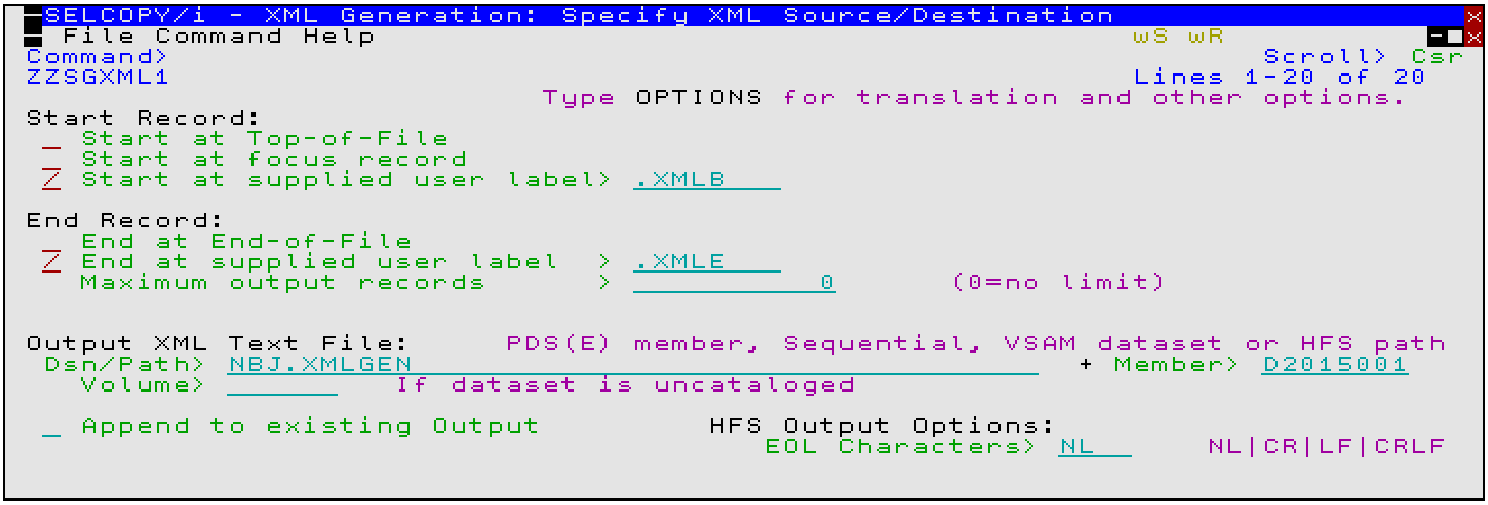
Function Keys

In addition to the standard interactive panel key assignments for scrolling and navigation, the PRINT panel supports the following:

F6	OPTIONS	Display PRINT processing options panel.

SDE XML Generation Panel

The **XML Generation** panel may be displayed by typing XMLGEN (XML) primary command without parameters from an existing Structured Data Browse/Edit session.



```
 SELCOPY/i - XML Generation: Specify XML Source/Destination
  File  Command  Help                                          wS wR
Command>                                                            Scroll> Csr
ZZSGXML1                                                      Lines 1-20 of 20
                           Type OPTIONS for translation and other options.
Start Record:
 _ Start at Top-of-File
 _ Start at focus record
 Z Start at supplied user label> .XMLB

End Record:
 _ End at End-of-File
 Z End at supplied user label  > .XMLE
   Maximum output records      >             0      (0=no limit)


Output XML Text File:      PDS(E) member, Sequential, VSAM dataset or HFS path
 Dsn/Path> NBJ.XMLGEN                                  + Member> D2015001
   Volume> ______    If dataset is uncataloged

 _ Append to existing Output          HFS Output Options:
                                          EOL Characters> NL__    NL|CR|LF|CRLF
```

Figure 32. SDE XML Generation Panel.

This panel allows the user to produce an exportable copy of selected records from the structured dataset as extended markup language (XML) text consisting of XML tags and tag content. The tag names correspond to the field names of the copybook/structure applied to the input dataset and the tag content to the field values expressed in character format.

The XMLGEN process will honour any SELECT issued prior to entry in order to restrict the fields generated and will operate only on un-excluded records that are currently in VIEW.

The XMLGEN process will be executed in the foreground.

Menu Bar Items

File

The File drop-down menu contains the single item "Exit" which simply closes the panel window. Note that, unlike CANCEL, CLOSE will save field values entered in the panel so that they may be redisplayed the next time the panel is opened.

Command

Generate the XMLGEN command line syntax and display it in a temporary CMX file text edit view. This command may be executed using ACTION point-and-shoot execution <F16> or copied into the user's HOME file and saved for future execution.

The user has the opportunity to edit the command prior to its execution and/or copying it to the home (CMX) command centre for future reference and re-execution.

Help

Display help for this panel view.

XML Source/Destination

Start Record:

Start at Top-of-File:
XML generation will start from the top of file.

Start at focus record
XML generation will start from the focus (cursor) record.

Start at supplied user label>
XML generation will start from the record specified by the supplied user label.

End Record:

End at End-of-File:
XML generation will continue until end of file.

End at supplied user label>
XML generation will continue up to and including the record specified by the supplied user label.

Maximum output records>
Specifies the number of selected records for which XML will be generated, at which the process will be automatically terminated.

Output XML Text File:
Input fields which together identify a single output sequential, VSAM or PDS/PDSE library data set, HFS file or PDS/PDSE library member.

DSN/Path>
Identifies the fully qualified data set name or an absolute or relative HFS file path.

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

If a DSN is specified for a data set that does not already exist, a prompt data set dialog will be opened to allocate the new output file.

Member>
If the DSN/Path> field contains the DSN of a PDS/PDSE library, then this field may specify the name of a new or existing member within that library.

A selectable list of members will be presented if the entered value contains wildcards characters "*" (asterisk) or "%" (percent), or is blanked out.

Volume>
Specifies the name of the output data set volume. This is required only if output is to an uncataloged data set.

Append to existing Output
Select this option if the generated records are to be appended to existing records in the output data set.

HFS Output Options:

EOL Characters>
Choose from one of the following End-Of-Line character combinations:

NL	X'15'	New Line.
CR	X'0D'	Carriage Return.
LF	X'0A'	Line Feed.
CRLF	X'0D0A'	Carriage Return + Line Feed.

XML Source and Options

Translation Options:

Non-printable chars>
Since XMLGEN output is supposed to be in a portable character format, this option is required to specify how non-printable characters are dealt with.

HEX
If a character field contains a non-printable character output the whole field in hex string format. For example a character field length 4 containing X'FFFFFFFF' would have its value represented as

```
X&apos;FFFFFFFF&apos;
```

If a non-printable character is found in a character field and this option is in effect the field XML tag will have the attribute **NONPRINT_CHAR="HEX"**.

ASIS
No special action is taken. All input bytes are copied to the output XML tag value. If a non-printable character is found in a character field and this option is in effect the field XML tag will have the attribute **NONPRINT_CHAR="ASIS"**.

SKIP
The field value is skipped. If a non-printable character is found in a character field and this option is in effect the field XML tag will have the attribute **NONPRINT_CHAR="SKIP"** and no content.

REPLACE
Each non-printable character in a character field is replaced with the specified value. If a non-printable character is found in a character field and this option is in effect the field XML tag will have the attribute **NONPRINT_CHAR="REPLACE"**. The default replace character is the period (full-stop) ".". The replacement character can be specified as:

with>

character

The actual replacement character. If no character is specified then period (.) is assumed. If the character is a lower case letter it will be changed to upper case.

'character' | *"character"*

The actual replacement character in single quotes (apostrophes) or double quotes. If the character is a lower case letter it will be translated to upper case.

c'*character*' | c"*character*"

The actual replacement character in single quotes (apostrophes) or double quotes with a **c** or **C** prefix. No case translation takes place.

x'*hex_value*' | x"*hex_value*"

The replacement character specified as a hexadecimal value.

HEX

This keyword does not represent a replacement character but requests that any substring of non-printable characters found in a character field is replaced with its value in hexadecimal format inside **<HEX> </HEX>** tags. For example a character field length 4 containing X'C1C2FFC3' would have its value represented as

```
AB<HEX>X&apos;FF&apos;</HEX>C
```

In this case the field XML tag wiill have the attribute **NONPRINT_CHAR="REPLACE_HEX"**.

XML special chars >

XML specifies 5 characters as of special syntactical significance. These characters are used to delimit XML constructs and must not appear as themselves in tag values. XML provides an escape sequence (character reference) which can be used to represent these special characters in tag values.

The XML special characters are:

Character	Name	Escape sequence
<	Less than	&lt;
>	Greater than	&gt;
'	Apostrophe	&apos;
"	Double quote	&quot;
&	Ampersand	&amp;

This option provides a way of dealing with any of the XML special characters found in character data fields.

ESCAPE

If a character field contains an XML special character replace it with its XML escape sequence. For example a character field length 4 containing 'A<>B' would have its value represented as:

```
A&lt;&gt;B
```

If an XML special character is found in a character field and this option is in effect the field XML tag will have the attribute **SPECIAL_CHAR="ESCAPE"**.

HEX

If a character field contains an XML special character output the whole field in hex string format. For example a character field length 4 containing 'A<>B' would have its value represented as

```
X&apos;C14C6EC2&apos;
```

If an XML special character is found in a character field and this option is in effect the field XML tag will have the attribute **SPECIAL_CHAR="HEX"**.

CDATA

If a character field contains an XML special character output the whole field asis in an XML character data (CDATA) section. CDATA sections in an XML document represent unparsed character data. For example a character field length 4 containing 'A<>B' would have its value represented as

```
<![CDATA[A<>B]]>
```

If an XML special character is found in a character field and this option is in effect the field XML tag will have the attribute **SPECIAL_CHAR="CDATA"**.

REPLACE

Each XML special character in a character field is replaced with the specified value. If an XML special character is found in a character field and this option is in effect the field XML tag will have the attribute **SPECIAL_CHAR="REPLACE"**. The default replace character is the underscore "_". The replacement character can be specified as:

with>

character

The actual replacement character. If no character follows REPLACE then underscore (_) is assumed. If the character is a lower case letter it will be changed to upper case.

'character' | *"character"*

The actual replacement character in single quotes (apostrophes) or double quotes. If the character is a lower case letter it will be translated to upper case.

c'*character*' | c"*character*"

The actual replacement character in single quotes (apostrophes) or double quotes with a **c** or **C** prefix. No case translation takes place.

x'*hex_value*' | x"*hex_value*"

The replacement character specified as a hexadecimal value.

HEX

This keyword does not represent a replacement character but requests that any substring of XML special characters found in a character field is replaced with its value in hexadecimal format inside **<HEX> </HEX>** tags. For example a character field length 4 containing 'A<>B' would have its value represented as:

```
A<HEX>X&apos;4C6E&apos;</HEX>B
```

In this case the field XML tag wiill have the attribute **SPECIAL_CHAR="REPLACE_HEX"**.

Invalid data values>

Non-character fields in structured data files may have invalid values which cannot be converted to character format. For example, a field defined as containing packed decimal (COBOL COMP-3) data may not contain a valid packed decimal value. This option provides a way of specifying how such fields are represented in the XML output.

HEX

If a non-character field contains an invalid data value output the whole field in hex string format. For example a packed decimal field length 4 containing X'00000000' would have its value represented as

```
X&apos;00000000&apos;
```

If an invalid value is found in a non-character field and this option is in effect the field XML tag will have the attribute **INVALID_DATA="HEX"**.

SKIP

The field value is skipped. If an invalid data value is found in a non-character field and this option is in effect the field XML tag will have the attribute **INVALID_DATA="SKIP"** and no content.

REPLACE

The invalid field data value is replaced with the specified value. If an invalid field data value is found in a non-character field and this option is in effect the field XML tag will have the attribute **INVALID_DATA="REPLACE"**. The default replace character is the asterisk "*". The replacement character can be specified as:

with>

character

The actual replacement character. If no character follows REPLACE then asterisk (*) is assumed. If the character is a lower case letter it will be changed to upper case.

'character' | *"character"*

The actual replacement character in single quotes (apostrophes) or double quotes. If the character is a lower case letter it will be translated to upper case.

c'*character*' | c"*character*"

The actual replacement character in single quotes (apostrophes) or double quotes with a **c** or **C** prefix. No case translation takes place.

x'*hex_value*' | x"*hex_value*"

The replacement character specified as a hexadecimal value.

CCSID Conversion>

Since the purpose of XMLGEN is to produce a portable export version of the data in a z/OS mainframe structured data file, and the output is character data, the coded character set identifiers (CCSIDs) of the input, output and of the XMLGEN internal constants themselves are of significance.

Even if the input and output is coded in an EBCDIC CCSID, these may differ, and both may differ from the CCSID of the XMLGEN internal constants. Since some of the special characters used in XML have different code points in different EBCDIC CCSIDs (for example square brackets) these must be dealt with consistently to produce correct XML output.

XMLGEN uses the z/OS character conversion support supplied by IBM modules CUNLINFO (for obtaining CCSID information) and CUNLCNV (for character conversion from one CCSID to another).

The internal XMLGEN CCSID (that of the constants used to build the XML syntax) is CCSID 285 (EBCDIC, SBCS UNITED KINGDOM).

XMLGEN assumes a default CCSID as follows:

Interactive
When executed interactively XMLGEN uses as default input CCSID that of the user's 3270 terminal.

Batch
When executed in batch XMLGEN uses as default input CCSID the value of the INI file variable **SDE.CCSID**. This variable is set automatically to the user's 3270 terminal CCSID (if not already set) during an interactive session. It can also be set using the structured data SET CCSID command.

Avaliable options are as follows:

`NONE`
The XML output dataset is produced using the default CCSID and the input dataset character fields are assumed to be in the same CCSID. The internal XMLGEN constants are converted from internal CCSID 285 to the default CCSID.

`ASCII`
Convert the output to ASCII. This is equivalent to specifying **CONVERT TO 819**. CCSID 819 is ISO 8859-1 ASCII.

`UNICODE`
Convert the output to UNICODE (UTF-16). This is equivalent to specifying **CONVERT TO 1200**. CCSID 1200 is the IBM bigendian UTF-16 CCSID which is automatically transformaed to the most recent UTF-16 standard.

`CONV`

`from>`

`from_ccsid`
The input character data fields are converted from this CCSID. If not supplied the default input CCSID is used.

`to>`

`to_ccsid`
The CCSID of the output XML text dataset. Internal XMLGEN character literals and input character data fields (and HFS line end characters if used) are converted to this CCSID.

`Miscellaneous Options:`

`View option>`
When XMLGEN is run interactively this option allows the user to request to view the output when the process completes.

`BROWSE`
Browse the output XML dataset.

`EDIT`
Edit the output XML dataset using the FileKit text editor.

`NOVIEW`
Do not view the output XML dataset. This option is forced when run in batch.

`Indentation value>`
Nested output XML tags corresponding to the hierarchy of group and elementary data fields in the input structure are indented by a default of one space for each data item level. This option allows the specification of a different indentation value.

`Output all field redefinitions:`
If the structure defined for the input dataset contains redefined fields this option controls whether the field redefinitions are output.

`Output all unnamed (FILLER) fields:`
If the structure defined for the input dataset contains unnamed or FILLER fields this option controls whether these fields are output.

`Output all fields as elementary:`
This option controls whether elements of a group field are output as children of their parent group tag. If activated then group field tags are not included and all elements are output at the top level within the record-type.

`Split long XML records:`
For each elementary input field XMLGEN builds one output record containing the field start tag, the field value (possibly with embedded HEX tags and special character escape sequences), and the field end tag. Depending on the options chosen and the nature of the input data, relatively long output records may result. If an output record is longer than the allocated logical record length of the output dataset this option controls how XMLGEN deals with the long output record.

`Activated`
Split the output record breaking it up into as many logical records as necessary. Records are split at the logical record length irrespective of the record content.

Not activated

Do not split the output record. Rather than truncate the output record XMLGEN terminates with an error message. This is the default.

Suppress output comment block:

This option controls whether an XML style comment block is generated at the top of the output dataset. This contains information about the host operating system, the id of the creator of the output file and the creation date and time, and details of any character conversion performed on the output character data.

Primary Commands

The following primary commands are supported.

OPTIONS (OPT)	Display XML processing options panel.

Function Keys

In addition to the standard interactive panel key assignments for scrolling and navigation, the XMLGEN panel supports the following:

F6	OPTIONS	Display XML processing options panel.

SDE CSV Generation Panel

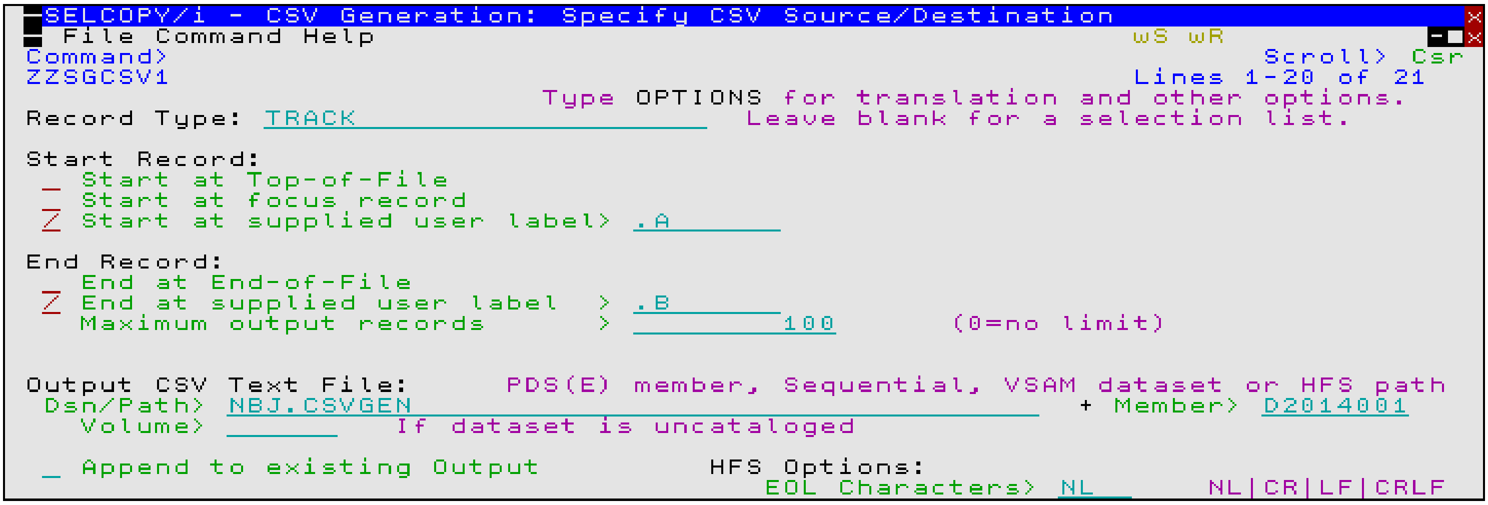
The **CSV Generation** panel may be displayed by typing **CSVGEN (CSV)** primary command without parameters from an existing Structured Data Browse/Edit session.

This panel allows the user to produce an exportable copy of selected records from the structured dataset as comma separated variable (CSV) text.

Where the current structure maps more than one record-type, the user must select a single record-type, with records belonging to all other record-types being bypassed. The initial setting for record-type is primed to that of the focus record when the CSVGEN command is issued.

The CSVGEN process will honour any **SELECT** issued prior to entry in order to restrict the fields generated from the chosen record-type, and will operate only on un-excluded records that are currently in **VIEW**.

The CSVGEN process will be executed in the foreground.



```
 SELCOPY/i - CSV Generation: Specify CSV Source/Destination
  File Command Help                                              wS wR
Command>                                                           Scroll> Csr
ZZSGCSV1                                                    Lines 1-20 of 21
                            Type OPTIONS for translation and other options.
Record Type: TRACK                     Leave blank for a selection list.

Start Record:
 _ Start at Top-of-File
 _ Start at focus record
 Z Start at supplied user label> .A

End Record:
   End at End-of-File
 Z End at supplied user label  > .B
   Maximum output records      >         100      (0=no limit)


Output CSV Text File:     PDS(E) member, Sequential, VSAM dataset or HFS path
 Dsn/Path> NBJ.CSVGEN                                + Member> D2014001
   Volume>           If dataset is uncataloged

 _ Append to existing Output             HFS Options:
                                           EOL Characters> NL      NL|CR|LF|CRLF
```

Figure 33. SDE CSV Generation Panel.

Menu Bar Items

File

The File drop-down menu contains the single item "Exit" which simply closes the panel window. Note that, unlike CANCEL, CLOSE will save field values entered in the panel so that they may be redisplayed the next time the panel is opened.

Command

Generate the CSVGEN command line syntax and display it in a temporary CMX file text edit view. This command may be executed using ACTION point-and-shoot execution <F16> or copied into the user's HOME file and saved for future execution.

The user has the opportunity to edit the command prior to its execution and/or copying it to the home (CMX) command centre for future reference and re-execution.

Help

Display help for this panel view.

CSV Source/Destination

RecType>

Identifies the name of a record-type record mapping defined within the structure. CSV will be generated only for records of this record-type.

The SDE: Select Record-Type panel will be automatically opened to display a selectable list of record-types if the structure contains multiple record types. Otherwise the single record-type will be inserted automatically.

Start Record:

Start at Top-of-File:

CSV generation will start from the top of file.

Start at focus record

CSV generation will start from the focus (cursor) record.

Start at supplied user label>

CSV generation will start from the record specified by the supplied user label.

End Record:

End at End-of-File:

CSV generation will continue until end of file.

End at supplied user label>

CSV generation will continue up to and including the record specified by the supplied user label.

Maximum output records>

Specifies the number of selected records for which CSV will be generated, at which the process will be automatically terminated.

Output CSV Text File:

Input fields which together identify a single output sequential, VSAM or PDS/PDSE library data set, HFS file or PDS/PDSE library member.

DSN/Path>

Identifies the fully qualified data set name or an absolute or relative HFS file path.

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

If a DSN is specified for a data set that does not already exist, a prompt data set dialog will be opened to allocate the new output file.

Member>

If the DSN/Path> field contains the DSN of a PDS/PDSE library, then this field may specify the name of a new or existing member within that library.

A selectable list of members will be presented if the entered value contains wildcards characters "*" (asterisk) or "%" (percent), or is blanked out.

Volume>

Specifies the name of the output data set volume. This is required only if output is to an uncataloged data set.

Append to existing Output

Select this option if the generated records are to be appended to existing records in the output data set.

HFS Output Options:

EOL Characters>

Choose from one of the following End-Of-Line character combinations:

NL	X'15'	New Line.
CR	X'0D'	Carriage Return.
LF	X'0A'	Line Feed.
CRLF	X'0D0A'	Carriage Return + Line Feed.

CSV Source and Options

Options:

This screen is displayed by typing the **OPTIONS (OPT)** primary command, which by default is on Function Key **F6**.

Separator character >

By default CSVGEN produces **comma** separated variables, but this option allows the user to specify any other character as the variable separator. The option may be specified as a single quoted or unquoted character literal, or as a hex value using **X'nn'** notation.

Quoted strings>

The **QUOTE** option controls when variable values are to be enclosed in double-quotes.

CHARacter	Quote character fields values only (default).
ALL	Quote all field values.
REQuired	Quote only if required i.e. if value contains a double-quote or the separator character.

CCSID Conversion>

Since the purpose of CSVGEN is to produce a portable export version of the data in a z/OS mainframe structured data file, and the output is character data, the coded character set identifiers (CCSIDs) of the input, output and of the CSVGEN internal constants themselves are of significance.

Even if the input and output is coded in an EBCDIC CCSID, these may differ, and both may differ from the CCSID of the CSVGEN internal constants. Since some of the special characters used in CSV have different code points in different EBCDIC CCSIDs (for example square brackets) these must be dealt with consistently to produce correct CSV output.

CSVGEN uses the z/OS character conversion support supplied by IBM modules CUNLINFO (for obtaining CCSID information) and CUNLCNV (for character conversion from one CCSID to another).

The internal CSVGEN CCSID (that of the constants used to build the CSV syntax) is CCSID 285 (EBCDIC, SBCS UNITED KINGDOM).

CSVGEN assumes a default CCSID as follows:

Interactive

When executed interactively CSVGEN uses as default input CCSID that of the user's 3270 terminal.

Batch

When executed in batch CSVGEN uses as default input CCSID the value of the INI file variable **SDE.CCSID**. This variable is set automatically to the user's 3270 terminal CCSID (if not already set) during an interactive session. It can also be set using the structured data SET CCSID command.

Avaliable options are as follows:

NONE

The CSV output dataset is produced using the default CCSID and the input dataset character fields are assumed to be in the same CCSID. The internal CSVGEN constants are converted from internal CCSID 285 to the default CCSID.

ASCII

Convert the output to ASCII. This is equivalent to specifying **CONVERT TO 819**. CCSID 819 is ISO 8859-1 ASCII.

UNICODE

Convert the output to UNICODE (UTF-16). This is equivalent to specifying **CONVERT TO 1200**. CCSID 1200 is the IBM bigendian UTF-16 CCSID which is automatically transformaed to the most recent UTF-16 standard.

CONV

from>

from_ccsid

The input character data fields are converted from this CCSID. If not supplied the default input CCSID is used.

to>

to_ccsid

The CCSID of the output CSV text dataset. Internal CSVGEN character literals and input character data fields (and HFS line end characters if used) are converted to this CCSID.

View option>

When CSVGEN is run interactively this option allows the user to request to view the output when the process completes.

BROWSE

Browse the output CSV dataset.

EDIT

Edit the output CSV dataset using the FileKit text editor.

NOVIEW

Do not view the output CSV dataset. This option is forced when run in batch.

Suppress output column headers record:

This option controls whether a CSV record containing the original field names is generated as the first output record.

Strip trailing blanks

This option controls controls whether trailing blanks are to be stripped from each variable. This option is particularly relevant to fixed length character fields.

Primary Commands

The following primary commands are supported.

OPTIONS (OPT)	Display CSV processing options panel.

Function Keys

In addition to the standard interactive panel key assignments for scrolling and navigation, the CSVGEN panel supports the following:

F6	OPTIONS	Display CSV processing options panel.

SDE JSON Generation Panel

The **JSON Generation** panel may be displayed by typing the JSONGEN (JSON) primary command without parameters from an existing Structured Data Browse/Edit session.

This panel allows the user to produce an exportable copy of a structured dataset as JavaScript Object Notation (JSON) text.

The JSONGEN process will honour any SELECT issued prior to entry in order to restrict the fields generated and will operate only on lines in the view that have not been suppressed or excluded.

The JSONGEN process will be executed in the foreground.

Menu Bar Items

File

The File drop-down menu contains the single item "Exit" which simply closes the panel window. Note that, unlike CANCEL, CLOSE will save field values entered in the panel so that they may be redisplayed the next time the panel is opened.

Command

Generate the JSONGEN command line syntax and display it in a temporary CMX file within a text edit view. This command may be executed immediately, using ACTION point-and-shoot execution on <F16>, or copied into the user's HOME file and saved for future execution.

The user has the opportunity to edit the command prior to its execution and/or copying it to the home (CMX) command centre for future reference and re-execution.

Help

Display help for this panel view.

JSON Source/Destination

Start Record:

Start at Top-of-File:
JSON generation will start from the top of file.

Start at focus record
JSON generation will start from the focus (cursor) record.

Start at supplied user label>
JSON generation will start from the record specified by the supplied user label.

End Record:

End at End-of-File:
JSON generation will continue until end of file.

End at supplied user label>
JSON generation will continue up to and including the record specified by the supplied user label.

Maximum output records>
Specifies the number of selected records for which JSON output will be generated, at which the process will be automatically terminated.

Output JSON Text File:
Input fields which together identify a single output sequential, VSAM or PDS/PDSE library data set, HFS file or PDS/PDSE library member.

DSN/Path>
Identifies the fully qualified data set name or an absolute or relative HFS file path.

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

If a DSN is specified for a data set that does not already exist, a prompt data set dialog will be opened to allocate the new output file.

Member>
If the DSN/Path> field contains the DSN of a PDS/PDSE library, then this field may specify the name of a new or existing member within that library.

A selectable list of members will be presented if the entered value contains wildcards characters "*" (asterisk) or "%" (percent), or is blanked out.

Volume>

Specifies the name of the output data set volume. This is required only if output is to an uncataloged data set.

Append to existing Output

Select this option if the generated records are to be appended to existing records in the output data set.

HFS Output Options:

EOL Characters>

Choose from one of the following End-Of-Line character combinations:

NL	X'15'	New Line.
CR	X'0D'	Carriage Return.
LF	X'0A'	Line Feed.
CRLF	X'0D0A'	Carriage Return + Line Feed.

JSON Source and Options

Translation Options:

CCSID Conversion>

Since the purpose of JSONGEN is to produce a portable (data-interchange format) version of the data in a z/OS mainframe structured data file, and the output is character data, the coded character set identifiers (CCSIDs) of the input, output and of the JSONGEN internal constants themselves are of significance.

Even if the input and output is coded in an EBCDIC CCSID, these may differ, and both may differ from the CCSID of the JSONGEN command's internal constants. Since some of the special characters used in JSON have different code points in different EBCDIC CCSIDs (for example quotation marks) these must be dealt with consistently to produce correct JSON output.

JSONGEN uses the z/OS character conversion support supplied by IBM modules CUNLINFO (for obtaining CCSID information) and CUNLCNV (for character conversion from one CCSID to another).

The internal JSONGEN CCSID (that of the constants used to build the JSON syntax) is CCSID 285 (EBCDIC, SBCS UNITED KINGDOM).

JSONGEN assumes a default CCSID as follows:

Interactive

When executed interactively JSONGEN uses as default input CCSID that of the user's 3270 terminal.

Batch

When executed in batch JSONGEN uses as default input CCSID the value of the INI file variable **SDE.CCSID**. This variable is set automatically to the user's 3270 terminal CCSID (if not already set) during an interactive session. It can also be set using the structured data SET CCSID command.

Avaliable options are as follows:

NONE

The JSON output dataset is produced using the default CCSID and the input dataset character fields are assumed to be in the same CCSID. The internal JSONGEN constants are converted from internal CCSID 285 to the default CCSID.

ASCII

Convert the output to ASCII. This is equivalent to specifying **CONVERT TO 819**. CCSID 819 is ISO 8859-1 ASCII.

UNICODE

Convert the output to UNICODE (UTF-16). This is equivalent to specifying **CONVERT TO 1200**. CCSID 1200 is the IBM big-endian UTF-16 CCSID which is automatically transformaed to the most recent UTF-16 standard.

CONV

from>

from_ccsid

The input character data fields are converted from this CCSID. If not supplied the default input CCSID is used.

to>

to_ccsid

The CCSID of the output JSON text dataset. Internal JSONGEN character literals and input character data fields (and HFS line end characters if used) are converted to this CCSID.

Miscellaneous Options:

View option>

When JSONGEN is run interactively this option allows the user to request to view the output when the process completes.

BROWSE

Browse the output JSON dataset.

EDIT

Edit the output JSON dataset using the FileKit text editor.

NOVIEW

Do not view the output JSON dataset. This option is forced when run in batch.

Indentation value>

Nested output JSON names corresponding to the hierarchy of group and elementary data fields in the input structure are indented by a default of one space for each data item level. This option allows the specification of a different indentation value.

Output all field redefinitions:

If the structure defined for the input dataset contains redefined fields this option controls whether the field redefinitions are output.

Split long JSON records

For each elementary input field JSONGEN builds one output record containing the field name and value. Depending on the nature of the input data, relatively long output records may result. If an output record is longer than the allocated logical record length of the output dataset, this option controls how JSONGEN deals with the long output record.

Activated

Split the output record breaking it up into as many logical records as necessary. Records are split at the logical record length irrespective of the record content.

Not activated

Do not split the output record. Rather than truncate the output record JSONGEN terminates with an error message. This is the default.

Primary Commands

The following primary commands are supported.

OPTIONS (OPT)	Display JSON processing options panel.

Function Keys

In addition to the standard interactive panel key assignments for scrolling and navigation, the JSONGEN panel supports the following:

F6	OPTIONS	Display JSON processing options panel.

Command Line (Primary) Commands

SDE commands may be issued from:

1. A Data Editor view command line.
2. A Text Editor view command line using the SDATA command.
3. An SDATA command in a line of text displayed in a Text Editor view. (Executed using the ACTION facility.)
4. A Text Editor or Data Editor REXX macro.
5. A programmable function key.

Multiple SDE commands may be issued in a single invocation by separating each command with the line end character (set to ";" semi-colon by default.)

Executing SDE Commands from a Text Editor View

The Text Editor primary command **SData** may be used to prefix a command string to be passed to the Structured Data Environment (SDE).

If execution of an SDE command from a Text Editor view causes an update to the display of data within a Data Editor view, then it is the current Data Editor view that is updated.

Following execution of the SDE command, focus will remain with the Text Editor view from which the command was executed unless EDIT or BROWSE was executed. In the case, focus will pass to the newly edited or browsed data in the Data Editor view.

The following examples are SDE commands that exist as lines of text in a file displayed using the Text Editor. Each may be executed by first positioning the cursor on the text line and then pressing the ACTION key (usually assigned to shift-F4 by default).

```
<sdata create structure   CBL.FILEKIT.STRUCT(COMPSTR)  from cobol CBL.COPYBOOK.COBOL(COMPDEF)
<sd edit  CBL.SDE.EMP   using CBL.FILEKIT.STRUCT(COMPSTR)
<sd select  Key,InvNumb,DeliveryDate  from Orders   in CBL.FILEKIT.STRUCT(COMPSTR)
```

Command Reference Syntax Conventions

How to read the Syntax Diagrams

The following rules apply to the syntax diagrams used in this command reference.

1. The diagrams should be read from left to right, from top to bottom, following the path of the line.
   - ♦ The >>- symbol indicates the beginning of a statement.
   - ♦ The ->< symbol indicates the end of a statement.
2. Required items appear on the horizontal line (the main path).

```
>>--- required_item --------------------------------><
```

3. Optional items appear below the main path.

```
>>--- required_item ---+----------------------+-----><
                       |                      |
                       +-- optional_item -----+
```

4. 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 ---+----------------------+-----><
```

5. If you can choose from two or more items, they appear vertically, in a stack.
6. If you must choose one of the items, one item of the stack appears on the main path.

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

7. If choosing one of the items is optional, the entire stack appears below the main path.

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

8. If one of the items is the default, it appears above the main path and the remaining choices are shown below.

```
                       +-- default_choice ----+
                       |                      |
>>--- required_item ---+----------------------+-----><
                       |                      |
                       +-- optional_choice1 --+
                       |                      |
                       +-- optional_choice2 --+
```

9. An arrow returning to the left, above the main line, indicates an item that can be repeated.

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

10. If the repeat arrow contains a comma, you must separate repeated items with a comma.

```
                       +- , ------------------+
                       v                      |
>>--- required_item ---+-- repeatable_item ---+-----><
```

11. A repeat arrow above a stack indicates that you can repeat the items in the stack.

```
                       +----------------------+
                       v                      |
>>--- required_item ---+----------------------+-----><
                       |                      |
                       +-- optional_choice1 --+
                       |                      |
                       +-- optional_choice2 --+
```

12. Uppercase characters in keywords indicate the minimum abbreviation allowed for that particular command or parameter and must be spelled exactly as shown.

13. Variables appear in all italic, lowercase letters and represent user-supplied names or values.

14. If punctuation marks, parentheses, arithmetic operators, or other such symbols are shown, you must enter them as part of the syntax.

15. Where a parameter, immediately following a command verb, begins with a non-alpha character, no separating blank is required between the command verb and the parameter. e.g. Add8, CHANGE/abc/xyz/

ARRC

Syntax:

```
>>-- ARRC -------------+---------------+------------------------------------->
                       |               |
                       +-- field_col --+

 >-+-----------------------------------------------------------------------+-><
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

Description:

ARRC is equivalent to SET ARRAYASCHARACTER ON and so displays the specified, single dimension array field of one byte character fields, as an updateable variable length character field of data type XVARCHAR.

Parameters:

field_col

The individual column identifying the array field to which the option will apply. The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

Default is the focus column.

FOR [RECORD] *record_type*

Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

IN [STRUCTURE] *struct_name*

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

See Also:

ARRX GRPC GRPX

ARRX

Syntax:

```
>>-- ARRX -------------+---------------+------------------------------------->
                       |               |
                       +-- field_col --+

 >-+-----------------------------------------------------------------------+-><
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

Description:

ARRX is equivalent to SET ARRAYASCHARACTER OFF and so redisplays an array field, which has been displayed as a single, variable length character field (ARRC), as a number of individual, single character fields.

Parameters:

field_col

The individual column identifying the array field to which the option will apply. The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

Default is the focus column.

`FOR [RECORD] record_type`

Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

`IN [STRUCTURE] struct_name`

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

See Also:

ARRC GRPC GRPX

AVERAGE

Syntax:

```
                   +------------------------- , ---------------------------+
                   v                                                       |
>>--+- AVerage -+--+- field_col -------------------------------------------+->
    |           |  |                                                       |
    +- AVG -----+  |                                       +- CHaracter -+ |
                   |                                       |             | |
                   +- field_pos -- , -- field_length -- , -+-------------+-+
                                                           |             |
                                                           +- Fixed -----+
                                                           +- Binary ----+
                                                           |             |
                                                           +- PD --------+
                                                           +- DECimal ---+
                                                           +- PACKed ----+
                                                           |             |
                                                           +- ZD --------+
                                                           +- Zoned -----+
                                                           |             |
                                                           +- HFP -------+
                                                           |             |
                                                           +- BFP -------+
                                                           |             |
                                                           +- DFP -------+


                                  +- .ZFIRST -- .ZLAST -+
                                  |                     |
 >--+- WHere --- expression ---+--+---------------------+--+---------------+->
    |                          |  |                     |  |               |
    |                          |  +- .name1 -+----------+  +- DIgits int --+
    | +- NX -----------+       |             |          |
    | +- NOTEXcluded --+       |             +- .name2 -+
    | |                |       |
    +-+----------------+-------+
      |                |
      +- EXcluded -----+
      +- X ------------+
      |                |
      +- ALL ----------+


 >--+--------------------------------------------+--+----------------+-------><
    |                                            |  |                |
    +-------+- RECord -+--------+- record_type --+  +- STEM stemvar -+
    |       |          |        |
    +- FOR -+          +- TYPE -+
```

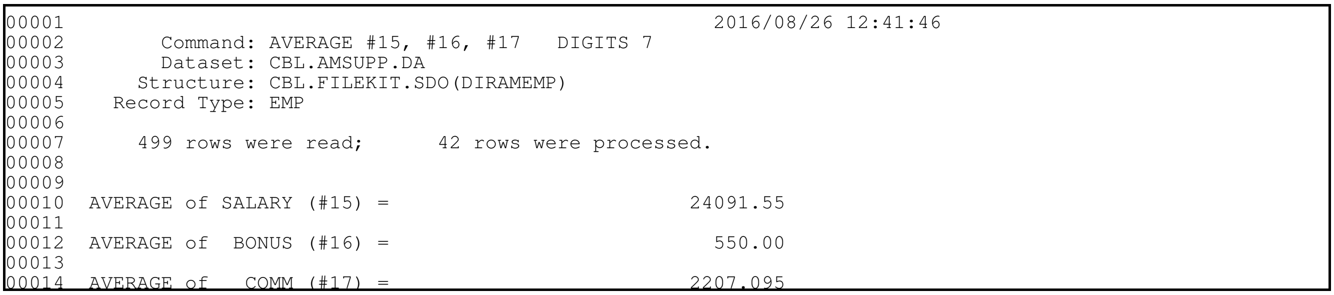
Description:

For browse or edit of formatted or unformatted data in the focus Data Editor view, AVERAGE will display or extract the average value of values belonging to columns in unformatted records or, if formatting is applied, records of a specific record type.

Multiple columns may be specified as a mixture of formatted record column ids (names and/or reference numbers) and field positions and length. An average value will be generated for each column specification.

Records may be included or excluded from the AVERAGE operation via a WHERE filter expression or using the current excluded status of lines within the display.

Unless STEM is specified for use in REXX procedures (edit macros), the AVERAGE values are reported in a temporary Text Edit view. e.g.



```
00001                                                       2016/08/26 12:41:46
00002        Command: AVERAGE #15, #16, #17   DIGITS 7
00003        Dataset: CBL.AMSUPP.DA
00004      Structure: CBL.FILEKIT.SDO(DIRAMEMP)
00005    Record Type: EMP
00006
00007      499 rows were read;      42 rows were processed.
00008
00009
00010  AVERAGE of SALARY (#15) =                          24091.55
00011
00012  AVERAGE of  BONUS (#16) =                            550.00
00013
00014  AVERAGE of   COMM (#17) =                          2207.095
```

For each reported column average, report lines may follow which detail the number of column values that were excluded from the process for the following reasons:

1. The value is null.
2. The value is invalid (not in the data type format).
3. For decimal floating point data type, the value is QNAN, SNAN or Infinity.

When the duration of a AVERAGE execution exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the oportunity to interrupt the operation using the Attn key.

Parameters:

field_col

An individual field column id within a formatted record display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. SALARY).

If *field_col* identifies an array, then AVERAGE treats each element of the array as requiring a separate average value. To perform AVERAGE on an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array.

field_pos

The start position of a field within the record data. For formatted records, this is a position in the expanded record.

field_length

Following *field_pos*, *field_length* defines the length of the field in the record data. The value of *field_length* must be valid for the corresponding *field_type*.

field_type

The data type of the data in the field identified by *field_pos* and *field_length*. *field_type* may be one of the following:

CHARACTER	Character - the default data type. *field_len* must be less than 32768. Numeric interpretation of character data supports use of the following: 1. Currency symbol (x'5B'). 2. A dot/period (".") representing decimal point. 3. Commas (","). 4. Character "E" (or "e") representing start of an exponent. 5. Unary plus ("+") or minus ("-") preceding the value and/or exponent value.
FIXED BINARY	Fixed Point Binary. *field_len* must be less or equal to 8.
PD DECIMAL PACKED	Packed Decimal. *field_len* must be less or equal to 16.
ZD ZONED	Zoned Decimal. *field_len* must be less or equal to 31.
HFP	Hexadecimal Floating Point. *field_len* must be 4, 8 or 16.
BFP	Binary Floating Point. *field_len* must be 4, 8 or 16.
DFP	Decimal Floating Point. *field_len* must be 4, 8 or 16.

WHERE *expression*

Only records of the specified record type *record_type* that satisfy the WHERE expression are included in the AVERAGE operation.

expression is a valid SDE expression which supports function calls, *record_type* field names/references, sub-expressions, arithmetic, relational and logical operators. The result of *expression* must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

When an expression is applied to the record, it must test "true" in order to be selected as a target of the AVERAGE operation.

By default, all data records in the selected range of lines are subject to the WHERE *expression* test. This includes data records that are of the selected record type and have EXCLUDED status at the time the AVERAGE command is executed. If one of these excluded records satisfies the *expression*, it will be included in the AVERAGE calculation.

`EXCLUDED`
`X`

Only records assigned the specified record type *record_type* and have EXCLUDED status are included in the AVERAGE operation. By default, only non-EXCLUDED records are selected.
AVERAGE does not include records that have the status NOTSELECTED or SUPPRESSED.

`NOTEXCLUDED`
`NX`

Only records assigned the specified record type *record_type* and have non-EXCLUDED status are included in the AVERAGE operation. This is the default action.
AVERAGE does not include records that have the status NOTSELECTED or SUPPRESSED.

`ALL`

All records (EXCLUDED and non-EXCLUDED) assigned the specified record type *record_type* are included in the AVERAGE operation. By default, only non-EXCLUDED records are selected.
AVERAGE does not include records that have the status NOTSELECTED or SUPPRESSED.

.name1

A label name identifying the first line of a range of lines on which the AVERAGE operation will operate. The preceding "." (dot) is mandatory.
Default is .ZFIRST.

.name2

A label name identifying the last line of a range of lines on which the AVERAGE operation will operate. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then *.name2* and *.name1* identify the first and last rows respectively.
Default is .ZLAST.

`DIGITS` *int*

DIGITS *int* represents the number of significant decimal digits (1-31) to be used in the average value. The value is rounded to the lowest significant digit.
Default is 21.

`[FOR] RECORD [TYPE]` *record_type*

Identifies the record type mapping *record_type* used by the AVERAGE operation. Only records assigned this record type are eligible to be included in the AVERAGE calculation.

If a *field_col* is specified, it must reference a column field in this record type definition.

Default is the default record type.

`STEM` *rexx_stemvar*

Applicable only to execution of AVERAGE within a REXX procedure edit macro, STEM indicates that the AVERAGE value for each of the specified columns be assigned to a REXX compound variable.

The mandatory *rexx_stemvar* argument specifies the character string to be used as the stem part of the compound variable name.

The following REXX compuond variables are set:

rexx_stemvar.AVG.0

The number of AVERAGE values returned (i.e. number of columns specified).

rexx_stemvar.AVG.i

The ith AVERAGE value corresponding to the ith column specification.

See Also:

MAXIMUM MINIMUM SUM TOTALS

BOTTOM

Syntax:

```
>>-- BOttom ------------------------------------------------------------------><
```

Description:

Display the last page of data.
Equivalent to the DOWN MAX command.

See Also:

DOWN TOP

BROWSE

Syntax:

```
>>-- Browse ---+-------------- fileid -----------------+---| SDE Opts |--+--->
               |                                       |                 |
               +---------- hfs_fileid ---| HFS Opts |--+                 |
               |                                       |                 |
               +---DDname=ddname ----------------------+                 |
               |                                                         |
               +-- DB2 --+---------+-----------------------| DB2 Opts |--+
                         |         |
                         +- (ssn) -+

             +-- PROFile SDEPROF -------+
             |                          |
 >-----------+--------------------------+------------------------------------>
             |                          |
             +-- PROFile macro_name ----+
             |                          |
             +-- NOPROFile -------------+

 >-----------+-------------------------------------+-------------------------><
             |                +--------------+     |
             |                v              |     |
             +-- INItcmd - ( -+- sd_command -+- ) -+
```

HFS Opts:

```
               +-- STD -----+
               |            |
    +-- EOL ---+------------+---------------+
    |          |            |               |
    |          +-- CR ------+               |
    |          +-- LF ------+               |
    |          +-- NL ------+               |
    |          +-- CRLF ----+               |
    |          +-- LFCR ----+               |
    |          +-- CRNL ----+               |
    |          +-- string --+               |  +- LRECL lrecl -+
    |                                       |  |               |
 |--+---------------------------------------+--+---------------+-------------|
    |                                       |
    +-- RECFM -+- F ------------------------+
               |                            |
               |     +- (0,2,0) ----------+ |
               |     |                    | |
               +- V -+- (off,len,origin) -+-+
```

SDE Opts:

```
|--+----------------------------------------------------------+------------->
   |                                                          |
   +- USING -+-------------+-+- struct_name ------------------+
             |             | |                                |
             +- STRUCTure -+ +------ = -----------------------+
             |                                                |
             +- HLAsm -----+--- copybook_name ----------------+
             |             |                                  |
             +- COBOL -----+                                  |
             |             |                                  |
             +- PL1 -------+                                  |
             |             |                                  |
             +- ADAta -----+                                  |
             |                                                |
             |              +-------------------------+       |
             |              v                         |       |
             +- SYMNAMes ( -+---- DFSORT symbols -----+- ) ---+

   +-- FORMat ------- TABle ------+
   |                              |
>--+------------------------------+--+-----------------------------+-------->
   |                              |  |                             |
   +-- FORMat -+--+-- SINGle -----+  +-- AUTostructure --+- APPLY -+
   |           |  |               |                      |         |
   +-- FMT ----+  +-- SNGL -------+                      +- OFF ---+
                  |               |                      |         |
                  +-- CHARacter --+                      +- ON ----+
                  |               |                      |         |
                  +-- HEX --------+                      +- SAVE --+

>--+-----------------------------------------------------------------+------>
   |                                                                 |
   +- KEYRange - ( - lowkey -+---------------+- ) -------------------+
   |                         |               |                       |
   |                         +- , - highkey -+                       |
   |                                                                 |
   +------------------------------+--+-------------------------------+
   |                              |  |                               |
   +-- FROM --+- KEY -- string ---+  +-- FOR -- n_recs --+-----------+
              |                   |                      |           |
              +- RBA ------ n ----+                      +- RECORDS -+
              |                   |
              +----------+- n ----+
              |          |
              +- RECord -+

   +-- View -------- * ---------+
   |                            |
>--+----------------------------+-------------------------------------------->
   |                            |
   |             +-- , --+      |
   |             |       |      |
   |         +---+-------+---+  |
   |         |               |  |
   |         V               |  |
   +-- View --- record_type -+--+

>-+----------------------------------+--+-------------------------------+---|
  |                                  |  |                               |
  +-- FILTer --+--- filter_fileid ---+  | +--------------------------+  |
               |                     |  | v                          |  |
               +--| Filter Clause |--+  +-+-- SELect select_syntax --+--+
```

Filter Clause:

```
        +----------------------------------------------------------+
        v                                                          |
|- ( -+--- INclude rec_type --+--------------+--+----------------+-+-+------>
      |                       |              |  |                |   |
      |                       +- WHere expr -+  +- LIMit n_hits -+   |
      |                                                              |
      | +----------------------------------------------------------+ |
      | v                                                          | |
      +--- EXclude rec_type --+--------------+--+----------------+-+-+
                              |              |  |                |
                              +- WHere expr -+  +- LIMit n_hits -+

>-----+--------------------+-------------------------------------------- ) -|
      |                    |
      +- Stopafter n_hits -+
```

DB2 Opts:

```
|--+-- table_name --+-+----| SQL Query Opts |---------------------+--------->
   |                |  |                                           |
   +-- view_name ---+ +-- USING --+-------------+-- struct_name --+
   |                              |             |                 |
   |                              +- STRUCTure -+                 |
   |                                                              |
   +-- SQL ( sql_syntax ) ----------------------------------------+
   |                                                              |
   +-- SQL sql_file ----------------------------------------------+

   +-- FORMat ------- TABle ------+
   |                              |
>--+------------------------------+----------------------------------------->
   |                              |
   +-- FORMat -+--+-- SINGle -----+
   |           |  |               |
   +-- FMT ----+  +-- SNGL -------+

>--+-----------------------------------------------------------------+------>
   |                                                                 |
   +-- SCROLL -------------------------------------------------------+
   |                                                                 |
   +------------------------------+--+-------------------------------+
   |                              |  |                               |
   +-- FROM --+----------+- n ----+  +-- FOR -- n_rows --+-----------+
              |          |                               |           |
              +- ROW ----+                               +- ROWS ----+

                                         +-- XMLLOBWidth -- 0 --------+
                                         |                            |
>--+---------------------------------+---+----------------------------+-----|
   |                                 |   |                            |
   +---- SELect (select_syntax) -----+   +-- XMLLOBWidth -- n_bytes --+
```

SQL Query Opts:

```
|--+------------------------+--+-----------------------------------------+--|
   |                        |  |                                         |
   +- WHERE (where_clause) -+  +- ORDER -+------+--+- (order_by_clause) -+
             |          |      |         |      |  |                     |
             +--- ? ----+      |         +- BY -+  |                     |
                               +- SORT ------------+                     |
                               |                                         |
                               +- SORTIndex -+-- index_name --+----------+
                                             |                |
                                             +-- Prime -------+
```

Description:

Browse a structured data set or HFS file, or browse a DB2 results table within SDE. If BROWSE is used, the displayed data cannot be altered in any way.

An SDE edit display window is opened and focus is passed to the new window. Depending on the value on the FORMAT parameter, the SDE display is either a multi record window view or a single record window view for the first record selected.

Structured Data Browse:

SDE browse will not attempt to load all records of the file into storage, thus allowing quick display of records even in very large data sets. Records are loaded and dropped from storage as required ensuring that available storage is never exceeded.

If the Structure Definition Object (SDO) specified by the USING STRUCTURE parameter is not already in storage, it is loaded from the appropriate Structure Definition File (SDF). Record Type Objects (RTO) within the specified SDO are then used to map the records in the data set.

For every record in the file, each RTO is checked until one is found whose criteria is matched by the record characteristics. This RTO is then used to map the record. If the record does not match any RTO criteria, then the first RTO in the SDO is used by default. The RTO automatically selected to map a record may be changed by the user via the USE WHEN command or the SDE Edit/Browse Utility Window.

By default, all fields within records that have an associated RTO are displayed as printable character, numeric data fields having first been converted to decimal.

If no SDO is specified, then FORMAT CHARACTER is applied to each record. i.e. the data is displayed as a single, variable (RECFM=V) or fixed (RECFM=F) length character field with field name "UnMapped". No data conversion is performed.

If the VIEW parameter is specified with a non-generic argument, the DRECTYPE setting is initialised as the first *record_type* parameter specified on the VIEW parameter. Otherwise, the DRECTYPE setting is initialised as the record type of the first visible record in the file.

Record data that does not meet the specifications of its field definition is considered to be invalid and is represented by asterisks.

DB2 Table Browse:

DB2 table browse is supported for in-storage edit/browse only. Therefore, if a results table is too large to be loaded into available storage a storage error occurs. The user must then restrict the number of rows to be loaded using further row selection criteria (i.e. FROM, FOR, SELECT, WHERE parameters).

On starting a DB2 table browse, a temporary SDO is created reflecting field characteristics obtained from the SQL Descriptor Area return by the prepared SQL SELECT execution. Furthermore, if the user does not already have an open connection to the required DB2 sub-system that is not for an SDE DB2 table edit view, then a new DB2 connection is made.

Parameters:

`fileid`

The fileid of the dataset, library member or PDSE library member generation containing the records to be browsed.

fileid may be any of the following:

◊ The DSN of a physical sequential data set.
◊ The DSN of a VSAM (KSDS, ESDS, RRDS, VRDS or LDS) data set.
◊ The library DSN and parenthesised member name of a PDS or PDSE library member.
◊ The library DSN, parenthesised member name and absolute or relative number of a PDSE library member generation as described under z/OS PDSE Library Member Generations. (PDSE version 2 with MAXGENS only.)
◊ The name of a member to be browsed from the same PDS or PDSE library as the member displayed in the current data edit window view.

`hfs_fileid`

A complete HFS (or zFS) file path containing data to be browsed.
hfs_fileid is identified as an HFS path if *fileid* is not a valid DSN for a sequential, VSAM or PDS/PDSE dataset. e.g. DSN begins with ".", contains invalid special characters (e.g. "/") or contains qualifiers of length greater than 8.

`ddname`

A previously allocated DD name (e.g. **"//MYFILE DD DSN=..."** in batch or **"ALLOC F(MYFILE)"** command under TSO) referring to the dataset(s) to be browsed.

ddname may refer to a concatenation of datasets or even a temporary dataset created by an earlier process.

HFS Opts

The following HFS options may be specified to determine the handling of data within the HFS file.

`EOL=STD|NL|CR|LF|CRLF|LFCR|CRNL|`*`string`*

Sets the EOLIN (input end-of-line) delimiter value used to determine the end of each record for non-RECFM F/V input. EOLIN delimiters are not included in the browsed record data or record length. EOL parameter elements are as follow:

STD	-	Any standard line-end.
NL	X'15'	New Line.
CR	X'0D'	Carriage Return.
LF	X'0A'	Line Feed.
string	-	A 2-byte user specified character or hex string.

STD is default so that the BROWSE operation scans the input data for any of the standard EOL combinations (not *string*), stopping when one is found. This EOL combination is used as EOLIN for the file.

Where EOL processing is specifically requested for BROWSE, if EOL characters are **not** found within the first lrecl bytes of data, then the file is displayed as RECFM F with records of length 80.

`RECFM F | V (`*`off,len,origin`*`)`

Specifies that the data is to be treated as containing Fixed or Variable length format records.

RECFM F indicates that all records are of a fixed length as defined by the LRECL argument.

RECFM V allows the user to specify the location of the record length fields within the data as follows:

off	Offset of the record length field from the start of the record.
len	Length of the record length field.
origin	The start of the record data at which the record length is applied.

Default is (0,2,0) which describes standard RECFM V organisation data sets.

`LRECL `*`lrecl`*

Specifies the maximum record length of input records.

Records terminated by an EOL sequence will wrap onto the next line of data if the record length exceeds *lrecl*. Where a record has wrapped, the prefix area contains the "==EOL>" flag.

For RECFM F data, *lrecl* is the fixed length of the records in the edit view. The data within the last record is padded with blanks up to the *lrecl* length if necessary.

If the record length field of a RECFM V record exceeds the *lrecl* value, then an error is returned.

RECFM V and EOL delimited records have default *lrecl* of 32752, wheras RECFM F records have default *lrecl* of 80.

SDE Opts

The following SDE options are applicable to browse of structured data found in HFS files or z/OS data sets only. See DB2 Opts for DB2 table browse options.

`USING (STRUCTURE)` *`struct_name`*

Identifies the SDO to be applied to the data records. *struct_name* is the SDF fileid assigned to the SDO in a CREATE STRUCTURE command.

"=" (equals) may be specified instead of *struct_name* to indicate the name of the structure being used in the current SDE window.

`USING HLASM | COBOL | PL1 | ADATA` *`copybook_name`*

Identifies the full fileid of *copybook_name*, an Assembler/COBOL/PL1 source member or Assembler/COBOL/PL1 ADATA output file to be used to format the data records.

Using *copybook_name* syntax has a processing overhead in that a CREATE STRUCTURE operation is performed to generate a temporary SDO. This overhead is greater if HLASM, COBOL or PL1 is specified since an assemble/compile of the source member is performed to first obtain the ADATA file output. Therefore, it is recommended that a non-temporary SDO is generated using CREATE STRUCTURE and that this is be used for all future formatting of the data (on EDIT, BROWSE, FSU, etc.)

`USING SYMNAMES (` *`DFSORT symbols`* `)`

Specifies DFSORT SYMNAME symbol definitions that are to be used to format the data records. The order in which symbol definitions are supplied dictate the order in which the fields will occur in the record type definition.

The symbol name definitions within the SYMNAMES parentheses may be supplied directly in-line and/or via input data sets/library members.

```
SYMNAMES( Card,06,04,CH  Dept,46,03,CH  Amount,49,06,PD  )
SYMNAMES( SYS1.MACLIB(EDGSMFSY)  SYS1.MACLIB(EDGSRCSY) )
SYMNAMES( CBL.DFSORT.SYM(CBLATRAC)  TCB,*,4,BI )
```

`FORMAT`
`FMT`

Specifies the format in which record data will be displayed in the initial SDE view.

`SINGLE` `SNGL`	Single record format.
`TABLE`	Table format. (Default)
`CHARACTER`	Multi record view with all records or record segments mapped as single, variable length character fields with field name "UnMapped" or "UnMappedSeg". No data conversion is performed.
`HEX`	Same as CHARACTER with the addition that the data is also displayed in Hex below the character display. Note that the Hex display occupies an additional 2 lines of data.

The data display format may be later altered using the FORMAT and/or ZOOM commands.

`AUTOSTRUCTURE`

Temporarily overrides the current setting for option AUTOSTRUCTURE. This option will specify whether or not a structure to dataset association is automatically defined and/or applied for the current BROWSE operation.

`APPLY`	Unless a structure is specified via a USING parameter, APPLY will attempt to use a structure associated with a filemask that matches the browsed *fileid*. If no association has been saved, then no structure is used.
`OFF`	No attempt will be made to automatically apply an associated structure and, if a structure is specified via a USING parameter, no attempt will be made to save an association between the structure and the browsed fileid.
`ON`	Equivalent to both APPLY and SAVE.
`SAVE`	If a structure is specified via a USING parameter, save an association between it and the browsed fileid. This association may be automatically applied when the fileid is next browsed or edited.

`KEYRANGE` *`(lowkey, highkey)`* `| KEYRANGE` *`(lowkey)`*

Records are to be displayed with keys in the given range. This option applies to VSAM KSDS files only.

If *highkey* is ommitted it is assumed to have the same value as *lowkey*.
If *lowkey* is shorter than the defined length of the KSDS key, it is assumed to be padded on the right with X'00'.
If *highkey* is shorter than the defined length of the KSDS key, it is assumed to be padded on the right with X'FF'.

The key values can be given in the form:

- Simple strings with no embedded blanks. These are translated to upper case.
- Quoted strings using apotrophes or double quotes. These are translated to upper case.
- Character strings of the form *C'...'* containing any characters. These are not translated to upper case.
- Hexadecimal strings of the form *X'...'* containing hex digits 0-9 a-f A-F. These are not translated to upper case.

`FROM`

Records are to be displayed beginning at a specific location within the data set. The first record at the specified location becomes record 1 within the SDE window view of the data set. Records that occur before this location are not included within the browse session.

If FILTER is specified, then record filtering occurs only on records selected using the FROM and FOR parameters.

Required record location is identified via one of the following:

`KEY` *`string`*
For KSDS data sets only, locate the record with a key string equal to *string*. If not found, the record with the next key string greater than *string* is used.

`RBA` *`n`*
For KSDS and ESDS data sets only, locate the record starting at the relative byte address specified by *n*.

The RBA must point at the start of the record otherwise a VSAM point error will occur. RBA address for each record is displayed as part of the record information columns within an SDE window view. Display of these columns is controlled using the SDE SET RECINFO CLI command.

`< RECORD >` *`n`*
For any data set organisation, locate the record number specified by *n*. Specification of parameter keyword RECORD is optional.

Default is to display records within the SDE window view starting at the first record in the data set.

`FOR` *`n_recs`* `< RECORDS >`

Specifies the number of data set records to be included within the SDE window browse session. Specification of parameter keyword RECORDS is optional.

If FILTER is specified, then record filtering occurs only on records selected using the FROM and FOR parameters.

Default is to include all records.

`VIEW` *`record_type`*
`VIEW *`

Identifies one or more record types for which records will be displayed in the initial SDE view. Records that are associated with other record types are not visible within the display but are displayed as shadow lines instead. * (asterisk) indicates all record types including the internal record type "UnMapped" (or "UnMappedSeg") which is used to map data records or record segments that have no associated RTO in the SDO.

Records within the display may be later suppressed or made visible using the VIEW command.

Default is VIEW *.

`FILTER` *`filter_fileid`* `|` **`Filter Clause`**

FILTER specifies additional record filtering criteria to further reduce the display of records for browse. All record filters are applied only to records that have been selected using the FROM and/or FOR parameters, otherwise it applies to all records in the structured data file.

FILTER parameters are specified via a filter clause which may be supplied as part of the BROWSE command or referenced via *filter_fileid*, a separate sequential data set, PDS/PDSE member or HFS file. *filter_fileid* must contain the keyword FILTER followed by a valid filter clause.

Scrolling vertically through the multi-record view will load, and potentially unload and reload, records in and out of storage. The filter is applied to each record as it is loaded or reloaded.

`SELECT` *`select_syntax`*

SELECT may be used to customise the sequence and number of record-type columns displayed when the browse view of formatted data is opened. The *select_syntax* argument represents parameters supported by the SELECT primary command with the exception of IN *struct_name* which is unnecessary.

Unlike parameter INITCMD, which may also be used to execute SELECT commands, the SELECT parameter column selection is performed before the profile macro is executed.

Filter Clause

A filter clause must be specified in "( )" (parentheses) and may contain comment data enclosed by "/*" and "*/". If filter clause is specified via *filter_fileid*, then comment data may also occur before and after the filter clause.

The filter clause may contain either INCLUDE or EXCLUDE selection sub-clauses but not both. When a filter is applied, the record data is tested against each selection sub-clause in the order specified until a true condition is returned. On encountering a true condition, both the hit count for the individual selection sub-clause and the overall hit count is incremented by one and the record included or excluded as approriate.

Once the hit count for an individual INCLUDE or EXCLUDE selection sub-clause matches its LIMIT threshold, then that sub-clause plays no further part in the filter when applied to subsequent browse input records. Similarly, once the overall hit count matches the STOPAFT threshold, no further record filtering occurs and the remainder or the records are excluded or included as appropriate.

The following options are supported by the filter clause.

`INCLUDE` *record_type*

INCLUDE denotes the start of an INCLUDE sub-clause which, together with optional parameters WHERE and LIMIT, identifies conditions for which a record is included in the subset of browsed records.

The INCLUDE sub-clause applies only to records that have been assigned the specified *record_type*. If the record is not assigned this record type, a false condition is returned for the sub-clause.

A number of INCLUDE sub-clauses may be specified for different record types or for the same record type with different WHERE expression conditions. If one or more INCLUDE sub-clauses are specified, then records that do not satisfy any of the sub-clause conditions will be excluded by default.

Note that *record_type* "Record" (with field name "UnMapped") may be used to perform a filter on the unformatted record data whether or not a structure (USING *struct_name*) has been specified. In this way, a filter may test **all** records regardless of their assigned record type.

INCLUDE and EXCLUDE parameters are mutually exclusive.

`EXCLUDE` *record_type*

EXCLUDE denotes the start of an EXCLUDE sub-clause which, together with optional parameters WHERE and LIMIT, identifies conditions for which a record is excluded from the subset of browsed records.

The EXCLUDE sub-clause applies only to records that have been assigned the specified *record_type*. If the record is not assigned this record type, a false condition is returned for the sub-clause.

A number of EXCLUDE sub-clauses may be specified for different record types or for the same record type with different WHERE expression conditions. If one or more EXCLUDE sub-clauses are specified, then records that do not satisfy any of the sub-clause conditions will be included by default.

Note that *record_type* "Record" (with field name "UnMapped") may be used to perform a filter on the unformatted record data whether or not a structure (USING *struct_name*) has been specified. In this way, a filter may test **all** records regardless of their assigned record type.

INCLUDE and EXCLUDE parameters are mutually exclusive.

`WHERE` *expr*

WHERE applies further filter conditions to records assigned to the record type specified by the last INCLUDE *record_type* or EXCLUDE *record_type* parameter processed.

expr is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of the WHERE expression must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

The WHERE expression is applied to each record assigned the record type *record_type* and, if the result is "true", the record is selected for include or exclude as indicated by the prevailing INCLUDE or EXCLUDE filter. If multiple INCLUDE/EXCLUDE *record_type* WHERE expressions exist for the same record type, then a logical OR is implied for all the expressions relating to that record type.

`LIMIT` *n_hits*

LIMIT *n_hits* may be included as part of a single INCLUDE (or EXCLUDE) sub-clause specification and applies only to that sub-clause.

When the number of records selected by an individual INCLUDE (or EXCLUDE) sub-clause reaches the *n_hits* value specified by LIMIT, then that sub-clause no longer forms part of the filter applied to subsequent input records.

LIMIT provides a method whereby the subset of records selected for browse is spread more evenly across the filter's sub-clause conditions than could be achieved by use of the STOPAFTER parameter alone.

By default, no LIMIT threshold is applied INCLUDE (or EXCLUDE) sub-clause.

`STOPAFTER` *n_hits*

When the total number of records selected by the INCLUDE (or EXCLUDE) sub-clauses reaches the *n_hits* value specified by STOPAFTER, then no further filter testing occurs.

If an INCLUDE filter, then all remaining untested records are excluded. If an EXCLUDE filter, then all remaining untested records are included.

By default, no STOPAFTER threshold is applied to the filter clause.

`DB2 < (`*ssn*`) >`

Indicates that browse is for a DB2 base or results table. *ssn* is optional and identifies the local DB2 sub-system name to which a connection will be made.

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 users default DB2 sub-system name as set by the DB2 Primary Options menu.

`table_name`

A DB2 base table name containing rows of data to be browsed.

table_name may be specified with 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default location is the local DB2 server and the default schema is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID).

`view_name`

A DB2 table view which references an SQL query used to generate a results table containing the rows of data to be browsed.

view_name may be specified with 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default location is the local DB2 server and the default schema is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID).

DB2 Opts

The following DB2 options are applicable to browse of DB2 table data only. See SDE Opts for structured data browse options.

`SQL (sql_syntax)`

Specifies that the result table generated by the SQL query *sql_syntax* is to be browsed.

If this option is chosen the table name, view name, SELECT(), WHERE() and ORDER BY() parameters must not be specified.

`SQL sql_file`

Specifies that the result table generated by the SQL query located in file *sql_file* is to be browsed.

If this option is chosen the table name, view name, SELECT(), WHERE() and ORDER BY() parameters must not be specified.

`WHERE (where_clause)`

Specifies a DB2 SQL WHERE clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *where_clause*.

If *where_clause* is null or "?" (question mark) the DB2 Row Selection Panel is opened before the browse is actioned. On closing the panel, the browse is actioned using the generated WHERE clause.

`SORT|ORDER BY (order_by_clause)`

Specifies a DB2 SQL ORDER BY clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *order-by-clause*.

`SORTINDEX index_name | PRIME`

Specifies *index_name*, the name of an existing DB2 Index for the table being browsed. The index identifies the key columns/expressions by which the table rows will be ordered in the display.

PRIME may be specified as an alternative to indicate that the primary index should be used.

If *index_name* is "?" (question mark), the DB2 Select Table Index window is opened before the browse is actioned, allowing selection of one of the indexes defined for the table. On selecting an entry, the window is closed and the browse operation continues using the SORTINDEX *index_name*.

`USING (STRUCTURE) struct_name`

Identifies the SDO to be applied to the DB2 data rows. *struct_name* is the SDF fileid assigned to the SDO in a CREATE STRUCTURE command.

If the USING STRUCTURE option is specified then SELECT, WHERE and ORDER BY options are ignored. The SDO may contain DB2 SQL query SELECT, WHERE and ORDER BY sub-clause options which are used to select and display DB2 table rows.

`SCROLL`

Use a DB2 scrollable INSENSITIVE cursor to fetch DB2 rows. This keyword is incompatible with the FOR and FROM keywords. See Using Scrollable Cursors for information on how the use of this keyword affects the browse session.

The use of DB2 scrollable cursors may be disabled by the installer of FileKit. In order to use them the system INI file must contain the *DB2.SCROLL=YES* variable.

`FROM < ROW > n`

Equivalent to FROM RECORD *n*, FROM ROW *n* specifies that rows are to be displayed beginning at row number *n* in the DB2 results table returned by the SQL query. The first row displayed becomes row 1 within the SDE window browse view. Rows that occur before this row number are not included within the browse session. Default is row 1.

`FOR n_rows < ROWS >`

Equivalent to FOR *n_recs* RECORDS, FOR *n_rows* ROWS specifies the maximum number of rows to be displayed from the DB2 results table returned by the SQL query. Rows that fall outside the range of rows specified by FROM *n* FOR *n_rows* are not included within the browse session.
Default is to display all selected rows.

`SELECT (select_syntax)`

DB2 table browse will fetch **all** columns from the selected table or view.

SELECT may be used to select a subset of these columns for initial display in the browse window view. Except for parameters FROM *record_type* and IN *struct_name* which are both unnecessary, the *select_syntax* argument represents parameters supported by the SELECT primary command.

Unlike parameter INITCMD, which may also be used to execute SELECT commands, the SELECT parameter column selection is performed before the profile macro is executed.

`XMLLOBWIDTH n_bytes`

Specifies the number of bytes (*n_bytes*) of text, at the start of an XML or large object (LOB) column, to be displayed for all XML and LOB columns in the edited table view.

`PROFILE macro_name`

Specifies the name of the REXX SDE edit macro to be executed as the profile when the data is browsed.

macro_name must exist in a library within the SDE macro path.

The PROFILE option only alters the profile used for the file currently being browsed. It does not define the profile macro to be used for subsequent SDE edit or browse.

The default is to execute a macro with member name SDEPROF if it exists.

`NOPROFILE`

Suppresses use of a profile macro when browsing the file.

The NOPROFILE option only suppresses use of a profile for the file currently being browsed. It does not suppress use of a profile macro for subsequent SDE edit or browse.

`INITCMD (sd_command ... )`

Specifies a list of one or more structured data commands to be executed when initialising the browse session. These commands are issued after the profile macro (if specified) has been executed but before the first view is displayed.

The commands are listed, separated by blanks, within parentheses following the *INITCMD* keyword. If a command contains blanks or special characters (including apostrophes or quotes) it must be delimited with apostrophes or quotes. When such a delimiter is used, instances of the delimiter character within the command string must be represented by a pair of delimiter characters. For example:

```
 INITCMD( 'f "O''Reilly" #3')
```

could be used to open the browse session with the dataset scrolled to the first record containing *O'Reilly* in field number 3.

Examples:

```
<sd browse CBL.SDE.MOD.ZJ2202   using CBL.FILEKIT.SDO(PRODL)  view ARCX1,PRODX
```

Browse records from data set "CBL.SDE.MOD.ZJ2202" within the SDE.
Initially display all records of type ARCX1 and PRODX where the record type objects (RTO) are found in the structure definition object (SDO) referenced by CBL.FILEKIT.SDO(PRODL).

```
<sd browse  db2(DB9G)  CBL.APIFUNC  where(FUNCMOD#REF = 0)
```

Browse filtered rows of DB2 table CBL.APIFUNC in subsystem DB9G.

See Also:

EDIT and CREATE STRUCTURE

CANCEL

Syntax:

```
>>--- CANcel ----------------------------------------------------------------><
```

Description:

Close **all** SDE window views for data displayed in the current SDE view.

If unsaved changes exist, the user will be prompted to save the changes, discard the changes or to terminate the CANCEL operation and return focus to the SDE edit view. The default action in this list being to discard the changes.

For DB2 edit, if the changes are discarded then a **ROLLBACK** is done, backing out changes to the previous commit point.

If a non-temporary structure (SDO) is used to map the record data and changes have been made to that structure during the course of the edit session (e.g. USE WHEN), then the user will be prompted to save the changed structure to its structured data file (SDF). See command, SAVESTRUCTURE.

See Also:

END FILE QQUIT

CCOLOUR

Syntax:

```
    +- TEMPorary -+
    |             |
>>--+-------------+-- CCOlour -+- field_col -+- | colour_specification | -+-->
    |             |            |             |                            |
    +- PERManent -+            |             +- OFF ----------------------+
                               |                                          |
                               +- OFF ------------------------------------+

 >-+-----------------------------------------------------------------------+-><
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

colour_specification

```
                        +- NONe ------+
                        |             |
 >-----+- Blue ------+--+-------------+--- WHEN expression ------------------>
       |             |  |             |
       +- Red -------+  +- BLInk -----+
       |             |  |             |
       +- Pink ------+  +- REVvideo --+
       |             |  |             |
       +- Green -----+  +- Uscore ----+
       |             |
       +- Turquoise -+
       |             |
       +- Yellow ----+
       |             |
       +- White -----+
       |             |
       +- Default ---+
```

Description:

CCOLOUR (column colour) may be used to apply preferred colouring to field columns (data elements) in records with an assigned record type (RTO) based on selection criteria.

Column colouring specifications defined by the CCOLOUR command are stored in the structure definiiton (SDO) if the *PERManent* command prefix is used. If not already loaded, the required SDO is loaded from its structured data file (SDF) and updated accordingly. If a non-temporary SDO, then this change to the SDO may be saved (written back to the SDF) by the user when the structure is dropped or on execution of the SAVESTRUCTURE command.

Any number of CCOLOUR specifications may exist for a field (data element) based on different WHEN *expressions*. When records satisfy the WHEN *expression* of more than one CCOLOUR specification, then the CCOLOUR specification used is the one that was defined last.

Parameters:

`TEMPorary`
: This column colour specification is for the current session only, and will not be saved in the structure definition file. This is the default if neither PERMANENT or TEMPORARY is given.

`PERManent`
: This column colour specification is to be saved in the structure definition.

`field_col`

The name of the field column (data element) to which the colour specification will be applied.

`BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE | DEFAULT`

Supported colours. If DEFAULT is specified, the default colour for record display is used.

`BLINK | REVERSE | USCORE | NONE`

Extended highlighting. The colour may blink, be displayed in reverse video or be underlined. Default is NONE.

`WHEN expression`

Specifies the criteria that must be satisfied before the defined colouring will occur.

expression is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of the WHEN expression must be numeric and is treated as a boolean value with zero indicating *FALSE* and non-zero indicating *TRUE*.

The WHEN expression is applied to each record assigned the record type *record_type* and, if the result is "true", the specified column colouring is applied to the field column. To apply column colouring for all instances of *field_col* omit the WHEN expression.

`OFF`

Remove all specified column colouring for the *field_col* from the SDO, reverting back to the default settings.

If no *field_col* is specified then column colouring is removed from all fields of the *record_type*.

`FOR (RECord) record_type`

The name of the record type (RTO) containing the *field_col*.
Default is the current (i.e. last referenced) record type.

`IN (STRUCTure) struct_name`

The name of the structure (SDO) in which *record_type* is defined.
Default is to the current (i.e. last referenced) structure. This is the structure used in the current SDE window view or explicitly referenced by an SDE command (e.g. USE WHEN)

Examples:

```
cco bonus red uscore when bonus>500.00
```

Define colouring of the *BONUS* field column in the current (default) record type as red underscore if the bonus is greater than *500.00*.

See Also:

RCOLOUR DROP SAVESTRUCTURE USE and the SET/QUERY/EXTRACT Option: COLOUR

CHANGE

Syntax:

```
                 +- EQ -+                                   (1) +- ANY ---+
                 |      |                                       |         |
>>--- Change --+-+------+----- string1 ------+----- string2 ----+---------+-->
               | |      |                    |                  |         |
               | +- op -+                    |                  +- FOCus -+
               |                             |
               +- VALID ---------------------+
               |                             |
               +- INVALID -------------------+
               |                             |
               +- ANYValue ------------------+

                      +- NEXT --+  +- CHARs --+
                      |         |  |          |
 >--------------------+---------+--+----------+--+------+-------------------->
                      |         |  |          |  |      |
                      +- ALL ---+  +- PREfix -+  +- EX -+
                      |         |  |          |  |      |
                      +- FIRST -+  +- SUFfix -+  +- NX -+
                      |         |  |          |  |      |
                      +- LAST --+  +- WORD ---+  +- X --+
                      |         |
                      +- PREV --+

   +-- #ALL ------------------------------------+ +- .ZFIRST ---- .ZLAST -+
   |                                            | |                       |
 >-+--------------------------------------------+-+-----------------------+-->
   |                                            | |                       |
   +-- pos1 ---+---------+----------------------+ +- .name1 --+-----------+
   |           |         |                      |             |           |
   |           +- pos2 --+                      |             +- .name2 --+
   |                                            |
   |      +----+---------+----------------+     |
   |      |    |         |                |     |
   |      |    +-- , ----+                |     |
   |      v                               |     |
   +-- ( -+-- field_col ------------------+- ) -+
          |                               |
          +--  field_col1:field_col2  ----+

   +-- DATA --+
   |          |
 >-+----------+--------------------------------------------------------------><
   |          |
   +-- TEXT --+
```

(1) Default (ANY or FOCUS) set by the RTSCOPE option.

Description:

Search DB2 table rows or data records in the current edit or browse view for the specified character string or numeric value (*string1*) and replace it with *string2*.

If the **FOCUS** option is specified then only records/segments assigned the default record type (visible and EXCLUDED records) are included in the search. Otherwise all records types are searched.

If the specified occurrence (ALL, FIRST, LAST, NEXT or PREV) of the search string or numeric value (*string1*) is found within a record, then:

1. If the record is EXCLUDED, it is made visible.
2. The cursor is positioned at the beginning of the string or numeric field.
3. If necessary, scrolling occurs to display the found data.
4. If the replace string (*string2*) satisfies the data type, precision and scale requirements for the field, then *string2* replaces *string2* in the field.

All occurrences of *string1* are highlighted in the text. (Enter the RESET FIND command to turn off the highlighting.)

Use of CHANGE with no parameters is equivalent to RCHANGE (assigned to function key **PF6** by default) and repeats the last change command executed, including all its specified parameters.

To find and optionally change the next occurrence of *string1*, execute a combination of RFIND (assigned to PF5 by default) followed by RCHANGE.

If *string1* and *string2* are character strings of unequal length, then the following will occur:

- If the length of *string1* is greater than the length of *string2*, then words to the right of the replaced string will be shifted left.

If parameter TEXT is specified and more than one blank exists before a word to the right of the replaced string, then additional blanks are inserted to maintain that word's position in the record.

If parameter DATA is specified (default), then pad characters, defined by the PAD option, will be inserted at the end of the character field data.

- If the length of *string1* is less than the length of *string2*, then words to the right of the replaced string will be shifted right. Note, however, that CHANGE will not increase the length of formatted data beyond its defined maximum field length.

  If parameter TEXT is specified, multiple, consecutive blanks are absorbed to leave at least one blank between each word. Only if no blanks are eligible to be absorbed will text to the right of the replaced string be shifted right.

If a structure has been applied to the edited records and a CHANGE operation alters the length of a record or updates a field referenced by a USE *record_type* WHEN condition, then the IDENTIFY command may be used to re-evaluate the record type (RTO) assigened to the changed record.

When the duration of a CHANGE or RCHANGE execution exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the oportunity to interrupt the operation using the Attn key.

The FORMAT of the SDE display affects the execution of CHANGE.

Unformatted Multi or Single Record Display (CHAR or UNFMT):

By default, a character compare for the supplied search string is performed against the entire length of unformatted data records.

The prevailing BOUNDS left and right column values define the area of the record within which the search occurs. i.e. the matched data must begin at or after the left bound and not exceed the right bound.

The BOUNDS columns may be overridden using *pos1* and *pos2* positional parameters.

field_col and #ALL parameters are not applicable to these display formats and are ignored.

Formatted Multi or Single Record Display (VFMT or FMT):

For formatted records, the search string is compared against **individual fields** that have been selected for display in the formatted data record. (See SELECT)

Fields are searched from left to right in the order that they appear in the display. This is true regardless of the order in which field columns are specified on the CHANGE command, or the order in which fields are encountered within the unformatted record.

The prevailing BOUNDS left and right column values define which of the fields within the formatted records are eligible to be searched. i.e. Fields are eligible only if they exist within the left and right bounds when applied to the expanded record data (which is not necessarily the same as the unformatted record data.)

Where a BOUNDS column occurs within a field definition, then the following rules apply:

1. For character data type fields, only the area of the field that falls within the BOUNDS columns is eligible for the search.
2. For numeric data type fields, the field is not eligible to be included in the search.

If one or more field columns are specified on the CHANGE command (using *field_col* or *field_col1*:*field_col2*) that do not reference at least one field defined by the BOUNDS columns as being eligible for search, then the following error message is returned:

```
ZZSD280E No fields selected within the current bounds for the CHANGE command.
```

If a field column is specified on the CHANGE command that is not part of the display (e.g. a group field or a field that has been removed from display by a SELECT command), then the following error message is returned:

```
ZZSD179E Data element field_col is not selected in the current view
    of record type record-type of structure struct_name.
```

For character data type fields, a string compare is performed. For numeric data type fields (binary, packed decimal, floating point, zoned, etc.), then the following will occur:

1. If *pos1*, *pos2* positional parameters are **not** specified, the search string is interpreted as a signed numeric value and an arithmetic compare is performed against the field's formatted numeric value.

   The length and data type of the numeric field, and the number of digits in the search value are not significant. e.g.

   ```
   CHANGE  67  23
   ```

   Finds numeric fields with value "67" (e.g. "0067", "67.00", "0.0670E+03") and character fields containing the string "67" (e.g. "167 Baker Street") and replaces with the value "23" using the appropriate data type

   If the search string is non-numeric, then numeric data type fields are not searched. Therefore, to bypass searching numeric data type fields, explicitly set the search string to be character or hex using 'string', C'string' or X'string' formats respectively. e.g.

   ```
   CHANGE  '67'    '23'
   ```

```
CHANGE  C'67'    C'23'
CHANGE  X'F6F7'  X'F2F3'
```

Finds only character fields containing the string "67" and replaces with character string "23".

2. If *pos1*, *pos2* positional parameters are specified, the search string is interpreted as being a character string and so a string compare is performed against the unformatted representation of the field data for fields falling entirely or partly within the range of record positions.

   Although the range of positions may span a number of fields, the search is still performed against individual fields within the range. i.e. a match for the search string will not occur for data that spans a field boundary.

   A match for the search string may occur on just part of the unformatted data representation of a numeric field. e.g.

   ```
   CHANGE   476  850    21  100
   ```

   This will find a match in any numeric field where unformatted representation of the data contains the string "476" and replaces it with "850". (e.g. a zoned decimal field with value "14760" would become "18500".)

Any match of the search string on data in a numeric field will highlight the entire formatted display of the field. If the numeric field is also the current, identified occurrence of the search string, the cursor is positioned at the start of the formatted numeric field display.

If replacing a string in a character field would exceed the defined maximum length of the field then no update will take place.

Parameters:

op

A relational operator used in the compare operation which determines the relationship that the data must have with the search string in order for it to be identified as a successful match.

Valid values for *op* are as follow:

Operator	Description
EQ	Data must be equal to *string1*. (Default)
NE	Data must be not equal to *string1*.
GT	Data must be greater than *string1*.
GE	Data must be greater than or equal to *string1*.
LT	Data must be less than *string1*.
LE	Data must be less than or equal to *string1*.

If a character string compare is performed, the EBCDIC values assigned to characters in the search and data strings determine the relationship (equal to, greater than or less than) between the two strings.

string1

The CHANGE search string. The seach string may be any of the following:

◊ An unquoted numeric value. The search string is treated as a numeric value when a numeric field is searched in a formatted record view. In all other cases a numeric search string is treated as a character string.

◊ An unquoted character string containing no commas or blanks. The search for the character string will be case-insensitive so that uppercase and lowercase characters are treated as being the same.

◊ A character string enclosed in single (') or double (") quotation marks. The search string may contain embedded commas and blanks and the character string will be case-insensitive. A string enclosed in quotes may still be interpreted as a numeric value.

Two adjacent quotation mark characters that are embedded in a search string which is enclosed by the same quotation mark characters, will be treated as a single occurrence of the character. e.g.

```
CHANGE  'Jim O''Brien'  'James O''Brien'
```

Find the character string "Jim O'Brien" and replaces it with "James O'Brien".

◊ A character string enclosed in single (') or double (") quotation marks with the prefix C. This is equivalent to specifying a quoted search string but that the string search will be case-sensitive. (e.g. C'Book')

◊ A hexadecimal string enclosed in single (') or double (") quotation marks with the prefix X.

◊ A picture string enclosed in single (') or double (") quotation marks with the prefix P.

Picture strings use special characters to represent a generic group of characters as described below. Any character in a picture string that is not one of these special characters is untranslated.

String	Description
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.

◊ A regular expression enclosed in single (') or double (") quotation marks with the prefix R.

Regular expressions use special characters for complex pattern matching. See Regular Expressions in Text Editor documentation for detailed description.

VALID

Intended for use with formatted records, VALID will search fields for valid data. i.e. data that satisfies the field's assigned data type.

INVALID

Intended for use with formatted records, INVALID will search fields for invalid data. i.e. data that does not satisfy the field's assigned data type.

ANYVALUE

The keyword ANYVALUE (ANYV) may be specified in place of the search string (string1) on a change operation in order to set a new value (specified as string2) irrespective of the current value.

string2

The CHANGE replace string used to replace *string1|VALID|INVALID|ANYVALUE*. The replace string may be any of the following:

◊ An unquoted numeric value. The replace string is treated as a numeric value when it replaces a value in a numeric field in a formatted view. In all other cases a numeric replace string is treated as a character string.

◊ An unquoted character string containing no commas or blanks. String2 may be null (").

◊ A character string enclosed in single (') or double (") quotation marks that may contain embedded commas and blanks. String2 may be null (C").

Two adjacent quotation mark characters that are embedded in a replace string which is enclosed by the same quotation mark characters, will be treated as a single occurrence of the character. (See example under *string1*.)

◊ A character string enclosed in single (') or double (") quotation marks with the prefix C. This is equivalent to specifying a quoted search string but that the string search will be case-sensitive. (e.g. C'Book')

◊ A hexadecimal string enclosed in single (') or double (") quotation marks with the prefix X.

◊ A picture string enclosed in single (') or double (") quotation marks with the prefix P.

Where a picture string is used as *string2* of a CHANGE command, then it must be the same length as the *string1* and may only contain the following special characters. Any character in a picture string that is not one of the special characters supported by picture strings, is untranslated.

String	Description
P'='	Same as the corresponding character in the search string.
P'<'	Change the corresponding character in the search string to lowercase.
P'>'	Change the corresponding character in the search string to uppercase.

◊ Where *string1* is a regular expression, *string2* may contain tag references to tagged sub-expressions of the regular expression search pattern defined by *string1*.

See Regular Expressions in Text Editor documentation for details on tagged sub-expression reference.

ANY

All record-types are included in the search provided they are not suppressed. See VIEW , VBASE , and V/V+/V-line-commands to suppress and unsuppress record-types.

FOCUS

Only the default record type is included in the search. This is normally the type of record at the top of the screen or at the cursor location. This option was the default in earlier versions of FileKit and can be made so again using the RTSCOPE option or via the settings panel (=0.4).

ALL

Where field conditions are satisfied, change all occurrences of *string1* to *string2*. A message is displayed providing the number of occurrences of *string1* that have been changed to *string2*. If NX is not specified, all excluded records that contain an occurrence of the *string1* are made visible whether or not *string1* is replaced.

FIRST

Search forwards from the top of the file data (i.e. the first position of the first data record) to find the first occurrence of *string1* and attempt to replace it with *string2*.

`LAST`

Search backwards from the bottom of the file data (i.e. the last position of the last data record) to find the last occurrence of *string1* and attempt to replace it with *string2*.

`NEXT`

Search forwards from the current cursor location to find the next occurrence of *string1* and attempt to replace it with *string2*. If the cursor is not within the window's data display area, the search begins at the first position of the first visible or excluded record within the display area that is of the default record type.

`PREV`

Search backwards from the current cursor location to find the previous occurrence of of *string1* and attempt to replace it with *string2*. If the cursor is not within the window's data display area, the backwards search begins at the first position of the first visible or excluded record within the display area that is of the default record type.

`CHARS`

For non-numeric search strings only, CHARS indicates that a successful match occurs if *string1* is found anywhere within the data being searched.

`PREFIX`

For non-numeric search strings only, PREFIX indicates that a successful match only occurs if *string1* is found at the start of a word within the data being searched. i.e. the matched text must precede an alphanumeric character and either be preceded by a non-alphanumeric character or be at the start of a line or field.

`SUFFIX`

For non-numeric search strings only, SUFFIX indicates that a successful match only occurs if *string1* is found at the end of a word within the data being searched. i.e. the matched text must be preceded by an alphanumeric character and either precede a non-alphanumeric character or be at the end of a line or field.

`WORD`

For non-numeric search strings only, WORD indicates that a successful match only occurs if *string1* is found to be a complete word within the data being searched. i.e. the matched text must either be preceded by a non-alphanumeric character or be at the start of a line or field, and either precede a non-alphanumeric character or be at the end of a line or field.

`EX`
`X`

Search EXCLUDED data records only. By default, both visible and EXCLUDED records are searched.
CHANGE does not search records that are NOTSELECTED or SUPPRESSED.

`NX`

Search only visible data records (i.e. not EXCLUDED). By default, both visible and EXCLUDED records are searched.
CHANGE does not search records that are NOTSELECTED or SUPPRESSED.

`pos1`

The first position of a range of positions within the data record to be searched.

For formatted records, this is a position in the expanded record . Only those fields, or parts of fields, that fall within the position range will be searched. Fields will be searched in the order that they occur within the display area.

pos1 may be a positive or negative integer value (not zero) and must be a value that is less than or equal to the maximum length of the data records or, for formatted record data, the length of the expanded record.

A negative value represents a position in the record relative to the end of the record. Therefore, where position 1 references the 1st character in the record, position -1 references the last character.

For all display formats, *string1* is treated as being non-numeric and the search is actioned on the character representation of the record data.

`pos2`

The last position of a range of positions within the data record to be searched.

Like *pos1*, *pos2* may be a positive or negative integer value (not zero). If *pos1* references a position within the record data which is higher than that referenced by *pos2*, then the *pos1* and *pos2* values are swapped.

If *pos2* is greater than the maximum length of the data records or, for formatted record data, greater than the length of the expanded record, then *pos2* is set equal to the maximum (or expanded) record length.

Default is *pos1* plus the length of the search string minus 1.

`#ALL`

Search all eligible field columns in the current formatted display.

`field_col`

An individual field column to be searched within a formatted display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID). If a field name is used, enclosing parentheses are **mandatory** Field names are automatically converted to a field reference and Referencing the same field column more than once will not cause an error.

If the field is an array, then all elements of the array are searched. To search an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array. An entry must exist for each dimension of the array. e.g. RoomSize(6,2,4) - a three dimensional array.

Specification of multiple field columns must either be enclosed in parentheses and/or separated by commas. The field columns may be specified in any order, however, the data is always searched from the first column in the display to the last.

Field column search specifications may be a combination of individual field columns and columns ranges. e.g. (JobID #6 Tax_Reference #12 #15:#20).

A search is only performed on field columns that are selected for display. Therefore, any field column specified on the CHANGE command that has not been selected for display, will be ignored.

`field_col1:field_col2`

The first and last fields of a range of field columns to be searched within a formatted record display display.

field_col1 and *field_col2* must be separated by ":" (colon) and if *field_col1* occurs after *field_col2* in the data record, then the values are swapped. *field_col1* and *field_col2* have the same specifications as *field_col*.

Specification of multiple field column ranges must either be enclosed in parentheses and/or separated by commas. Field column search specifications may be a combination of individual field columns and field columns ranges. e.g. (FirstName:LastName, #2, Salary:Bonus, EmpNo, #10:#15). If field names are used, enclosing parentheses are **mandatory**.

`.name1`

A label name identifying the first record of a range of data records to be searched. The preceding "." (dot) is mandatory. Default is .ZFIRST.

`.name2`

A label name identifying the last record of a range of data records to be searched. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then the order is reversed.
If CHANGE PREV is executed and *.name1* is specified, the default is .ZFIRST. Otherwise the default is .ZLAST.

`DATA`

DATA (as opposed to TEXT) indicates that records are to be treated as data so that CHANGE performs no special treatment of any multiple, consecutive blanks that follow the replaced string.
DATA is default.

`TEXT`

TEXT (as opposed to DATA) indicates that records are to be treated as formatted text so that CHANGE operaties identically to the ISPF edit style CHANGE command. i.e blanks following the replaced string may be absorbed or added in order to maintain text positions.

Examples:

```
change  22.3  303.25
```

Search all fields in the display belonging to the default record type for the next occurrence of the search string/value 22.3. Numeric fields are tested for numeric value 22.3, character fields are tested for character string "22.3" anywhere within the field. Where the field width and, for numeric fields, the precision and scale definitions are satisfied by the replacement string/value 303.25, text update will takes place.

```
change  all  'ing' 's'  suffix  ex  (DsName:ProcLib, JobStep)
```

Search selected character fields in the display that belong to excluded records of the default record type, for all occurrences of the word suffix string "ing" and replace the text with "s".

```
change  all  anyvalue ' '  (Remarks)
```

Reset all values in the "Remarks" field to blank.

See Also:

EXCLUDE FIND ONLY RCHANGE RFIND SELECT

CHAR

Syntax:

```
>>-- CHAR -------------------------------------------------------------------><
```

Description:

Display records or record segments in multi-record, unformatted character view by temporarily disabling the record-type (RTO) assigned to each record or record segment.

Compare with FORMAT CHAR which displays the records/segments in multi-record, unformatted view but does not disable the assigned record-type.

Note that unformatted records have record type "UnMapped", unformatted segments of a segmented record have record type "UnMappedSeg".

See Also:

FORMAT MAP (FMT) UNFMT VFMT ZOOM

CLIPBOARD

Description:

SDE CLI command, CLIPBOARD, performs the same operation as the CBLe CLI command CLIPBOARD. See CLIPBOARD in CBLe Text Edit documentation.

CMSG

Description:

SDE CLI command, CMSG, performs the same operation as the CBLe CLI command CMSG. See CMSG in CBLe Text Edit documentation.

COPY

Syntax:

```
                    +--- FROM --- 1 -----------+   +--- TO ---- * --------+
                    |                          |   |                      |
>>- COPY -- fileid -+--------------------------+-+-+----------------------+-+->
                    |                          | | |  (1)                 | |
                    | +- FROM -+               | | | +- TO --+            | |
                    | |        |               | | | |       |            | |
                    +-+--------+- start_line  -+ | +-+-------+- end_line -+ |
                                                 |                          |
                                                 |           +- * --------+ |
                                                 |           |            | |
                                                 +----- FOR -+------------+-+
                                                             |            |
                                                             +- n_lines --+

          +- AFTER---+
          |          |
 >--------+----------+--+---------+-------------------------------------------><
          |          |  |         |
          +- BEFORE -+  +- .name -+
```

Note:

1. TO is mandatory if *start_line* is not specified.

Description:

Copy lines from an existing sequential or VSAM data set, PDS/PDSE library member or HFS file into the focus SDE Edit view.

The line within the focus SDE edit data which identifies the target of the copy, is specified by a line label name. If no line label name is specified, the target line is the first line containing line (prefix area) command "A" (AFTER) or "B" (BEFORE), otherwise the focus line.

COPY supports copy of lines belonging to files that can be loaded entirely in available storage. Beware that if an attempt is made to copy lines from a file which cannot be loaded in available storage, a shortage of storage error will be displayed and the focus SDE edit view will be closed without saving any outstanding changes.

Specify COPY with no parameters to open the SDE COPY File panel .

Parameters:

fileid

Specifies the fileid of the source file from which records are to be copied.

If *fileid* is a less than 8 characters in length and is a valid member name, the target file will be a member of member name *fileid* belonging to the same PDS/PDSE library referenced in the focus SDE view. If the focus SDE view does not display a library member, the *fileid* will be treated as an HFS file within the current HFS working directory.

If *fileid* includes a volume id, then the source file may be a cataloged or uncataloged data set or PDS/PDSE library which exists in that volume's VTOC. e.g. VOLWKA:DEV.UNCATLG.FILE.

[FROM] *start_line*

Identifies the first record number in the group of records belonging to *fileid* to be copied into the focus SDE edit view. Default is record 1.

[TO] *end_line*

Identifies the last record number in the group of records belonging to *fileid* to be copied into the focus SDE edit view.

If no *start_line* value has been specified, the "TO" keyword is required if *end_line* is to be specified. If *start_line* has been specified, then *end_line* must be greater than the *start_line* value.

Default is "*" (asterisk), the last record belonging to *fileid*.

FOR *n_lines*

Specifies the number of consecutive lines in the group of records belonging to *fileid* to be copied into the focus SDE edit view.

All remaining *fileid* records are copied if the *n_lines* value is greater than the number of remaining *fileid* records.

Default is "*" (asterisk), all remaining *fileid* records.

AFTER | BEFORE

Identifies whether lines are to be copied after or before the target line in the focus SDE view.
Default is AFTER.

.*name*

A label name identifying the target line of copy.

If not specified, then the first line containing line (prefix area) command "A" or "B" is the target line, otherwise the focus line.

Examples:

```
copy    SYSA.CBL.MACLIB(FS23T901)  22  *  before  .MAC
```

Copy records from line 22 to the end of the file from member "SYSA.CBL.MACLIB(FS23T901)" before the line in the focus SDE view identified by line label ".MAC".

See Also:

CLIPBOARD CREATE REPLACE

CREATE

Syntax:

```
>>-- CREate -----+----------+------+---------------------+-------------------><
                 |          |      |                     |
                 +- fileid -+      +- .name1 -- .name2 --+
                                     (1)
```

Note:

1. If a group of lines are not specified using line label names, then a group of lines must be specified using one of the following line (prefix area) commands:
   - **C** or **CC** (Copy), if the group of lines in the current file is to be preserved following successful execution of CREATE.
   - **M** or **MM** (Move), if the group of lines in the current file is to be deleted following successful execution of CREATE.

Description:

Create a new sequential or VSAM data set, PDS/PDSE library member or HFS file, containing a group of lines extracted from the current SDE edit or browse view.

If the target file already exists, then the following message is returned:

```
  ZZSD229E File fileid already exists.
```

If output is to a new member of an existing PDS/PDSE library or to an HFS file, the target file will be created automatically created. When CREATE defines a new HFS file, the permission bits are set to 740 (rwxr-----).

If output is to a non-existant data set or to a member of a non-existant PDS/PDSE library, the Allocate Non-VSAM panel is displayed in order to allocate the new data set.

Beware that the specified *fileid* character string must not be "LIST" (minimum abbreviation LI), "STRUCTURE" (minimum abbreviation "STRUCT") or "VIEW" (minimum abbreviation VI) which conflict with SDE command verbs CREATE LIST, CREATE STRUCTURE and CREATE VIEW.

Specify CREATE with no parameters to open the SDE CREATE/REPLACE file panel.

Parameters:

fileid

Specifies the fileid of the target file to be created.

If *fileid* is a less than 8 characters in length and is a valid member name, the target file will be a member of member name *fileid* belonging to the same PDS/PDSE library referenced in the focus SDE view. If the focus SDE view does not display a library member, the *fileid* will be treated as an HFS file within the current HFS working directory.

.*name1*

A label name identifying the first line of the group of lines to be copied to the target file.

If not specified, then the group of lines must marked using "C" or "M" line (prefix area) commands.

.*name2*

A label name identifying the last line of the group of lines to be copied to the target file. If *.name1* has been specified, *.name2* is mandatory.

Examples:

```
create    DEV.USER223.COB.COPY(QX95R001)  .GRBEG .GREND
```

Create new library member "DEV.USER223.COB.COPY(QX95R001)" containing records in the current SDE view which fall between defined line label names .GRBEG and .GREND inclusively.

See Also:

COPY REPLACE

CREATE LIST

Syntax:

```
>>-- CREate LIst -- list_name ------- FROM fileid --------------------------->


 >-- USING --+--------------+- struct_name ---+---------------------+------->< 
             |              |                 |                     |
             +- STRUCTure --+                 +-- TITLE title_text --+
```

Description:

Create an in storage list from records within a structured data set, that may then be displayed in a FileKit List window via the DISPLAY LIST command.

Only records of one record type may be accurately displayed within a list window. By default, the first record type object (RTO) within the SDO is used to map **all** the records within the file when generating the list.

A list column entry is generated for every field name defined by the RTO. Therefore, data within named fields that are part of a named group of fields will be repeated in different columns.

The List window displaying the list data will support most of the standard list window features as follow:

- Field Descriptor Block (FDB)
- Edit View
- Selecting, Sorting and Filtering
- Sorting with the Cursor

On exiting the List window display, the in-storage list is destroyed and storage is freed.

Parameters:

`list_name`
The name to be assigned to the in-storage list data for reference in a DISPLAY LIST command.

`FROM fileid`
Identifies the fileid of the data set containing the structured records.
fileid may be the DSN of a sequential or VSAM data set or the DSN of a PDS/PDSE member.

`USING (STRUCTURE) struct_name`
Identifies the SDO to be applied to the data records. *struct_name* is the SDF fileid assigned to the SDO in a CREATE STRUCTURE command.

`TITLE title_text`
Specifies the text to be displayed in the List window's title bar.

Examples:

```
<sd create  list  AMLIST   from CBL.AMSUPP.DA   using CBL.FILEKIT.SDO(AMMAP)   title "Contact Address Details"
```

Generate in-storage list AMLIST.

```
<sd display list  AMLIST
```

Open a list window for the in-storage list AMLIST.

See Also:

DISPLAY LIST   CREATE STRUCTURE

CREATE STRUCTURE

Syntax:

```
>>-- CREATE STRUCTure --- struct_name --------------------------------------->

 >--+----------------------+--+-----------------------------------+------------>
    |                      |  |                                   |
    +- TITLE short_title -+  +- DESCRiption long_description ---+

 >--+---------- | XRef Definition | ----------+------------------------------>
    |                                         |
    +---------- | CopyBook Definition | ------+
    |                                         |
    +---------- | DB2 Definition | -----------+
    |                                         |
    +---------- | Direct Definition | --------+

 >--+-----------------------------------------------------------------+------>
    |                                                                 |
    |                     +- NOTHING ----------------------------+      |
    |                     |                                     |      |
    +--- REPLACING    ( --+-------------------------------------+- ) -+
                          |          +------------------------+ |
                          |          v                        | |
                          +--------+-+- text_1 - BY - text_2 -+-+
                          |        |
                          +- ONLY -+

 >--+-----------------------------------------------------------------+------>
    |                                                                 |
    +--- SYSADATA     ( ---- sysadata_dataset_name --------------- ) -+

 >--+-----------------------------------------------------------------+------>
    |                                                                 |
    +--- COPYBOOKProc ( --+- COMPILER -+-------------------------- ) -+
                          |            |
                          +- INTERNAL -+

 >--+-----------------------------------------------------------------+------>
    |                                                                 |
    | +-------------------------------------------------------------+ |
    | v                                                             | |
    +-+- OPTion       ( --+- colwidth_command --+--------------- ) -+-+
                          |                     |
                          +- ccolour_command ---+
                          |                     |
                          +- rcolour_command ---+

 >--+-----------------------------------------------------------------+------>
    |                                                                 |
    |                     +---------------------+                     |
    |                     v                     |                     |
    +--- INITCMD      ( --+- sd_command --------+----------------- ) -+

    +--- NOREPlace -+                       +-- CASE(IGNORE)  ---+
    |               |                       |                    |
 >--+---------------+---+---------------+---+--------------------+------------><
    |               |   |               |   |                    |
    +--- REPlace ---+   +-- TEMPorary --+   +-- CASE(RESPECT)  --+
```

XRef Definition:

```
 |----+--------------------------------------------------+-------------------->
      |                +--------------------+             |
      |                v                    |             |
      +- LIBrary -- (---- copybook_library -+- ) --------+

    +------------------------------------------------------+
    v                                                      |
 >------ RECord  -- ( -- | RecordType Definition | -- ) ---+-----------------|
```

RecordType Definition:

```
|--+- NAME -------- record_type --+------------------------------------+--+->
   |                              |                                    |  |
   |                              |          +- + (plus) --+           |  |
   |                              |          |             |           |  |
   |                              +- OFFset -+-------------+- n_bytes -+  |
   |                                         |             |              |
   |                                         +- - (minus) -+              |
   |                                                                      |
   |                                                                      |
   +- STARTField -- data_element -----------------------------------------+
     (1)

              +- COBol -+                       +- LEVEL 1 ------------+
              |         |                       |                      |
>-- SOURCE -+-+---------+--- source_fileid -----+----------------------+--+->
            | |         |                       |                      |  |
            | +- PL1 ---+                       +- LEVel level_number -+  |
            |                                                             |
            +--- HLAsm ----- source_fileid -------------------------------+
            |                                                             |
            |                                                             |
            +--- ADAta ------ adata_fileid -------------------------------+
            |                                                             |
            |                                                             |
            +--- SDO ---------- sdo_fileid -------------------------------+
            |                                                             |
            |                                                             |
            +--- STRUCTure (-- | Record Definition | --) -----------------+
            |                                                             |
            |                                                             |
            |                +------------------+                         |
            |                v                  |                         |
            +--- SYMNAMes  (-+- DFSORT symbols -+-) ----------------------+
            |                                                             |
            |                                                             |
            |                +------ , ---------+                         |
            |                v                  |                         |
            +--- FIELDS -----+- data_element ---+--- FROM -- record_type -+


       +-- PRImary ----+
       |               |
>---+--+---------------+----+-------+-+- WHEN -+-- expression ------+-------|
    |  |               |    |       | |        |                    |
    |  +-- SECondary --+    +- USE -+ +- IF ---+                    |
    |                                                               |
    +----- DEFault -------------------------------------------------+
```

(1) STARTFIELD may be used only with SOURCE COBOL, PL1 or HLASM.

CopyBook Definition:

```
         + COBOL +  +-------------------- , ------------------------------+
         |       |  v                                                     |
|- FROM -+-------+--+-----------------------------------+- copybook_name -+->
         |       |  |                                   |
         + PL1 --+  +-+---------+-- 1 --+- level1_name -+
         |       |  | +- LEVel -+       |
         + ADAta +  |                   |
         |       |  +- RECord -+--------+
         + HLAsm +             +- TYPe -+

>--------+-----------------------------+------------------------------------|
         |          +------ , -------+ |
         |          v                | |
         +- SELECT --- level1_entry -+-+
```

DB2 Definition:

```
|- FROM -- DB2 -+---------+--+-- table-name --+--+------------------------+-|
                |         |  |                |  |                        |
                +- (ssn) -+  +-- view-name ---+  +- ( --| DB2 Opts |-- ) -+
```

DB2 Opts:

```
|--+------------------------+--+-----------------------------------------+-->
   |                        |  |                                         |
   +- WHERE (where-clause) -+  +- ORDER -+------+--+-- (orderby-clause) -+
             |          |      |         |      |  |
             +--- ? ----+      |         +- BY -+  |
                               +- SORT ------------+

                                                  +- COMMITONCLEANSAVE -+
                                                  |                     |
>--+------------------------------------------+---+---------------------+--->
   |                                          |   |                     |
   +- WITH -+- CS ----------------------------+   +- COMMITONSAVE ------+
            |                                 |   |                     |
            +- UR ----------------------------+   +- COMMITONEXIT ------+
            |                                 |
            +- RR --+-+-----------------------+
            |       | |                       |
            +- RS --+ +- KEEP -+-- EXclusive -+
                               |              |
                               +-- UPDate ----+
                               |              |
                               +-- SHR -------+

                                                       + PROTECTPRIMEKEY -+
                                                       |                  |
>--+----------------+---+---------+--+--------------+--+------------------+->
   |                |   |         |  |              |  |                  |
   +- COMMITONLOAD -+   +- AUDit -+  +- SKIPLocked -+  +--- EDITPRIMEKEY -+

>--+---------------------------------+--------------------------------------|
   |                                 |
   +---- SELect (select_syntax) -----+
```

Direct Definition:

```
      +---------------------------- , -----------------------------+
      |                                                            |
      |                      (2) +------------------------------+  |
      v                          v                              |  |
|- ( -+--| Record Definition |---+------------------------------+--+- ) ----|
                                 |                              |
                                 +--| Expand Clause |-----------+
                                 |                              |
                                 +-+-------+- WHEN expression --+
                                   |       |
                                   +- USE -+

       +- NAMES(ASM) -------------------+
       |                                |
|------+--------------------------------+-----------------------------------|
       |                                |
       +- NAMES( -+- COBOL -------+- ) -+
                  +- DB2 ---------+
                  +- PL1 ---------+
```

(2) **Expand Clause** and **WHEN** may each be specified once per record definition.

Record Definition:

```
                                  +--------- , ----------+      +- EBCdic +
                                  v                      |      |         |
|-- record_type -- STRUCTure - ( -+-| Field Definition |-+- ) --+---------+-|
                                                                |         |
                                                                +- ASCii -+
```

Expand Clause:

```
|-- EXPAND -- ( EXPLOC (expr) --- EXPSIZE (expr) --+------------------+- ) -|
                                                   |                  |
                                                   +-- WHEN (expr) ---+
```

Field Definition:

```
|--+--------------+---| Data Definition |---+----------------------------+-->
   |              |                         |                            |
   +- field_name -+                         +-- PICture picture_string --+

                                                      +-- EBCdic -+
                                                      |           |
>--+----------------------------------------------+---+-----------+--------->
   |                                              |   |           |
   |               + ----------- , ---------+     |   +-- ASCii --+
   |               v                        |     |
   +- DIMensions( ---+-- dim-- -----------+-+- ) -+
                     |                    |
                     +-- (min,max,d_name)-+

                  +- (TRAILing,INCluded) -----------------+
                  |                                       |
>--+----- SIGNed -+---------------------------------------+-+--------------->
   |              |                                       | |
   |              |                     +- ,INCluded --+  | |
   |              |                     |              |  | |
   |              +- ( -+- LEADing --+--+--------------+--+ |
   |                    |            |  |              |    |
   |                    +- TRAILing -+  +- ,SEParate --+    |
   |                                                        |
   +--- UNSIGNed -------------------------------------------+

   +-- ALIGNed ---+
   |              |
>--+--------------+---------------------------------+-----------+----------->
   |              |                                 |           |
   +-- UNALIGNed -+                                 +-- ZEROs --+

>--+------------------+--+-----------------------------------------------+--|
   |                  |  |                             +---- , ------+   |
   +- REMarks remark -+  |                             v             |   |
                         +- ENUMeration -+--------+- ( -- enumerator +) -+
                                         |        |                      |
                                         +- enam -+                      |
                                         |                               |
                                         +- enam ------------------------+
```

Data Definition:

```
|---+- BINTeger ---+-------------------------------------------------+------|
    |               |                                                |
    |               +- ( n_bits   ) ---------------------------------+
    |                                                                |
    +- BIT ---------+------------------------------------------------+
    |               |                                                |
    |               +- ( n_bits   ) ---------------------------------+
    |                                                                |
    +- CHARacter ---+------------------------------------------------+
    |               |                                                |
    |               +- ( n_bytes  ) ---------------------------------+
    |                                                                |
    +- CHARVarying -+------------------------------------------------+
    |               |                                                |
    |               +- ( n_bytes  ) ---------------------------------+
    |                                                                |
    +- CHARZ -------+------------------------------------------------+
    |               |                                                |
    |               +- ( n_bytes  ) ---------------------------------+
    |                                                                |
    |               +- ( Decimal ) -+                                |
    |               |               |                                |
    +- DATE --------+---------------+--------------------------------+
    |               |               |                                |
    |               +- ( Binary  ) -+                                |
    |               |               |                                |
    |               +- ( Catalog ) -+                                |
    |               |               |                                |
    |               +- ( VTOC    ) -+                                |
    |                                                                |
    +- DECimal -----+------------------------------------------------+
    |               |                                                |
    |               +- (precision+-------------+- ) -----------------+
    |                            |             |                     |
    |                            +- , scale ---+                     |
    |                                                                |
    +- FIXed -------+------------------------------------------------+
    |               |                                                |
    |               +- (precision+-------------+- ) -----------------+
    |                            |             |                     |
    |                            +- , scale ---+                     |
    |                                                                |
    +- FLOATBin ----+------------------------------------------------+
    |               |                                                |
    |               +- ( n_bytes  ) ---------------------------------+
    |                                                                |
    +- FLOATDec ----+------------------------------------------------+
    |               |                                                |
    |               +- ( precision ) --------------------------------+
    |                                                               |
    +- FLOAThex ----+-----------------------------------------------+
    |               |                                               |
    |               +- ( n_bytes  ) --------------------------------+
    |                                                               |
    +- HEXadecimal -+-----------------------------------------------+
    |               |                                               |
    |               +- ( n_bytes  ) --------------------------------+
    |                                                               |
    +- HEXVAR ------+-----------------------------------------------+
    |               |                                               |
    |               +- ( n_bytes  ) --------------------------------+
    |                                                               |
    +- INTeger -----+-----------------------------------------------+
    |               |                                               |
    |               +- ( n_bytes  ) --------------------------------+
    |                                                               |
    +- IPaddress ---+-----------------------------------------------+
    |               |                                               |
    |               +- ( n_bytes  ) --------------------------------+
    |                                                               |
    +- PCHAR -------+-----------------------------------------------+
    |               |                                               |
    |               +- ( pl1_picture_string  ) ---------------------+
    |                                                               |
    +- PFIXED ------+-----------------------------------------------+
    |               |                                               |
    |               +- ( pl1_picture_string  ) ---------------------+
    |                                                               |
    +- PFLOAT ------+-----------------------------------------------+
    |               |                                               |
    |               +- ( pl1_picture_string  ) ---------------------+
    |                                                               |
    |                   +-------- , ------------+                   |
    |                   v                       |                   |
    +- STRUCTure --- ( ---| Field Definition |--+--- ) -------------+
    |                                                               |
    |               +- ( Decimal ) -+                               |
    |               |               |                               |
    +- TIME --------+---------------+-------------------------------+
    |               |               |                               |
    |               +- ( Binary  ) -+                               |
    |               |               |                               |
    |               +- ( Stck    ) -+                               |
    |               |               |                               |
    |               +- ( STCKE   ) -+                               |
    |               |               |                               |
    |               +- ( Unix    ) -+                               |
    |               |               |                               |
    |               +- ( DECIMAL2) -+                               |
    |               |               |                               |
    |               +- ( DECIMAL3) -+                               |
    |                                                               |
    |               +- ( TIMEDec ) -+                               |
    |               +- ( Decimal ) -+                               |
    |               |               |                               |
    +- TIMESTamp ---+---------------+-------------------------------+
    |               |               |                               |
    |               +- ( TIMEBin ) -+                               |
    |               |               |                               |
    |               +- ( Binary  ) -+                               |
    |               |               |                               |
    |               +- ( Stck    ) -+                               |
    |               |               |                               |
    |               +- ( STCKE   ) -+                               |
    |               |               |                               |
    |               +- ( Hfsdir  ) -+                               |
    |               |               |                               |
    |               +- ( SMf     ) -+                               |
    |               |               |                               |
    |               +- ( TIMEDEC2) -+                               |
    |               +- ( DECIMAL2) -+                               |
    |               |               |                               |
    |               +- ( TIMEDEC3) -+                               |
    |               +- ( DECIMAL3) -+                               |
    |                                                               |
    |                   +-------- , ------------+                   |
    |                   v                       |                   |
    +- UNION ------- ( ---| Field Definition |--+--- ) -------------+
    |                                                               |
    +- VARCHAR -----+-----------------------------------------------+
    |               |                                               |
    |               +- ( max_bytes+-------------+--------------+- )-+
    |                             |             |              |    |
    |                             +- , len_bytes-+- , exclusive-+   |
    |                                                               |
    +- VARHEX ------+-----------------------------------------------+
    |               |                                               |
    |               +- ( max_bytes+-------------+--------------+- )-+
    |                             |             |              |    |
    |                             +- , len_bytes-+- , exclusive-+   |
    |                                                               |
    +- XVARCHAR ----+-----------------------------------------------+
    |               |                                               |
    |               +- ( max_bytes, len_field) ---------------------+
    |                                                               |
    +- ZONEd -------+---------------------------------------------+
                    |                                             |
                    +- (precision+-------------+- ) -----------------+
                                 |             |
                                 +- , scale ---+
```

Description:

Create a Structure Definition Object (SDO) to be used by the FileKit Structured Data Environment (SDE) and, if TEMPORARY is not specified, save it to a Structure Definition File (SDF).

An SDO contains the names and attributes of record types and fields in a dataset or the rows and columns in a DB2 table.

The source of structure definition information can be:

- One or more COBOL copybooks containing data element declarations.
- One or more PL/1 copybooks containing data element declarations.
- One or more HLASM (assembler) copybooks or macros containing DSECT declarations.
- An ADATA file containing the SYSADATA output of a COBOL or PL/1 compilation or an HLASM assembly.
- An existing Structure Definition Object (SDO).
- One or more DFSORT style SYMNAMES datasets.
- A FileKit structure declaration using the syntax described in *Direct Definition* below.

Parameters:

`struct_name`

The name assigned to the structure being created, *struct_name* is the SDF fileid assigned to the SDO which will subsequently be referenced via the *USING* parameter of *EDIT* and *BROWSE* commands.

Unless parameter TEMPORARY is specified, *struct_name* must be the DSN of a new or existing data set or PDS(E) member to which the file structure definition is saved.

`TITLE short_title`

Specifies a title to be assigned to the structure.

The structure's title is displayed as part of the structure (SDO) list and may be displayed or updated via the SET/QUERY/EXTRACT TITLE option.

The *short_title* text string should be no longer than 64 characters and may be enclosed in quotation marks (") or apostrophes (').

`DESCRIPTION long_description`

Specifies a description to be assigned to the structure.

The structure's description and may be displayed or updated via the SET/QUERY/EXTRACT DESCRIPTION option.

The *long_description* text string should be no longer than 512 characters and may be enclosed in quotation marks (") or apostrophes (').

`REPLACING`

Specifies text replacing options to be implemented when processing COBOL copybooks.

When creating structures from COBOL copybooks, FileKit generates a dummy COBOL program with a working storage section which consists of COPY statements for each included copybook member and an optional REPLACE statement. This program is then compiled and the resulting SYSADATA dataset used to generate the data structure.

The syntax of the COBOL REPLACE compiler directive statement is:

```
                +-----------------------------------------------+
                v                                               |
>>- REPLACE -+- ==pseudo_text_1== - BY - ==pseudo_text_1== -+-><
```

The pairs of pseudo text strings are obtained from the following sources:

1. The FileKit site INI file variable pairs *SITECOBOLREPLACE.nnFR* (the change from string) and *SITECOBOLREPLACE.nnTO* (the change to string) where nn = 01 - 12.

2. The FileKit user INI file variable pairs *USERCOBOLREPLACE.nnFR* (the change from string) and *USERCOBOLREPLACE.nnTO* (the change to string) where nn = 01 - 12.

   These variables can be viewed and modified in the COBOL Replacing Options fields of the COBOL Compiler Options (=0.4.1) panel.

3. The CREATE STRUCTURE command RELACING clause.

By default the generated REPLACE statement contains the changes from all of these sources. This action can be modified by the following keywords:

`NOTHING`

Do not generate a REPLACE statement. The COBOL copybook text is processed as is. This may lead to compile errors due to invalid COBOL data names.

`ONLY`

Only use the replacing text from the REPLACING clause of the CREATE STRUCTURE command. The site and user INI file variables are not used.

`SYSADATA ( *sysadata_dataset_name* )`

When creating structures from COBOL or PL1 copybooks FileKit uses the language compliers to process the copybook source and produce a SYSADATA dataset.

By default the SYSADATA dataset is a dynamically allocated temporary dataset which is deleted when the CREATE STRUCTURE command has been processed.

There may be a requirement to examine or save this SYSADATA file (for example for problem diagnosis). This option allows the specification of a user allocated permanent dataset or PDS(E) member which will be used instead of the temporary dataset.

The SYSADATA dataset or PDS(E) member must have record format variable blocked (RECFM=VB) and logical record length greated than 1000 (LRECL>1000).

`COPYBOOKPROC ( *COMPILER|INTERNAL* )`

Specify whether the language compiler or FileKit internal code is to process copybooks for this command.

By default the current setting of the COPYBOOKPROC option determines the copybook processor.

`OPTION ( ... )`

The *OPTION* keyword can be used to set certain optional structure attributes. These attributes are normally set interactively using a structured edit primary command but can be pre-defined when the structure is created using this keyword.

In each case the required structured edit primary command is placed in parentheses following the *OPTION* keyword. The supported optional commands are:

colwidth_command

The *colwidth_command* option is SET COLWIDTH syntax which is used to modify the displayed width of character columns.

ccolour_command

The *ccolour_command* option is CCOLOUR syntax which is used to conditionally apply a specific colour and hilight to a particular column.

rcolour_command

The *rcolour_command* option is RCOLOUR syntax which is used to conditionally apply a specific colour and hilight to a particular row.

`INITCMD (*sd_command* ... )`

The *INITCMD* keyword can be used to define a set of structured edit primary commands which are to be executed whenever an EDIT or BROWSE session is started using this structure. These commands are executed before any *INITCMD* specifications for edit or browse.

Any number of commands may be listed in parentheses following the *INITCMD* keyword enclosed in apostrophes (') or quotation marks (").

`REPLACE`
`NOREPLACE`

Either overwrite or give an error if *struct_name* already exists.
Default is NOREPLACE.

`TEMPORARY`

Create the SDO in storage without automatically saving it to an SDF data set. A temporary SDO cannot be saved.

Where TEMPORARY is used, *struct_name* is simply a named reference for the SDO and does not need to be a valid DSN. As for standard SDOs, when a temporary SDO is created, it will remain in storage until it is dropped using the DROP or DDROP command or the Data Editor session is ended.
Default is to save the SDO to an SDF allocated to DSN *struct_name*.

`CASE(IGNORE)`
`CASE(RESPECT)`

Respect or ignore the case setting of any variable names used in the structure definition. Default is **CASE(IGNORE)**.

XRef Definition

XRef Definition syntax is an alternative, more flexible method of generating a FileKit structure (SDO) than using CopyBook Definition or Direct Definition syntax.

Features of XRef Definition which are not supported by these other methods are:

1. Define segment record type structures for mapping the individual segments (primary and secondary) of segmented records.

2. Define a record type based on any COBOL group or PL1 major/minor structure within a COBOL/PL1 copybook or ADATA file.

3. In a single CREATE STRUCTURE execution, define multiple record types from a mixture of COBOL and/or PL1 copybooks, ADATA files, existing FileKit structures (SDOs) and/or in-line record type direct definition syntax.

4. Specification of a default record type.

5. Assignment of USE WHEN selection criteria to each non-default primary and secondary record type definition generated from COBOL/PL1 source. (This is mandatory.)

`LIBRARY (`*`copybook_library`*` ... )`

Specifies a library search chain for COBOL or PL1 copybook lookup. If a library containing a required source copy book is in the search chain, then *copybook_library* need not be specified following the SOURCE COBOL/PL1 parameter.

One or more blank separated *copybook_library* PDS/PDSE library data set names are specified in "( )" (parentheses).

`RECORD (` **`RecordType Definition`** `)`

A separate RECORD specification is required for each record type to be defined within the structure.

RecordType Definition

RecordType Definition incorporates all parameters supported by each RECORD parameter specification in XRef Definition.

`NAME` *`record_type`*

Identifies the name of the record type to be defined in the SDO.

Unless SOURCE references FileKit SDE record definition syntax via the STRUCTURE sub-parameter, *record_type* must match the name of a group (major or minor structure) field in the source (copybook, ADATA, SDO) object.

`OFFSET <+|->` *`n_bytes`*

Specifies a positive or negative offset from the start of the record (or segment) at which the record type will start mapping data.

Use of an offset is documented in detail under SET/QUERY/EXTRACT option USEOFFSET.

`STARTFIELD` *`data_element`*

Applicable only to SOURCE COBOL, PL1 or HLASM, STARTFIELD may be used instead of NAME and OFFSET, to identify the start field (*data_element*) at which record data will be mapped.

The record type definition name (*record_type*) will be that of a COBOL group, PL1 structure or HLASM DSECT in which the start data element is defined. If a LEVEL number is specified, the record type definition will be the group containing *data_element* which matches the specified level number. Otherwise, it is the group containing *data_element* with the highest level number.

The offset of *data_element* within the selected group defines a negative OFFSET value to be applied when mapping record data.

`SOURCE`

The SOURCE parameter identifies the input file (data set or PDS/PDSE library member) and the type of input to be used to generate the record type definition.

If COBOL or PL1, then the appropriate compiler will be invoked to compile the source file and, if the compiler return code is within the range of acceptable values (See options MAXCOBOLRC and MAXPL1RC), use the resulting ADATA file to build the record type definition (RTO). If the source is an existing COBOL or PL1 ADATA output file, then the RTO is generated directly from the ADATA source (no compilation is required.)

SDO source may be used to copy an RTO definition from an existing SDO structure, whereas STRUCTURE indicates that the RTO is to be constructed using in-line record definition syntax.

`COBOL|PL1|HLASM` *`source_fileid`*

COBOL, PL1 or HLASM specifies the program language of the statements in the copybook source data set or library member identified by *source_fileid*.

source_fileid contains the COBOL group field definition, major or minor PL1 structure or Assembler DSECT from which the record type definition will be generated. This group/structure/DSECT name must match the *record_type* name as specified by the NAME parameter.

If *source_fileid* specifies a PDS/PDSE library member name without a library DSN, then the library search chain specified by LIBRARY is used to find the member.

If *source_fileid* specifies a PDS/PDSE library member name with a library DSN, *library(member_name)*, then the library DSN is added to the end of the library search chain. Note that the first occurrence of the member within the library chain will be used. This may be a member with the same member name that exists in a different library to that specified by *source_fileid*.

`ADATA` *`adata_fileid`*

Specifies the name (*adata_fileid*) of an ADATA output data set which was created as a result of an Enterprise COBOL or PL1 compilation or an HLASM assembly.

adata_fileid contains the 01 COBOL group field definition, major PL1 structure or Assembler DSECT from which the record type definition will be generated. This group/structure/DSECT name must match the *record_type* name as specified by the NAME parameter.

`LEVEL` *`level_number`*

Applicable only to SOURCE type COBOL or PL1, LEVEL identifies the data element level number, *level_number*, of the COBOL group field or PL1 structure which constitutes the record definition.

When used with the NAME parameter, the *record_type* name at the specified *level_number* will be used. Default is LEVEL 1.

When used with the STARTFIELD parameter, *level_number* identifies the level of a group or structure within the hierarchy in which the start *data_element* is defined. This group/structure will be used to build the record definition. If no LEVEL is specified, the default for STARTFIELD will be a value equal to the highest level of the group in which *data_element* is defined.

`SDO` *`sdo_fileid`*

Specifies *sdo_fileid*, a sequential data set or library and member name of a previously generated FileKit structure (SDO).

sdo_fileid contains the record type definition from which the new record type definition will be created. This record type definition name must match the *record_type* name as specified by the NAME parameter.

`STRUCTURE (`**`Record Definition`**`)`

Specifies the CREATE STRUCTURE Record Definition syntax that will be used to create a a record type definition of name *record_type* as specified by the NAME parameter.

`SYMNAMES (`*`DFSORT symbols, ...`*`)`

Specifies DFSORT SYMNAME symbol definitions that are to be used to define consecutive fields in the new record type definition. The order in which symbol definitions are supplied dictate the order in which the fields will occur in the record type definition.

The symbol name definitions within the SYMNAMES parentheses may be supplied directly in-line and/or via input data sets/library members.

```
SYMNAMES( Card,06,04,CH  Dept,46,03,CH  Pounds,49,06,PD  Dollars,55,06,CH  )
SYMNAMES( SYS1.MACLIB(EDGSMFSY)  SYS1.MACLIB(EDGSRCSY) )
SYMNAMES( CBL.DFSORT.SYM(CBLATRAC)  TCB,*,4,BI )
```

`FIELDS` *`data_element, ...`*

Specifies the name of one or more field data elements (*data_element*) that exist in the record definition identified by the FROM *record_type* parameter. Multiple *data_element* specifications must be comma separated.

The new record type definition will be identical to that specified by FROM *record_type* except that all field data elements that have not been selected by the FIELDS specification, are marked as hidden. Data in records that are mapped to hidden field elements are not included in the formatted record data.

This technique of hiding fields can be useful when only one instance of a redefined field (or group field) mapping is to be displayed in a data editor view. By default, the data editor will redisplay data for each field mapping involved in a redefine (union).

All fields elements in a group (structure) can be selected by specifying the group field name as the *data_element* with a "(*)" suffix. e.g. XREF_KEY(*) selects all field elements in the group field XREF_KEY.

`FROM` *`record_type`*

Specifies the name of a record type definition *record_type* which has been defined by previous RECORD() syntax in the current CREATE STRUCTURE command.

This *record_type* identifies the record type definition from which the FIELDS parameter selects field data elements for the new record type.

`PRIMARY`

Identifies the record type as being a non-default primary (base) segment.

A primary segment maps data at the start (first segment) of a segmented record or, for non-segmented records, maps all data in a record. Primary segment definitions are never selected to map data beginning at any position other than position 1 of a segmented record.

Specification of USE WHEN criteria is mandatory in order to identify the conditions by which this record type is allocated to a record/base segment.

`SECONDARY`

Applicable to segmented records only, SECONDARY identifies the record type as being a secondary segment.

A secondary segment maps data that immediately follows any primary segment or any other secondary segment defined within the structure. Secondary segment definitions are never selected to map data beginning at position 1 of a segmented record.

Specification of USE WHEN criteria is mandatory in order to identify the conditions by which this record type is allocated to a secondary segment.

If no secondary segment record type criteria is satisfied by the record data, then the remainder of that record is displayed using the default secondary segment "UnmappedSeg". This maps the remaining segmented record data as a character data field of length equal to the remaining record data length.

`DEFAULT`

Identifies the record type as being the default primary (base) segment.

DEFAULT may be assigned tp only one record type definition in the SDO. Furthermore, USE WHEN criteria must not be specified for default record type definitions.

The default record type is assigned to a record (or primary segment) when it does not satisfy the USE WHEN selection criteria of any of the defined primary record type definitions.

`<USE> WHEN|IF` *`expression`*

Specifies the selection criteria that must be satisfied by the record (or record segment) data, in order for it to be assigned the associated record type mapping.

The USE WHEN *expression* is a valid SDE expression which supports function calls, record type field names and references, sub-expressions, arithmetic, relational and logical operators.

The result of the USE WHEN expression must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

When the selection criteria involves testing fields in unformatted record data, the SDE expression should simply use function SUBSTR() on the record field RECORD. e.g. To assign a record type based on a character string "A11" at position 11 of the record data, specify the following:

```
USE IF  SUBSTR(RECORD,11,3) = 'A11'
```

To test fields in the record data as if already formatted by the record type structure you are attempting to assign, simply reference the required field names in the SDE expression. e.g. To assign a record type based on values in 2 numeric fields, SEQUENCE_ID and CUST_REF, both defined within the record type structure...

```
USE IF  ( SEQUENCE_ID > 301   AND   CUST_REF = 10233 )
```

Note that fields SEQUENCE_ID and CUST_REF may have been defined as packed decimal, integer, floating point, etc. If the record data is invalid for the assigned data type, the expression will return a "false" condition.

CopyBook Definition

Defines the one or more external copy book(s) or an ADATA compile output file to be used to generate the SDO. Multiple copy books are concatenated to produce a single copy book and then individual record type definitions (RTOs) established as follow:

1. If Auto Rec-Types is set OFF (see the Structured Data Edit Settings panel), then each level 1 entry will generate a new record type (RTO) mapping.

   Note that, for COBOL copybooks only, if LEVEL 1 or RECORD TYPE is not specified and the first data declaration statement is not a level 1 entry, then a level 1 entry with group name "SELCOPY-01" is generated at the start of the copy book concatenation.

2. If Auto Rec-Types is set ON, then for each level 1 entry where all the 2nd level entries are a union (REDEFINES) of each other, a new record type will be created for each 2nd level entry instead of the level 1 entry.

`FROM COBOL|PL1|HLASM|ADATA`

Identifies the input as a copy book with programming language COBOL, PL/1 or Assembler. Alternatively identifies input as being ADATA output generated from a Enterprise COBOL, PL/1 compilation or High level Assembler assembly.

COBOL and PL/1 copy books that have been compiled using the following compiler options, generate ADATA output.

ADATA	COBOL compiler option.
ADATA	HLASM assemble option.
XINFO(SYN,SYM)	PL/1 compiler option.

Using an already generated ADATA output reduces the inevitable overhead that occurs if FileKit itself has to re-compile a copy book in order to create the SDO.

Default is COBOL.

`(LEVEL) 1` *`level1_name`*
`RECORD (TYPE)`

LEVEL 1 or RECORD TYPE followed by *level1_name* will insert a COBOL 01 goup, PL/1 struct or Assembler DSECT record as appropriate at the start of the file data specified by *copybook_name*. This syntax may be used where the record structure name is not included in the copybook file name. The record-type name generated will match the name specified by *level1_name*.

`copybook_name`

The complete fileid (DSN and member name) of a COBOL copybook, PL/1 include file, Assembler DSECT source or ADATA file.

`SELECT` *`level1_entry`*

Selects the individual "01" level entry names from within the concatenation of specified copy books for which record type mappings (RTOs) are to be generated. All data declaration entries associated with "01" level entries that are not selected, will be ignored. Multiple names must be separated with commas.

For HLASM an "01" level name is a *DSECT* name.
Default is to generate an RTO for each "01" level entry in the copy book concatenation.

DB2 Definition

DB2 definition syntax provides the method of defining an SDO containing a single record type structure (RTO) based on columns returned by a prepared SQL query statement. CREATE STRUCTURE uses information stored in the returned SQLDA to determine the data type, etc. for each field definition in the RTO. To do this, a BIND of the FileKit DB2 plan must have been performed for any DB2 sub-system containing tables involved in the CREATE STRUCTURE operation.

Furthermore, SDE DB2 processing options may be specified on CREATE STRUCTURE and so are saved in the RTO. These options are used by default when the (SDO) structure is used to process the table data.

`FROM DB2 < (`*`ssn`*`) >`

Indicates that structure generated is derived from a DB2 results table. *ssn* is optional and identifies the local DB2 sub-system name to which a connection will be made.

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 users default DB2 sub-system name as set by the DB2 Primary Options menu.

`table_name`

A DB2 base table name identifying the columns that constitute the structure's only record type (RTO) definition.

table_name may be specified with 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default location is the local DB2 server and the default schema is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID).

`view_name`

A DB2 table view which references an SQL query used to generate a results table containing the columns that constitute the structure's only record type (RTO) definition.

view_name may be specified with 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default location is the local DB2 server and the default schema is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID).

DB2 Opts

The following DB2 options are saved in the structure and are used to generate the dynamic SQL query statement, apply further mapping to DB2 column data and also set default values for SDE DB2 table processing. These options are invoked if the structure is used to process the DB2 table. For example, via the EDIT or BROWSE command USING STRUCTNAME parameter.

`WHERE (`*`where-clause`*`)`

Specifies a DB2 SQL WHERE clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *where-clause*.

If *where-clause* is null or "?" (question mark) the DB2 Row Selection Panel is opened before the create structure is actioned. On closing the panel, the create structure operation is actioned using the generated WHERE clause.

`SORT|ORDER BY (`*`order-by-clause`*`)`

Specifies a DB2 SQL ORDER BY clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *order-by-clause*.

`SKIPLOCKED`

Saves an SDE DB2 table processing default for option SKIPLOCKED.
Ignored unless an isolation level of Cursor Stability (CS) or Read Stability (RS) is in effect, SKIPLOCKED specifies that any selected rows that are already locked by another process should be skipped and not be included in the edit display. See "*DB2 SQL Reference*" for details on the SKIP LOCKED DATA clause.
Default is to allow display of locked rows whenever possible.

`PROTECTPRIMEKEY | EDITPRIMEKEY`

Saves an SDE DB2 table processing default for option PROTECTPRIMEKEY / EDITPRIMEKEY.
Specifies whether data occupying columns that comprise the table's primary key is eligible for update (EDITPRIMEKEY) or is read only (PROTECTPRIMEKEY).
Default is PROTECTPRIMEKEY.

`WITH CS|UR|RR|RS  < KEEP  EXCLUSIVE|UPDATE|SHR >`

Saves an SDE DB2 table processing default for option WITH.
Specifies a DB2 SQL isolation-clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for details on the *isolation-clause* and "*DB2 Performance Monitoring and Tuning Guide*" for details on the effects of isolation level on concurrency and protection of DB2 table data.

The KEEP sub-parameter options are applicable to isolation levels RR (Repeatable Read) and RS (Read Stability) only.
Default isolation level (as set by the FileKit DB2 package and plan BIND) is CS.

`COMMITONCLEANSAVE | COMMITONSAVE | COMMITONEXIT`

Saves an SDE DB2 table processing default for option COMMITONCLEANSAVE / COMMITONSAVE / COMMITONEXIT.
Specifies when a COMMIT should be executed for changed data.

COMMITONCLEANSAVE	COMMIT only if SAVE is executed without errors.
COMMITONSAVE	COMMIT on SAVE regardless of errors.
COMMITONEXIT	COMMIT only on exit of the edit session.

Default is COMMITONCLEANSAVE.

`COMMITONLOAD`

Saves an SDE DB2 table processing default for option COMMITONLOAD.
Performs a COMMIT following the initial load of rows to be edited, thus releasing all DB2 table locks performed during load of the data. This includes any explicit table locks applied via the LOCKTABLE parameter.
Default is not to perform a COMMIT following load of the table rows.

`AUDIT`

Saves an SDE DB2 table processing default for option AUDIT.
AUDIT will open a new FileKit DB2 audit data set to record changes to the edited table made during this edit session. See DB2 Auditing for details.
Default is not to perform edit auditing.

`SELECT (`*`select_syntax`*`)`

DB2 table edit/browse will fetch all columns from the selected table or view. SELECT may be used to select a subset of these columns for initial display in the edit/browse window view. Except for parameters FROM *record_type* and IN *struct_name* which are both unnecessary, the *select_syntax* argument represents parameters supported by the SELECT primary command.

Unlike parameter INITCMD, which may also be used to execute SELECT commands, the SELECT parameter column selection is performed before the profile macro is executed.

Direct Definition

Direct definition syntax provides the method of directly defining an SDO containing one or more record structures (RTOs), each comprising one or more fields, without first having a copy book.

`NAMES`

Field names specified in the *record_definition* are verified so that they conform with the naming standards of the specified programming or query language. Supported standards are as follow:

ASM | ASSEMBLER | HLASM	IBM High Level Assembler.
COBOL	IBM COBOL.
DB2	IBM DB2 SQL.
PL1 | PLI	IBM Enterprise PL/1.

Record Definition

Definition of a single record structure (RTO) within the structure definition object (SDO).

`record_type`

The record type name used to identify the new record type object (RTO).

`STRUCTURE`

The data type of an RTO. An RTO may be considered to be a single field of type STRUCTURE which itself comprises a number of individual fields.

`EBCDIC`
`ASCII`

Unless overridden by EBCDIC or ASCII specified as a parameter on a field definition, all character fields within the RTO are to be treated as either EBCDIC or ASCII.
Default is EBCDIC.

`USE WHEN `*`expression`*

Often an individual record can be assigned a record type by matching on record length. Where several record types are potential matches based on record length, then **USE WHEN** should be used to specify criteria that must be satisfied before the record type can be assigned to a particular record.

expression is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of the WHEN expression must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

If a WHEN expression associated with a record type returns a "true" result for a data record, then that record type is assigned to the record.

Expand Clause

If an Expand Clause is included as part of a record-type definition, then it defines a CSRCE compressed area of data within any record mapped by the record-type. Compressed data in these records will be expanded before being processed by the FileKit utility.

`EXPAND`

EXPAND keyword identifies the start of an Expand Clause for the record-type definition. Each record-type definition may contain at most one Expand Clause.

`EXPLOC(`*`expr`*`)`

This parameter is mandatory and specifies an SDE expression (*expr*) that resolves to be the position of the start of the compressed data within the record. e.g. EXPLOC(DSTART) where DSTART is a field defined in the non-compressed area of record-type mapping, and has a value which is equal to the record start position of the compressed data.

`EXPSIZE(`*`expr`*`)`

This parameter is mandatory and specifies an SDE expression (*expr*) that resolves to be the length of the compressed data within the record. e.g. EXPSIZE(DLEN) where DLEN is a field defined in the non-compressed area of record-type mapping, and has a value which is equal to the length of the compressed data.

`WHEN` *`expr`*

Optional parameter that specifies an SDE expression (*expr*) that resolves to be true (1) or false (0) value. WHEN identifies a condition for which the result must be true in order for data expansion to occur.

Field Definition

Definition of a single field within the record RTO.

`field_name`

The field name identifier.

`PICture` *`picture_string`*

Any COBOL-style picture string. e.g. `PIC S9(7)`
This is treated **purely as comment** data.

`DIMensions`

Used to define an array of given datatype.

dim	An integer that defines the fixed number of elements.
min	An integer that defines the minimum number of elements.
max	An integer that defines the maximum number of elements.
d_name	The field name that defines the variable number of elements.

e.g.

```
dim(12)
dim((0,44,ObjNum))
dim((0,10,OutPrtNum),(0,10,OutPunNum))
```

`EBCDIC`
`ASCII`

For character fields only, treat as either EBCDIC or ASCII.
Default is as defined on the Record Definition, otherwise EBCDIC.

`SIGNED`
`UNSIGNED`

For numeric fields only, treat as signed or unsigned. Default is SIGNED.

The sub-parameters **LEADing|TRAILing** and **SEParate|INCluded** are applicable to zoned decimal only and correspond directly to the COBOL SIGN clause.

`ALIGNED`
`UNALIGNED`

Determines whether the field will be boundary aligned for its data type.
Default is UNALIGNED.

`ZEROS`

Applicable to numeric fields only, ZEROS indicates that leading zeros are to be displayed. For alpha-numeric fields, this parameter is ignored.

`REMarks` *`remark`*

A comment string placed in either single or double quotes.

`ENUMeration`

Create an enumeration definition or specify the name of an existing enumeration for use with the current field.

enam may be specified to allocate a name to the enumeration definition or as a reference to an enumeration already defined within the current field definition clause.

Each *enumerator* should be specified in the form *display*=*value*, where *display* will be displayed when this field contains *value*. e.g.

```
enum(HLAsm=16,Cobol=17,Pl1=40)
```

display should be enclosed in quotes if it is to contain blanks. *display* may be specified without a corresponding *value*, in which case the previous value plus 1 will be implied.

Data Definition

Data type of a single field within the record RTO.

`BINTeger`

Defines a field to be interpreted as binary whole-number occupying *n_bits* bits, where *n_bits* defaults to **1**.

`BIT`

Defines a field to be interpreted as a series of ON|OFF (1|0) switches occupying *n_bits* bits, where *n_bits* defaults to **1**.

`CHARacter`

Defines a field to be interpreted as a fixed-length character string occupying *n_bytes* bytes, where *n_bytes* defaults to **1**.

`CHARVarying`

Defines a field to be interpreted as a variable-length character string padded with blanks to occupy a fixed *n_bytes*, where *n_bytes* defaults to **1**. The actual length of the string is defined by a 2 byte binary integer at the start of the field so that the overall length of the field is 2+*n_bytes*.
CHARVARYING is equivalent to a PL/1 field which is declared as being CHARACTER VARYING.

`CHARZ`

Defines a field to be interpreted as a variable-length character string occupying a fixed *n_bytes* bytes, where *n_bytes* defaults to **1**. No length field exists, instead the character string is terminated by a null character (x'00') which increases the overall length of the field by 1. (i.e. *n_bytes*+1)
CHARZ is equivalent to a PL/1 field which is declared as being CHARACTER VARYINGZ.

`DECimal`

Defines a field to be interpreted as a packed-decimal number allowing *precision* number of digits in total, of which *scale* number of digits will be displayed following the decimal-point.
Default for *precision* is **7**.
Default for *scale* is **0**.

`DATE`

Defines a date field with the source data interpreted in one of the following formats:

`DECIMAL`	Packed decimal format of length 4 bytes (X'0cyy,dddF'). Decimal values for year of century (yy) and Julian day of year (ddd). 'F' is a 4-bit sign and 'c' is the century indicator X'0'(1900) or X'1'(2000). e.g. 14th February 2011 is X'0111,045F'.
`BINARY`	Binary format of length 6 bytes (X'yyyy,00mm,00dd'). Binary values for year including century (yyyy), month of year (mm) and day of month (dd). e.g. 20th October 2011 is X'07DB,000A,0014'.
`CATALOG`	ICF Catalog Date format of length 4 bytes (X'yydd,dFcc'). Decimal values for year of century (yy) and Julian day of year (ddd). 'F' is a 4-bit sign and 'cc' is the century indicator X'00'(1900) or X'01'(2000). e.g. 16th July 2008 is X'0819,7F01'.
`VTOC`	VTOC format of length 3 bytes (X'yydddd'). Binary values for years since 1900 (yy) and Julian day of year (dddd). e.g. 19th August 2005 is X'6900E7'.

Default date format is **DECIMAL**.

`FIXed`

Defines a field to be interpreted as a binary number allowing *precision* number of digits in total, of which *scale* number of digits will be displayed following the decimal-point.
Default for *precision* is **7**.
Default for *scale* is **0**.

`FLOATBIN`

Defines a field to be interpreted as a Binary (IEEE 754) Floating-point number occupying *n_bytes* bytes, where *n_bytes* may be 4, 8 or 16 (defaults to **4**).

`FLOATDEC`

Defines a field to be interpreted as a Decimal Floating-point value of *precision* number of significant digits, where *precision* may be 7, 16 or 34 (defaults to **34**). Precision 7 occupies 4 bytes of data, 16 occupies 8 bytes of data and precision 34 occupies 16 bytes.

`FLOATHEX`

Defines a field to be interpreted as a Hexadecimal Floating-point number occupying *n_bytes* bytes, where *n_bytes* may be 4, 8 or 16 (defaults to **4**).

`HEXadecimal`

Defines a field to be interpreted as printable hexadecimal character string occupying *n_bytes* bytes, where *n_bytes* defaults to **1**.

`HEXVAR`

Defines a field to be interpreted as a variable-length printable hexadecimal string padded with blanks to occupy a fixed *n_bytes*, where *n_bytes* defaults to **1**. The actual length of the string is defined by a 2 byte binary integer at the start of the field so that the overall length of the field is 2+*n_bytes*.

`INTeger`

Defines a field to be interpreted as a binary whole-number occupying *n_bytes* bytes, where *n_bytes* defaults to **4**.

`IPaddress(4|16)`

Defines a field to be interpreted as an IP address occupying either **4** or **16** bytes.

IPADDRESS(4)

Assumed to be an IPv4 address. The value will display as 4, 3-digit decimal values each separated by a "." (dot/period) with an overall length of 15 bytes. e.g. *`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. e.g. `0123:4567:89AB:CDEF:0123:4567:89AB:CDEF`

`PCHAR`

Defines a field to be interpreted as a PL/1 style PICTURE string representing a character data item (i.e. no numerical interpretation). The length of the field is determined by the quoted *pl1_picture_string*.
pl1_picture_string may contain any valid PL/1 picture character for character data. If invalid characters (e.g. numeric picture characters E, V, Z, etc.) are specified then an error will occur. Default is PCHAR('X').

`PFIXED`

Defines a field to be interpreted as a PL/1 style PICTURE string representing a FIXED point numerical character data item. The length of the field is determined by the quoted *pl1_picture_string*.
pl1_picture_string may contain any valid PL/1 picture character for numeric character data except for exponent characters "E" and "K". If invalid characters are specified then an error will occur. Default is PFIXED('S9').

`PFLOAT`

Defines a field to be interpreted as a PL/1 style PICTURE string representing a FLOAT point numerical character data item. The length of the field is determined by the quoted *pl1_picture_string*.
pl1_picture_string may contain any valid PL/1 picture character for numeric character data. If invalid characters are specified then an error will occur. Default is PFLOAT('S9ES99').

`STRUCTure`

Defines a field to be interpreted as a structure containing one or more fields specified using Field Definition syntax. One or more of these fields may themselves be structures and so nesting of structures is supported.
Each field definition should be separated by a comma.

`TIME`

Defines a time field with the source data interpreted in one of the following formats:

`BINARY`	Binary format of length 4 bytes (X'nnnn,nnnn'). A 32-bit unsigned binary value for number of hundredths of seconds (0.01 second) elapsed in a day.
`DECIMAL`	Packed decimal format of length 4 bytes as returned by the TIME macro with option DEC (X'HHMM,SSTH'). Decimal values for hours (HH), minutes (MM), seconds (SS), tenths of second (T) and hundredths of second (H). Note that the data is unsigned.
`DECIMAL2`	Packed decimal format of length 4 bytes (X'00HH,MMSS'). Decimal values for hours (HH), minutes (MM) and seconds (SS). Note that the data is unsigned.
`DECIMAL3`	Packed decimal format of length 4 bytes (X'0HHM,MSSC'). Decimal values for hours (HH), minutes (MM) and seconds (SS). Note that the data **is** signed with X'C'.
`STCK`	STCK format of length 8 bytes. A 64-bit unsigned binary value elapsed time value in the same format as that returned by a STCK operation (i.e. bits 0-63 of the system TOD clock).
`STCK,`*`nbits`*	The low order, unsigned 32-bits (4-bytes) of a STCK format value which has been shifted right a number of bits specified by *nbits*. (This STCK format is often used for elapsed time values in SMF records.) The following shows the number of microsecond (us) time units represented by a single bit in the 32-bit value identified by TIME(STCK,*nbits*). TIME(STCK,12) 1us TIME(STCK,16) 16us TIME(STCK,19) 128us TIME(STCK,22) 1024us

	TIME(STCK,32)	1048576us

STCKE

STCKE format of length 16 bytes. A 128-bit unsigned binary value elapsed time value in the same format as that returned by an STCKE operation (i.e. an 8-bit Epoch Index, followed by 104-bit TOD clock and 16-bit Programmable Field).

UNIX

UNIX format time of length 4 bytes. A 32-bit unsigned binary value which is the number of seconds elapsed since midnight.

Default time format is **DECIMAL**.

TIMESTAMP

Defines a timestamp (date and time) field with the source data interpreted in one of the following formats:

BINARY	Binary format of length 10 bytes (X'00dd,nnnn,nnnn'). See descriptions of BINARY format for DATE and TIME parameters.
TIMEDEC DECIMAL	Packed decimal format of length 8 bytes (X'0cyy,dddF,HHMM,SSTH'). See descriptions of DECIMAL format for DATE and TIME parameters.
TIMEDEC2 DECIMAL2	Packed decimal format of length 8 bytes (X'0cyy,dddF,00HH,MMSS'). See descriptions of DECIMAL format for the DATE parameter and DECIMAL2 format for the TIME parameter.
TIMEDEC3 DECIMAL3	Packed decimal format of length 8 bytes (X'0cyy,dddF,0HHM,MSSC'). See descriptions of DECIMAL format for the DATE parameter and DECIMAL3 format for the TIME parameter.
TIMEBIN	Packed decimal/binary format of length 8 bytes (X'0cyy,dddF,nnnn,nnnn'). See descriptions of DECIMAL format for DATE and BINARY format for TIME parameters.
STCK	STCK format of length 8 bytes. A 64-bit unsigned binary value elapsed time value in the same format as that returned by a STCK operation (i.e. bits 0-63 of the system TOD clock).
STCK,*n-bytes*	The high order bytes of a store clock value of length *n-bytes*. The source value is padded with nulls to length 8-bytes so that it is a 64-bit unsigned binary value in the same format as the system TOD clock. STCK,4 is used as the timestamp format in a number of SMF records fields.
STCKE	STCKE format of length 16 bytes. A 128-bit unsigned binary value elapsed time value in the same format as that returned by an STCKE operation (i.e. an 8-bit Epoch Index, followed by 104-bit TOD clock and 16-bit Programmable Field).
HFSDIR	HFS directory timestamp of length 4 bytes. A 32-bit unsigned binary value for number of seconds elapsed since 1970/01/01 00:00:00.
SMF	SMF record timestamp format (X'nnnn,nnnn,yyyy,dddF') which is the same as TIMEBIN but the opposite way around (i.e. time component first) See descriptions of DECIMAL format for DATE and BINARY format for TIME parameters.

Default timestamp format is **TIMEDEC (DECIMAL)**.

UNION

Defines a field to be interpreted as a union of one or more fields specified using Field Definition syntax.
Each field definition should be separated by a comma.
Unions allow the same data to be interpreted as more than one data-type.

VARCHAR

Defines a field to be interpreted as a variable-length character string. The maximum possible length of the string is defined as **max_bytes** bytes, defaulting to **0**. The actual length of the string is defined by a binary integer field of length **len_bytes** bytes at the start of the field.
Default for *len_bytes* is **2**. The value of the *exclusive* parameter indicates whether this length integer includes or exlcudes the length field itself. Permissable values are EXCLUSIVE (the default) and INCLUSIVE.

VARHEX

Defines a field to be interpreted as a variable-length printable hexadecimal string. The maximum possible length of the string is defined as **max_bytes** bytes, defaulting to **0**. The actual length of the string is defined by a binary integer field of length **len_bytes** bytes at the start of the field.
Default for *len_bytes* is **2**. The value of the *exclusive* parameter indicates whether this length integer includes or exlcudes the length field itself. Permissable values are EXCLUSIVE (the default) and INCLUSIVE.

XVARCHAR

Defines a field to be interpreted as a variable-length character string. The maximum possible length of the string is defined as *max_bytes* bytes, defaulting to **0**. The actual length of the string is defined by field name *len_field*, which must be numeric.

ZONEd

Defines a field to be interpreted as a zoned-decimal character number allowing *precision* number of digits in total, of which *scale* number of digits will be displayed following the decimal-point.
Default for *precision* is **1**.
Default for *scale* is **0**.

Examples:

```
<sd create structure          CBL.FILEKIT.STRUCT(EMP) from cobol CBL.COPYBOOK.COBOL(EMP)
<sd edit  CBL.SDE.EMP   using CBL.FILEKIT.STRUCT(EMP)
```

Sample commands as they may appear in the user's HOME CMX file. The first command will generate a new SDO structure from a single COBOL copy book, the second will edit a file using the SDO to apply record mapping.

```
<sd create structure   CBL.FILEKIT.SDO(FILSTRUC)        \
  (                                                  \
    REC-TYPE01 struct                                \
    (  REC-TYPE              char(2),                \
       NAME                  char(20),               \
       EMPLOYEE-NO           int(2) unsigned,        \
       AGE                   int(2) unsigned,        \
       SALARY                dec(7),                 \
       MONTH                 int(4) dim(12) unsigned,\
                             char(2)                 \
    )  use when REC-TYPE='01'                        \
   ,                                                 \
    REC-TYPE02 struct                                \
    (  REC-TYPE              char(2),                \
       NAME                  char(20),               \
       JOB-TITLE             char(14),               \
       ADDR1                 char(20),               \
       ADDR2                 char(20),               \
       POSTCODE              char(4)                 \
    )  use when REC-TYPE='02'                        \
  )                                                  \
  names(COBOL)     replace    case respect
```

Sample command as it may appear in the user's HOME CMX file. This command will generate a new SDO structure consisting of 2 record type definintions.

```
<||sd create structure CBL.FILEKIT.SDO(FILSTRUC) \
  (                                                   \
    REC-TYPE01 struct                                 \
    (  REC-TYPE              char(2),                 \
       NAME                  char(20),                \
       EMPLOYEE-NO           int(2) unsigned,         \
       AGE                   int(2) unsigned,         \
       SALARY                dec(7),                  \
       MONTH                 int(4) dim(12) unsigned, \
                             char(2)                  \
    )  use when  REC-TYPE='01'                        \
   ,                                                  \
    REC-TYPE02 struct                                 \
    (  REC-TYPE              char(2),                 \
       NAME                  char(20),                \
       JOB-TITLE             char(14),                \
       ADDR1                 char(20),                \
       ADDR2                 char(20),                \
       POSTCODE              char(4)                  \
    )  use when (                                     \
                    (  REC-TYPE < '02'                \
                     & REC-TYPE > '05'                \
                    )                                 \
                  |                                   \
                    (  REC-TYPE < '22'                \
                     & REC-TYPE > '25'                \
                    )                                 \
                )                                     \
  )                                                   \
  names(COBOL)     replace    case respect
```

Sample command as it may appear in the user's HOME CMX file. This command will generate a new SDO structure consisting of 2 record type definintions. Additional USE WHEN criteria is specified to identify the circumstances under which the particular record mapping is to be used.

See Also:

EDIT   BROWSE

CSVGEN

Syntax:

```
>>--- CSVgen ----------------------------------------------------------------><
```

Description:

CSVGEN with no parameters will open the SDE CSV Generation Panel to generate comma separated variable output for the contents of the focus SDE data edit or browse display.

If the focus window is not an SDE window view, the general CSV Generation dialog panel is displayed.

If parameters are specified, then the general FileKit CSVGEN primary command is executed instead.

See Also:

JSONGEN XMLGEN PRINT

CURSOR

Syntax:

```
>>-- CURsor --+- HOME ---------------+---------------------------------------><
              |                      |
              +- CMdline ------------+
```

Description:

Position the cursor within the current SDE window view. The CURSOR command is most often used in macros.

Note: The CURSOR command does not alter the window view itself. i.e. the current line and column remains unchanged.

Parameters:

`HOME`
: If the cursor is on the command line, the cursor is positioned at the line and column at which it was positioned when it was last in the display area. If this line is no longer visible within the current SDE window view, the cursor is positioned in column 1 of the current line.

  If the cursor is in the file area, the cursor is positioned at column 1 of the command line.

`CMDLINE`
: The cursor is positioned at column 1 of the command line.

CUT

Environments:

CUT primary command exists in the following application environments.

Text Edit	Text Editor (both XEDIT and ISPF interfaces.)
Data Edit	SDE Data Editor.

Syntax:

```
            +-- Copy --+  +-- REPlace -+  +-- NX ------+
            |          |  |            |  |            |
>>-- CUT ---+----------+--+------------+--+------------+---------------------><
            |          |  |            |
            +-- Move --+  +-- APPend --+
```

Description:

CUT may be used instead of primary command CLIPBOARD to copy or move a group of lines to the FileKit clipboard. Once in the clipboard, PASTE may be used to copy the data from the clipboard to another text edit view.

Before text can be copied to the clipboard, it must first be marked within the text display of an edit view. This may be done using prefix commands C(n), CC, M(n) or MM. An error message is displayed if there is no block marked in the current file. Records are copied to the clipboard in their unformatted state. Therefore, formatted records of different record types may be included in the marked group of lines.

If M(n) or MM is used to select a group of lines to be moved to the clipboard, then CUT COPY is equivalent to CUT MOVE.

Parameters COPY and REPLACE are default and so CUT with no parameters is equivalent to CLIPBOARD COPY.

Parameters:

`COPY`

Copy the group of lines to the clipboard. If M(n) or MM is used to select a group of lines, then COPY MOVE is performed. Otherwise the group of lines within the display area is preserved following the copy.

CUT COPY is equivalent to CLIPBOARD COPY.

`MOVE`

Move the group of lines to the clipboard so that it no longer exists within the display area.

CUT MOVE is equivalent to CLIPBOARD CUT.

`REPLACE`

Replaces any text that already exists in clipboard storage. This is default.

`APPEND`

Append data to a new line following the existing clipboard text.

CUT COPY APPEND is equivalent to CLIPBOARD APPEND COPY. CUT MOVE APPEND is equivalent to CLIPBOARD APPEND CUT.

`NX`

Indicates that only visible (i.e. non-excluded and non-suppressed) lines that exist within the selected group of lines are to be included in the copy or move operation.

Examples:

`cut`

Take a copy of a group of lines in the current text edit view selected using CC pair of line commands.

See Also:

CLIPBOARD COPY CREATE PASTE REPLACE

DELETE

Syntax:

```
                                          +- .ZFIRST -+  +- .ZLAST --+
                                          |           |  |           |
>>-- DELete ----- ALL ---------+------+---+-----------+--+-----------+-------><
                               |      |   |           |  |           |
                               +- EX -+   +- .name1 --+  +- .name2 --+
                               |      |
                               +- NX -+
                               |      |
                               +- X --+

               +--- 1 ------+             +- .ZCSR ---+  +- .ZLAST --+
               |            |             |           |  |           |
>>-- DELete ---+------------+--+------+---+-----------+--+-----------+-------><
               |            |  |      |   |           |  |           |
               +- n_lines  -+  +- EX -+   +- .name1 --+  +- .name2 --+
                               |      |
                               +- NX -+
                               |      |
                               +- X --+
```

Description:

Delete visible data records (not excluded) and/or EXCLUDED groups of records.

Deletion of records is not allowed for Update-in-place editing unless the data set is an RRDS or VRDS. For these types of files, deleted records become empty slots.

A group of EXCLUDED records corresponds to a single line for deletion regardless of the number of excluded lines it represents or whether excluded shadow lines are set on or off. (See SET SHADOW.)

DELETE is not sensitive to record type and so may delete records of different record types as part of the same execution.

Records groups that are NOTSELECTED or are SUPPRESSED are not eligible for deletion. Similarly, if EXCLUDED shadow lines are set off, EXCLUDED record groups are also not eligible for deletion unless parameter EX or X is specified.

Also see the "D(n)" and "DD" delete prefix area commands.

Parameters:

`n_lines`

Selects a number of lines from the top of the range upon which the DELETE operation will apply.

The *n_lines* value includes visible records and EXCLUDED, SUPPRESSED and NOTSELECTED shadow lines. Each shadow line corresponds to a single line in the *n_lines* line count, regardless of the number or records within the record group which it represents.

In general, if shadow lines set off, then they are **not** included in the *n_lines* line count. The exception to this is when EX or X is specified, in which case EXCLUDED shadow lines are always included within the *n_lines* line count, regardless of whether excluded shadow lines are set on or off.

Only lines that are eligible for deletion and are included within the lines selected by *n_lines*, will be deleted. e.g. SUPPRESSED shadow lines may be included within the selected lines but are not eligible for deletion and are, therefore, not deleted.

The default value of *n_lines* is 1.

`ALL`

All lines within the specified or implied range are selected for DELETE.

`EX`
`X`

Only EXCLUDED record groups are eligible for deletion. Visible data records are not eligible.

Only when EX or X is used will all EXCLUDED record groups that fall within the selected number of records, be deleted regardless of whether excluded shadow lines are set on or off.

Default is that both EXCLUDED record groups and visible data records are eligible for delete, in which case, if excluded shadow lines are set off, EXCLUDED record groups are not eligible for delete.

`NX`

Only visible data records are eligible for deletion. EXCLUDED record groups are not eligible.
Default is that both EXCLUDED record groups and visible data records are eligible for delete.

`.name1`

A label name identifying the first record of a range of data records to be deleted. The preceding "." (dot) is mandatory.

Default is **.ZFIRST** if **ALL** is specified, otherwise the
default is **.ZCSR** (the focus line.)

`.name2`

A label name identifying the last record of a range of data records to be deleted. The preceding "." (dot) is mandatory.

If *.name2* occurs before *.name1* in the display, then the order is reversed.

The default is **.ZLAST**.

Examples:

```
del 3
```

Delete all eligible lines within the first 3 lines of data starting at the focus line. Eligible lines are visible data records and shadow lines representing an EXCLUDED record group.

```
delete nx
```

Delete the focus line if the focus line is a visible data record.

```
del  all  x   .fix  .fixe
```

Delete all EXCLUDED record groups that exist within the range of lines starting at label name .FIX, ending at label name .FIXE. (The setting for excluded shadow lines is irrelevant.)

```
del  20   ex
```

Delete all EXCLUDED record groups that exist within the first 20 lines of data starting at the focus line. Each EXCLUDED record group constitutes a single line regardless of whether excluded shadow lines are set on or off. Therefore, even with excluded shadow lines set off, 10 consecutive excluded records are treated as a single line for deletion.

See Also:

INSERT

DISPLAY LIST

Syntax:

```
>>- DISplay LIst -- list-name --+-------------------+------------------------->
                                |                   |
                                +-- select_clause --+

 >------+------------------+----+-------------------+------------------------><
        |                  |    |                   |
        +-- where_clause --+    +--- sort_clause ---+
```

Description:

Open a FileKit List window to display an in-storage list generated via the CREATE LIST command.

The List window displaying the list data will support most of the standard list window features as follow:

- Field Descriptor Block (FDB)
- Edit View
- Selecting, Sorting and Filtering
- Sorting with the Cursor

On exiting the List window display, the in-storage list is destroyed and storage is freed.

Parameters:

list_name
The name of the in-storage list data to be displayed.

select_clause
Identifies a valid SELECT Clause to display only those selected columns.

where_clause
Identifies a valid WHERE Clause to filter rows of data.

sort_clause
Identifies a valid SORT Clause to sort the specified columns of data.

Examples:

```
>sd display list AMLIST    select KEY,COMPNAME,FIRSTNAME,LASTNAME,DEPT   where lastname=JONES   sort DEPT a
```

Open a list window for the in-storage list AMLIST with select, sort and filter clauses applied.

See Also:

CREATE LIST

DISPLAY RECTYPES

Syntax:

```
>>-+- DISplay RECTypes --+--+---------------+--------------------------------><
   |                     |  |               |
   +- LR ----------------+  +- struct_name -+
```

Description:

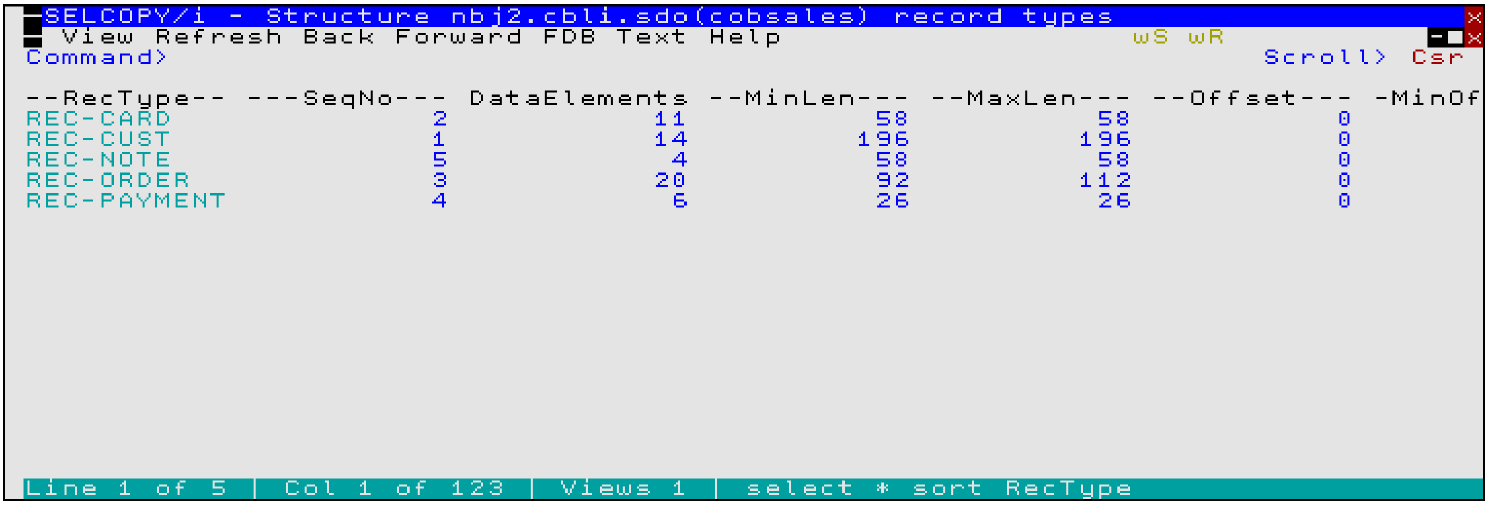
Open a FileKit List window to display each record_type in the specified structure definition object (SDO).

Parameters:

struct_name
The name of the SDO structure.
Default is the name of the SDO structure used in the current SDE window.

Examples:

`display rectypes     NBJ2.FILEKIT.SDO(COBSALES)`
Open a record type list window for the structure definition file NBJ2.FILEKIT.SDO(COBSALES).



```
SELCOPY/i - Structure nbj2.cbli.sdo(cobsales) record types
 View Refresh Back Forward FDB Text Help                         wS wR
Command>
                                                              Scroll> Csr

--RecType-- ---SeqNo--- DataElements --MinLen--- --MaxLen--- --Offset--- -MinOf
REC-CARD              2           11          58          58           0
REC-CUST              1           14         196         196           0
REC-NOTE              5            4          58          58           0
REC-ORDER             3           20          92         112           0
REC-PAYMENT           4            6          26          26           0

Line 1 of 5 | Col 1 of 123 | Views 1 | select * sort RecType
```

Figure 34. Display Record Type List View.

Columns Displayed:

Name	Type	Description
RecType	ALPair	Record type
SeqNo	Int	Record type sequence number
DataElements	Int	Number of Data Elements
MinLen	Int	Minimum length
MaxLen	Int	Maximum length
Offset	Int	Offset
MinOffset	Int	Minimum Offset
UseWhen	ALPair	Record type identification criteria
Flag1	Hex	RTO flags
Flag2	Hex	RTO flags
UseNever	BitFlag	This record-type mapping is NEVER to be applied.
UseAlways	BitFlag	This record-type mapping is ALWAYS to be applied.

DISPLAY STRUCTURE

Syntax:

```
>>-+- DISplay STRUCTure -+--+---------------+--------------------------------><
   |                     |  |               |
   +- LS ----------------+  +- struct_name -+
```

Description:

Open a FileKit List window to display the field definitions for each record_type in the specified structure definition object (SDO).

The List window displaying supports the standard list window features as follow:

- Field Descriptor Block (FDB)
- Edit View
- Selecting, Sorting and Filtering
- Sorting with the Cursor

Parameters:

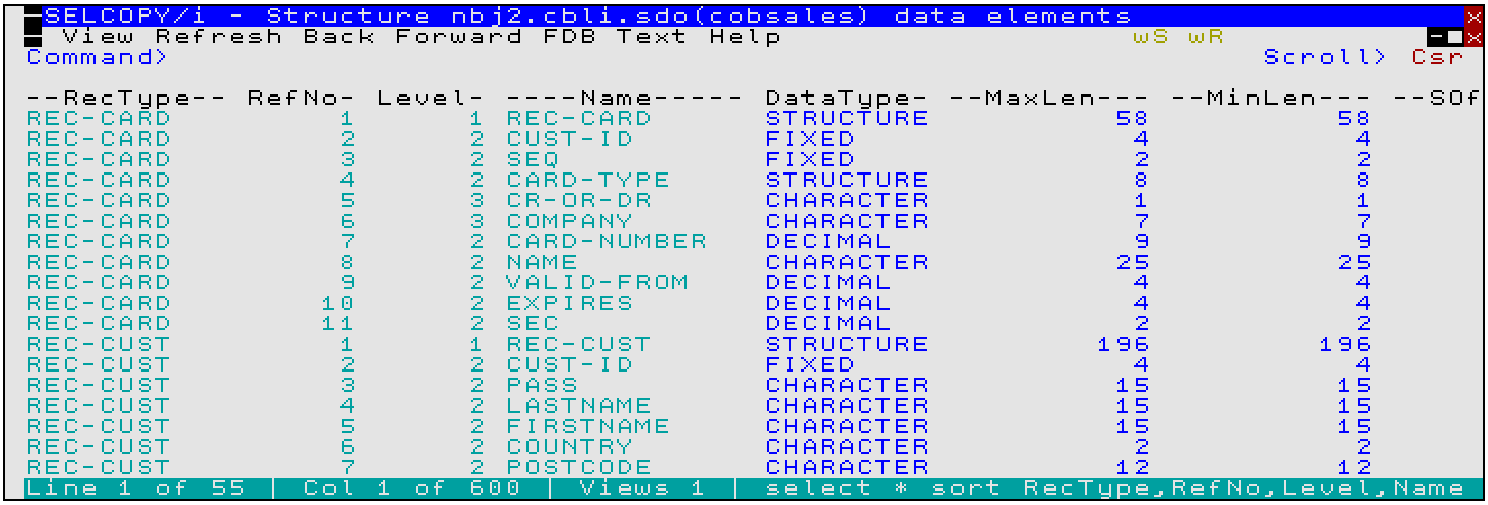
`struct_name`

The name of the SDO structure.
Default is the name of the SDO structure used in the current SDE window.

Examples:

```
display structure   NBJ2.FILEKIT.SDO(COBSALESC)
```

Open a list window for the structure definition PDSE member NBJ2.FILEKIT.SDO(COBSALES).



```
SELCOPY/i - Structure nbj2.cbli.sdo(cobsales) data elements
 View Refresh Back Forward FDB Text Help                        wS wR
Command>                                                          Scroll> Csr

--RecType-- RefNo- Level- ----Name----- DataType- --MaxLen--- --MinLen--- --SOf
REC-CARD         1      1 REC-CARD      STRUCTURE          58          58
REC-CARD         2      2 CUST-ID       FIXED               4           4
REC-CARD         3      2 SEQ           FIXED               2           2
REC-CARD         4      2 CARD-TYPE     STRUCTURE           8           8
REC-CARD         5      3 CR-OR-DR      CHARACTER           1           1
REC-CARD         6      3 COMPANY       CHARACTER           7           7
REC-CARD         7      2 CARD-NUMBER   DECIMAL             9           9
REC-CARD         8      2 NAME          CHARACTER          25          25
REC-CARD         9      2 VALID-FROM    DECIMAL             4           4
REC-CARD        10      2 EXPIRES       DECIMAL             4           4
REC-CARD        11      2 SEC           DECIMAL             2           2
REC-CUST         1      1 REC-CUST      STRUCTURE         196         196
REC-CUST         2      2 CUST-ID       FIXED               4           4
REC-CUST         3      2 PASS          CHARACTER          15          15
REC-CUST         4      2 LASTNAME      CHARACTER          15          15
REC-CUST         5      2 FIRSTNAME     CHARACTER          15          15
REC-CUST         6      2 COUNTRY       CHARACTER           2           2
REC-CUST         7      2 POSTCODE      CHARACTER          12          12
Line 1 of 55 | Col 1 of 600 | Views 1 | select * sort RecType,RefNo,Level,Name
```

Figure 35. Display Structure List View.

Columns Displayed:

Name	Type	Description
RecType	ALPair	Record type
RefNo	Int	Reference number
Level	Int	Level
Name	ALPair	Name
DataType	ALPair	Data type
MaxLen	Int	Maximum length
MinLen	Int	Minimum length
SOffset	Int	Offset in structure
POffset	Int	Offset in parent
BitOff	Int	Bit offset
FType	Char	Field content type
Precision	Int	Precision
Scale	Int	Scale
Picture	ALPair	Picture
Dims	Int	Total dimensions
Dim	Int	Dimension at current level
MaxArray	Int	Maximum number of array elements
MaxIV	Int	Maximum integer value of reference
MinIV	Int	Minimum integer value of reference
Flags1	Hex	Flags
VarLen	BitFlag	Variable inherent length

VarDim	BitFlag	Occurs in array of variable dimension
VarSOff	BitFlag	Offset in structure is variable
VarPOff	BitFlag	Offset in parent is variable
HasVarOff	BitFlag	Group has elements with variable offsets
VarRef	BitFlag	Referred to in size or dimension
UFlags	Hex	Usage flags
Aligned	BitFlag	Aligned
Signed	BitFlag	Signed
Filler	BitFlag	Unnamed
DFlags	Hex	Display flags
LZeros	BitFlag	Display with leading zeros
ASCII	BitFlag	Display with ASCII to EBCDIC translation
Num	Enum	Numeric Type
CPMax	Int	Maximum number of create parameters
CPAct	Int	Actual number of create parameters
CPType01	Enum	Create parameter type 01
CPIntV01	Int	Create parameter integer value 01
CPChar01	ALPair	Create parameter character value 01
CPType02	Enum	Create parameter type 02
CPIntV02	Int	Create parameter integer value 02
CPChar02	ALPair	Create parameter character value 02
CPType03	Enum	Create parameter type 03
CPIntV03	Int	Create parameter integer value 03
CPChar03	ALPair	Create parameter character value 03
CPType04	Enum	Create parameter type 04
CPIntV04	Int	Create parameter integer value 04
CPChar04	ALPair	Create parameter character value 04
DimMax01	Int	Maximum dimension 01
DimMin01	Int	Minimum dimension 01
DimExp01	ALPair	Dimension expression 01
DimMax02	Int	Maximum dimension 02
DimMin02	Int	Minimum dimension 02
DimExp02	ALPair	Dimension expression 02
DimMax03	Int	Maximum dimension 03
DimMin03	Int	Minimum dimension 03
DimExp03	ALPair	Dimension expression 03
DimMax04	Int	Maximum dimension 04
DimMin04	Int	Minimum dimension 04
DimExp04	ALPair	Dimension expression 04
DimMax05	Int	Maximum dimension 05
DimMin05	Int	Minimum dimension 05
DimExp05	ALPair	Dimension expression 05
Rem	ALPair	Remarks string

DOWN

Syntax:

```
>>- DOwn ---+-------------------+--------------------------------------------><
            |                   |
            +-- Cursor ---------+
            |                   |
            +-- CSR ------------+
            |                   |
            +-- Data -----------+
            |                   |
            +-- Half -----------+
            |                   |
            +-- Max ------------+
            |                   |
            +-- Page -----------+
            |                   |
            +-- n_lines --------+
```

Description:

Scroll the view of the data within the SDE window downwards towards the bottom of the file data.

Where no parameter is specified, the scroll amount will be the value specified in the **Scroll>** field in the top right corner of the window display.

Note that the first data record in any multi record view will **always** be preceded by its complete group of column header lines.

For single record views, the standard headers are always visible at the top of the display area and are not included as part of the scrollable text. Each non-header line within a single record view is identified as being a **field data line**.

Where the cursor is positioned at an offset within a column header line of a multi record view, the cursor is deemed to be located at the same offset within the data record that immediately follows the group of header lines. Therefore, DOWN CURSOR will operate successfully when the cursor is positioned within a header line.

UP and DOWN scroll commands will cause the window display area to be adjusted by a number of lines determined by the number of multi record view data records and shadow lines, or single record view field data lines. Other lines in the display area, i.e. Header lines and blank filler lines that precede a group of header lines or follow the "End of Data" record, are not included in this calculation.

Attempting to scroll down beyond the last entry (data record, shadow line or field data line) will make the "End of Data" record the first line of the scrolled display.

Parameters:

`CURSOR`
`CSR`

The data record, shadow line or field data line on which the cursor is positioned becomes the first line of the scrolled display.
If the cursor is positioned on a header line, the data record or field data line immediately following the header line becomes the first line in the display area.
If the cursor is positioned outside the display area or on the first line within the display area, then DOWN PAGE is executed instead.

`DATA`

Scroll down to display one page (display window depth) less one line of data.
The last data record, shadow line or field data line in the current display area becomes the first line of the scrolled display.

`HALF`

Scroll down half a page of data.
The data record, shadow line or field data line that is half way down the page of data in the current display area becomes the first record of the scrolled display.

`MAX`

Scroll down to display the last page of data.
Where more than one page of data exists, the "End of Data" line becomes the last line of the scrolled display. Otherwise, the first line of data becomes the first line of the scrolled display.
Equivalent to the BOTTOM command.

`PAGE`

Scroll down to display the next whole page of data.
The data record, shadow line or field data line following the last line of the current display area becomes the first line of the scrolled display.

n_lines
Scroll down a specified number of lines.
The data record, shadow line or field data line that is *n_lines* below the current line becomes the first line of the scrolled display.

Examples:

`down 5`
Scroll the display area down 5 lines.

See Also:

LEFT RIGHT UP

DROP, DDROP

Syntax:

```
>>- DROP --- struct_name ---------------------------------------------------><

>>- DDROP -- struct_name ---------------------------------------------------><
```

Description:

Drop from storage a Structure Definition Object (SDO) used within SDE, An SDO (file-structure) provides the information required to interpret indiviudal logical fields within a file's data records.

If changes have been made to the structure since it was last saved using SAVESTRUCTURE, (USE WHEN criteria may have been added for instance) then the user will be prompted with a dialog asking whether or not the changes are to be saved.

DDROP removes the SDO without prompting the user to save changes.

Parameters:

struct_name
The name of the structure (SDO) being dropped.

See Also:

CREATE STRUCTURE SAVESTRUCTURE

DUPLICATE

Syntax:

```
                  +--- 1 -----+
                  |           |
>>-- DUPlicate ---+-----------+---------------------------------------------><
                  |           |
                  +- n_lines -+
```

Description:

Duplicate the focus line record a specified number of times.

By default, DUPLICATE operates on both visible data records (not excluded) and EXCLUDED record groups. Records groups that are NOTSELECTED or are SUPPRESSED are not affected.

Parameters:

n_lines
Number of times the focus line is to be duplicated.
The default is 1 line.

Examples:

```
dup 3
```

Duplicate the focus line 3 times.

See Also:

INSERT DELETE

EDIT

Syntax:

```
>>-- Edit ---+-------------- fileid -----------------+---| SDE Opts |--+----->
             |                                       |                 |
             +---------- hfs_fileid ---| HFS Opts |--+                 |
             |                                                         |
             +-- DB2 --+---------+--+-- table_name --+---| DB2 Opts |--+
                       |         |  |                |
                       +- (ssn) -+  +-- view_name ---+

             +-- PROFile SDEPROF -------+
             |                          |
 >-----------+--------------------------+------------------------------------>
             |                          |
             +-- PROFile macro_name ----+
             |                          |
             +-- NOPROFile -------------+

 >-----------+-------------------------------------+-------------------------><
             |                +--------------+     |
             |                |              |     |
             |                v              |     |
             +-- INItcmd - ( -+- sd_command -+- ) -+
```

HFS Opts:

```
               +-- STD -----+
               |            |
    +-- EOL ---+------------+---------------+
    |          |            |               |
    |          +-- CR ------+               |
    |          +-- LF ------+               |
    |          +-- NL ------+               |
    |          +-- CRLF ----+               |
    |          +-- LFCR ----+               |
    |          +-- CRNL ----+               |
    |          +-- string --+               |  +- LRECL lrecl -+
    |                                       |  |               |
 |--+---------------------------------------+--+---------------+-------------|
    |                                       |
    +-- RECFM -+- F ------------------------+
               |                            |
               |     +- (0,2,0) ----------+ |
               |     |                    | |
               +- V -+- (off,len,origin) -+-+
```

SDE Opts:

```
 |--+----------------------------------------------------------+------------->
    |                                                          |
    +- USING -+-------------+-+- struct_name ------------------+
              |             | |                                |
              +- STRUCTure -+ +------ = -----------------------+
              |                                                |
              +- HLAsm -----+--- copybook_name ----------------+
              |             |                                  |
              +- COBOL -----+                                  |
              |             |                                  |
              +- PL1 -------+                                  |
              |             |                                  |
              +- ADAta -----+                                  |
              |                                                |
              |              +-------------------------+       |
              |              v                         |       |
              +- SYMNAMes ( -+---- DFSORT symbols -----+- ) ---+
    +-- FORMat ------- TABle ------+  +- REUse -----+
    |                              |  |             |
 >--+------------------------------+--+-------------+-+------------+-------->
    |                              |  |             | |            |
    +-- FORMat -+--+-- SINGle -----+  +- AUXiliary -+ +- READonly -+
    |           |  |               |  |                            |
    +-- FMT ----+  +-- SNGL -------+  +- UPDate -------------------+
                   |               |
                   +-- CHARacter --+
                   |               |
                   +-- HEX --------+


 >--+-----------------------------------------------------------------+----->
    |                                                                 |
    +- KEYRange - ( - lowkey -+---------------+- ) -------------------+
    |                         |               |                       |
    |                         +- , - highkey -+                       |
    |                                                                 |
    +------------------------------+--+-------------------------------+
    |                              |  |                               |
    +-- FROM --+- KEY -- string ---+  +-- FOR -- n_recs --+-----------+
               |                   |                      |           |
               +- RBA ------ n ----+                      +- RECORDS -+
               |                   |
               +----------+- n ----+
               |          |
               +- RECord -+

    +-- View -------- * ---------+
    |                            |
 >--+----------------------------+--+-----------------------------+--------->
          |                            |  |                             |
          |             +-- , --+      |  +-- AUTostructure --+- APPLY -+
          |             |       |      |                      |         |
          |         +---+-------+---+  |                      +- OFF ---+
          |         |               |  |                      |         |
          |         V               |  |                      +- ON ----+
          +-- View --- record_type -+--+                      |         |
                                                              +- SAVE --+

 >-+----------------------------------+--+-------------------------------+---|
   |                                  |  |                               |
   +-- FILTer --+--- filter_fileid ---+  | +--------------------------+  |
                |                     |  | v                          |  |
                +--| Filter Clause |--+  +-+-- SELect select_syntax --+--+
```

Filter Clause:

```
        +------------------------------------+
        v                                    |
|- ( -+-- INclude rec_type -+--------------+-+-+-+--------------------+- ) -|
      |                     |              |   | |                    |
      |                     +- WHere expr -+   | +- Stopafter n_hits -+
      |                                        |
      | +------------------------------------+ |
      | v                                    | |
      +-- EXclude rec_type -+--------------+-+-+
                            |              |
                            +- WHere expr -+
```

DB2 Opts:

```
|--+----| SQL Query Opts |---------------------+--------------------------->
   |                                           |
   +-- USING --+-------------+-- struct_name --+
               |             |
               +- STRUCTure -+

                                                  +- COMMITONCLEANSAVE -+
                                                  |                     |
>--+------------------------------------------+---+---------------------+--->
   |                                          |   |                     |
   +- WITH -+- CS ----------------------------+   +- COMMITONSAVE ------+
            |                                 |   |                     |
            +- UR ----------------------------+   +- COMMITONEXIT ------+
            |                                 |
            +- RR --+-+-----------------------+
            |       | |                       |
            +- RS --+ +- KEEP -+-- EXclusive -+
                               |              |
                               +-- UPDate ----+
                               |              |
                               +-- SHR -------+
>--+------------+-+--------------------------+-+---------------+-+--------+->
   |            | |                          | |               | |        |
   + SKIPLocked + +- LOCKTABLE +- EXclusive -+ + COMMITONLOAD -+ + AUDit -+
                               |             |
                               +- SHr -------+

   +- PROTECTPRIMEKEY -+  +-- FORMat ------- TABle ------+
   |                   |  |                              |
>--+-------------------+--+------------------------------+--+------------+-->
   |                   |  |                              |  |            |
   +- EDITPRIMEKEY ----+  +-- FORMat -+--+-- SINGle -----+  +- READonly -+
                          |           |  |               |
                          +-- FMT ----+  +-- SNGL -------+

>--+-----------------------------------------------------------------+------>
   |                                                                 |
   +-- SCROLL -------------------------------------------------------+
   |                                                                 |
   +------------------------------+--+-------------------------------+
   |                              |  |                               |
   +-- FROM --+----------+- n ----+  +-- FOR -- n_rows --+-----------+
              |          |                               |           |
              +- ROW ----+                               +- ROWS ----+

                                         +-- XMLLOBWidth -- 0 --------+
                                         |                            |
>--+---------------------------------+---+----------------------------+-----|
   |                                 |   |                            |
   +---- SELect (select_syntax) -----+   +-- XMLLOBWidth -- n_bytes --+
```

SQL Query Opts:

```
|--+------------------------+--+-----------------------------------------+--|
   |                        |  |                                         |
   +- WHERE (where-clause) -+  +- ORDER -+------+--+-- (orderby-clause) -+
             |          |      |         |      |  |                     |
             +--- ? ----+      |         +- BY -+  |                     |
                               +- SORT ------------+                     |
                               |                                         |
                               +- SORTIndex -+-- index_name --+----------+
                                             |                |
                                             +-- Prime -------+
```

Description:

Edit a structured data set or HFS file, or edit a DB2 results table within SDE.

An SDE edit display window is opened and focus is passed to the new window. Depending on the value on the FORMAT parameter, the SDE display is either a multi record window view or a single record window view for the first record selected.

For both Structured Data Edit alpha-numeric (AN) fields and DB2 Table Edit character (CH or VCHAR) fields that contain non-printable text, only the printable characters may be edited. These editable areas of the character data occupy immoveable fixed positions and lengths within the field which are represented by underscores. The non-printable characters may only be edited by setting HEX ON and updating the hex display.

Structured Data Edit:

If the Structure Definition Object (SDO) specified by the USING STRUCTURE parameter is not already in storage, it is loaded from the appropriate Structure Definition File (SDF). Record Type Objects (RTO) within the specified SDO are then used to map the records in the data set.

For every record in the file, each RTO is checked until one is found whose criteria is matched by the record characteristics. This RTO is then used to map the record. If the record does not match any RTO criteria, then the first RTO in the SDO is used by default. The RTO automatically selected to map a record may be changed by the user via the USE WHEN command or the SDE Edit/Browse Utility Window.

By default, all fields within records that have an associated RTO are displayed as printable character, numeric data fields having first been converted to decimal.

If no SDO is specified, then FORMAT CHARACTER is applied to each record. i.e. the data is displayed as a single, variable (RECFM=V) or fixed (RECFM=F) length character field with field name "UnMapped". No data conversion is performed.

If the VIEW parameter is specified with a non-generic argument, the DRECTYPE setting is initialised as the first *record_type* parameter specified on the VIEW parameter. Otherwise, the DRECTYPE setting is initialised as described in *"Default Record Type"*

With the exception of alpha-numeric (AN) fields, record data that does not meet the specifications of its field definition is considered to be invalid and is represented by asterisks.

DB2 Table Edit:

DB2 table edit is supported for in-storage edit/browse only. Therefore, if a results table is too large to be loaded into available storage a storage error occurs. The user must then restrict the number of rows to be loaded using further row selection criteria (i.e. FROM, FOR, SELECT, WHERE parameters).

On starting a DB2 table edit, a temporary SDO is created reflecting field characteristics obtained from the SQL Descriptor Area return by the prepared SQL SELECT execution. Furthermore, a new DB2 connection is made and, optionally, a new FileKit DB2 audit log is allocated for each new SDE edited DB2 base or results table. Opening second and subsequent SDE edit views of the same table data does not open a new DB2 connection or audit log data set.

Parameters:

`fileid`

The fileid of the dataset, library member or PDSE library member generation containing the records to be edited.

fileid may be any of the following:

◊ The DSN of a physical sequential data set.
◊ The DSN of a VSAM (KSDS, ESDS, RRDS, VRDS or LDS) data set.
◊ The library DSN and parenthesised member name of a PDS or PDSE library member.
◊ The library DSN, parenthesised member name and absolute or relative number of a PDSE library member generation as described under z/OS PDSE Library Member Generations. (PDSE version 2 with MAXGENS only.)
◊ The name of a member to be edited from the same PDS or PDSE library as the member displayed in the current data edit window view.

`hfs_fileid`

A complete HFS (or zFS) file path containing data to be edited.
hfs_fileid is identified as an HFS path if *fileid* is not a valid DSN for a sequential, VSAM or PDS/PDSE dataset. e.g. DSN begins with ".", contains invalid special characters (e.g. "/") or contains qualifiers of length greater than 8.

HFS Opts

The following HFS options may be specified to determine the handling of data within the HFS file.

`EOL=STD|NL|CR|LF|CRLF|LFCR|CRNL|`*`string`*

Sets the EOLIN (input end-of-line) delimiter value used to determine the end of each record for non-RECFM F/V input. EOLIN delimiters are not included in the edited record data or record length. EOL parameter elements are as follow:

STD	-	Any standard line-end.
NL	X'15'	New Line.
CR	X'0D'	Carriage Return.
LF	X'0A'	Line Feed.
string	-	A 2-byte user specified character or hex string.

STD is default so that the EDIT operation scans the input data for any of the standard EOL combinations (not *string*), stopping when one is found. This EOL combination is used as EOLIN for the file.

`RECFM F | V (`*`off,len,origin`*`)`

Specifies that the data is to be treated as containing Fixed or Variable length format records.

RECFM F indicates that all records are of a fixed length as defined by the LRECL argument.

RECFM V allows the user to specify the location of the record length fields within the data as follows:

off	Offset of the record length field from the start of the record.
len	Length of the record length field.
origin	The start of the record data at which the record length is applied.

Default is (0,2,0) which describes standard RECFM V organisation data sets.
The length field will be displayed as part of the data, so, unless editing the data using a suitable associated structure (SDO), the user must take care not to corrupt the length field and also maintain it for any change in record length.

`LRECL `*`lrecl`*

Specifies the maximum record length of input records.

Records terminated by an EOL sequence will wrap onto the next line of data if the record length exceeds *lrecl*. Where a record has wrapped, the prefix area contains the "==EOL>" flag. Furthermore, read-only edit is forced in order to suppress save of a wrapped record as multiple, individual records.

For RECFM F data, *lrecl* is the fixed length of the records in the edit view. If the file size is not a multiple of the fixed format lrecl value then, an error occurs and edit is cancelled. For display purposes only, using BROWSE with any *lrecl* value will display the data with the last record padded with blanks up to the *lrecl* length.

If the record length field of a RECFM V record exceeds the *lrecl* value, then an error is returned.

RECFM V and EOL delimited records have default *lrecl* of 32752, wheras RECFM F records have default *lrecl* of 80.

SDE Opts

The following SDE options are applicable to edit of structured data found in HFS files or z/OS data sets only. See DB2 Opts for DB2 table edit options.

`USING (STRUCTURE) struct_name`

Identifies the SDO to be applied to the data records. *struct_name* is the SDF fileid assigned to the SDO in a CREATE STRUCTURE command.

"=" (equals) may be specified instead of *struct_name* to indicate the name of the structure being used in the current SDE window.

`USING HLASM | COBOL | PL1 | ADATA copybook_name`

Identifies the full fileid of *copybook_name*, an Assembler/COBOL/PL1 copy book or Assembler/COBOL/PL1 ADATA output file to be used to format the data records.

Using *copybook_name* syntax has a processing overhead in that a CREATE STRUCTURE operation is performed to generate a temporary SDO. This overhead is greater if HLASM, COBOL or PL1 is specified since an assemble/compile of the copybook is performed to first obtain the ADATA file output. Therefore, it is recommended that a non-temporary SDO is generated using CREATE STRUCTURE and that this is used for future formatting of the data (on EDIT, BROWSE, FSU, etc.)

`USING SYMNAMES ( DFSORT symbols )`

Specifies DFSORT SYMNAME symbol definitions that are to be used to format the data records. The order in which symbol definitions are supplied dictate the order in which the fields will occur in the record type definition.

The symbol name definitions within the SYMNAMES parentheses may be supplied directly in-line and/or via input data sets/library members.

```
SYMNAMES( Card,06,04,CH  Dept,46,03,CH  Amount,49,06,PD  )
SYMNAMES( SYS1.MACLIB(EDGSMFSY)  SYS1.MACLIB(EDGSRCSY) )
SYMNAMES( CBL.DFSORT.SYM(CBLATRAC)  TCB,*,4,BI )
```

`FORMAT`
`FMT`

Specifies the format in which record data will be displayed in the initial SDE view.

`SINGLE` `SNGL`	Single record format.
`TABLE`	Table format. (Default)
`CHARACTER`	Multi record view with all records or record segments mapped as single, variable length character fields with field name "UnMapped" or "UnMappedSeg". No data conversion is performed.
`HEX`	Same as CHARACTER with the addition that the data is also displayed in Hex below the character display. Note that the Hex display occupies an additional 2 lines of data.

The data display format may be later altered using the FORMAT and/or ZOOM commands.

`AUTOSTRUCTURE`

Temporarily overrides the current setting for option AUTOSTRUCTURE. This option will specify whether or not a structure to dataset association is automatically defined and/or applied for the current EDIT operation.

`APPLY`	Unless a structure is specified via a USING parameter, APPLY will attempt to use a structure associated with a filemask that matches the edited *fileid*. If no association has been saved, then no structure is used.
`OFF`	No attempt will be made to automatically apply an associated structure and, if a structure is specified via a USING parameter, no attempt will be made to save an association between the structure and the edited fileid.
`ON`	Equivalent to both APPLY and SAVE.
`SAVE`	If a structure is specified via a USING parameter, save an association between it and the edited fileid. This association may be automatically applied when the fileid is next browsed or edited.

`READONLY`

Perform read-only edit of the data.

Unless the fileid of the edited data set is changed, attempts to save changes made to the data during the SDE edit session will give an error.

`REUSE`

Where possible, full edit capabilities of record move, copy, insert, change and delete are to be permitted for the data set being edited. Also, where the data set organisation and SDO structure permits, data changes may result in changes to record lengths.

FileKit SDE first determines whether the entire data set may be comfortably loaded into available storage. If not, parameter REUSE is ignored and AUXILIARY edit becomes default.

Full (REUSE) edit is not possible for VSAM RRDS and ESDS data sets defined with NOREUSE, in which case parameter REUSE is ignored and the ESDS/RRDS data set is edited using Update-in-place edit. For all other data sets, parameter REUSE is the default.

`UPDATE`

Update-in-place Edit is to be used so that existing records may only be altered and replaced without changing the record length. Records cannot be moved, copied, inserted or deleted.

Use of Update-in-place Edit has significant performance advantages over full (REUSE) edit when editing large, non-KSDS data sets.

UPDATE is default if the edited data set is a VSAM ESDS or RRDS file defined with NOREUSE or the data set is non-KSDS and its load into storage has been interrupted by the user via the Load Threshold break-in modal window.

`AUXILIARY`

Select Auxiliary edit for non-KSDS data sets. AUXILIARY is ignored for edit of a KSDS data set.

Auxiliary edit is identical to full (REUSE) edit except that the data set to be edited is first copied to an auxiliary data set on disk. This allows full editing capabilities for data sets that are too large to be loaded into available storage.

`KEYRANGE` *`(lowkey, highkey)`* `| KEYRANGE` *`(lowkey)`*

Records are to be displayed with keys in the given range. This option applies to VSAM KSDS files only.

If *highkey* is ommitted it is assumed to have the same value as *lowkey*.

If *lowkey* is shorter than the defined length of the KSDS key, it is assumed to be padded on the right with X'00'.

If *highkey* is shorter than the defined length of the KSDS key, it is assumed to be padded on the right with X'FF'.

The key values can be given in the form:

- Simple strings with no embedded blanks. These are translated to upper case.
- Quoted strings using apotrophes or double quotes. These are translated to upper case.
- Character strings of the form *C'...'* containing any characters. These are not translated to upper case.
- Hexadecimal strings of the form *X'...'* containing hex digits 0-9 a-f A-F. These are not translated to upper case.

Use of the KEYRANGE parameter will force Update-in-place Edit.

`FROM`

Records are to be displayed beginning at a specific location within the data set. The first record at the specified location becomes record 1 within the SDE window view of the data set. Records that occur before this location are not included within the edit session.

If FILTER is specified, then record filtering occurs only on records selected using the FROM and FOR parameters.

Required record location is identified via one of the following:

`KEY` *`string`*	For KSDS data sets only, locate the record with a key string equal to *string*. If not found, the record with the next key string greater than *string* is used.
`RBA` *`n`*	For KSDS and ESDS data sets only, locate the record starting at the relative byte address specified by *n*. The RBA must point at the start of the record otherwise a VSAM point error will occur. RBA address for each record is displayed as part of the record information columns within an SDE window view. Display of these columns is controlled using the SDE SET RECINFO CLI command.
`< RECORD >` *`n`*	For any data set organisation, locate the record number specified by *n*. Specification of parameter keyword RECORD is optional.

Use of the FROM parameter will force Update-in-place Edit.

Default is to display records within the SDE window view starting at the first record in the data set.

`FOR` *`n_recs`* `< RECORDS >`

Specifies the number of data set records to be included within the SDE window edit session. Specification of parameter keyword RECORDS is optional.

If FILTER is specified, then record filtering occurs only on records selected using the FROM and FOR parameters.

Use of the FOR parameter will force Update-in-place Edit.

Default is to include all records.

`VIEW` *`record_type`*
`VIEW *`

Identifies one or more record types for which records will be displayed in the initial SDE view. Records that are associated with other record types are not visible within the display but are displayed as shadow lines instead. ***** (asterisk) indicates all record types including the internal record type "UnMapped" (or "UnMappedSeg") which is used to map data records that have no associated RTO in the SDO.

Records within the display may be later suppressed or made visible using the VIEW command.

Default is VIEW *.

`FILTER` *`filter_fileid`* `|` **`Filter Clause`**

FILTER specifies additional record filtering criteria to further reduce the display of records for edit. All record filters are applied only to records that have been selected using the FROM and/or FOR parameters, otherwise it applies to all records in the structured data file.

Enough available storage must exist to load all records selected by the filter otherwise a storage error occurs and the edit is terminated.

FILTER parameters are specified via a filter clause which may be supplied as part of the EDIT command or referenced via *filter_fileid*, a separate sequential data set, PDS/PDSE member or HFS file. *filter_fileid* must contain the keyword FILTER followed by a valid filter clause.

The filter is applied only once during initial load of the data. A record will not be excluded from the display of edited records if its record type or data is changed in a way that no longer satisfies the filter criteria.

Use of FILTER will force Update-in-place Edit.

`SELECT` *`select_syntax`*

SELECT may be used to customise the sequence and number of record-type columns displayed when the edit view of formatted data is opened. The *select_syntax* argument represents parameters supported by the SELECT primary command with the exception of IN *struct_name* which is unnecessary.

Unlike parameter INITCMD, which may also be used to execute SELECT commands, the SELECT parameter column selection is performed before the profile macro is executed.

Filter Clause

A filter clause must be specified in "( )" (parentheses) and may contain comment data enclosed by "/*" and "*/". If filter clause is specified via *filter_fileid*, then comment data may also occur before and after the filter clause.

The filter clause may contain either INCLUDE or EXCLUDE selection sub-clauses but not both. When a filter is applied, the record data is tested against each selection sub-clause in the order specified until a true condition is returned. On encountering a true condition, both the hit count for the individual selection sub-clause and the overall hit count is incremented by one and the record included or excluded as approriate.

Once the hit count for an individual INCLUDE or EXCLUDE selection sub-clause matches its LIMIT threshold, then that sub-clause plays no further part in the filter when applied to subsequent edit input records. Similarly, once the overall hit count matches the STOPAFT threshold, no further record filtering occurs and the remainder or the records are excluded or included as appropriate.

The following options are supported by the filter clause.

`INCLUDE` *`record_type`*

INCLUDE denotes the start of an INCLUDE sub-clause which, together with optional parameters WHERE and LIMIT, identifies conditions for which a record is included in the subset of edited records.

The INCLUDE sub-clause applies only to records that have been assigned the specified *record_type*. If the record is not assigned this record type, a false condition is returned for the sub-clause.

A number of INCLUDE sub-clauses may be specified for different record types or for the same record type with different WHERE expression conditions. If one or more INCLUDE sub-clauses are specified, then records that do not satisfy any of the sub-clause conditions will be excluded by default.

Note that *record_type* "Record" (with field name "UnMapped") may be used to perform a filter on the unformatted record data whether or not a structure (USING *struct_name*) has been specified. In this way, a filter may test **all** records regardless of their assigned record type.

INCLUDE and EXCLUDE parameters are mutually exclusive.

`EXCLUDE` *`record_type`*

EXCLUDE denotes the start of an EXCLUDE sub-clause which, together with optional parameters WHERE and LIMIT, identifies conditions for which a record is excluded from the subset of edited records.

The EXCLUDE sub-clause applies only to records that have been assigned the specified *record_type*. If the record is not assigned this record type, a false condition is returned for the sub-clause.

A number of EXCLUDE sub-clauses may be specified for different record types or for the same record type with different WHERE expression conditions. If one or more EXCLUDE sub-clauses are specified, then records that do not satisfy any of the sub-clause conditions will be included by default.

Note that *record_type* "Record" (with field name "UnMapped") may be used to perform a filter on the unformatted record data whether or not a structure (USING *struct_name*) has been specified. In this way, a filter may test **all** records regardless of their assigned record type.

INCLUDE and EXCLUDE parameters are mutually exclusive.

`WHERE` *`expr`*

WHERE applies further filter conditions to records assigned to the record type specified by the last INCLUDE *record_type* or EXCLUDE *record_type* parameter processed.

expr is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of the WHERE expression must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

The WHERE expression is applied to each record assigned the record type *record_type* and, if the result is "true", the record is selected for include or exclude as indicated by the prevailing INCLUDE or EXCLUDE filter. If multiple INCLUDE/EXCLUDE *record_type* WHERE expressions exist for the same record type, then a logical OR is implied for all the expressions relating to that record type.

`LIMIT` *`n_hits`*

LIMIT *n_hits* may be included as part of a single INCLUDE (or EXCLUDE) sub-clause specification and applies only to that sub-clause.

When the number of records selected by an individual INCLUDE (or EXCLUDE) sub-clause reaches the *n_hits* value specified by LIMIT, then that sub-clause no longer forms part of the filter applied to subsequent input records.

LIMIT provides a method whereby the subset of records selected for edit is spread more evenly across the filter's sub-clause conditions than could be achieved by use of the STOPAFTER parameter alone.

By default, no LIMIT threshold is applied INCLUDE (or EXCLUDE) sub-clause.

`STOPAFTER` *`n_hits`*

When the total number of records selected by the INCLUDE (or EXCLUDE) sub-clauses reaches the *n_hits* value specified by STOPAFTER, then no further filter testing occurs.

If an INCLUDE filter, then all remaining untested records are excluded. If an EXCLUDE filter, then all remaining untested records are included.

By default, no STOPAFTER threshold is applied to the filter clause.

`DB2 < (`*`ssn`*`) >`

Indicates that edit is for a DB2 base or results table. *ssn* is optional and identifies the local DB2 sub-system name to which a connection will be made.

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 users default DB2 sub-system name as set by the DB2 Primary Options menu.

`table_name`

A DB2 base table name containing rows of data to be edited.
table_name may be specified with 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default location is the local DB2 server and the default schema is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID).

`view_name`

A DB2 table view which references an SQL query used to generate a results table containing the rows of data to be edited.

view_name may be specified with 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default location is the local DB2 server and the default schema is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID).

DB2 Opts

The following DB2 options are applicable to edit of DB2 table data only. See SDE Opts for structured data edit options.

`WHERE (`*`where-clause`*`)`

Specifies a DB2 SQL WHERE clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *where-clause*.

If *where-clause* is null or "?" (question mark) the DB2 Row Selection Panel is opened before the edit is actioned. On closing the panel, the edit is actioned using the generated WHERE clause.

`SORT|ORDER BY (`*`order-by-clause`*`)`

Specifies a DB2 SQL ORDER BY clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *order-by-clause*.

`SORTINDEX` *`index_name`* `| PRIME`

Specifies *index_name*, the name of an existing DB2 Index for the table being edited. The index identifies the key columns/expressions by which the table rows will be ordered in the display.

PRIME may be specified as an alternative to indicate that the primary index should be used.

If *index_name* is "?" (question mark), the DB2 Select Table Index window is opened before the edit is actioned, allowing selection of one of the indexes defined for the table. On selecting an entry, the window is closed and the edit operation continues using the SORTINDEX *index_name*.

`USING (STRUCTURE)` *`struct_name`*

Identifies the SDO to be applied to the DB2 data rows. *struct_name* is the SDF fileid assigned to the SDO in a CREATE STRUCTURE command.

If the USING STRUCTURE option is specified then SELECT, WHERE and ORDER BY options are ignored. The SDO may contain DB2 SQL query SELECT, WHERE and ORDER BY sub-clause options which are used to select and display DB2 table rows. Other DB2 table edit options, apart from LOCKTABLE, FORMAT, READONLY, FROM and FOR, may also exist in the SDO and so define default values which may be overridden by specific parameters on the EDIT command. e.g. WITH isolation-clause, SKIPLOCKED, COMMIT.

`SKIPLOCKED`

Ignored unless an isolation level of Cursor Stability (CS) or Read Stability (RS) is in effect, SKIPLOCKED specifies that any selected rows that are already locked by another process should be skipped and not be included in the edit display. See "*DB2 SQL Reference*" for details on the SKIP LOCKED DATA clause.
Default is to allow display of locked rows whenever possible.

`PROTECTPRIMEKEY | EDITPRIMEKEY`

Specifies whether data occupying columns that comprise the table's primary key is eligible for update (EDITPRIMEKEY) or is read only (PROTECTPRIMEKEY).
Default is PROTECTPRIMEKEY.

`WITH CS|UR|RR|RS  < KEEP  EXCLUSIVE|UPDATE|SHR >`

Specifies a DB2 SQL isolation-clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for details on the *isolation-clause* and "*DB2 Performance Monitoring and Tuning Guide*" for details on the effects of isolation level on concurrency and protection of DB2 table data.

The KEEP sub-parameter options are applicable to isolation levels RR (Repeatable Read) and RS (Read Stability) only.
Default isolation level (as set by the FileKit DB2 package and plan BIND) is CS.

`COMMITONCLEANSAVE | COMMITONSAVE | COMMITONEXIT`

Specifies when a COMMIT should be executed for changed data.

`COMMITONCLEANSAVE`	COMMIT only if SAVE is executed without errors.
`COMMITONSAVE`	COMMIT on SAVE regardless of errors.
`COMMITONEXIT`	COMMIT only on exit of the edit session.

Default is COMMITONCLEANSAVE.

`LOCKTABLE EXCLUSIVE|SHR`

For DB2 base tables only, executes a DB2 SQL command LOCK TABLE before loading data from the table. Use this option with **caution** as other users and applications may be prevented from accessing the table. If LOCKTABLE is specified the lock is held until the next COMMIT is actioned.

LOCKTABLE EXCLUSIVE prevents another process from performing any operations on the table whilst it is being edited, unless the process is running with an isolation level of Uncommitted Read (UR) in which case read-only (dirty read) operations may be performed.

LOCKTABLE SHR prevents anything other than read-only operations to be performed on the table whilst it is being edited.

See "*DB2 SQL Reference*" for details on the effects of LOCK TABLE options.
Default is not to perform any explict table locking prior to load. (Recommended)

`COMMITONLOAD`

Perform a COMMIT following the initial load of rows to be edited, thus releasing all DB2 table locks performed during load of the data. This includes any explicit table locks applied via the LOCKTABLE parameter.
Default is not to perform a COMMIT following load of the table rows.

`AUDIT`

Open a new FileKit DB2 audit data set to record changes to the edited table made during this edit session. See Audit Trail Functions for details.
Default is not to perform edit auditing.

`FORMAT`

Specifies the format (TABLE or SINGLE) in which record data will be displayed in the initial SDE view. See FORMAT for structured data edit for details.
Default is FORMAT TABLE.

`READONLY`
Perform read-only edit of the data. Attempts to save changes made to the data during the SDE edit session will give an error.

`SCROLL`
Use a DB2 scrollable SENSITIVE STATIC cursor to fetch DB2 rows. This keyword is incompatible with the FOR and FROM keywords. See Using Scrollable Cursors for information on how the use of this keyword affects the edit session.

The use of DB2 scrollable cursors may be disabled by the installer of FileKit. In order to use them the system INI file must contain the *DB2.SCROLL=YES* variable.

`FROM < ROW > n`
Equivalent to FROM RECORD *n*, FROM ROW *n* specifies that rows are to be displayed beginning at row number *n* in the DB2 results table returned by the SQL query. The first row displayed becomes row 1 within the SDE window edit view. Rows that occur before this row number are not included within the edit session.
Default is row 1.

`FOR n_rows < ROWS >`
Equivalent to FOR *n_recs* RECORDS, FOR *n_rows* ROWS specifies the maximum number of rows to be displayed from the DB2 results table returned by the SQL query. Rows that fall outside the range of rows specified by FROM *n* FOR *n_rows* are not included within the edit session.
Default is to display all selected rows.

`SELECT (select_syntax)`
DB2 table edit will fetch **all** columns from the selected table or view.

SELECT may be used to select a subset of these columns for initial display in the edit window view. Except for parameters FROM *record_type* and IN *struct_name* which are both unnecessary, the *select_syntax* argument represents parameters supported by the SELECT primary command.

Unlike parameter INITCMD, which may also be used to execute SELECT commands, the SELECT parameter column selection is performed before the profile macro is executed.

`XMLLOBWIDTH n_bytes`
Specifies the number of bytes (*n_bytes*) of text, at the start of an XML or large object (LOB) column, to be displayed for all XML and LOB columns in the edited table view.

Note that XML and LOB text cannot be updated directly in the table edit view. To update data of this type, primary command XMLEDIT must be used.

`PROFILE macro_name`
Specifies *macro_name*, the name of the REXX SDE edit macro to be executed as the profile when the data is edited. The *macro_name* must exist in a library within the SDE macro path.

The PROFILE option only alters the profile used for the file currently being edited. It does not define the profile macro to be used for subsequent SDE edit.

The default is to execute a macro with member name SDEPROF if it exists.

`NOPROFILE`
Suppresses use of a profile macro when editing the file.
The NOPROFILE option only suppresses use of a profile for the file currently being edited. It does not suppress use of a profile macro for subsequent SDE edit.

`INITCMD (sd_command ... )`
Specifies a list of one or more structured data commands to be executed when initialising the edit session. These commands are issued after the profile macro (if specified) has been executed but before the first view is displayed.

The commands are listed, separated by blanks, within parentheses following the *INITCMD* keyword. If a command contains blanks or special characters (including apostrophes or quotes) it must be delimited with apostrophes or quotes. When such a delimiter is used, instances of the delimiter character within the command string must be represented by a pair of delimiter characters. For example:

```
  INITCMD( 'f "O''Reilly" #3')
```

could be used to open the edit session with the dataset scrolled to the first record containing *O'Reilly* in field number 3.

Examples:

```
<sd edit CBL.SDE.MOD.ZJ2202  using CBL.FILEKIT.SDO(PRODL)  view ARCX1,PRDMST,PRODX
```
Edit data set "CBL.SDE.MOD.ZJ2202" using Structure Definintion Object (SDO) "CBL.FILEKIT.SDO(PRODL)" to map records.
Initially, all records of type ARCX1, PRDMST and PRODX are displayed.

```
<sd edit  CBL.SAMP01.KSDS  using CBL.FILEKIT.SDO(SAMP01)  from key XMHW003 for 50
```
Edit VSAM KSDS data set "CBL.SAMP01.KSDS" using SDO "CBL.FILEKIT.SDO(SAMP01)" to map 50 records starting at the record with key "XMHW003" (or the next record in key sequence if this key is not found.)

```
<sd edit  db2(CBLA)  ZZS.ZZSSYSMOD  where(SYSMOD LIKE 'RS%')  editprimekey
```

Edit filtered rows of DB2 table ZZS.ZZSSYSMOD in subsystem CBLA allowing update of the table's primary key columns.

See Also:

BROWSE CREATE STRUCTURE EDITDIALOG

EDITDIALOG

Syntax:

```
>>-- EDITDialog -------------------------------------------------------------><
```

Description:

Open the Structured Edit dialog window to edit or browse a data set using an SDE structure (SDO).

This dialog window may also be opened by selecting **Structured Edit** from the File menu in the CBLe frame window menu bar.

EMSG

Description:

SDE CLI command, EMSG, performs the same operation as the CBLe CLI command EMSG. See EMSG in CBLe Text Edit documentation.

END

Syntax:

```
>>--+-- END ---+-------------------------------------------------------------><
    |          |
    +-- QUIT --+
```

Description:

Close an SDE window view opened for EDIT or BROWSE of data. Set on **PF3** by default.

If additional SDE views exist for the data, then these will remain open.

For EDIT only, if only one SDE view exists for the data and that data includes unsaved changes, then one of the following will occur depending on the status of the AUTOSAVE option:

AUTOSAVE	Action Taken
ON (PROMPT or NOPROMPT)	Changes to the data are saved without prompting the user before doing so.
OFF NOPROMPT	Changes to the data are discarded. (Same as QQUIT)
OFF PROMPT	User is prompted to save the changes, discard the changes or to terminate the END command and return to the SDE edit view.

Furthermore, if a non-temporary structure (SDO) is used to map the record data and changes have been made to that structure during the course of the edit session (e.g. USE WHEN), then the user will be prompted to save the changed structure to its structured data file (SDF) regardless of AUTOSAVE status. See command, SAVESTRUCTURE.

See Also:

SET AUTOSAVE CANCEL DROP, DDROP QQUIT

ENUMS

Syntax:

```
>>--+-- ENUMs ----+----------------------+-----------------------------------><
    |             |   |                  |
    +-- ENUML ----+   +-- struct_name ---+
    |             |
    +-- LENums ---+
    |             |
    +-- LSENums --+
```

Description:

Open a FileKit List window to display the enum definitions for each record_type in the specified structure definition object (SDO).

The List window displaying supports the standard list window features as follow:

- Field Descriptor Block (FDB)
- Edit View
- Selecting, Sorting and Filtering
- Sorting with the Cursor

Parameters:

struct_name

The name of the SDO structure.
Default is the name of the SDO structure used in the current SDE window.

If no *struct_name* is specified and the cursor is positioned on an ENUM field in the display, then the list will contain only the ENUM values for that field definition.

ERROR

Syntax:

```
>>--+-- ERRor -----+---------------------------------------------------------><
    +-- SQLError --+
```

Description:

Display the SQL error panel describing the non-zero SQLCODE returned from the last attempt to save the focus row.

This command is only applicable to DB2 table edit. It has the same effect as the E prefix command entered in the prefix area of the focus row.

See the DB2 Save SQL Error Panel for a description of how SQL errors are displayed.

EXCLUDE

Syntax:

```
                 +- EQ -+             (1) +- ANY ---+
                 |      |                 |         |
>>-+- EXclude -+-+------+- string --+-----+---------+------------------------>
   |        |  | |      |           |     |         |
   +- X ----+  | +- op -+           |     +- FOCus -+
               |                    |
               +- VALID ------------+
               |                    |
               +- INVALID ----------+

   +- NEXT --+  +- CHARs --+
   |         |  |          |
>>-+---------+--+----------+--+------+--------------------------------------->
   |         |  |          |  |      |
   +- ALL ---+  +- PREfix -+  +- EX -+
   |         |  |          |  |      |
   +- FIRST -+  +- SUFfix -+  +- NX -+
   |         |  |          |  |      |
   +- LAST --+  +- WORD ---+  +- X --+
   |         |
   +- PREV --+

   +-- #ALL ------------------------------------+ +- .ZFIRST ---- .ZLAST -+
   |                                            | |                       |
 >-+--------------------------------------------+-+-----------------------+--><
   |                                            | |                       |
   +-- pos1 ---+---------+----------------------+ +- .name1 --+-----------+
   |           |         |                      |             |           |
   |           +- pos2 --+                      |             +- .name2 --+
   |                                            |
   |      +----+---------+----------------+     |
   |      |    |         |                |     |
   |      |    +-- , ----+                |     |
   |      v                               |     |
   +-- ( -+-- field_col ------------------+- ) -+
          |                               |
          +--  field_col1:field_col2  ----+
```

(1) Default (ANY or FOCUS) set by the RTSCOPE option.

Description:

Exclude data records in the current SDE edit or browse view that satisfy a search for the specified character string or numeric value.

If the **FOCUS** option is specified then only records/segments assigned the default record type (visible and EXCLUDED records) are included in the search. Otherwise all records types are searched.

Record groups that are already excluded will remain excluded, therefore, execution of successive EXCLUDE commands has a cumulative effect.

EXCLUDE employs the same string search and field highlighting methods as the FIND command. All occurrences of the search string or numeric value that are not excluded by the EXCLUDE command, are highlighted in the text. A record group which has been excluded via the EXCLUDE command, will contain the highlighted field matching the search string if it is subsequently redisplayed. (e.g. using RESET EXCLUDED) Enter the RESET FIND command to turn off the highlighting.

Use of EXCLUDE with no parameters repeats the last EXCLUDE command executed including all its specified parameters.

The FORMAT of the SDE display affects the execution of EXCLUDE.

Unformatted Multi or Single Record Display (CHAR or UNFMT):

By default, a character compare for the supplied search string is performed against the entire length of unformatted data records.

The prevailing BOUNDS left and right column values define the area of the record within which the search occurs. i.e. the matched data must begin at or after the left bound and not exceed the right bound.

The BOUNDS columns may be overridden using *pos1* and *pos2* positional parameters.

field_col and #ALL parameters are not applicable to these display formats and are ignored.

Formatted Multi or Single Record Display (VFMT or FMT):

For formatted records, the search string is compared against **individual fields** that have been selected for display in the formatted data record. (See SELECT)

Fields are searched from left to right in the order that they appear in the display. This is true regardless of the order in which field columns are specified on the EXCLUDE command, or the order in which fields are encountered within the unformatted record.

The prevailing BOUNDS left and right column values define which of the fields within the formatted records are eligible to be searched. i.e. Fields are eligible only if they exist within the left and right bounds when applied to the expanded record data (which is not necessarily the same as the unformatted record data.)

Where a BOUNDS column occurs within a field definition, then the following rules apply:

1. For character data type fields, only the area of the field that falls within the BOUNDS columns is eligible for the search.
2. For numeric data type fields, the field is not eligible to be included in the search.

If one or more field columns are specified on the EXCLUDE command (using *field_col* or *field_col1*:*field_col2*) that do not reference at least one field defined by the BOUNDS columns as being eligible for search, then the following error message is returned:

```
ZZSD280E No fields selected within the current bounds for the EXCLUDE command.
```

If a field column is specified on the EXCLUDE command that is not part of the display (e.g. a group field or a field that has been removed from display by a SELECT command), then the following error message is returned:

```
ZZSD179E Data element field_col is not selected in the current view
    of record type record-type of structure struct_name.
```

For character data type fields, a string compare is performed. For numeric data type fields (binary, packed decimal, floating point, zoned, etc.), then the following will occur:

1. If *pos1*, *pos2* positional parameters are **not** specified, the search string is interpreted as a signed numeric value and an arithmetic compare is performed against the field's formatted numeric value.

   The length and data type of the numeric field, and the number of digits in the search value are not significant. e.g.

   ```
   EXCLUDE  67
   ```

   Excludes records containing a numeric field with value "67" (e.g. "0067", "67.00", "0.0670E+03") or a character field containing the string "67" (e.g. "167 Baker Street").

   If the search string is non-numeric, then numeric data type fields are not searched. Therefore, to bypass searching numeric data type fields, explicitly set the search string to be character or hex using 'string', C'string' or X'string' formats respectively. e.g.

   ```
   EXCLUDE    '67'
   EXCLUDE   C'67'
   EXCLUDE   X'F6F7'
   ```

   Excludes records only if a character field contains the string "67".

2. If *pos1*, *pos2* positional parameters are specified, the search string is interpreted as being a character string and so a string compare is performed against the unformatted representation of the field data for fields falling entirely or partly within the range of record positions.

   Although the range of positions may span a number of fields, the search is still performed against individual fields within the range. i.e. a match for the search string will not occur for data that spans a field boundary.

   A match for the search string may occur on just part of the unformatted data representation of a numeric field. e.g.

   ```
   EXCLUDE    476    21  100
   ```

   This will find a match in any numeric field where unformatted representation of the data contains the string "476" and exclude the record. (e.g. a zoned decimal field with value "14760" or "-4762")

Parameters:

op

A relational operator used in the compare operation which determines the relationship that the data must have with the EXCLUDE search string in order for it to be identified as a successful match.

Valid values for *op* are as follow:

Operator	Description
EQ	Data must be equal to *string*. (Default)
NE	Data must be not equal to *string*.
GT	Data must be greater than *string*.
GE	Data must be greater than or equal to *string*.
LT	Data must be less than *string*.
LE	Data must be less than or equal to *string*.

If a character string compare is performed, the EBCDIC values assigned to characters in the search and data strings determine the relationship (equal to, greater than or less than) between the two strings.

string

The EXCLUDE search string. The seach string may be any of the following:

◊ An unquoted numeric value. The search string is treated as a numeric value when a numeric field is searched in a formatted record view. In all other cases a numeric search string is treated as a character string.

◊ An unquoted character string containing no commas or blanks. The search for the character string will be case-insensitive so that uppercase and lowercase characters are treated as being the same.

◊ A character string enclosed in single (') or double (") quotation marks. The search string may contain embedded commas and blanks and the character string will be case-insensitive.

Two adjacent quotation mark characters that are embedded in a search string which is enclosed by the same quotation mark characters, will be treated as a single occurrence of the character. e.g.

```
EXCLUDE  'Jim O''Brien'
```

Find the character string "Jim O'Brien".

◊ A character string enclosed in single (') or double (") quotation marks with the prefix C. This is equivalent to specifying a quoted search string but that the string search will be case-sensitive. (e.g. C'Book')

◊ A hexadecimal string enclosed in single (') or double (") quotation marks with the prefix X.

◊ A picture string enclosed in single (') or double (") quotation marks with the prefix P.

Picture strings use special characters to represent a generic group of characters as described below. Any character in a picture string that is not one of these special characters is untranslated.

String	Description
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.

◊ A regular expression enclosed in single (') or double (") quotation marks with the prefix R.

Regular expressions use special characters for complex pattern matching. See Regular Expressions in Text Editor documentation for detailed description.

If a no search string is specified and there are no other parameters, then the command is treated as RFIND.

However, if a no search string is specified but one or more *field_col* is supplied then it is treated as a special case that means accept any value in the specified field column(s). This is a way of excluding the next/prev/all occurrence(s) of a record that includes the specified field column(s).

"X ALL" with no search string or field column(s) is another special case and will exclude all data records of any record type.

"X ALL FOC" with no search string or field column(s) will exclude all data records of the focus record type only.

VALID

Intended for use with formatted records, VALID will search fields for valid data. i.e. data that satisfies the field's assigned data type.

INVALID

Intended for use with formatted records, INVALID will search fields for invalid data. i.e. data that does not satisfy the field's assigned data type.

ANY

All record-types are included in the search provided they are not suppressed. See VIEW , VBASE , and V/V+/V-line-commands to suppress and unsuppress record-types.

FOCUS

Only the default record type is included in the search. This is normally the type of record at the top of the screen or at the cursor location. This option was the default in earlier versions of FileKit and can be made so again using the RTSCOPE option or via the settings panel (=0.4).

ALL

Search forwards from the top of the file data (i.e. the first position of the first data record) and excluded all records containing an occurrence of the string. EXCLUDE ALL with no other parameters excludes all records of the default record type.

`FIRST`

Search forwards from the top of the file data (i.e. the first position of the first data record) to exclude the first record of the default record type to contain an occurrence of the search string.

`LAST`

Search backwards from the bottom of the file data (i.e. the last position of the last data record) to exclude the last record of the default record type to contain an occurrence of the search string.

`NEXT`

Search forwards from the current cursor location to exclude the next record of the default record type to contain an occurrence of the search string. If the cursor is not within the window's data display area, the search begins at the first position of the first visible or excluded record within the display area that is of the default record type.

`PREV`

Search backwards from the current cursor location to exclude the previous record of the default record type to contain an occurrence of the search string. If the cursor is not within the window's data display area, the backwards search begins at the first position of the first visible or excluded record within the display area that is of the default record type.

`CHARS`

For non-numeric search strings only, CHARS indicates that a successful match occurs if the search string is found anywhere within the data being searched.

`PREFIX`

For non-numeric search strings only, PREFIX indicates that a successful match only occurs if the search string is found at the start of a word within the data being searched. i.e. the matched text must precede an alphanumeric character and either be preceded by a non-alphanumeric character or be at the start of a line or field.

`SUFFIX`

For non-numeric search strings only, SUFFIX indicates that a successful match only occurs if the search string is found at the end of a word within the data being searched. i.e. the matched text must be preceded by an alphanumeric character and either precede a non-alphanumeric character or be at the end of a line or field.

`WORD`

For non-numeric search strings only, WORD indicates that a successful match only occurs if the search string is found to be a complete word within the data being searched. i.e. the matched text must either be preceded by a non-alphanumeric character or be at the start of a line or field, and either precede a non-alphanumeric character or be at the end of a line or field.

`pos1`

The first position of a range of positions within the data record to be searched.

For formatted records, this is a position in the expanded record . Only those fields, or parts of fields, that fall within the position range will be searched. Fields will be searched in the order that they occur within the display area.

pos1 may be a positive or negative integer value (not zero) and must be a value that is less than or equal to the maximum length of the data records or, for formatted record data, the length of the expanded record.

A negative value represents a position in the record relative to the end of the record. Therefore, where position 1 references the 1st character in the record, position -1 references the last character.

For all display formats, the string is treated as being non-numeric and the search is actioned on the character representation of the record data.

`pos2`

The last position of a range of positions within the data record to be searched.

Like *pos1*, *pos2* may be a positive or negative integer value (not zero). If *pos1* references a position within the record data which is higher than that referenced by *pos2*, then the *pos1* and *pos2* values are swapped.

If *pos2* is greater than the maximum length of the data records or, for formatted record data, greater than the length of the expanded record, then *pos2* is set equal to the maximum (or expanded) record length.

Default is *pos1* plus the length of the search string minus 1.

`#ALL`

Search all eligible field columns in the current formatted display.

`field_col`

An individual field column to be searched within a formatted display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID). If a field name is used, enclosing parentheses are **mandatory** Field names are automatically converted to a field reference and Referencing the same field column more than once will not cause an error.

If the field is an array, then all elements of the array are searched. To search an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array. An entry must exist for each dimension of the array. e.g. RoomSize(6,2,4) - a three dimensional array.

Specification of multiple field columns must either be enclosed in parentheses and/or separated by commas. The field columns may be specified in any order, however, the data is always searched from the first column in the display to the last.

Field column search specifications may be a combination of individual field columns and columns ranges. e.g. (JobID #6 Tax_Reference #12 #15:#20).

A search is only performed on field columns that are selected for display. Therefore, any field column specified on the EXCLUDE command that has not been selected for display, will be ignored.

`field_col1:field_col2`

The first and last fields of a range of field columns to be searched within a formatted record display display.

field_col1 and *field_col2* must be separated by ":" (colon) and if *field_col1* occurs after *field_col2* in the data record, then the values are swapped. *field_col1* and *field_col2* have the same specifications as *field_col*.

Specification of multiple field column ranges must either be enclosed in parentheses and/or separated by commas. Field column search specifications may be a combination of individual field columns and field columns ranges. e.g. (FirstName:LastName, #2, Salary:Bonus, EmpNo, #10:#15). If field names are used, enclosing parentheses are **mandatory**.

`.name1`

A label name identifying the first record of a range of data records to be searched. The preceding "." (dot) is mandatory. Default is .ZFIRST.

`.name2`

A label name identifying the last record of a range of data records to be searched. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then the order is reversed.
If EXCLUDE PREV is executed and *.name1* is specified, the default is .ZFIRST. Otherwise the default is .ZLAST.

Examples:

```
exclude last 100
```

Exclude the last record in the display (of the default record type) that contains a numeric field with value 100 or a character field containing the string "100".

```
ex c'Spec' prefix  (#10:#20, TitleTrack)
```

Exclude the first record in the display (of the default record type) that contains the exact string "Spec" as a word prefix within any of the selected character fields.

```
x all
```

Exclude all records of the default record type.

See Also:

CHANGE FIND ONLY RFIND SELECT

EXTRACT

Description:

See SET/QUERY/EXTRACT Options.

FILE

Syntax:

```
>>-- FILE --------+-------------+-------------------------------------------><
                  |             |
                  +-- NEWGEN ---+
                  |             |
                  +-- NOGEN ----+
```

Description:

Close **all** SDE window views for data displayed in the current SDE view. Any unsaved changes are saved.

If a non-temporary structure (SDO) is used to map the record data and changes have been made to that structure during the course of the edit session (e.g. USE WHEN), then the user will be prompted to save the changed structure to its structured data file (SDF). See command, SAVESTRUCTURE.

Parameters:

`NEWGEN`

Applicable only if output is to a version 2 PDSE library member that supports member generations. Data will be saved to a new primary member generation.
This option overrides the default set by the GENSAVE option.

`NOGEN`

Applicable only if output is to a version 2 PDSE library member that supports member generations. Data will be saved to back to the same member generation referenced in the focus edit view.
This option overrides the default set by the GENSAVE option.

See Also:

CANCEL END QQUIT and SET/QUERY/EXTRACT Option: GENSAVE

FILEIO

Syntax:

```
>>--+- FILEIO -+--- filename ---+----| Open/Close_Operations | -----+-------><
    |          |                |                                   |
    +- FIO ----+                +----| Input_Operations | ----------+
                                |                                   |
                                +----| Output_Operations | ---------+
                                |                                   |
                                +----| Selection_Operations | ------+
                                |                                   |
                                +----| Test_Data_Operations | ------+
                                |                                   |
                                +----| Miscellaneous_Operations | --+
                                |                                   |
                                +----| Options | -------------------+
```

Description:

For use in FileKit REXX macros, FILEIO may be used to perform file input/output on any:

- ♦ Physical sequential data set,
- ♦ PDS/PDSE library member or member generation
- ♦ VSAM data set,
- ♦ Unix (HFS/ZFS) file
- ♦ DB2 table or SQL result table
- ♦ Multiple members of a library or concatenation of libraries **(BPAM)**

Where **BPAM** access to multiple members across a concatenation of multiple libraries is requested, then the user may choose to either:

- ♦ Process all matching member names from each library in sequence, *or*
- ♦ Treat the concatenation as a **library search path**, processing only the first occurrence of each member name.

FILEIO maybe thought of as similar to **EXECIO**, but unlike EXECIO (which is a TSO/E REXX extension), FILEIO may be executed in REXX macros run by FILEKITB (the FileKit batch interface) or via a FileKit VTAM login.

FILEIO also has features that make it a great deal more powerful than **EXECIO**.

Just like EXECIO, FILEIO operations include the following:

- ♦ Read a single unformatted record into a named REXX variable.
- ♦ Write a single unformatted record from a REXX variable or literal.
- ♦ Read multiple unformatted records into a named REXX stem variable.
- ♦ Write multiple unformatted records from a named REXX stem variable.

However, FILEIO operations also include the following:

- ♦ Read a single record formatted by a structure (e.g. COBOL/PL1) and transfer (selected) field values into separate REXX variables.
- ♦ Write a single record formatted by a structure, populating field values from separate REXX variables.
- ♦ Perform direct reads on any VSAM (KSDS/ESDS/RRDS/VRDS) file.
- ♦ Perform direct reads on any unix (HFS/ZFS) file.
- ♦ Update, insert and delete records.
- ♦ Filter records by content using powerful selection criteria including reference by COBOL/PL1 field name.
- ♦ Extract information from SMF records and navigate their extremely complex structures.
- ♦ Generate Test Data including sequenced or random numbers, dates/times etc ... even the name of a person.
- ♦ Disguise existing data by performing "keyed" look-up/substitution

Unlike EXECIO, the FILEIO feature allows the user to refer to files directly by dataset name, without need for a prior allocation of the dataset to a **"DD" name**. However, for convenience and brevity, an (up to 8-character) user name may be assigned on the OPEN operation, meaning that from then on any FILEIO operations for that file may refer to it by its short user name. (See **"NAME"** option below).

Parameters:

filename

An identifier for the file(s) or DB2 table on which FILEIO should operate.

FILEIO accepts any of the following:

A **DD name** previously defined by batch **JCL** or (TSO) **ALLOC** command. e.g.

```
address "CBLSDATA" /* REXX */
"alloc f(INDD) shr reuse da('SYS1.MACLIB(ACB)')"
"fileio  INDD             read       REC1"
say     "SYS1.MACLIB(ACB) Record 1:" REC1
"fileio  INDD             close"
```

A **fully qualified dataset name** with optional bracketed member name. e.g.

```
address "CBLSDATA" /* REXX */
"fileio  SYS1.MACLIB(ACB) read       REC1"
say     "SYS1.MACLIB(ACB) Record 1:" REC1
"fileio  SYS1.MACLIB(ACB) close"
```

A **user name** defined on a previous **FILEIO OPEN** operation using the **NAME** option. e.g.

```
address "CBLSDATA" /* REXX */
arg MyFileId
"fileio" MyFileId "open    name MyFILE"

"fileio  MyFILE    read       REC1"
say      MyFileId "Record 1:" REC1
"fileio  MyFILE    close"
```

A **DB2** table or view name. e.g.

```
address "CBLSDATA" /* REXX */
"fileio DSN8910.EMP  open   db2(DBCG)   name SAMPLE",
      "select(LASTNAME,FIRSTNME,PHONENO)",
      "sort  (LASTNAME,FIRSTNME)"

do forever
  'fileio SAMPLE FVALUE READ'; if rc <> 0 then leave
  say SAMPLE.LASTNAME SAMPLE.FIRSTNME SAMPLE.PHONENO
end
"fileio SAMPLE close"
```

A z/OS **Unix HFS**/**ZFS** file ID. e.g.

```
address "CBLSDATA" /* REXX */
"fileio  /Z23A/samples/hobbies read       REC1"
say     "/Z23A/samples/hobbies Record 1:" REC1
"fileio  /Z23A/samples/hobbies close"
```

A **PDS**/**PDSE library** name with optional bracketed **generic member name mask(s)**. e.g.

```
address "CBLSDATA" /* REXX */
"fileio  SYS1.MACLIB(BPX*)  open  library  name MyMAC"
"fileio MyMAC options MEMBER"

do forever
  "fileio MyMAC  read  MYREC"

  say MyMAC.1.MEMBER "Record 1:" MYREC

  "fileio MyMAC  nextmember"
  if rc <> 0 then leave
end

"fileio  MyMAC close"
```

A **DD Name** for a **concatenation of libraries** defining a **member search path** with optional bracketed **generic member name mask(s)**. e.g.

```
address "CBLSDATA" /* REXX */

  /* Library search path */
"alloc f(SOURCE) reuse shr da( 'MY.COBOL.SOURCE.REL130' ",
                             " 'MY.COBOL.SOURCE.REL120' ",
                             " 'MY.COBOL.SOURCE.REL110' ",
                             " 'MY.COBOL.SOURCE.REL100' )",

"fileio SOURCE(INP*,OUT*)  open  ConcatenatedLibraryDirectory"
"fileio SOURCE  options LIBNAME MEMBER"

do forever
  "fileio SOURCE  read  MYREC"

  say SOURCE.1.LIBNAME"("SOURCE.1.MEMBER")" "Record 1:" MYREC

  "fileio SOURCE  nextmember"
  if rc <> 0 then leave
end

"fileio  SOURCE close"
```

Open/Close_Operations:

```
|---+--+- OPENread ----+--| Open_Options |-+------------------------------+-+-|
    |  |               |                    |                              | |
    |  +- OPENWrite ---+                    +- USING | Mapping_Structure |-+ |
    |  |               |                                                    |
    |  +- OPENUpdate --+                                                    |
    |  |               |                                                    |
    |  +- OPENInsert --+                                                    |
    |                                                                       |
    +---- CLOSE ------------------------------------------------------------+
```

Open_Options:

```
|---+------------------+--+--------------------------------+--+-------+----->
    |                  |  |                                |  |       |
    +- NAME user_name -+  +- CONCATENATEDLIBRARYDIRectory -+  +- IOP -+
                          +- CLD --------------------------+
                          |                                |
                          +- LIBRARY ----------------------+
                          +- PDS --------------------------+
                          |                                |
                          +- | DB2_options |---------------+
                          |                                |
                          +- | HFS_Options |---------------+

  >---+----------------+--+-------------+--+----------+-----------------------|
      |                |  |             |  |          |
      +- ILIMit nrecs -+  +- LRECL nnn -+  +- APPend -+
```

Mapping_Structure:

```
      +- SDO ---+
      |         |
|--+--+---------+--- copybook_or_mapping_file ---+--------------------------|
   |  |         |                                |
   |  +- COBol -+                                |
   |  |         |                                |
   |  +- PL1 ---+                                |
   |  |         |                                |
   |  +- ADAta -+                                |
   |                                             |
   +---- SYMNAMES ( DFSORT_symbols ) ------------+
```

`OPENREAD | OPENWRITE | OPENUPDATE | OPENINSERT`

The type of open to be performed for subsequent I/O operations.

OPENREAD (OPENR)	The file is opened for input only
OPENWRITE (OPENW)	The file is opened for output only
OPENUPDATE (OPENU)	The file is opened for input, but any record may be replaced with an updated version. In addition, for **VSAM KSDS** and **RRDS** files, records may deleted.
OPENINSERT (OPENI)	The file is opened for insert. This is necessary when inserting records **out of key sequence** to a **VSAM KSDS** using the WRITE, WRITEV, WRITEX, SWRITE or FWRITE operations. When purely using the INSERT, INSERTV, INSERTX or FINSERT operations then **open for insert is implied** and need not be explicitly coded.

`CLOSE`

Closes the named file.

In the FileKit **online** environment, beware that if your REXX macro exits early, before issuing an explicit CLOSE operation for any opened files, then they will remain open and possibly locked to other users.

To close all files opened in your FileKit session, you may issue either: `FILEIO * CLOSE` or `FILEIO CLOSEALL`.

In the **batch** environment, all files opened by FILEIO are **automatically closed** at the end of each FILEKITB job step.

`USING SDO|COBOL|PL1|ADATA` *`copybook_or_mapping_file`*

Indicates that FileKit should apply a mapping to records read or written to this file. The advantages of this are:

1. Individual field values may be extracted from the record into separate REXX variables
   See FVALUE operation.

2. The user does not need to be concerned about knowing the position, length and data-type of individual fields, only their names.

3. By using FILEIO's built in record selection capabilties, processing by REXX itself may be limited to records containing any required combination of field values. This dramatically improves run times over using REXX itself to filter records based on content.
   (See VIEW, WHERE, FILTER, TOFCONDITION and EOFCONDITION operations.)

The following keywords indicate the type of mapping dataset specified:

`COBOL`
`(COB)`
	copybook_or_mapping_file is a COBOL copybook.

`PL1`	*copybook_or_mapping_file* is a PL1 copybook.

`ADATA`
`(ADA)`
	copybook_or_mapping_file is an Assembler SYSADATA file.

`SDO`	*copybook_or_mapping_file* is a FileKit Structured Data Object.

`USING SYMNAMES (` *`DFSORT symbols`* `)`

Specifies DFSORT SYMNAME symbol definitions that are to be used to format the data records. The order in which symbol definitions are supplied dictate the order in which the fields will occur in the record type definition.

The symbol name definitions within the SYMNAMES parentheses may be supplied directly in-line and/or via input data sets/library members.

```
  SYMNAMES   ( Card,06,04,CH  Dept,46,03,CH  Amount,49,06,PD  )
  SYMNAMES   ( SYS1.MACLIB(EDGSMFSY)  SYS1.MACLIB(EDGSRCSY) )
  SYMNAMES   ( CBL.DFSORT.SYM(CBLATRAC)  TCB,*,4,BI )
```

`APPEND`

Specifies that, where the underlying file organisation supports it, any records written to this file will be added to the end, preserving all existing records.

APPEND is supported for the following:

◊ z/OS Standard Sequential files.
◊ z/OS Unix HFS/ZFS files.
◊ VSAM ESDS file defined with the REUSE option.

`CONCATENATEDLIBRARYDIRectory | CLD`

Indicates that the list of PDS/PDSE libraries specified by *filename* is to be treated as a **library search path** for all members that match the optionally supplied mask(s).

Members will be processed in sequence using the **fast BPAM access method.**

For example, three libraries are supplied as a search path with a member mask of "*" (meaning all member names are to be processed).

1. MY.LIB**1** contains members **C, F** and **Q**
2. MY.LIB**2** contains members **F, H, Q** and **Z**
3. MY.LIB**3** contains members **B, C** and **D**

The command "`FILEIO MY.LIB1,MY.LIB2,MY.LIB3(*) openread CLD`" will cause subsequent input operations to process members in the following order:

1. Member **B** from MY.LIB**3**
2. Member **C** from MY.LIB**1**
3. Member **D** from MY.LIB**3**
4. Member **F** from MY.LIB**1**
5. Member **H** from MY.LIB**2**
6. Member **Q** from MY.LIB**1**
7. Member **Z** from MY.LIB**2**

Return Codes set by FILEIO input operations (e.g. READ, SREAD or FVALUE READ):

- **rc=***2* indicates end of member
- **rc=***4* indicates end of last member

It is also worth noting that the *SREAD* operation (which inputs multiple records into a REXX stem variable) will terminate with **return code = 2** at the end of each member. Meaning that the specified stem will only ever contain records from one member at a time. (See *LIBRARY* parameter for example.)

The sequence of libraries will typically be supplied as a **DD name** allocated to a **concatenation of libraries** e.g.

```
address "CBLSDATA" /* REXX */

  /* Library search path */
"alloc f(SOURCE) reuse shr da( 'MY.COBOL.SOURCE.REL130' ",
                             " 'MY.COBOL.SOURCE.REL120' ",
                             " 'MY.COBOL.SOURCE.REL110' ",
                             " 'MY.COBOL.SOURCE.REL100' )",

"fileio SOURCE(INP*,OUT*)  open  CLD  name MyCOBOL"
"fileio MyCOBOL options LIBNAME MEMBER"

do forever
  "fileio MyCOBOL  read  MYREC"

  say MyCOBOL.1.LIBNAME"("MyCOBOL.1.MEMBER")" "Record 1:" MYREC

  "fileio MyCOBOL  nextmember"
  if rc <> 0 then leave
end

"fileio  MyCOBOL close"
```

However, the sequence of libraries may also be supplied as a comma separated (no blanks) list of dataset names e.g.

```
address "CBLSDATA" /* REXX */


"fileio MYSRC.R150,MYSRC.R130,MYSRC.R120,MYSRC.R110(WIN*)",
    "open  CLD   name MySRC"


"fileio MySRC options LIBNAME MEMBER"

do forever
  "fileio MySRC  read  MYREC"

  say MySRC.1.LIBNAME"("MySRC.1.MEMBER")" "Record 1:" MYREC

  "fileio MySRC  nextmember"
  if rc <> 0 then leave
end

"fileio  MySRC close"
```

LIBRARY | PDS

Multiple members of a PDS/PDSE library may be processed in sequence using the **fast BPAM access method.**

An optional bracketed library **member name mask** may follow directly after the fully qualified library dataset name specified as *filename*. e.g.

```
SYS1.MACLIB(ATB*)
```

Multiple member masks may be specified each separated by a **comma** (no blanks).

```
SYS1.MACLIB(ATB*,BPX*,GIM*)
```

If *filename* refers to a **DD name** (previously defined by batch **JCL** or TSO **ALLOC** command) which is allocated to a **concatenation of libraries**, then they will be treated as a **library search path**, just as though CONCATENATEDLIBRARYDIRectory (CLD) had been coded instead.

If the library is a **PDSE Version 2** defined with **member generations** then older generations maybe processed too.

To reference an individual member generation, FileKit supports specification of an absolute or relative generation number following the member name with a single separating "." (dot/period) between. A member generation may be identified by its relative or absolute generation number. e.g.

```
MY.GEN25.XEC(APESUB.-5)
MY.SELCI.SDO(DB2FUNC1.-1)
```

A member generation mask is the same as a single member generation but with either:

1. A **null** or **wildcard character "*"** specified in place of the generation number. This indicates that all generations are to be selected.

2. A relational operator inserted between the "." (dot/period) and the (absolute or relative) generation number. This indicates that only generation numbers that satisfy the numeric comparison will be selected.

   Supported relational operators are:

Operator	Description
=	Equal.
\= ¬= != <>	Not equal, less than or greater than.
>	Greater than.
>=	Greater than or equal.
<	Less than.
<=	Less than or equal.

Member generations that satisfy both the member mask and generation mask will be selected. e.g. The following are equivalent and will select all generations of all members:

```
MY.JCLLIB(*.*)
MY.JCLLIB(*.)
MY.JCLLIB(*.<*)
```

To select all member generations whose member name begins with *CALL* and whose absolute generation number is greater than or equal to *12*:

```
MY.JCLLIB(CALL*.>=12)
```

To select all generations except the base (i.e. relative generation 0):

```
MY.JCLLIB(NBJX.<0)
```

To select all member generations whose member name begins with *SS* and whose relative generation number is *-1*:

```
MY.JCLLIB(SS*.-1)
```

To select all member generations whose entries match one of the member and generation masks:

```
MY.JCLLIB(SS*.>=-3,ADA%%%%.>=-5,CBL*.0)
```

Return Codes set by FILEIO input operations e.g READ/SREAD/FVALUE READ:

- **rc=***2* indicates end of member.
- **rc=***4* indicates end of last member.

It's also worth pointing out that the *SREAD* operation (which inputs multiple records into a REXX stem variable) will terminate with **return code = 2** at the end of each member. Meaning that the specified stem will only ever contain records from one member at a time.

For example, suppose a library **MY.TEST.LIB** has a number of members, the first of which contains **150 records**, and the following is executed:

```
address "CBLSDATA" /* REXX */

"fileio MY.TEST.LIB(*)  open  library  name MyLIB"
"fileio MyLIB  sread MYSTEM  100"
"fileio MyLIB  sread MYSTEM  100"
```

On execution of the **first** SREAD operation, FILEIO will set the following:

- **rc**=*0*
- **MYSTEM.*0***=*100*
- **MYSTEM.*1*** to **MYSTEM.*100*** populated by records *1-100*

On execution of the **second** SREAD operation, FILEIO will set the following:

- **rc**=*2*
- **MYSTEM.*0***=*50*
- **MYSTEM.*1*** to **MYSTEM.*50*** populated by records *101-150*
- **MYSTEM.*51*** to **MYSTEM.*100*** will be unchanged

When processing multiple members in seqeunce it's obviously useful to know the **current member name**.

This is acheived by activating the **MEMBER option**, which instructs FILEIO to set REXX variable ***<filename>*.1.MEMBER** to the member name each time it changes. e.g.

```
address "CBLSDATA" /* REXX */

arg LibraryDSN   /* e.g. SYS1.MACLIB(A*,B*,C*) */

   /* Display name and 1st 4 records for each member */
"fileio" LibraryDSN "open  library  name MyLIB"
"fileio  MyLIB       option MEMBER"
do forever
  "fileio MyLIB  sread MYSTEM  4"

  say "MEMBER="MyLIB.1.MEMBER /* Display member name */

  do i = 1 to MYSTEM.0
    say MYSTEM.i
  end

  say "                 ------------------          "
  say " "

  "fileio MyLIB  nextmember"
  if rc <> 0 then leave

end

"fileio  MyLIB close"
```

`IOP`

If FILEIO is called from a REXX macro running within an online FileKit session (TSO/ISPF foreground) then option IOP may be used to display an **"I/O Progress" window** for *filename*.

As well as communicating progress, in terms of records read or written, its primary purpose is to give the user an oppunity to **interrupt the process** by pressing the **"Attention" key** (once the keyboard has been unlocked).

The **IOP** and **NOIOP** options allow the user to switch this feature on and off as desired. For example, while testing it's good idea to switch on the IOP, but if your process uses REXX's **"say"** feature to display information on your TSO terminal, then having FILEIO IOP causing a screen refresh every second or so would be very annoying.

For input files, the IOP window is displayed at the top right of the screen.

For output files, the IOP window is displayed halfway down on right of the screen.

Once the **"Attention" key** interrupt has been registered all input and output operations for *filename* will set **return code 4** (RC=4).

IOP/NOIOP may also be specified after OPEN, using the OPTIONS operation.

`ILIMit` *`nrecs`*

Input limit specifies the maximum number of records to be read from *filename*.

This means a file containing *nrecs***+1** records is treated as if it contains only *nrecs* records.

FETCHLIMit is a synonym for ILIMIT.

For BPAM multi-member library input (using either LIBRARY or CLD options), **ILIMIT** is applied separately for each member.

Example:

```
address "CBLSDATA" /* REXX */

arg LibraryDSN   /* e.g. SYS1.MACLIB(A*,B*,C*) */

   /* Display name and 1st 10 records for each member */
"fileio" LibraryDSN "open  library  name MyLIB   iLimit 10"
"fileio  MyLIB       option MEMBER"
do forever
  "fileio MyLIB  sread MYSTEM"

  say "MEMBER="MyLIB.1.MEMBER /* Display member name */

  do i = 1 to MYSTEM.0
    say MYSTEM.i
  end

  say "                 ------------------          "
  say " "

  "fileio MyLIB  nextmember"
  if rc <> 0 then leave

end

"fileio  MyLIB close"
```

`LRECL` *`nnn`*

LRECL is also described as part of **HFS_Options.**

Apart from for HFS files the **LRECL** is seldom required. However, it is also useful when processing very long **spanned records**.

When FILEIO detects a spanned file defined with **LRECL=X** it is unable to immediately determine the maximum size of its records, merely that they **may exceed 32K**.

In this case FILEIO's default action is to start by allocating work buffers with a size of **64K**. This buffer size may be overidden by coding **LRECL=*nnn*** on the OPEN.

For both **input** and **ouput** files, even if their records do exceed the default 64K, coding LRECL is not essential since FILEIO will automatically reallocate its buffers as necessary.

So (for non-HFS files) coding LRECL is useful only to **improve efficiency**. e.g.

◊ If your longest record is much smaller than 64K, then you may wish to code LRECL to reduce the amount of storage allocated to buffers.
◊ If your longest record is much larger than 64K, then you may wish to code LRECL to reduce the amount of cpu used to reallocate and copy larger and larger buffers each time.

`NAME` *`user_name`*

Use the NAME option to assign a short (up to 8-characters) user name that may be specified in place of *filename* on any subsequent FILEIO operations.

This is especially useful if the *filename* is passed to your procedure as a parameter, or is otherwise built in a REXX variable.

For example the following two sequences perform identically, but the second is slightly neater and more concise.

```
address "CBLSDATA" /* REXX */

arg InputFile OutputFile Mapping FieldName SelectValue

"fileio" InputFile      "openread   using COBOL" Mapping
"fileio" OutputFile     "openwrite"

"fileio" InputFile      "where ("FieldName"="SelectValue")"

do forever
  "fileio" InputFile    "read   MYREC"
  if rc = 4 then leave

  "fileio" OutputFile   "writev MYREC"
end

"fileio" InputFile      "close"
"fileio" OutputFile     "close"
```

```
address "CBLSDATA" /* REXX */

arg InputFile OutputFile Mapping FieldName SelectValue

"fileio" InputFile      "openread  name III   using COBOL" Mapping
"fileio" OutputFile     "openwrite name OOO"

"fileio  III where ("FieldName"="SelectValue")"

do forever
  "fileio III read   MYREC"; if rc = 4 then leave
  "fileio OOO writev MYREC"
end

"fileio III close"
"fileio OOO close"
```

DB2_Options:

```
|-+-------+--+--------------------------+--+--------------+--+----------+--->
  |       |  |                          |  |              |  |          |
  +- DB2 -+  +- LOCKTable +- EXclusive -+  +- SKIPLocked -+  +- SCROLL -+
                          +- SHare -----+
                          +- SHR -------+

>-+---------------------------------------------+--+----------------+------->
  |                                             |  |                |
  +-- WITH --+- CS --+-+------------------------+  +- EDITPRIMEKEY -+
             +- RR --+ |                        |
             +- RS --+ +- KEEP --+- EXclusive --+
             +- UR --+           +- UPDate -----+
                                 +- SHare ------+
                                 +- SHR --------+

>-+------------------------+-+-----------------------------------------+-+-->
  |                        | |                                         | |
  +- WHERE (where_clause) -+ +- ORDER -+------+--+- (order_by_clause) -+ |
  |                          |         |      |  |                     | |
  |                          |         +- BY -+  |                     | |
  |                          +- SORT ------------+                     | |
  |                          |                                         | |
  |                          +- SORTIndex -+--+-- index_name --+-------+ |
  |                          +- SORTIx ----+  |                |         |
  |                                           +-- Prime -------+         |
  |                                                                      |
  +--- SQL -+- ( sql_syntax ) --+----------------------------------------+
            |                   |
            +--- sql_file ------+

>-+----------------------------------+--------------------------------------|
  |                                  |
  |            +----- , ------+      |
  |            v              |      |
  +- SELect ( -+- field_name -+- ) --+
```

DB2

The **DB2** keyword identifies *filename* as a DB2 table or view name. Strictly speaking it is unnecessary if any of the other above options exclusive to DB2 are also supplied.

EDITPRIMEKEY

Specify **EDITPRIMEKEY** if you deliberately intend to update the DB2 table's primary key column(s), otherwise any attempts to do so will cause an error.

LOCKTABLE EXCLUSIVE | SHR

For DB2 base tables only, executes a DB2 SQL command LOCK TABLE before loading data from the table. Use this option with **caution** as other users and applications may be prevented from accessing the table. If LOCKTABLE is specified the lock is held until the next COMMIT is actioned.

LOCKTABLE EXCLUSIVE prevents another process from performing any operations on the table whilst it is open, unless the process is running with an isolation level of Uncommitted Read (UR) in which case read-only (dirty read) operations may be performed.

LOCKTABLE SHR prevents anything other than read-only operations to be performed on the table whilst it is open.

See "*DB2 SQL Reference*" for details on the effects of LOCK TABLE options.
Default is not to perform any explict table locking prior to load. (Recommended)

SCROLL

Use a DB2 scrollable SENSITIVE STATIC cursor to fetch DB2 rows. See Using Scrollable Cursors for information on how the use of this keyword affects processing.

The use of DB2 scrollable cursors may be disabled by the installer of FileKit. In order to use them the system INI file must contain the *DB2.SCROLL=YES* variable.

SELECT (*select_clause*)

Specifies a DB2 SQL SELECT clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *select_clause*.

SORT (*order-by-clause*)

Specifies a DB2 SQL ORDER BY clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *order-by-clause*.

SKIPLOCKED

Ignored unless an isolation level of Cursor Stability (CS) or Read Stability (RS) is in effect, SKIPLOCKED specifies that any selected rows that are already locked by another process should be skipped. See "*DB2 SQL Reference*" for details on the SKIP LOCKED DATA clause.
Default is to include locked rows whenever possible.

`SORTIX`

Specifies *index_name*, the name of an existing DB2 Index for the table being opened. The index identifies the key columns/expressions by which the table rows will be ordered.

`SQL (`*`sql_syntax`*`)`

Specifies that the result table generated by the SQL query *sql_syntax* is to input. The *sql_syntax* must be specified in **brackets**. If SQL is specified then, although *filename* must still be provided, it will be ignored. Additionally an explicit SELECT, WHERE and SORT options must not be specified.

`SQL `*`sql_file`*

Specifies that the result table generated by the SQL query located in file *sql_file* is to be input. If SQL is specified then although *filename* must still be provided, it will be ignored. Additionally an explicit SELECT, WHERE and SORT options must not be specified.

`WITH CS|UR|RR|RS  < KEEP  EXCLUSIVE|UPDATE|SHR >`

Specifies a DB2 SQL isolation-clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for details on the *isolation-clause* and "*DB2 Performance Monitoring and Tuning Guide*" for details on the effects of isolation level on concurrency and protection of DB2 table data.

The KEEP sub-parameter options are applicable to isolation levels RR (Repeatable Read) and RS (Read Stability) only. Default isolation level (as set by the FileKit DB2 package and plan BIND) is CS.

`WHERE (`*`where_clause`*`)`

Specifies a DB2 SQL WHERE clause to be included in the prepared SQL select statement. See "*DB2 SQL Reference*" for syntax of the *where_clause*.

HFS_Options:

```
              +-- STD -----+
              |            |
   +-- EOL ---+------------+---------------+
   |          |            |               |
   |          +-- CR ------+               |
   |          +-- LF ------+               |
   |          +-- NL ------+               |
   |          +-- CRLF ----+               |
   |          +-- LFCR ----+               |
   |          +-- CRNL ----+               |
   |          +-- string --+               |  +- LRECL lrecl -+
   |                                       |  |               |
|--+---------------------------------------+--+---------------+-------------|
   |                                       |
   +-- NOEOL ------------------------------+
   |                                       |
   +-- RECFM -+- F --+---------------------+
              +- V --+
              +- V1 -+
              +- V2 -+
              +- V3 -+
```

`EOL=STD|NL|CR|LF|CRLF|LFCR|CRNL|`*`string`*

Sets the EOL (input end-of-line) delimiter value used to determine the end of each record for non-RECFM F/V input. EOL delimiters are not included in the returned record data or record length. EOL parameter elements are as follow:

`STD`	-	Any standard line-end.
`NL`	X'15'	New Line.
`CR`	X'0D'	Carriage Return.
`LF`	X'0A'	Line Feed.
`string`	-	A 2-byte user specified character or hex string.

STD is default so that FILEIO operation scans the input data for any of the standard EOL combinations (not *string*), stopping when one is found. This EOL combination is used as EOL for the file.

`NOEOL`

The **"no end-of-line"** option indicates to FILEIO that the length of each record read or written to the HFS file is variable and determined by the preceeding setting of the **current record length** via the **RECLEN=*nnn*** operation.

This option allows the user to read/write chunks of varying size from a z/OS unix file, with the length of each chunk being determined as processing proceeds.

For example, to read a z/OS unix file with the following repeating layout:

1. A fixed 1-byte binary field containing the **length** of a Company Name
2. A character field containing the Company Name (len=*??*)

3. A fixed 2-byte binary field containing the **length** of a number of comma separated email addresses
4. A character field containing the email addresses (len=*??*)

```
address "CBLSDATA" /* REXX */

parse arg HfsFile

"fileio" HfsFile "openread name MyHFS noeol"

do forever
  "fileio MyHFS reclen 1"
  "fileio MyHFS read  L1" ; if rc <> 0 then leave
  L1=c2d(L1)
  "fileio MyHFS reclen" L1
  "fileio MyHFS read  CompanyNam"
  say '"'CompanyNam'"'

  "fileio MyHFS reclen 2"
  "fileio MyHFS read  L2"
  L2=c2d(L2)
  "fileio MyHFS reclen" L2
  "fileio MyHFS read  EmailAddrs"
  say '"'EmailAddrs'"'
  say ' ---------------------------------------'
  say ' '

end
"fileio MyHFS close"
exit
```

`LRECL` *`lrecl`*

Specifies the maximum record length of input records.

Records terminated by an EOL sequence will be broken into multiple records if the record length exceeds *lrecl*.

For RECFM F data, *lrecl* is the fixed length of the records read/written.

If the record length field of a RECFM V record exceeds the *lrecl* value, then an error is returned.

RECFM V and EOL delimited records have default *lrecl* of 32752, wheras RECFM F records have default *lrecl* of 80.

`RECFM F | V | V1 | V2 | V3`

Specifies that the data is to be treated as containing Fixed or Variable length format records.

RECFM F indicates that all records are of a fixed length as defined by the LRECL argument.

RECFM V/**V1**/**V2**/**V3**, where RECFM V is a synonym for RECFM V1, tells FILEIO that the data length is held in a binary record prefix in one of the following formats:

RECFM V1	A 4-byte IBM-style Record Descriptor Word (**RDW**) comprising a 2-byte binary length followed 2 more bytes that are unused by FILEIO. The first two bytes contain the length of both the prefix and the following data i.e. the record data length **+4**.
RECFM V2	A 2-byte **VARCHAR**-style prefix. The first two bytes contain the length of following data only.
RECFM V3	A 4-byte **LONGVARCHAR**-style prefix. The first four bytes contain the length of following data only.

Input_Operations:

```
|--+-+- READ --------+--+------------+--+------------------------+-+---------|
   | |               |  |            |  |                        | |
   | +- READSEGment -+  +- rexxvar --+  +- KEY string -----------+ |
   |                                    |                        | |
   |                                    +- RBA nnnn -------------+ |
   |                                    |                        | |
   |                                    +- RECno nnnn -----------+ |
   |                                    |                        | |
   |                                    +- TTR ttr --------------+ |
   |                                    |                        | |
   |                                    +- RECLen record_length -+ |
   |                                                               |
   +-+- FVALUE ---+---+---------+----------------------------------+
   | |            |   |         |                                  |
   | +- FVALUEQP -+   +- READ --+                                  |
   | |            |                                                |
   | +- FVALUEQF -+                                                |
   |                                                               |
   +--- CREAD -----------------------+---------+-------------------+
   |                                 |         |                   |
   |                                 +- nrecs -+                   |
   |                                                               |
   +-+- SREAD --------+-- stem_name -+---------+-------------------+
   | |                |              |         |                   |
   | +- SREADSEGment -+              +- nrecs -+                   |
   |                                                               |
   +--- RECLen record_length --------------------------------------+
   |                                                               |
   +--- SKIP nrecs ------------------------------------------------+
```

READ

The READ operation inputs a single logical record and optionally places the whole of its contents in the REXX variable *rexxvar*.

The **read direction** is **forward by default**. To read a VSAM file **backwards** use the ModRPL operation to set option BWD.

READSEGment

Identical to READ above except that the next logical segment is returned instead of the next complete physical record.

If no further segments are available from the current physical record, then the next physical record is read, broken up in to logical segments, and the first (prime) segment is returned.

If the USING *copybook_or_mapping_file* option specified on the OPEN operation for *filename* does not refer to a *segmented* file mapping, then **READSEGMENT** will behave identically to the READ operation.

FVALUE | FVALUEQP | FVALUEQF <READ>

For **formatted input** only, these operations are used in conjunction with the USING *copybook_or_mapping_file* option above this operation sets a REXX variable for each field in an input record, in much the same fashion as "EXTRACT /FVALUE/" does while in a browse or edit session.

The generated REXX variable names are the record field column names, and their assigned values are the character representation of the individual field's contents within the record.

One difference from the EXTRACT operation is that each of the REXX variable names are prefixed with *user_name* (or *filename* if NAME is not specified on OPEN) followed by a dot. This allows a calling REXX procedure to maintain independent variables from FILEIO operations performed on multiple input files concurrently.

For **FVALUE** the name of the generated REXX variable is the elementary field name with no leading record-type and/or parent structure (group) name(s) added.
e.g. *user_name.***RELEASE_YYYY**

For **FVALUEQP** the name of the generated REXX variable is the partially qualified field name with only parent structure (group) name(s) added, each separated by a full stop, but no leading record-type.
e.g. *user_name.***RELEASE_DATE.RELEASE_MM**

For **FVALUEQF** the name of the generated REXX variable is the fully qualified field name with both leading record-type and parent structure names added.
e.g. *user_name.***TRACK.RELEASE_DATE.RELEASE_DD**

Field names that are elements of an array, and so have array suffices, generate REXX compound variables where the field name is the variable name stem and each dimension of the array is represented within the variable name tail.   e.g. Field name "AddrLine(6)" generates variable *user_name.***AddrLine.6**.

Any occurrence of the "-" (minus) character in a field column name (as supported by COBOL) is translated to "_" (underscore) in the generated variable name thus avoiding REXX "Bad Arithmetic Conversion" errors.   e.g. Field name "XX-HRMN" generates variable *user_name.***XX_HRMN**.

For example, operating on a record containing elementary fields names: "TRACK-NUM", "ARTIST" and "NAME", the command "`FILEIO MYDD FVALUE`" will set the following REXX variables:

◊ **MYDD**.TRACK_NUM
◊ **MYDD**.ARTIST
◊ **MYDD**.NAME

Similarly, the command "`FILEIO USER123.SELCTRN.ZZST1DAT FVALUE READ`" will read the next record then set the following REXX variables:

◊ **USER123.SELCTRN.ZZST1DAT**.TRACK_NUM
◊ **USER123.SELCTRN.ZZST1DAT**.ARTIST
◊ **USER123.SELCTRN.ZZST1DAT**.NAME

The optional **READ** keyword tells FILEIO whether or not to read the next record (or logical segment) from the input file before performaing the FVALUE operation. If **READ** is omitted then the record previously input using an unformatted "`FILEIO mydd READ`" operation will be operated upon.

In the case of segmented record mappings, the **READ** option moves on to the next secondary segment until all secondaries are exhausted, in which case the next primary is read.

You may use the "`FILEIO` *`filename`* `SKIP 0`" operation (see below) to bypass all remaining secondary segments on the current physical record.

Significant improvement in performance may be achieved by executing an unformatted "`FILEIO` *`filename`* `READ`" operation in order to do some basic record selection, before executing a potentially CPU-intensive "FVALUE" operation, that may set dozens of REXX variables.

Example:

```
address "CBLSDATA" /* REXX */

"alloc f(INDD) reuse shr  da('USER123.SELCTRN.ZZST1DAT')"
"fileio  INDD  openread using cobol USER123.SELCTRN.SAM1(ZZST1CPC)"
"fileio  INDD  select  ARTIST,ALBUM,TRACK-NUM,NAME from TRACK"
do forever
  'fileio INDD read RecData';
  if rc <> 0 then leave        /* End of File? */

  if pos('BRUCE SPRINGSTEEN' ,translate( RecData ) ) > 0 then
    do;
        'fileio INDD fvalue'
        say 'Artist =' INDD.ARTIST
        say 'Album  =' INDD.ALBUM
        say 'TrkNum =' INDD.TRACK_NUM
        say 'Track  =' INDD.NAME
        say ' ---------------------------------- '
    end
end
"fileio  INDD  close"
return 0
```

In addition FILEIO will set the following standard REXX variables:

◊ "*mydd*.**FVALUE.0**" is set to the number of record field name/value variables set.
◊ "*mydd*.**FVALUE.*n***" is set to the REXX variable name which is set for the *n*th field in the record/segment.

Because only variables for "selected" field columns of the current record-type are generated, the SELECT operation will influence the effect of the FVALUE operation.

`CREAD` *`nrecs`*

Input the next *nrecs* number of records and append them direcly onto the FileKit clipboard.

The specified value for *nrecs* defines a **maximum** number of records to be read by this execution of FILEIO. Input will stop when end of file is encountered and so the actual number of records read may be less than the *nrecs* value. If no value for *nrecs* is specified, the default is to read **all** records to end of file.

`SREAD` *`stem nrecs`*

Input the next *nrecs* number of records and assign each record's data to a REXX compound variable with a stem of *stem* and a tail equal to the number of records read so far by this execution of the FILEIO SREAD operation.

Specification of a stem symbol *stem* is mandatory. If *stem* does not end with a "." (dot/period), then one is added automatically. The derived compound variable names allow for easy indexing as data belonging to the first record read by this execution of FILEIO will be assigned to *stem*.1, the second to *stem*.2, etc. Once the operation is complete, compound variable *stem*.0 is assigned a value equal to the number of records read by this execution of FILEIO SREAD. Note that values assigned to other compound variables with the same stem remain unchanged. e.g. If only 5 records are read, *stem*.6 is unchanged.

The following illustrates how each record's data may be processed in a loop.

```
"FILEIO  INDD  SREAD  INREC  20 "
 do  x = 1 to INREC.0
     say  INREC.x
 end
```

The specified value for *nrecs* defines a **maximum** number of records to be read by this execution of FILEIO. Input will stop when end of file is encountered and so the actual number of records read may be less than the *nrecs* value. If no value for *nrecs* is specified, the default is to read **all** records to end of file.

When processing **multiple members of a PDS**/**PDSE** library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, input will also stop (and **RC=2** set) at the **end of each member**. This means that the stem variable will only ever contain records from one member at a time, which is especially convenient when copying members from one library to another using **SREAD** and **SWRITE**.

```
address "CBLSDATA" /* REXX */
  /* Copy members from a "search path" of libraries */

 arg InpLibPath OutLibrary
 /*  e.g. InpLibPath = "MY.INPUT.LIB1,MY.INPUT.LIB2(A*,P*,Y*)"
 |        OutLibrary = "MY.OUTPUT.LIB"
 */

 MemberCount=0
 CurrentMember=''
 "fileio" InpLibPath "openread  cld   name InPath"
 "fileio" OutLibrary "openwrite pds   name OutLib"

 "fileio InPath options member"

 do forever
   "fileio InPath sread MYREC 100"; rrc=rc
       /* RC=2 just means end of member was reached */
   if rrc >= 4 then leave

   /* Handle new member, even an empty one */
   if InPath.1.Member <> CurrentMember then
     do; "fileio OutLib member" InPath.1.Member
         wrc = rc; if wrc <> 0 then leave
         MemberCount=MemberCount+1
         CurrentMember = InPath.1.Member
     end

   "fileio OutLib swrite MYREC"   ; wrc=rc
   if wrc > 0  then exit wrc
 end
 "fileio InPath close"
 "fileio OutLib close"

 'msg' MemberCount 'members copied'

 exit 0
```

`SREADSEGMENT` *`stem nrecs`*

Identical to SREAD above except that **logical segments** are returned instead of complete physical records.

If the USING *copybook_or_mapping_file* option specified on the OPEN operation for *filename* does not refer to a *segmented* file mapping, then **SREADSEGMENT** will behave identically to the SREAD operation.

`SKIP` *`nrecs`*

The SKIP operation may be used to bypass unwanted input records. This means the records are "read" but no REXX variables are set, making it significantly quicker than bypassing the records using the READ or SREAD operations.

The number of records to skip, *nrecs*, refers to physical records (not record segments).

In the case where your input is *segmented* the SKIP operation always starts by bypassing any "unread" secondary segments belonging to the current physical record.

This means that executing **"SKIP 0"** before the next READSEG operation, guarantees it will start by reading the next physical record.

`KEY` *`string`*

For **VSAM KSDS** data sets only, performs a **direct read** on the record with a key value equal to *string*. The key *string* value may be supplied in any of the following formats:

◊ Unquoted string with no embedded blanks. Alpha characters are translated to **upper case**.

◊ Quoted string using apotrophes or double quotes. Alpha characters are translated to **upper case**.

◊ Character string of the form *C'...'* containing any characters. Alpha character case is preserved.

◊ Hexadecimal string of the form *X'...'* containing hex digits 0-9, a-f and A-F. Double quotes may be used instead of single quotes if necessary.

If a record with the specified key is not found, the **default action** is that the record with the **next highest key** is returned. This is option KeyGE.

To alter the default use the ModRPL operation to set option KeyEQ.

Also, see the KEY parameter of the OPTIONS operation, which will extract the key of the most recent input record into a REXX variable.

`RBA` *`nnnn`*

Applicable to **VSAM ESDS**, **VSAM KSDS** and **z/OS Unix (HFS/ZFS)** files only, this option performs a **direct read** on the record with a **relative byte address** equal to *nnnn*.

For VSAM files, the RBA must point at the start of the record otherwise a VSAM point error will occur.

The RBA number *nnnn* may be supplied in either of the following formats:

◊ Unquoted decimal numeric value. e.g. *1538*

◊ Hexadecimal value of the form *X'...'* containing hex digits 0-9, a-f and A-F. e.g. *X'3DE827'* Double quotes may be used instead of single quotes if necessary.

For very large VSAM files, the ModRPL operation may be used to set option XRBA (eXtended RBA).

See RECID parameter of the OPTIONS operation which will extract the RBA of the most recent input record into a REXX variable.

`RECno` *`nnnn`*

For **VSAM RRDS** and Variable RRDS data sets only, this option performs a **direct read** by **record number**.

The record number *nnnn* may be supplied in either of the following formats:

◊ Unquoted decimal numeric value. e.g. *1538*
◊ Hexadecimal value of the form *X'...'* containing hex digits 0-9, a-f and A-F. e.g. *X'3DE827'* Double quotes may be used instead of single quotes if necessary.

Also, see the RECNO parameter of the OPTIONS operation which will extract the most recent input record number into a REXX variable.

`TTR` *`ttr`*

For z/OS **standard sequential files** and **PDS/PDSE Library Members** only, this option performs a **direct read** by **volume, track, record (block)** and **block offset** combination.

The *ttr* combination must be supplied as an 8-byte character string in the format: **X'*vvttttttttrroooo*'**

◊ *v* = **volume** (00=1st)
◊ *t* = **track** (00=1st)
◊ *r* = physical **record** i.e. block (01=1st)
◊ *o* = block **offset** of logical record

See RECID parameter of the OPTIONS operation which will extract the TTR (in the above format) for the most recently read record into a REXX variable.

The following example processes a file containing two different record types:

1. A **"Header"** record (contains '**H**' in **position 1**)
2. A **"Detail"** record (contains '**D**' in **position 1**)

Each **"Header"** record in the file is followed by zero or more (up to 9999) **"Detail"** records.

A **count** needs to be made of the number of **"Detail"** records that follow each **"Header"**. The count should then be maintained as a character field at **position 18** of the header.

On encountering a **"Detail"** record, the process will simply maintain a counter.
On encountering a **"Header"** record, the process will perform the following:

1. Make a note of its **TTR** (RECID)
2. Do a **direct read by TTR** to **jump back** to the previous **"Header"**
3. Update the previous **"Header"** with the count figure
4. Do a **direct read by TTR** to **jump forward** again to the current **"Header"**
5. Reset the counter and continue sequentially

```
address "CBLSDATA" /* REXX */

'fileio SAMP.TTR.TEST OpenUpdate  name MYFILE'
'fileio MYFILE        option RECID'

D_Count= 0; Eof  = 0; Prev_H = ''
do forever
  'fileio MYFILE      read MYREC'
  if rc <> 0 then Eof=1

  C1 = left(MYREC,1)   /* 'H' or 'D' */

  if ( C1 = 'H' | Eof ) & Prev_H <> '' then
    do; Curr_H = MYFILE.1.RECID
           /* Jump back to previous Header rec */
        'fileio MYFILE      read MYREC  ttr' x2c(Prev_H)

           /* Update previous header with count */
        MYREC = overlay(right(D_Count,4),MYREC,18)
        'fileio MYFILE      update' MYREC

           /* Jump forward again to current Header rec */
        'fileio MYFILE      read MYREC  ttr' x2c(Curr_H)
        D_Count= 0   /* Reset count of Detail records */
    end

  if   Eof then leave

  if   C1 = 'D'                           /* Detail record? */
    then D_Count = D_Count+1              /* Count Detail records */
    else Prev_H  = MYFILE.1.RECID  /* Save TTR of Header */

end
'fileio MYFILE        close'
```

`RECLEN` *`record_length`*

Applicable only to **z/OS Unix HFS/ZFS** input files with the NOEOL option specified. **RECLEN** is both an option on the READ operation and a separate operation by itself, and sets the length of the next (and subsequent) record(s) read.

The *record_length* may be supplied in either of the following formats:

- ◊ Unquoted decimal numeric value. e.g. *1538*
- ◊ Hexadecimal value of the form *X'...'* containing hex digits 0-9, a-f and A-F. e.g. *X'3DE827'* Double quotes may be used instead of single quotes if necessary.

For example, to read a z/OS unix file with the following repeating layout:

1. A fixed 1-byte binary field containing the **length** of a Company Name
2. A character field containing the Company Name (len=*??*)
3. A fixed 2-byte binary field containing the **length** of a number of comma separated email addresses
4. A character field containing the email addresses (len=*??*)

```
address "CBLSDATA" /* REXX */

parse arg HfsFile

"fileio" HfsFile "openread name MyHFS noeol"

do forever
  "fileio MyHFS reclen 1"
  "fileio MyHFS read  L1" ; if rc <> 0 then leave
  L1=c2d(L1)
  "fileio MyHFS reclen" L1
  "fileio MyHFS read  CompanyNam"
  say '"'CompanyNam'"'

  "fileio MyHFS reclen 2"
  "fileio MyHFS read  L2"
  L2=c2d(L2)
  "fileio MyHFS reclen" L2
  "fileio MyHFS read  EmailAddrs"
  say '"'EmailAddrs'"'
  say ' ----------------------------------------'
  say ' '

end
"fileio MyHFS close"
exit
```

Output_Operations:

```
|--+--- DELete ---------------------------------------------+----------------|
   |                                                        |
   +-+- INSert -------+--- text_literal --------------------+
   | |                |                                     |
   | +- UPDate -------+                                     |
   | |                |                                     |
   | +- WRite --------+                                     |
   |                                                        |
   +-+- INSERTV|INSV -+--- rexxvar -------------------------+
   | |                |                                     |
   | +- UPDATEV|UPDV -+                                     |
   | |                |                                     |
   | +- WRITEV |WRV --+                                     |
   |                                                        |
   +-+- INSERTX|INSX -+--- hex_literal ---------------------+
   | |                |                                     |
   | +- UPDATEX|UPDX -+                                     |
   | |                |                                     |
   | +- WRITEX |WRX --+                                     |
   |                                                        |
   +-+- SWRite -------+--- stem_name -----------------------+
   | |                |                                     |
   | +- SWRITEX ------+                                     |
   |                                                        |
   +-+- FINSert --------+--- | Field_Values | --------------+
   | |                  |                                   |
   | +- FUPDate --------+                                   |
   | |                  |                                   |
   | +- FWRite ---------+                                   |
   |                                                        |
   +--- COMMIT ---------------------------------------------+
```

Field_Values:

```
|--+---------------+--+--------------------+--+--------------------------+-->
   |               |  |                    |  |                          |
   +- record_type -+  |  +- , -------+     |  |          +-- , -------+  |
                      |  v           |     |  |          v            |  |
                      +- ( field_name +-) -+  +- Values( field_value +-)-+

>--+----------------------------+-----------------------------------------|
   |                            |
   +- RANDomize ----------------+
   |                            |
   |            +-- , ------+   |
   |            v           |   |
   +- RANDomize( field_name +-)-+
```

`DELETE`

For **VSAM** data sets and **DB2 Tables** only, this operation will delete the current record or table row.

For **VSAM** data sets only, the OPENUPDATE operation must be used to open the file.

`INSERT` *`text_literal`*

For **VSAM KSDS** data sets only, this operation will insert a record **out of key sequence**. The *text_literal* is the record text to be inserted and must be supplied as an **unquoted character string**.

```
address "CBLSDATA" /* REXX */

do forever
  "fileio TEST.SEQ     read  My_Rec"   /* Records out of key seq */
  if rc <> 0 then leave

  "fileio TEST.KSDS  insert" My_Rec
end
"fileio TEST.SEQ   close"
"fileio TEST.KSDS  close"
```

An explicit OPENUPDATE operation is **unnecessary** (implied by this operation), but if coded then any WRITE operation will also be treated as an out of sequence insert.

`UPDATE` *`text_literal`*

This operation will replace the most recently read record with an updated version. The *text_literal* is the record text to be inserted and must be supplied as an **unquoted character string**.

An explicit OPENUPDATE operation **is necessary** prior to execution of **INSERT**.

```
address "CBLSDATA" /* REXX */

"fileio  TEST.FILE  OpenUpdate"
"fileio  TEST.FILE    read  Rec1"

Rec1 = overlay(date(s) time(), Rec1, 17)  /* Update timestamp at pos 17 */

"fileio  TEST.FILE  update" Rec1
"fileio  TEST.FILE  close"
```

WRITE *text_literal*

This operation will write a new output record. The *text_literal* is the record text to be inserted and must be supplied as an **unquoted character string**.

See also the APPEND option for the OPENWRITE operation.

```
address "CBLSDATA" /* REXX */

do forever
  "fileio TEST.FILE      read   My_Rec"
  if rc <> 0 then leave

  if datatype(left(My_Rec,4) <> "NUM"
    then My_Rec = overlay("0000", My_Rec, 1)

  "fileio TEST.FILE.COPY write" My_Rec
end
"fileio TEST.FILE      close"
"fileio TEST.FILE.COPY close"
```

An explicit OPENWRITE operation is **unnecessary** (implied by this operation), unless APPEND is required to ensure existing records are not overwritten.

INSERTV *rexxvar*

For **VSAM KSDS** data sets only, this operation will insert a record **out of key sequence**. The record to be inserted is taken from the contents of named REXX variable *rexxvar*.

```
address "CBLSDATA" /* REXX */

do forever
  "fileio TEST.SEQ   read     My_Rec"   /* Records out of key seq */
  if rc <> 0 then leave

  "fileio TEST.KSDS  insertV  My_Rec"
end
"fileio TEST.SEQ   close"
"fileio TEST.KSDS  close"
```

An explicit OPENUPDATE operation is **unnecessary** (implied by this operation), but if coded then any WRITEV operation will also be treated as an out of sequence insert.

UPDATEV *rexxvar*

This operation will replace the most recently read record with an updated version. The replacement record is taken from the contents of named REXX variable *rexxvar*.

An explicit OPENUPDATE operation **is necessary** prior to execution of **INSERT**.

```
address "CBLSDATA" /* REXX */

"fileio  TEST.FILE  OpenUpdate"
"fileio  TEST.FILE  read     Rec1"

Rec1 = overlay(date(s) time(), Rec1, 17)  /* Update timestamp at pos 17 */

"fileio  TEST.FILE  updateV  Rec1"
"fileio  TEST.FILE  close"
```

WRITEV *rexxvar*

This operation will write a new output record. The new record is taken from the contents of named REXX variable *rexxvar*.

See also the APPEND option for the OPENWRITE operation.

```
address "CBLSDATA" /* REXX */

do forever
  "fileio TEST.FILE      read    My_Rec"
  if rc <> 0 then leave

  if datatype(left(My_Rec,4) <> "NUM"
    then My_Rec = overlay("0000", My_Rec, 1)

  "fileio TEST.FILE.COPY writeV  My_Rec"
end
"fileio TEST.FILE      close"
"fileio TEST.FILE.COPY close"
```

An explicit OPENWRITE operation is **unnecessary** (implied by this operation), unless APPEND is required to ensure existing records are not overwritten.

`INSERTX` *`hex_literal`*

This operation behaves just like INSERT except that the record, *hex_literal,* must be supplied as an **unquoted** set of character codes expressed in **hexadecimal notation** (i.e. containing only hex digits 0-9, a-f and A-F) which will be translated to a normal character string. For example, "`INSERTX C1C2C3`" performs identically to "`INSERTX ABC`".

`UPDATEX` *`hex_literal`*

This operation behaves just like UPDATE except that the record, *hex_literal,* must be supplied as an **unquoted** set of character codes expressed in **hexadecimal notation** (i.e. containing only hex digits 0-9, a-f and A-F) which will be translated to a normal character string. For example, "`UPDATEX C1C2C3`" performs identically to "`UPDATEX ABC`".

`WRITEX` *`hex_literal`*

This operation behaves just like WRITE except that the record, that the record, *hex_literal,* must be supplied as an **unquoted** set of character codes expressed in **hexadecimal notation** (i.e. containing only hex digits 0-9, a-f and A-F) which will be translated to a normal character string. For example, "`WRITEX C1C2C3`" performs identically to "`WRITEX ABC`".

`SWRITE` *`stem_name`*

The stem write operation will write new output records using values assigned to REXX compound variables with a stem of *stem_name*.

Records 1 to *n* are written so that each record's data is the corresponding value assigned to variables *stem_name*.1 to *stem_name.n*, where *n* is the value assigned to *stem_name*.0.

Specification of a stem symbol *stem_name* is mandatory. If *stem_name* does not end with a "." (dot/period), then one is added automatically.

See also the APPEND option for the OPENWRITE operation.

```
address "CBLSDATA" /* REXX */

"fileio TEST.FILE     SRead    My_Stem"

do i=1 to My_Stem.0
  if datatype(left(My_Stem.i,4) <> "NUM"
    then My_Stem.i = overlay("0000", My_Stem.i, 1)
end

"fileio TEST.FILE.COPY SWrite  My_Stem"
```

An explicit OPENWRITE operation is **unnecessary** (implied by this operation), unless APPEND is required to ensure existing records are not overwritten.

`SWRITEX` *`stem_name`*

This operation behaves just like SWRITE except that each of the values assigned to variables *stem_name*.1 to *stem_name.n* is first converted from **printable hex** before being written.

`FINSERT | FUPDATE | FWRITE` **`Field_Values`**

The **Formatted output** operations perform the same basic function as INSERT, UPDATE and WRITE, except that instead of supplying **raw** record data, a list of field names and their corresponding values are supplied. FILEIO will build a record based on these values according to the set of record mappings specified by USING *copybook_or_mapping_file*.

The syntax for FINSERT/FUPDATE/FWRITE is modelled on the INSERT command that you may have used in an Data-Edit macro.

`record_type`

For **FINSERT** and **FWRITE only**, this is the name of the record type of the record to be written, and **must be omitted** for **FUPDATE**.

The *record_type* may also be omitted if there is only one record type in the mapping structure.

`(field_name1,field_name2,field_name3, ...)`

A list of fields for which a corresponding value (*field_value1*,*field_value2*,*field_value3 ...*) is also supplied, and is to be generated in the formatted output record.

Each field may be identified either by its reference number (e.g. *#12*) or its name (e.g. *SeqNUM*). To specify multiple fields each must be separated by a comma and enclosed in parentheses.

If the field is a **struct** or a **union**, then an error message is returned. If the field is part of an array, then an individual array element must also be specified in parentheses as a subscript to the field. e.g. *#13(3)* meaning the **3rd element** of a single dimension array. A subscript entry must exist for each dimension of the array. e.g. *RoomSize(6,2,4)* refers to an element within a **three dimensional array**.

If a list of fields is not specified then as many of the set of the selected fields as there are supplied values are used as the default.

`VALUES( field_value1,field_value2,field_value3, ... )`

A list of values corresponding to each entry in the field list.

If a character string field value contains special characters, blanks or commas, then it must be delimited by quotation mark (") or apostrope (') characters. If the value contains characters which match the delimiting characters, then each occurrence of this character must be escaped by prefixing it with the escape character. The escape character is the same as the delimiting character. e.g. **VALUES('He said "It''s John''s bike."')**

`RANDOMIZE(field_name1,field_name2,field_name3, ...)`

A list of fields for which "random" **test data will be generated**.

If one does not already exist for the named field then a default RANDOMIZER object will be created, with characteristics based on the data-type of the field.

See Test_Data_Operations for more information on how to create a "RANDOMIZER object" associated with each field.

`RANDOMIZE`

If no list of fields is specified in brackets then, unless the field is assigned a specific value using the **VALUES** keyword the following rules apply:

Operation	RANDOMIZER Rule
FWRITE FINSERT	Specified without an explicit list of columns, this option causes a new test data value to be generated for **all** fields. If any field does not already have a RANDOMIZER object defined, then one will be automatically created using default characteristics based on the **data-type** of the field. Since the **default** action is that a new test data value will be generated for all fields that already have an existing RANDOMIZER object defined, in order to deactivate it you must specify **RANDOMIZE()**.
FUPDATE	Specified without an explicit list of columns, this option causes a new test data value to be generated for all fields that already have an existing RANDOMIZER object defined.

Fields within the written records do not require explicitly assigned values.

If fields are not explicitly assigned values (either because no list of field values is specified, or because the record type contains more fields than there are values in the values list), then default values are assigned as follows:

◊ If a USE WHEN expression has been defined that sets **record-type identification criteria**, then an attempt is made to populate any fields involved with values that satisfy the expression.

◊ For **FUPDATE** only, the already existing value is maintained

◊ A "Test Data" value is **generated** if either:
1. **RANDOMIZE(*field_col*)** is explicitly requested, or
2. A RANDOMIZER object already exists for *field_col*.

◊ 0 (zero) for numeric fields (e.g. binary or packed-decimal).

◊ Nulls for bit and hex fields.

◊ Blanks for all other field types.

```
address "CBLSDATA" /* REXX */

'create struct MY.FILEKIT.SDO(TUNES)           ',
   '(Artist       struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="A"                       ',
   '                                            ',
   ',Album        struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="L"                       ',
   '                                            ',
   ',Track        struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Num                 int(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="T"                       ',
   '                                            ',
   ')                                           ',
   'replace                                     '


'fileio MY.TUNES   openwrite  name MyFile       ',
    'using MY.FILEKIT.SDO(TUNES)'

'fileio MyFile  fwrite ARTIST  (Name) V("Adele") '
'fileio MyFile  fwrite ALBUM   (Name) V("21"   ) '

'fileio MyFile  fwrite TRACK                    ',
     'VALUES("T","1","Rolling In the Deep")'

'fileio MyFile  fwrite TRACK                    ',
     'VALUES("T","2","Rumour Has It"      )'

'fileio MyFile  fwrite TRACK                    ',
     'VALUES("T","3","Turning Tables"     )'

'fileio MyFile  close'

'browse MY.TUNES  using MY.FILEKIT.SDO(TUNES)'
```

COMMIT

COMMIT only applies to DB2 output and determines when updates are committed to the database (with the SQL COMMIT statement).

A COMMIT is automatically actioned by the CLOSE operation.

Selection_Operations:

```
|--+-+- FILTer ---------+---+--------- selection_file -----------+------+---|
   | |                  |   |                                    |      |
   | +- TOFCondition ---+   +- ( -- | Selection_Criteria | -- ) -+      |
   | |                  |                                               |
   | +- EOFCondition ---+                                               |
   |                                                                    |
   +--- MEMBer --+-- library_member_name -------------+-----------------+
   |             |                                    |                 |
   |             +-- library_member_name.generation --+                 |
   |                                                                    |
   +--- NEXTMEMber -----------------------------------------------------+
   |                                                                    |
   +--- PREVMEMber -----------------------------------------------------+
   |                                                                    |
   |               +----- , ------+                                     |
   |               v              |                                     |
   +--- SELect --+-+- field_name -+-+--+--------------------+-----------+
   |             |                  |  |                    |           |
   |             +-- * -------------+  +- FROM record_type -+           |
   |                                                                    |
   |              +------ , ------+                                     |
   |              v               |                                     |
   +--- VIEW -----+- record_type -+-------------------------------------+
   |                                                                    |
   +--- WHere (expr) -+------------------+------------------------------+
                      |                  |
                      +- IN record_type -+
```

Selection_Criteria:

```
     +----------------------------------------------------------+
     v                                                          |
|--+-+- INclude rec_type --+--------------+--+----------------+-+-+--------->
   |                       |              |  |                |   |
   |                       +- WHere expr -+  +- LIMit n_hits -+   |
   |                                                              |
   | +----------------------------------------------------------+ |
   | v                                                          | |
   +-+- EXclude rec_type --+--------------+--+----------------+-+-+
                           |              |  |                |
                           +- WHere expr -+  +- LIMit n_hits -+

>--+--------------------+---------------------------------------------------|
   |                    |
   +- Stopafter n_hits -+
```

FILTER (**Selection_Criteria**)

Use the FILTER operation to define input **record selection criteria**.

Please note that the WHERE operation provides most of the same functionality as FILTER and (if adequate) may well provide a slight performance improvement over FILTER.

The filter clause may be supplied either in-line (must be in parentheses) or from a named dataset.

expr is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of the WHERE expression must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

The WHERE expression is applied to each record assigned the record type *rec_type* and, if the result is "true", the record is selected for include or exclude as indicated by the prevailing INCLUDE or EXCLUDE filter. If multiple INCLUDE/EXCLUDE *record_type* WHERE expressions exist for the same record type, then a logical OR is implied for all the expressions relating to that record type.

Example of a simple WHERE expression:

```
where Track << "rock'n'roll"  or Track << "rock and roll"
```

Example of a more complex WHERE expression:

```
where (      TrackNum = 1

        &    substr(record,17,3) = "INS"

        & (  Artist    = "Adele"   /* "and" is equally valid for "&" */
           | Artist    = "Prince"  /* "or"  is equally valid for "|" */
           | Track    << "Blues"   /* "<<" means "contains" */
           | Date     >> "2022"    /* ">>" means "begins with" */
          )
      )
```

See description of the BROWSE command for further details on the FILTER clause.

```
address "CBLSDATA" /* REXX */

'create struct MY.FILEKIT.SDO(TUNES)           ',
   '(Artist       struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="A"                       ',
   '                                            ',
   ',Album        struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="L"                       ',
   '                                            ',
   ',Track        struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Num                 int(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="T"                       ',
   '                                            ',
   ')                                           ',
   'replace                                     '

'fileio MY.TUNES   openread   name MyFile       ',
    'using MY.FILEKIT.SDO(TUNES)'

'fileio MyFile  filter (include TRACK where Num > 40)'

do forever
  'fileio MyFile  fvalue read'
  if rc <> 0 then leave
  say MyFile.RecTyp MyFile.Num MyFile.Name
end

'fileio MyFile  close'
```

The example above will provide identical results to the following, but its use of FILTER will **dramatically improve performance**.

```
'create struct MY.FILEKIT.SDO(TUNES)           ',
   '(Artist       struct                        ',
------------ 21 line(s) not displayed ------------
   ')                                           ',
   'replace                                     '


'fileio MY.TUNES   openread   name MyFile       ',
    'using MY.FILEKIT.SDO(TUNES)'

do forever
  'fileio MyFile  fvalue read'
  if MyFile.RecTyp = 'T' & MyFile.Num > 40
    then say MyFile.RecTyp MyFile.Num MyFile.Name
end
'fileio MyFile  close'
```

`TOFCondition ( `**`Selection_Criteria`**` )`

Use the TOFCONDITION operation to set a **logical top of file** condition.

This means that all input records are **bypassed** until the first that satisfies the specifed selection criteria. As soon as a match occurs TOFCONDITION is deactivated.

Having satisfied (and deactivated) an earlier one, further TOFCONDITIONs may be be specified in order to proceed efficiently through your file.

```
address "CBLSDATA" /* REXX */

'fileio  CBL.SMF.T030  openread   name SMF30',
      'using CBL.INST.CBL21042.SZZSDIST.SDO(T030)'

'fileio SMF30  TOFCondition',
      '(include SMF030_Common_Address_Space_Work ',
      '    where zTME >= "2019/07/12 13:00:00"   ',
      ')'

'fileio SMF30  EOFCondition',
      '(include SMF030_Common_Address_Space_Work ',
      '    where zTME >= "2019/07/12 14:00:00"   ',
      ')'

'fileio SMF30  view SMF030_Identification'
'fileio SMF30  select zJOBNAME,zUSERID,zRST,zGRP'
do forever
  'fileio SMF30  fvalue read'
  if rc <> 0 then leave

  say 'Timestamp:' SMF30.zRST
  say 'Job Name :' SMF30.zJOBNAME
  say 'UserId   :' SMF30.zUSERID
  say 'Group    :' SMF30.zGRP
  say '       --------------------------------'
  say ' '

end
'fileio SMF30  close'
```

EOFCondition (**Selection_Criteria**)

Use the EOFCONDITION operation to set a **logical end of file** condition.

This means that **return code 4** will be set as soon as a record is encountered that satisfies the specifed selection criteria.

```
address "CBLSDATA" /* REXX */

'fileio  CBL.SMF.T030  openread   name SMF30',
      'using CBL.INST.CBL21042.SZZSDIST.SDO(T030)'

'fileio SMF30  TOFCondition',
      '(include SMF030_Common_Address_Space_Work ',
      '    where zTME >= "2019/07/12 13:00:00"   ',
      ')'

'fileio SMF30  EOFCondition',
      '(include SMF030_Common_Address_Space_Work ',
      '    where zTME >= "2019/07/12 14:00:00"   ',
      ')'

'fileio SMF30  view SMF030_Identification'
'fileio SMF30  select zJOBNAME,zUSERID,zRST,zGRP'
do forever
  'fileio SMF30  fvalue read'
  if rc <> 0 then leave

  say 'Timestamp:' SMF30.zRST
  say 'Job Name :' SMF30.zJOBNAME
  say 'UserId   :' SMF30.zUSERID
  say 'Group    :' SMF30.zGRP
  say '       --------------------------------'
  say ' '

end
'fileio SMF30  close'
```

When **RC=4** is set, the record that triggered the EOFCONDITION is left in a **"read pending"** state, in case you decide you wish to continue processing your file after this point. In which case, in order to proceed you must first deactivate the earlier EOFCONDITION using a null EOFCONDITION operation. e.g.

```
'fileio MYFILE  EOFCondition  ()'     /* Unset existing EOFCOND */
```

Having satisfied (and deactivated) an earlier one, further EOFCONDITIONs may then be be specified in order to proceed efficiently through your file.

Selection_Criteria | *selection_file*

A **Selection_Criteria** clause must be specified in "( )" (parentheses) and may contain comment data enclosed by "/*" and "*/". If **Selection_Criteria** clause is specified via a *selection_file*, then comment data may also occur before and after the *selection_criteria* clause.

The **Selection_Criteria** clause may contain **either** *INCLUDE* or *EXCLUDE* selection sub-clauses **but not both**.

When **Selection_Criteria** is applied, the record data is tested against each selection sub-clause in the order specified until a true condition is returned. On encountering a true condition, both the hit count for the individual selection sub-clause and the overall hit count is incremented by one and the record included or excluded as approriate.

Note that, for TOFCondition or EOFCondition to be triggered, **Selection_Criteria** must cause the record to be **included**. i.e. one of the following must be true:

1. The record satisfies **ANY** of the *INCLUDE* sub-clauses.
2. The record satisfies **NONE** of the *EXCLUDE* sub-clauses.

Once the hit count for an individual *INCLUDE* or *EXCLUDE* selection sub-clause matches its *LIMIT* threshold, then that sub-clause plays no further part in the selection process. Similarly, once the overall hit count matches the *STOPAFT* threshold, no further record filtering occurs and the remainder or the records are excluded or included as appropriate.

The following options are supported by the *selection_criteria* clause.

`INCLUDE` *`rec_type`*

INCLUDE denotes the start of an INCLUDE sub-clause which, together with optional parameters *WHERE* and *LIMIT*, identifies conditions for which a record is included.

The INCLUDE sub-clause applies only to records that have been assigned the specified *rec_type*. If the record is not assigned this record type, a false condition is returned for the sub-clause.

A number of INCLUDE sub-clauses may be specified for different record types or for the same record type with different **WHERE expression** conditions. If one or more INCLUDE sub-clauses are specified, then records that do not satisfy any of the sub-clause conditions will be **excluded by default**.

Note that *rec_type* "Record" (with field name "UnMapped") may be used to perform selection criteria on the unformatted record data, irrespective of a structure (USING *copybook_or_mapping_file*) being specified. In this way, *selection_criteria* may test **all** records regardless of their assigned record type.

INCLUDE and EXCLUDE parameters are mutually exclusive.

`EXCLUDE` *`rec_type`*

EXCLUDE denotes the start of an EXCLUDE sub-clause which, together with optional parameters *WHERE* and *LIMIT*, identifies conditions for which a record is excluded.

The EXCLUDE sub-clause applies only to records that have been assigned the specified *rec_type*. If the record is not assigned this record type, a false condition is returned for the sub-clause.

A number of EXCLUDE sub-clauses may be specified for different record types or for the same record type with different **WHERE expression** conditions. If one or more EXCLUDE sub-clauses are specified, then records that do not satisfy any of the sub-clause conditions will be **included by default**.

Note that *rec_type* "Record" (with field name "UnMapped") may be used to perform selection criteria on the unformatted record data, irrespective of a structure (USING *copybook_or_mapping_file*) being specified. In this way, *selection_criteria* may test **all** records regardless of their assigned record type.

INCLUDE and EXCLUDE parameters are mutually exclusive.

`WHERE` *`expr`*

WHERE applies further selection conditions to records assigned to the record type specified by the last INCLUDE *rec_type* or EXCLUDE *rec_type* parameter processed.

expr is a valid FileKit Data-Edit expression which supports function calls, *rec_type* field names and references, sub-expressions, arithmetic, relational and logical operators.

The result of the WHERE expression must be numeric and is treated as being **Boolean** in nature with a zero value indicating a **"false"** condition and any non-zero value indicating a **"true"** condition.

The WHERE expression is applied to each record assigned the record type *rec_type* and, if the result is "true", the record is selected for include or exclude as indicated by the prevailing INCLUDE or EXCLUDE selection. If multiple INCLUDE/EXCLUDE *rec_type* WHERE expressions exist for the same record type, then a **logical OR** is implied for all the expressions relating to that record type.

`LIMIT` *`n_hits`*

LIMIT *n_hits* may be included as part of a single INCLUDE (or EXCLUDE) sub-clause specification and applies only to that sub-clause.

When the number of records selected by an individual INCLUDE (or EXCLUDE) sub-clause reaches the *n_hits* value specified by LIMIT, then that sub-clause no longer forms part of the selection applied to subsequent input records.

LIMIT provides a method whereby the subset of records selected is spread more evenly across the selection's sub-clause conditions than could be achieved by use of the *STOPAFTER* parameter alone.

By default, no LIMIT threshold is applied to any INCLUDE (or EXCLUDE) sub-clause.

`STOPAFTER` *`n_hits`*

When the total number of records selected by the INCLUDE (or EXCLUDE) sub-clauses reaches the *n_hits* value specified by STOPAFTER, then no further selection testing occurs.

If an INCLUDE selection, then all remaining untested records are excluded. If an EXCLUDE selection, then all remaining untested records are included.

By default, no STOPAFTER threshold is applied to the *selection_criteria* clause.

Batch JCL Example

```
//FIOSAMP  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SDEIN     DD *
  macro FIOTMPDD

//FIOTMPDD  DD *
  address "CBLSDATA" /* REXX */

  'fileio  CBL.SMF.T030  openread   name SMF30',
        'using CBL.INST.CBL21042.SZZSDIST.SDO(T030)'

  'fileio SMF30  TOFCondition SDETOFC'
  'fileio SMF30  EOFCondition SDEEOFC'

  'fileio SMF30  view SMF030_Common_Address_Space_Work',
                    ',SMF030_Identification'

  'fileio SMF30  select zTME',
                'from SMF030_Common_Address_Space_Work'

  'fileio SMF30  select zJOBNAME,zUSERID,zRST,zGRP',
                'from SMF030_Identification'

  'fileio SMF30  option rectype'

  do forever
    'fileio SMF30  fvalue read'
    if rc <> 0 then leave

    if SMF30.1.rectype = 'SMF030_COMMON_ADDRESS_SPACE_WORK'
      then iterate

    msg 'SMF Date/Time :' SMF30.zTME
    msg 'RDR Date/Time :' SMF30.zRST
    msg 'Job Name      :' SMF30.zJOBNAME
    msg 'UserId        :' SMF30.zUSERID
    msg 'Group         :' SMF30.zGRP
    msg '       --------------------------------'
    msg ' '

  end
  'fileio SMF30  close'
/*
//SDETOFC   DD *
 TOFCondition
  (include SMF030_Common_Address_Space_Work
         where zTME >= "2019/07/12 13:00:00"
  )
//SDEEOFC   DD *
 EOFCondition
  (include SMF030_Common_Address_Space_Work
         where zTME >= "2019/07/12 14:00:00"
  )
/*
```

MEMBER *library_member_name | library_member_name.generation*

When processing multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the MEMBER operation may be used to switch directly to a named member, *library_member_name*, or member generation, *library_member_name.generation*.

For **input**, all remaining records from the current member will be bypassed, and the next input operation will start at the first record of the named member/generation.

For **output**, the current member will be closed (STOWed), updating its directory timestamp and other statistics, and the next output operation will write the first record of the named member/generation.

Related OPTIONS:

MEMBER	Extracts the **current member name** to a REXX variable whenever it changes, allowing your calling program to monitor which member it is dealing with at any time.

```
address "CBLSDATA" /* REXX */
  /* Copy members from a "search path" of libraries */

 arg InpLibPath OutLibrary
 /*  e.g. InpLibPath = "MY.INPUT.LIB1,MY.INPUT.LIB2(A*,P*,Y*)"
 |        OutLibrary = "MY.OUTPUT.LIB"
 */

 MemberCount=0
 CurrentMember=''
 "fileio" InpLibPath "openread  cld   name InPath"
 "fileio" OutLibrary "openwrite pds   name OutLib"

 "fileio InPath options member"

 do forever
   "fileio InPath sread MYREC 100"; rrc=rc
       /* RC=2 just means end of member was reached */
   if rrc >= 4 then leave

   /* Handle new member, even an empty one */
   if InPath.1.Member <> CurrentMember then
     do; "fileio OutLib member" InPath.1.Member
         wrc = rc; if wrc <> 0 then leave
         MemberCount=MemberCount+1
         CurrentMember = InPath.1.Member
     end

   "fileio OutLib swrite MYREC"   ; wrc=rc
   if wrc > 0  then exit wrc
 end
 "fileio InPath close"
 "fileio OutLib close"

 'msg' MemberCount 'members copied'

 exit 0
```

NEXTMEMBER

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the NEXTMEMBER operation may be used to switch directly to the next member.

All remaining records from the current member will be bypassed, and the next input operation will start at the first record of the next member.

Related OPTIONS:

MEMBER Extracts the **current member name** to a REXX variable whenever it changes, allowing your calling program to monitor which member it is dealing with at any time.

MEMLIST Extracts the **list of members** to a REXX stem variable. Returned information includes:
◊ Library name (varies for CLD input only)
◊ Member name
◊ Last modified timestamp
◊ Number of records
◊ Name of user that performed last modification

```
address "CBLSDATA" /* REXX */
  /* Display 1st 5 recs from each member of PDS library search-path */

 arg MyLib

 "fileio" MyLib "openread  cld   name InLib"

 "fileio InLib options libname member recno"

 do forever
   "fileio InLib sread MYREC 5"; rrc=rc

   say 'MEMBER='InLib.1.libname'('InLib.1.Member')'
   do i=1 to MYREC.0
     say right(InLib.i.recno,5,'0') MYREC.i
   end
   say '----------------------------------------------'
   say ' '
   say ' '
   say ' '

   "fileio InLib nextmember"; rrc=rc
   if rrc > 0 then leave
 end
 "fileio InLib close"
```

PREVMEMBER

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the PREVMEMBER operation may be used to switch directly to the previous member.

All remaining records from the current member will be bypassed, and the next input operation will start at the first record of the previous member.

Related OPTIONS:

MEMBER	Extracts the **current member name** to a REXX variable whenever it changes, allowing your calling program to monitor which member it is dealing with at any time.
MEMLIST	Extracts the **list of members** to a REXX stem variable. Returned information includes: ◊ Library name (varies for CLD input only) ◊ Member name ◊ Last modified timestamp ◊ Number of records ◊ Name of user that performed last modification

```
address "CBLSDATA" /* REXX */
  /* Display 1st 5 recs from each member of PDS library search-path */
  /* processing member names in reverse order.                      */

 arg MyLib

 "fileio" MyLib "openread  cld   name InLib"

 "fileio InLib options  memlist  libname  member  recno  noiop"

   /* Switch directly to the last member */
   /* before proceeding in reverse order */
 i          = InLib.memlist.0
 parse var InLib.memlist.i Lib'('LastMember')' .

 "fileio InLib member" LastMember

 do forever
   "fileio InLib sread MYREC 5"; rrc=rc

   say 'MEMBER='InLib.1.libname'('InLib.1.Member')'
   do i=1 to MYREC.0
     say right(InLib.i.recno,5,'0') MYREC.i
   end
   say '----------------------------------------------'
   say ' '
   say ' '
   say ' '

   "fileio InLib prevmember"; rrc=rc
   if rrc > 0 then leave
 end
 "fileio InLib close"
```

`SELECT field_name1, field_name2, etc FROM record_type`

For files opened with an associated record mapping, use the SELECT operation to restrict which **fields** are **active**, and define their order.

Each field may be identified either by its reference number (e.g. *#12*) or its name (e.g. *SeqNUM*). To specify multiple fields each must be separated by a comma.

"**FROM** *record_type*" may be omitted if any of the following are true:

◊ Only one record type is defined by the mapping
◊ A VIEW operation has selected only *record_type*
◊ *record_type* is the first record type defined in the mapping

In particular, SELECT will affect subsequent use of the following operations:

FVALUE/QP/QF

This operation sets REXX variables for "selected" fields only. When dealing with record-types defined with dozens of fields, only a few of which are relevant to your application, then use of **SELECT** can minimize the number REXX variables set, resulting in dramatically lower CPU usage and improved run times.

FINSERT / FUPDATE / FWRITE

For these operations, a list of fields (for which new values are also supplied) may be stated explcitly on the command. However, if field names are not stated, then it is assumed that values are being supplied for "selected" fields only (in the correct order).

```
address "CBLSDATA" /* REXX */

'fileio  CBL.SMF.T030  openread   name SMF30',
      'using CBL.INST.CBL21042.SZZSDIST.SDO(T030)'

'fileio SMF30  view SMF030_Identification'
'fileio SMF30  select zJOBNAME,zUSERID,zRST,zGRP'
do forever
  'fileio SMF30  fvalue read'
  if rc <> 0 then leave

  say 'Timestamp:' SMF30.zRST
  say 'Job Name :' SMF30.zJOBNAME
  say 'UserId   :' SMF30.zUSERID
  say 'Group    :' SMF30.zGRP
  say '       --------------------------------'
  say ' '

end
'fileio SMF30  close'
```

VIEW *record_type1, record_type2, etc.*

For files opened with an associated record mapping, use the VIEW operation to restrict which **record-types** are **"visible"** to input operations. The result being that READ, SREAD, FVALUE READ, etc. operations will automatically bypass records that are not "visible".

To specify multiple record types each must be separated by a comma.

A leading **plus (+)** or **minus (-)** sign before the list of record types indicates that this operation is adding to, or removing from, the list of visible record types.

```
address "CBLSDATA" /* REXX */

'create struct MY.FILEKIT.SDO(TUNES)           ',
   '(Artist       struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="A"                       ',
   '                                            ',
   ',Album        struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="L"                       ',
   '                                            ',
   ',Track        struct                        ',
   '         (                                  ',
   '           RecTyp             char(  1)     ',
   '          ,Num                 int(  1)     ',
   '          ,Name            varchar(100)     ',
   '         )                                  ',
   '  use when RecTyp="T"                       ',
   '                                            ',
   ')                                           ',
   'replace temp                                '


'fileio MY.TUNES   openread   name MyFile       ',
    'using MY.FILEKIT.SDO(TUNES)'

'fileio MyFile  view Artist'
'fileio MyFile  select Name'

do forever
  'fileio MyFile  fvalue read'
  if rc <> 0 then leave
  say MyFile.Name        /* Artist name */
end
'fileio MyFile  close'
```

The example above will provide identical results to the following, but its use of VIEW (and SELECT) will **significantly improve performance**.

```
'create struct MY.FILEKIT.SDO(TUNES)           ',
   '(Artist       struct                        ',
------------ 21 line(s) not displayed ------------
   ')                                           ',
   'replace temp                                '


'fileio MY.TUNES   openread   name MyFile       ',
    'using MY.FILEKIT.SDO(TUNES)'

do forever
  'fileio MyFile  fvalue read'
  if MyFile.RecTyp = 'A'
    then say MyFile.Name  /* Artist name */
end
'fileio MyFile  close'
```

`WHERE (`*`expr`*`) IN `*`record_type`*

Use the WHERE operation to define input **record selection criteria**.

Please note that WHERE provides the same basic functionality as the FILTER operation, and may provide a slight performance improvement if none of FILTER's additional features are needed.

So, you may prefer to use a combination of WHERE and VIEW operations instead of FILTER, if **none** of the following features are required:

◊ LIMIT *n_hits*
◊ STOPAFTER *n_hits*
◊ Selection of primary *segmented* records based on values in a secondary segment.

Note that you may require a **VIEW** and **multiple WHERE operations** to reproduce the functionality of single **FILTER**.

expr **must be supplied in parentheses**.

"**IN** *record_type*" may be omitted if any of the following are true:

♦ No record mapping is in effect
♦ Only one record type is defined by the mapping
♦ A VIEW operation has selected only *record_type*
♦ *record_type* is the first record type defined in the mapping

expr is a valid FileKit Data-Edit expression which supports function calls, *rec_type* field names and references, sub-expressions, arithmetic, relational and logical operators.

The result of the WHERE expression must be numeric and is treated as being **Boolean** in nature with a zero value indicating a **"false"** condition and any non-zero value indicating a **"true"** condition.

The WHERE expression is applied to each record assigned the record type *record_type* and, if the result is "false", the record is **bypassed**.

Use of WHERE (or FILTER) does not depend on a copybook mapping being applied. e.g.

```
"fileio MYFILE  where                                   " ,
               "(      substr(record,09,2)   > c'72'   " ,
               " & (   substr(record,14,1)   = c'R'    " ,
               "    or substr(record,24,1)   = c'E'    " ,
               "    or substr(record,27,1)   = c'P'    " ,
               "    or substr(record,34,1)   = c'G'    " ,
               "    or substr(record,37,1)   = c'V'    " ,
               "   )                                   " ,
               ")                                      " ,
```

The two examples that follow will provide identical results, but the first, through its use of **WHERE**, will deliver **dramatically improved performance**.

Example 1:

```
address "CBLSDATA" /* REXX */

'create struct MY.FILEKIT.SDO(TUNES)           ',
   '(Artist       struct                        ',
   '          (                                 ',
   '            RecTyp            char(  1)     ',
   '           ,Name           varchar(100)     ',
   '          )                                 ',
   '  use when RecTyp="A"                       ',
   '                                            ',
   ',Album        struct                        ',
   '          (                                 ',
   '            RecTyp            char(  1)     ',
   '           ,Name           varchar(100)     ',
   '          )                                 ',
   '  use when RecTyp="L"                       ',
   '                                            ',
   ',Track        struct                        ',
   '          (                                 ',
   '            RecTyp            char(  1)     ',
   '           ,Num                int(  1)     ',
   '           ,Name           varchar(100)     ',
   '          )                                 ',
   '  use when RecTyp="T"                       ',
   '                                            ',
   ')                                           ',
   'replace                                     '


'fileio MY.TUNES   openread   name MyFile       ',
    'using MY.FILEKIT.SDO(TUNES)'

'fileio MyFile  view Track'   /* Bypass Artist and Album recs */

            /* "<<"  operator means "Contains" */
'fileio MyFile  where (Name << "blues" ',
                  ' or Name << "soul"  ',
                  ' or Name << "rock"  ',
                  '   )'

do forever
  'fileio MyFile  fvalue read'
  if rc <> 0 then leave
  say MyFile.RecTyp MyFile.Num MyFile.Name
end

'fileio MyFile  close'
```

Example 2:

```
'create struct MY.FILEKIT.SDO(TUNES)           ',
   '(Artist       struct                        ',
------------ 21 line(s) not displayed ------------
   ')                                           ',
   'replace                                     '

'fileio MY.TUNES   openread   name MyFile       ',
    'using MY.FILEKIT.SDO(TUNES)'

'fileio MyFile  view Track'   /* Bypass Artist and Album recs */

do forever
  'fileio MyFile  fvalue read'
  NameUpper = translate(MyFile.Name)
  if pos('BLUES',MyFile.RecTyp) > 0,
   | pos('SOUL' ,MyFile.RecTyp) > 0,
   | pos('ROCK' ,MyFile.RecTyp) > 0,
    then say MyFile.RecTyp MyFile.Num MyFile.Name
end

'fileio MyFile  close'
```

Test_Data_Operations:

```
|-- RANDomize -- field_name -+----------------+- | Randomizer Definition | -|
                             |                |
                             +- FOR rec_type -+
```

Randomizer Definition:

```
  (1) +--- | Character Values | ---+
      +---- | Numeric Values | ----+
      +--- | Date/Time Values | ---+
      +----- | Time Values | ------+
      |                            |
 |--+-+----------------------------+--+-----------------------+-------+------|
    | |                            |  |                       |       |
    | +----- | List Values | ------+  |         (2)           |       |
    | |                            |  |           +- SEQRL -+ |       |
    | +--- | Key List Values | ----+  |           |         | |       |
    | |                            |  +- CHAIN ---+---------+-+       |
    | +--- | Sentence Values | ----+              |         |         |
    | |                            |              +- SEQLR -+         |
    | +- | Personal Name Values | -+                                  |
    | |                            |                                  |
    | +-- LITeral -- "string" -----+                                  |
    |                                                                 |
    +------- | Pattern Value | ---------------------------------------+
    |                                                                 |
    +-- REPlacement --+--------------------------+--------------------+
                      |                          |
                      +- ( replace_expression ) -+
```

(1) Default value types matches that of the source field data type.

Numeric Values	Default for any numeric data type field.
Character Values	Default for any character or hex data type field.
Date/Time Values	Default for any DATE or TIMESTAMP data type field.
Time Values	Default for any TIME data type field.

(2) The last CHAIN element to have SEQRL or SEQLR specified will apply.

Character Values:

```
                     (2)
    +---- CHARS ------ "default_chars_list" --+
    |  (1)                                    |
    |  +- CHars --+                           |
    |  |          |                           |
 |--+--+----------+-+-------------------------+---- | Index Values | --------|
                    |                         |
                    +- "chars_list" ----------+
                    +- ALPHA -----------------+
                    +- ALPHANumeric ----------+
                    +- HEX -------------------+
                    +- HEXEVEN ---------------+
                    +- NUMeric ---------------+
                    +- LALPHA ----------------+
                    +- LALPHANumeric ---------+
                    +- LHEX ------------------+
                    +- LHEXEVEN --------------+
                    +- MALPHA ----------------+
                    +- MALPHANumeric ---------+
```

(1) CHARS keyword is mandatory for *"chars_list"*

(2) "*default_chars_list*" characters are: "A-Z", "0-9", "+-=,./*()_!:@#$" and the blank character.

Numeric Values:

```
                  (1)        (2)                              (4)
   +- RANGE -------- * -------- * ---------------------------+
   |                                                         |
|--+---------------------------------------------------------+-+---------+-|
   |                                                         | |         |
   +- RANGE ------ lo_num -+----------+----+---------------+-+ +- ZEROS -+
   |                       |          |    |               | |
   |                       +- hi_num -+    +- BASE string -+ |
   |                     (2)                                 |
   |                                                         |
   |                        (2)                              |
   |            +- 1 ---------- * ------ +1 ----+            |
   |            |                               |            |
   +- SEQuence -+-------------------------------+------------+
   |            |                               |            |
   |            +- lo_num -+------------------+-+            |
   |                       |                  |              |
   |                   (2) +- hi_num -+-------+              |
   |                                  |       |              |
   |                              (3) +- inc -+              |
   |          (3)                                            |
   +- ADJust -- inc --+-----------+--+------------------+----+
                      |           |  |                  |
                      +- PERCENT -+  +- (source_field) -+
```

(1) Default low number (*lo_num*) for a RANGE is "*", the minimum value supported for the numeric field.

(2) Default high number (*hi_num*) is "*", the maximum supported for the field. (Character => "9" for length #digits).

(3) Negative integer increment (*inc*) values are supported. Default increment for SEQUENCE is: +1
The low number (*lo_num*) may be **higher** than the high number (*hi_num*), in which case the default increment is -1.

(4) Numbers may be expressed with or without a decimal point and may include an exponent. The scale of the generated value will equal the largest scale supplied on *lo_num*, *hi_num* and *inc* values.

Date/Time Values:

```
(1)                                                           (3)
  +- DATE -+              +- RANGE -- 2001/01/01 00:00:00.00 -- hi_ts ----+
  |        |              |                                               |
|-+--------+-+----------+-+-----------------------------------------------+-|
             |          | |                                               |
             + "str_ts" + +--- NOW ---------------------------------------+
           (2)            |                                               |
                          +--- TODay -------------------------------------+
                          |                                               |
                          +-+- PAST ---+-+--------------------------------+
                          | |          | |                                |
                          | +- FUTure -+ |          +- DAYs --------+     |
                          |              |          |               |     |
                          |              +- int ----+---------------+-----+
                          |                         |               |     |
                          |                         +- HOURs -------+     |
                          |                         +- MINutes -----+     |
                          |                         +- SECs --------+     |
                          |                                               |
                          +- RANGE -- lo_ts -+--------+-+---------------+-+
                          |                  |        | |               | |
                          |                  + hi_ts -+ +- BASE string -+ |
                          |               (3)                             |
                          |                                   + DAYs ---+ |
                          |                                   |         | |
                          +- SEQuence lo_ts -+--------------+-+---------+-+
                          |                  |              | |         | |
                          |              (3) + hi_ts -+-----+ + HOURs --+ |
                          |                           |     | + MINutes + |
                          |                       (4) + inc + + SECs ---+ |
                          |                                               |
                          |                + DAYs ---+                    |
                          |                |         |                    |
                          +- ADJust - inc -+---------+-+----------------+-+
                                     (4)   |         | |                |
                                           + HOURs --+ + (source_field) +
                                           + MINutes +
                                           + SECs ---+
```

(1) DATE keyword required if source field is **not** a FileKit DATE or TIMESTAMP data type. (e.g. a character field)

(2) Omit "*str_ts*" for a DATE or TIMESTAMP source field.
"*str_ts*" is a quoted date/time string containing any character, but the following are substituted with date/time elements:
"CC", "CI", "YYYY", "YY", "MM", "MMM", "Mmm", "DD", "DDD", "Ddd", "JJJ", "WW", "WWS", "hh", "mm", "ss", "ttt"
Default combination is: **"CCYY/MM/DD hh:mm:ss.ttt"**, or **"CCYYMMDDhhmmssttt"** for numeric source fields.

(3) Default high date/time (*hi_ts*) is: 2042/09/17 23:53:47.37

(4) Negative integer increment (*inc*) values are supported. Default increment for SEQUENCE is: +1 DAY
The low date/time (*lo_ts*) may be **higher** than the high date/time (*hi_ts*), in which case the default increment is -1 DAY.

Time Values:

```
(1)                                                            (3)
  +- TIME -+              +- RANGE ------------- 00:00:00.00 -- hi_ti -----+
  |        |              |                                                |
|-+--------+-+----------+-+------------------------------------------------+-|
             |          | |                                                |
             + "str_ti" + +--- NOW ----------------------------------------+
           (2)            |                                                |
                          +-+- PAST ---+-+---------------------------------+
                          | |          | |                                 |
                          | +- FUTure -+ |            +- SECs --------+    |
                          |              |            |               |    |
                          |              +- int ------+---------------+----+
                          |                           |               |    |
                          |                           +- HOURs -------+    |
                          |                           +- MINutes -----+    |
                          |                                                |
                          +- RANGE -- lo_ti -+--------+-+----------------+-+
                          |                  |        | |                | |
                          |                  + hi_ti -+ +- BASE string -+  |
                          |                (3)                             |
                          |                                   + SECs ---+  |
                          |                                   |         |  |
                          +- SEQuence lo_ti -+--------------+-+----------+-+
                          |                  |              | |         |  |
                          |                  + hi_ti -+-----+ + HOURs --+  |
                          |                (3)        |     | + MINutes +  |
                          |                           + inc +              |
                          |                         (4)                    |
                          |                                                |
                          |                + SECs ---+                     |
                          |                |         |                     |
                          +- ADJust - inc -+---------+-+-----------------+-+
                                     (4)   |         | |                |
                                           + HOURs --+ + (source_field) +
                                           + MINutes +
```

(1) TIME keyword required if source field is **not** a FileKit TIME data type. (e.g. a character field)

(2) Omit "*str_ti*" for a TIME source field.
"*str_ti*" is a quoted time string containing any character, but the following are substituted with time elements:
"hh", "mm", "ss", "ttt"
Default combination is: **"hh:mm:ss.ttt"**, or **"hhmmssttt"** for numeric source fields.

(3) Default high time (*hi_ti*) is: 23:59:59.99

(4) Negative integer increment (*inc*) values are supported. Default increment for SEQUENCE is: +1 SEC
The low time (*lo_ti*) may be **higher** than the high time (*hi_ti*), in which case the default increment is -1 SEC.

List Values:

```
                  +--------+        +- VALLOCATION (1) ------------------+
                  v        |        |                                    |
|-- LIST --+-- ( -+- item -+- ) --+-+------------------------------------+-->
           |                      | |                                    |
           +------ list_file -----+ +- VALLOCATION ( pos -+---------+- ) -+
                                                          |         |
                                                          +- ,len -+

>---- | Index Values | ------------------------------------------------------|
```

Key List Values:

```
                   +--------+
                   v        |
|- KEYLIST -+-- ( -+- item -+- ) --+--- VALLOCATION ( pos -+---------+- ) --->
            |                      |                       |         |
            +------ list_file -----+                       +- ,len -+

>-- KEYLOCATION ( pos -+--------+- ) --+--------------------------+----------|
                       |        |      |                          |
                       +- ,len -+      +- KEY( key_expression ) -+
```

Sentence Values:

```
  +- LIST -+       +--------+       +- VALLOCATION (1) ------------------+
  |        |       v        |       |                                    |
|-+--------+-+- ( -+- item -+- ) -+-+------------------------------------+-->
             |                    | |                                    |
             +---- list_file -----+ +- VALLOCATION ( pos -+---------+- ) -+
                                                          |         |
                                                          +- ,len -+

>---- | Index Values | ------ VOCab -----+--------+-+----------+-------------|
                                         |        | |          |
                                         +- CAP1 -+ +- PERIOD -+
```

Personal Name Values:

```
                  +----- ANY ---------+
                  |                   |
|-- PERSon --- ( --+-------------------+-- ) ---- | Index Values | ---------|
                  |                   |
                  +----- BOY ---------+
                  +----- FULL --------+
                  +----- FULL1 -------+
                  +----- FULL2 -------+
                  +----- FULL3 -------+
                  +----- GIRL --------+
                  +----- LAST --------+
                  +----- TITLE -------+
                  +----- TITLE2 ------+
```

Pattern Value:

```
|-- PATTERN -- "pattern_string" ---+------------------------+-+---------+---|
                                   |                        | |         |
                                   |            +- SEQRL -+ | +- ECHO --+
                                   |            |         | |
                                   +- SEQuence -+---------+-+
                                   |            |         | |
                                   |            +- SEQLR -+ |
                                   |                        |
                                   +- BASE string ----------+
```

Index Values:

```
                                (2)                            (4)
    +- RANGE -------- 1 -------- * ----------------------------+
(1) |                                                          |
 |--+----------------------------------------------------------+-------------|
    |                                                          |
    +- RANGE ------ lo_num -+----------+-----+---------------+-+
    |                       |          |     |               | |
    |                       +- hi_num -+     +- BASE string -+ |
    |                     (2)                                  |
    |                                                          |
    |                           (2)                            |
    |            +- 1 ---------- * ------ +1 ----+             |
    |            |                               |             |
    +- SEQuence -+-------------------------------+-------------+
                 |                               |
                 +- lo_num -+------------------+-+
                            |                  |
                        (2) +- hi_num -+-------+
                                       |       |
                                   (3) +- inc -+
```

(1) An **Index Value** is a numeric index into a string of characters or list items.

(2) Default high number (*hi_num*) is "*", the number of random characters, or the number of items in the random value list.

(3) Negative integer increment (*inc*) values are supported. Default increment is: +1
The low number (*lo_num*) may be **higher** than the high number (*hi_num*), in which case the default increment is -1.

(4) Numbers *lo_num*, *hi_num* and *inc* are integer values.

RANDOMIZE *field_name* FOR *rec_type* **Randomize_Options**

Use the RANDOMIZE option to define test data generation options for any individual field(s) in your record layout structure. e.g.

```
'fileio' MyFile 'openwrite name MUSIC  using cobol USER123.SELCTRN.SAM1(ZZST1CPC)'

'fileio MUSIC    rand  TRACK-NUM  range 1 30                 '
'fileio MUSIC    rand  TRACK-ID   chars "0123456789ABCDEF"   '
'fileio MUSIC    rand  ARTIST     MAlphaNum                  '  /* Mixed case alpha or numeric */

'fileio MUSIC    option randomize                            '
'fileio MUSIC    writerepeat 10000'    /* Generate 10,000 records from scratch */
'fileio MUSIC    close'
```

If no mapping structure is applied then the whole record may be treated as a single field named **"Record"** or **"UnMapped"**.

As well as "random" numbers and character data, the RANDOMIZER may be used to:

◊ Generate a **sequence** of numbers based on a supplied increment/decrement value. e.g.

```
'fileio' MyFile 'openwrite name HR  using cobol PROD.COBOL.COPYBK(R789TH2D)'

'fileio HR  rand  SALARY seq 500   600.30   +0.05   '

        /* Will produce "500.00",    on 1st record
        |                "500.05",    on 2nd record
        |                "500.10" etc
        |         up to "600.30"     ... after which the sequence will repeat.
        */

'fileio HR  option randomize'
'fileio HR  writerepeat 10000'    /* Generate 10,000 records from scratch */
'fileio HR  close'
```

◊ Generate **date**/**time** fields either randomly or in sequence. e.g.

```
'fileio' MyFile 'openwrite name HR  using cobol PROD.COBOL.COPYBK(R789TH2D)'

'fileio HR  rand  START-DATE date "CCYY-MM-DD (DDD)"       range 1980/01/01 2030/12/31 '
                           /* e.g. "2002-10-27 (SUN)" */

'fileio HR  rand  LASTUPDATE date "Mmm DD CCYY hh:mm:ss"              ',
                                                 'seq  "2003/04/10 08:30" 2003/04/20 +5 SECS '

                           /* i.e. "Apr 10 2003 08:30:00"
                           |       "Apr 10 2003 08:30:05"
                           |       "Apr 10 2003 08:30:10"
                           |       "Apr 10 2003 08:30:15"
                           |        etc
                           */

'fileio HR  option randomize'
'fileio HR  writerepeat 10000'    /* Generate 10,000 records from scratch */
'fileio HR  close'
```

◊ Adjust existing numeric or date/time values using a supplied increment/decrement value. e.g.

```
'fileio' MyFile 'openupdate   name HR  using cobol PROD.COBOL.COPYBK(R789TH2D)'

'fileio HR  rand  NAME-POS    adjust +8  '

'fileio HR  rand  ORDER-DATE  date "CCYY-MM-DD (DDD)"     adjust -30 days '

'fileio HR  rand  SALARY      adjust +3.75 percent '

'fileio HR  option randomize'

do forever
  'fileio HR  read    MyRec'
  if rc <> 0 then leave

  'fileio HR  updatev MyRec'
end

'fileio HR  close'
```

◊ Perform a **calculation** based on current value(s) in this and/or other field(s). e.g.

```
'fileio' MyFile 'openupdate   name HR  using cobol PROD.COBOL.COPYBK(R789TH2D)'

'fileio HR  rand  BONUS       replacement(BONUS*2)'      /* Double it! */

'fileio HR  rand  GAS-RATE    rep(  (GAS-COST-0.2848) / GAS-KWHs  )'

'fileio HR  option randomize'
do forever
  'fileio HR  read    MyRec'
  if rc <> 0 then leave

  'fileio HR  updatev MyRec'
end
'fileio HR  close'
```

◊ Pick from a **list** of values, either randomly or in sequence. Any list may be supplied as a **separate file** or from **in-line** values. e.g.

```
'fileio' MyFile 'openwrite name TRACE  using cobol PROD.COBOL.COPYBK(R789TH2E)'

'fileio TRACE  rand  CALLER          list=MY.RAND.LIST(MODULES)'  /* Take list from a file */

'fileio TRACE  rand  FIRST-NAME  seq list( "George", "Paul", "Ringo", "John" )'

'fileio TRACE    option randomize                            '
'fileio TRACE    writerepeat 10000'    /* Generate 10,000 records from scratch */
'fileio TRACE    close'
```

◊ **Substitute** values by performing a **keyed lookup** into a supplied list. e.g.

```
'fileio' MyFile 'openwrite name HR  using cobol PROD.COBOL.COPYBK(R789TH2D)'

'fileio HR  rand  FIRST-NAME  KeyLocation(1,10) ValLocation(11,10)',
           '  /*  keyList MY.RAND.LIST(FIRSTNAME)  */             ',
           '      keyList(                                        ',
           '               "Annabel   Alison    "                 ',
           '               "Edward    David     "                 ',
           '               "Heidi     Etta      "                 ',
           '               "Jack      James     "                 ',
           '               "Laurence  John      "                 ',
           '               "Paul      Nicholas  "                 ',
           '               "Peter     Paul      "                 ',
           '               "Pasqual   Peter     "                 ',
           '               "Simon     Ricky     "                 ',
           '              )                                       '

'fileio HR  option randomize'
do forever
  'fileio HR  read    MyRec'
  if rc <> 0 then leave

  'fileio HR  updatev MyRec'
end
'fileio HR  close'
```

◊ Fill character fields using a supplied list of **"vocabulary"**. The selected words will be blank separated. Optionally the first character of the first word may be uppercased, and a period (full-stop) added to the end. e.g.

```
'fileio' MyFile 'openwrite name HR  using cobol PROD.COBOL.COPYBK(R789TH2D)'

'fileio HR    rand  NOTES-TXT  vocab    cap1  period             ',
            '   /*     List MY.RAND.LIST(WORDLIST)  */           ',
            '          List(                                     ',
            '                "a"     "an"    "the"   "and"       ',
            '                "have"  "that"  "for"   "you"       ',
            '                "with"  "say"   "this"  "they"      ',
            '                "but"   "his"   "from"  "not"       ',
            '                "ask"   "need"  "too"   "feel"      ',
            '                "three" "state" "never" "become"    ',
            '                "night" "high"  "real"  "each"      ',
            '                "most"  "other" "much"  "family"    ',
            '                                                    ',
            '                "a"     "an"    "the"   "and"       ',
            '                "a"     "an"    "the"   "and"       ',
            '                                                    ',
            '                "A complete sentance?"              ',
            '               )                                    '

'fileio HR  option randomize'
'fileio HR  writerepeat 10000'    /* Generate 10,000 records from scratch */
'fileio HR  close'
```

◊ Generate data according to a supplied **"pattern"** string. e.g.

```
'fileio' MyFile 'openwrite name STOCK  using cobol PROD.COBOL.COPYBK(R789TH2FF'

'fileio STOCK rand   PART-ID   pattern "A(J-N)#[-]#(1001-1999)[-]A(JGE,DJG,NBJ)"   '

    /* Example output "K5-1758-NBJ"
    |                 "J8-1044-JGE"
    |                 "M1-1346-DJG"
    */

'fileio STOCK  option randomize'
'fileio STOCK  writerepeat 10000'    /* Generate 10,000 records from scratch */
'fileio STOCK  close'
```

◊ To go beyond any limitations of the built-in "pattern" string syntax, multiple components may be manually **"chained"** together. e.g.

```
'fileio' MyFile 'openwrite name STOCK  using cobol PROD.COBOL.COPYBK(R789TH2FF'

'fileio STOCK rand   PART-ID            chars "JND"     len=1                       '
'fileio STOCK rand   PART-ID  chain     range 1,3       len=1                       '
'fileio STOCK rand   PART-ID  chain     lit   "-"                                   '
'fileio STOCK rand   PART-ID  chain     range 901,999   len=3 zeros sequence        '
'fileio STOCK rand   PART-ID  chain     lit   "-"                                   '
'fileio STOCK rand   PART-ID  chain     range 1001,1999 len=4                       '
'fileio STOCK rand   PART-ID  chain     lit   "-"                                   '
'fileio STOCK rand   PART-ID  chain     list ( "JGE", "DJG", "NBJ", "NGH", "CLS" )  '

  /* Example output "D3-901-1758-CLS"
  |                 "J1-902-1044-NBJ"
  |                 "N2-903-1346-DJG"
  */

'fileio STOCK  option randomize'
'fileio STOCK  writerepeat 10000'    /* Generate 10,000 records from scratch */
'fileio STOCK  close'
```

Apart from FILEIO, the following methods may also be used to generate record data, each of them including options to generate field values according to their respective randomizer objects:

◊ Online, while editing a dataset using the Data-Editor, by defining your test data generation options using the SET RANDOMIZER command, then entering the INSERT and/or REPLACELINE commands.

◊ In batch or online, the FSU Utility may be used to either **copy** or **update** an existing file, generating "randomized" test data for certain fields, while preserving existing values in others.

FOR [RECORD] *rec_type*

Identifies the record-type mapping in which the specified *field_name*/*field_ref* is defined.

Default is the default record type.

ADJUST [+|-]*inc*

ADJUST (or ADJ) causes the existing (numeric or date/time) value to be incremented or decremented by the supplied number *inc*. e.g.

```
rand DESC-LEN    adjust -15

rand HOURLY-RATE adj +1.38

rand EXPIRY-DATE adj 365
```

ADJUST *inc* PERCENT

The increment/decrement as described above is expressed as a percentage. e.g.

```
rand HOURLY-RATE adj +4.75 PERCENT

rand HOURLY-RATE adj +4.75 %    /* Valid with blank before "%" */

rand HOURLY-RATE adj +4.75%     /* Not valid */
```

ADJUST *inc* DAYS|HOURS|MINUTES|SECS

Specify **DAYS** or **SECS** to indicate the unit of increment/decrement for **date** and **time** fields. For **date** and **date+time** fields the default increment/decrement unit is **days**, and for **time** fields the default unit is **seconds**, so to adjust a **date+time** field by a number of seconds you must use the **SECS** keyword.

```
rand ORDER-DATE  date "CCYY-MM-DD (DDD)"     adjust -30 days

rand LAST-CHG    date "MM/DD/CCYY hh:mm:ss"  adjust +60 secs

rand DURATION    time                        adjust +3600    /* Add one hour */
```

ADJUST *inc* ... (*source_field*)

The increment/decrement may be applied to a value that exists in a different field, for example **EXPIRY-DATE** should be an adjustment based on the existing value in **START-DATE**. e.g.

```
rand EXPIRY-DATE date "CCYY-MM-DD"     adj 365 days ( START-DATE )
```

BASE *string*

Specify a fixed "base" in order get a **repeatable** set of results from the randomizer.

string is a character string of up to 8 bytes that is used to seed the random number generation algorithm.

If BASE is not explicitly coded then a default is generated using the current TOD clock value combined with the field's unique reference number.

For **ADJUST, KEY, REPLACEMENT** and **SEQUENCE** options, BASE specification is not relevant as the process does not involve generation of a random number at any stage.

```
rand GAS-KWHS    range 00,100  base "ABCDEFGH"

rand ELC-KWHS    range 22,150  base X'1234'
```

CAP1

The **CAP1** option will cause the first letter of generated character strings to be upper-cased.

```
rand FirstName   cap1 alpha

rand Descrip     cap1 vocab list(every,good,boy,deserves,favour) period
```

CHAIN

The **CHAIN** option may be used to produce a concatenated result from multiple randomizers. Typically this would be used to generate a combination of numbers, alpha and special characters in a data "pattern".

For most straight forward data patterns the **PATTERN** *"pattern_string"* feature is recommended instead of coding multiple chained randomizers for the same field.

```
rand NAME       list=("Mr ", "Mrs ", "Miss ")
rand NAME chain list=MY.FIRST.NAMES.LIST
rand NAME chain lit ' '
rand NAME chain alpha  len=1               /* Get middle initial */
rand NAME chain list=(" "," "," ",". ")    /* Get occasional "." */
rand NAME chain list=MY.LAST.NAMES.LIST
```

CHARS

The CHARS (CH) option indicates that a character string is to be generated, and is the default for all fields that are not of a numeric data-type.

An optional quoted string may follow to supply the array of characters from which a value may be selected (either at random or in sequence).

Alternatively, one of a number of shortcut keywords may be supplied e.g. ALPHA, to indicate the eligible list of characters.

If no keyword or quoted string is supplied, the following characters are used:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 +-=,./*()_!:@#$
```

CHARS *"chars_list"*

A quoted string supplying the list of characters eligible for selection. e.g.

```
rand ARITH-OP    chars "+-/*"

rand PREFIX      chars 'EHPRXehprx' sequence
```

CHARS *chars_keyword*

A keyword that implies a list of eligible characters.

Note that the **CHARS** keyword itself may be omitted if any of the following are supplied.

Keyword	*Description*	*Implied Character List*
ALPHA	Upper case Alpha only	ABCDEFGHIJKLMNOPQRSTUVWXYZ
ALPHANumeric	Upper case Alpha + Num	ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789
HEX	Hexadecimal digits	0123456789ABCDEF 0123456789
HEXEVEN	Even value Hexadecimal digits	02468ACE 0123456789
NUMeric	Numeric only	0123456789
LALPHA	Lower case Alpha only	abcdefghijklmnopqrstuvwxyz
LALPHANumeric	Lower case Alpha + Num	0123456789 abcdefghijklmnopqrstuvwxyz
LHEX	Lower case Hexadecimal digits	0123456789abcdef 0123456789
LHEXEVEN	Lower case even value Hexadecimal digits	02468ace 0123456789

MALPHA	Mixed case Alpha only	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
MALPHANumeric	Mixed case Alpha + Num	ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 abcdefghijklmnopqrstuvwxyz

e.g.

```
rand LASTNAME    chars lalpha cap1
rand PASSWORD    malphanum         /* CHARS is implied */
```

DATE

Use the **DATE** keyword to identify the named field as a date or date+time field.

If no date format string is supplied, and the field is not defined using one of FileKit's built-in date/time formats, then for character fields the default is **"CCYY/MM/DD hh:mm:ss.ttt"**. For fields defined with a numeric data-type (e.g. binary, packed decimal etc) the default format is **"CCYYMMDDhhmmssttt"**.

```
rand START-DATE  date   range 2001/01/01 2010/12/12
```

DATE *"str_ts"*

A character string that defines the format of the date/time (timestamp) field for which test data should be generated or adjusted.

The following (case-sensitive) format codes are supported, with all other characters treated as literals and transferred directly to the output.

Code	*Description*	*Examples*
CC	2-digit Century	"19" or "20"
CI	2-digit Century Indicator	"00" for 19xx "01" for 20xx
YYYY	4-digit Year (Same as coding "CCYY")	"1923"
YY	2-digit Year	"23"
MM	2-digit Month	"08"
MMM	3-char Month name in upper case	"JUL"
Mmm	3-char Month name in mixed case	"Dec"
DD	2-digit Day of the month	"31"
DDD	3-char Day name in upper case	"WED"
Ddd	3-char Day name in mixed case	"Sat"
JJJ	3-digit Julian Day of the year	"365"
WW	2-digit Week number (Monday start)	"52"
WWS	2-digit Week number (Sunday start)	"52"
hh	2-digit Hour of the day	"00" to "24"
mm	2-digit Minute of the hour	"00" to "59"
ss	2-digit Second of the minute	"00" to "59"
ttt	3-digit Thousandth of the second	"000" to "999"

e.g.

```
rand START-DATE  date "(Ddd)  DD-Mmm CCYY" /* e.g. "(Thu)  02-May 2002" */

rand PD_TSTMP    date "MMDDCCYYhhmmss"     /* e.g. For packed dec -> */
                                           /*        X'025122019154521C' */
                                           /*   meaning "15:45:21"      */
                                           /*        on "25th Dec 2019" */
```

DATE ... TODAY

Ensures that any date/time value produced will be for the current date.

Equivalent to coding **"RANGE '*yyyy/mm/dd* 00:00:00' '*yyyy/mm/dd* 23:59:59'"**, where *yyyy/mm/dd* is the current date.

DATE ... PAST

Ensures that any date/time value produced will be earlier than the current date/time.

Equivalent to coding **"RANGE '2001/01/01 00:00:00' '*yyyy/mm/dd hh:mm:ss*'"**, where *yyyy/mm/dd hh:mm:ss* is the current date/time.

`DATE ... PAST int DAYS|HOURS|MINUTES|SECS`

Ensures that any date/time value produced will be earlier than the current date/time, but no earlier than *int* number of **DAYS**/**HOURS**/**MINUTES**/**SECS**.

`DATE ... FUTURE`

Ensures that any date/time value produced will be later than the current date/time.

Equivalent to coding **"RANGE '*yyyy/mm/dd hh:mm:ss*' '2042/09/17 23:53:47' "**, where *yyyy/mm/dd hh:mm:ss* is the current date/time.

`DATE ... FUTURE int DAYS|HOURS|MINUTES|SECS`

Ensures that any date/time value produced will be later than the current date/time, but no later than *int* number of **DAYS**/**HOURS**/**MINUTES**/**SECS**.

`KEY (key_expression)`

When using **KEYLIST** to perform a translation via a keyed lookup, the default key is the value of the field for which you are defining the randomizer.

Use the **KEY** (*key_expression*) option if the key value should be derived from one or more different fields.

The *key_expression* must be a valid Data-Edit expression

```
rand FUNC keylist=MY.RAND.LIST(FUNCNAM1)
          KeyLoc(01,08) ValLoc(11,20)
          key=(MODULE)


rand FUNC keylist=MY.RAND.LIST(FUNCNAM2)
          KeyLoc(01,12) ValLoc(15,20)
          key=(  cat( MODULE,  '|',   right(strip(ext(PARM1),'L'),3,'0')  ))

                                  /* e.g. "SDEFSQX9|028  SQXColWidth" */
                                  /*      "SDEFSQX9|02C  SQXAutoSave" */
```

`KEYLIST`

Use **KEYLIST** to supply a list of keys and their corresponding substitution values in order to perform a translation via a keyed lookup.

The list may be supplied from **in-line** values or as a **separate file**.

Each line of the list should contain a value that will be referenced by keyed lookup and a corresponding substitution value, the position and length of which should be defined using the
KeyLoc(*pos,len*) and
ValLoc(*pos,len*) options.

The default key is the value of the field that you are defining the randomizer for, but the **KEY** *(key_expression)* option may be used if the key value should be derived from one or more different fields.

`KEYLIST (item [, ...])`

The list may be supplied in-line as a series of blank or comma separated strings enclosed in brackets.

Each string must be quoted if it contains blanks or special characters.

```
rand FIRST-NAME  KeyLocation(1,10) ValLocation(11,10)
                 keyList(

                       /*  From      To        */
                       /*  ----      --        */

                          "Annabel   Alison    "
                          "Edward    David     "
                          "Heidi     Etta      "
                          "Jack      James     "
                          "Laurence  John      "
                          "Paul      Nicholas  "
                          "Peter     Paul      "
                          "Pasqual   Peter     "
                          "Simon     Ricky     "
                        )
```

`KEYLIST list_file`

The list may be supplied as a separate file.

```
rand FIRST-NAME  KeyLocation(1,10) ValLocation(11,10)
                 keyList MY.RAND.LIST(FIRSTNAME)
```

`KEYLOCATION(pos [,len])`

Use **KEYLOC** to define the **position** and **length** within each list line of the key to be "looked up", in order to perform a value substitution.

The position and length of the substitution value should also be supplied using the **VALLOC** option.

```
rand FIRST-NAME  KeyLoc(1,10) ValLoc(11,10)  StripTrailing
                 keyList(

                       /*  Before    After   */
                       /*  ------    -----   */

                          "Annabel   Alison    "
                          "Edward    David     "
                          "Heidi     Etta      "
                          "Jack      James     "
                          "Laurence  John      "
                          "Paul      Nicholas  "
                          "Peter     Paul      "
                          "Pasqual   Peter     "
                          "Simon     Ricky     "
                        )
```

LALPHA

The LAPLPHA option indicates that a character string is to be generated, containing **lower-case alphabetic** ("a" to "z") characters only.

Add the **CAP1** option to generate a field that is all lower case character, but with the first character upper cased. e.g.

```
rand LASTNAME    lalpha cap1
```

LALPHANUMERIC

The LAPLPHANUMERIC (LALPHAN) option indicates that a character string is to be generated, containing **lower-case alphabetic** ("a" to "z") and **numeric** ("0" to "9") characters only.

Note that if the **SEQUENCE** option is added, then numerics ("0" to "9") will be produced ahead of the lower-case alpha ("a" to "z").

```
rand LASTNAME    lalphan
```

LENGTH *len*

When generating data for **character** fields, unless a limit is imposed by some other option, the default is to generate data to fill the **whole field**.

Specify **LENGTH *len*** to limit the amount of data generated e.g. to generate 16 bytes of data in 20-byte field.

LENGTH *len* is also commonly used with the **CHAIN** option when generating a pattern of data consisting of multiple components. e.g.

```
rand NAME       list=("Mr ", "Mrs ", "Miss ")
rand NAME chain list=MY.FIRST.NAMES.LIST
rand NAME chain lit ' '
rand NAME chain alpha  len=1                /* Get middle initial */
rand NAME chain list=(" "," "," "," ",". ")    /* Get occasional "." */
rand NAME chain list=MY.LAST.NAMES.LIST
```

LIST

Use **LIST** to supply a list of possible values to be generated.

The list may be supplied from **in-line** values or as a **separate file**.

Each line of the list should contain a value. If the whole line is not to be used you may supply the value's **position** and **length** using the
ValLoc(*pos,len*) option.

LIST (*item* [, ...])

The list may be supplied in-line as a series of blank or comma separated strings enclosed in brackets.

Each string must be quoted if it contains blanks or special characters.

```
rand FIRST-NAME  list( "James G. Evans"
                       "Nicholas B. Jones"
                       "Daniel Gribble"
                       "Laurence A. Cross"
                       "Douglas J. Hegarty"
                     )
```

LIST *list_file*

The list may be supplied as a separate file.

```
rand FIRST-NAME  list MY.RAND.LIST(FIRSTNAME) ValLoc(11,10)
```

LITERAL *"string"*

The generated value will be a fixed character or numeric literal.

For numeric fields the value will automatically be converted to the correct data-type (e.g. fixed point binary or packed decimal).

LIT *"string"* is also commonly used with the **CHAIN** option when generating a pattern of data consisting of multiple components. e.g.

```
rand PART-ID            chars "JND"     len=1
rand PART-ID  chain     range 1,3       len=1
rand PART-ID  chain     lit   "-"
rand PART-ID  chain     range 901,999   len=3 zeros sequence
rand PART-ID  chain     lit   "-"
rand PART-ID  chain     range 1001,1999 len=4
rand PART-ID  chain     lit   "-"
rand PART-ID  chain     list ( "JGE", "DJG", "NBJ", "NGH", "CLS" )

  /* Example output "D3-901-1758-CLS"
  |                 "J1-902-1044-NBJ"
  |                 "N2-903-1346-DJG"
  */
```

MALPHA

The MAPLPHA (mixed-case alpha) option indicates that a character string is to be generated, containing **both upper- and lower-case alphabetic** ("A to "Z" and "a" to "z") characters only.

```
rand VERY-WEAK-PASSWORD    malpha
```

MALPHANUMERIC

The MAPLPHANUMERIC (MALPHAN) option indicates that a character string is to be generated, containing **upper and lower-case alphabetic** ("a" to "z"), and **numeric** ("0" to "9") characters only.

Note that if the **SEQUENCE** option is added, then upper-case alpha ("A" to "Z") are produced first, followed by numerics ("0" to "9"), then lower-case alpha ("a" to "z").

```
rand WEAK-PASSWORD    malphan
```

NUMERIC

The NUMERIC (NUM) option indicates that a character string is to be generated, containing **numeric** ("0" to "9") characters only.

For fields defined with a numeric data-type (e.g. binary, packed decimal etc), coding the "NUM" keyword is unnecessary (just code "RANGE n1 n2").

```
rand PIN    num  len=4      /* PIN is a char field */
```

PATTERN *"pattern_string"*

Data may be generated according to a fixed pattern consisting of upper-/lower-case characters, numbers and literals.

"pattern_string"

Defines the layout of the data to be generated.

The following (case-sensitive) format codes are supported.

Code	*Description*	*Examples*
A	Any Upper-case Alpha (A-Z)	
A(*a1-a2*)	Upper-case Alpha in range *a1* to *a2*	A(P-V) = "PQRSTUV"

A(*a1,a2,a3...*)	List of (case-sensitive char) literals	A("J","N", "D")
		A("Jim","Nick", "Dan")

a	Any lower-case alpha (a-z)	
a(*a1-a2*)	Lower-case alpha in range *a1* to *a2*	a(p-v) = "pqrstuv"

a(*a1,a2,a3...*)	List of (case-sensitive char) literals	a("j","n", "d")
		a("Jim","Nick", "Dan")

# or **N**	Any numeric digit (0-9)	
#(*nnn1-nnn2*)	Any number in range *nnn1* to *nnn2*	#(101-200)

#(*n1,n2,n3...*)	List of (numeric) literals	a("1","3", "5")
		a("32,768","32.768","32768.00",)

[*literal*]	Any literal	[>>]
X	Upper-case HEX digits (0-F)	X(8-F) = "89ABCDEF"
x	Lower-case HEX digits (0-f)	x = "0123456789abcdef"
H	Upper-case even HEX digits (0-E)	H(4-C) = "468AC"
h	Lower-case even HEX digits (0-e)	h = "02468ace"

e.g.

```
rand PART-ID   pattern "A(J-N)#[-]a#(1001-1999)[-]A(JGE,DJG,NBJ)"

  /* Example output "K5-g1758-NBJ"
  |                 "J8-e1044-JGE"
  |                 "M1-j1346-DJG"
  */

rand FNAM      pat "[<=-= ]Aaaa[ ]A[. ]Aaaaaaa[ =-=>]"

  /* Example output "<=-= Kuhi R. Wohudiu =-=>"
  |                 "<=-= Ijyt W. Pytsltm =-=>"
  |                 "<=-= Vkth S. Hyewjjs =-=>"
  */

rand NAME  seq pat "A('Thomas', 'Tom', 'T.S.')[ Evans]"

  /* Output         "Thomas Evans"
  |                 "Tom Evans"
  |                 "T.S. Evans"
  */

rand PART  seq pat "A(A,T,X)[-]#(1050-1001)"  seqLR

  /* Output         "A-1050"
  |                 "T-1050"
  |                 "X-1050"
  |                 "A-1049"
  |                 "T-1049"
  |                 "X-1049"
  |                 "A-1048"
  |                 "T-1048"
  |                 etc
  */
```

"pattern_string" ECHO

Whenever a pattern string is used, FileKit turns each component into a separate RANDOMIZER, the results of which are concatenated using the **CHAIN** option described earlier.

The **ECHO** option causes FileKit to display the "chain" of generated "SET RANDOMIZER" commands on the message queue, instead of executing them.

The feature may be used as a helpful shortcut to coding the chain yourself, if and when limits of the pattern string are encountered.

e.g. Your pattern contains a sequence number that needs to be decremented by -5 instead of -1.

```
rand PART       chars "ATX"   seq  seqLR
rand PART chain lit "-"
rand PART chain len=4 zeros   seq 1050 1001 -5

  /* Output         "A-1050"
  |                 "T-1050"
  |                 "X-1050"
  |                 "A-1045"
  |                 "T-1045"
  |                 "X-1045"
  |                 "A-1040"
  |                 "T-1040"
  |                 etc
  */
```

PERCENT | %

Indicates that the increment/decrement value on an ADJUST (or ADJ) operation is expressed as a **percentage** e.g.

```
rand HOURLY-RATE adj +4.75 PERCENT

rand HOURLY-RATE adj +4.75 %    /* Valid with blank before "%" */

rand HOURLY-RATE adj +4.75%     /* Not valid */
```

PERIOD

Applicable to the **VOCAB** operation only, use **PERIOD** option to ensure a full-stop (".") is added at the end of the generated string.

If the generated string already ends in ".", "?", or "!' then a period will not be added.

```
rand DESC-TXT  vocab    cap1  period
             /*     List MY.RAND.LIST(WORDLIST)  */
                    List(
                          "a"     "an"    "the"   "and"
                          "have"  "that"  "for"   "you"
                          "with"  "say"   "this"  "they"
                          "but"   "his"   "from"  "not"
                          "ask"   "need"  "too"   "feel"
                          "three" "state" "never" "become"
                          "night" "high"  "real"  "each"
                          "most"  "other" "much"  "family"

                          "a"     "an"    "the"   "and"
                          "a"     "an"    "the"   "and"

                          "A complete sentance?"
```

PERSON

PERSON (PERS) provides options for generating the **name of a person**. e.g.

```
rand FIRST-NAME      person          /* e.g. "Jacob", "Emma"    etc */
rand FIRST-NAME      person(BOY)     /* e.g. "Jacob", "Michael" etc */
rand FIRST-NAME      person(LAST)    /* e.g. "Smith", "Johnson" etc */
rand CONTACT-NAME    person(FULL)    /* e.g. "Emma Smith"       etc *
rand CONTACT-NAME    person(FULL2)   /* e.g. "Mrs Erin Fields"  etc *
```

PERSON (*person_keyword*)

Keyword	*Description*	*Example*
ANY	First-name (Male/Female)	"Chloe"
BOY	First-name (Male)	"Mark"
FULL **FULL1**	First-name (Male/Female) **+** Last-name	"Mark Smith"
FULL2	Title (Male/Female) plus First-name (Male/Female) **+** Last-name	"Mrs Mark Smith" (can't guarantee compatibility!)
FULL3	Title (M/F - ext choice) **+** First-name (Male/Female) **+** Last-name	"Major General Mark Smith"
GIRL	First-name (Female)	"Chloe"
LAST	Last-name	"Smith"
TITLE **TITLE1**	Title (Male/Female)	"Miss"
TITLE2	Title (M/F - ext choice)	"Rear Admiral"

PERSON (***person_keyword*** **)** basically provides a shortcut to the product supplied list files **"*ProdHlq*.SZZSSAM2(ZZSPERxx)"**. e.g.

```
rand PERSON  name(FULL3)


  /* Above is equivalent to coding ... */

rand PERSON       list "ProdHlq.SZZSSAM2(ZZSPERT2)"  /* PERSON(TITLE2) */
rand PERSON chain literal " "
rand PERSON chain list "ProdHlq.SZZSSAM2(ZZSPERAN)"  /* PERSON(ANY)    */
rand PERSON chain literal " "
rand PERSON chain list "ProdHlq.SZZSSAM2(ZZSPERLA)"  /* PERSON(LAST)   */
```

RANGE *low_val high_val*

Defines the range of **numeric**, **date/time** or **time** values to be generated.

If *high_val* is omitted, then the default is the maximum value able to fit in the field (e.g. 32767 for a 2-byte signed binary field).

For date/time fields (although later dates may be generated) *high_val* defaults to **2042/09/17 23:53:47.37**, which is the highest timestamp supported by the standard TOD clock (STCK).

If **RANGE** is omitted altogether, then the default for *low_val* follows the same principle, except that date/time fields default to **2001/01/01 00:00:00.00**.

```
rand HOURLY-RATE range 8.51 1250

rand START-DATE  range 1980/01/01 2030/12/31  date "CCYY-MM-DD (DDD)"

rand END-TIME    range 15:00 17:30            time "hh-mm-ss"
```

REPLACEMENT (*replace_expression*)

Use the **REPLACEMENT** (or REP) option to generate a value that is calculated based on the existing value in the same and/or separate field(s).

The *replace_expression* must be a valid Data-Edit expression

```
rand BONUS       replacement(BONUS*2)     /* Double existing value!  */

rand PREV-UPD    replacement(LAST-UPD)    /* Just copy another field */

rand GAS-RATE    rep(  (GAS-COST-0.2848) / GAS-KWHs   ) /* Fancy calc */
```

SEQUENCE

Use the **SEQUENCE** (or SEQ) option to indicate that, instead of at random, values are to be generated in sequence, the next being a fixed increment/decrement on the previous.

The increment defaults to "+1". e.g.

```
rand REFNO seq                 /* A 4-byte signed binary field */

      /* Will produce "1"     on 1st record
      |                "2"     on 2nd record
      |                "3"     on 3rd record
      |                "4"     on 4rd record etc
      |
      */

rand MIDIN seq chars "PRS"

      /* Will produce "P"     on 1st record
      |                "R"     on 2nd record
      |                "S"     on 3rd record
      |                "P"     on 4rd record etc
      |
      */

rand CNAME seq list( "Jim", "Dan", "Nick" )

      /* Will produce "Jim"   on 1st record
      |                "Dan"   on 2nd record
      |                "Nick"  on 3rd record
      |                "Jim"   on 4rd record etc
      |
      */
```

SEQ *lo_num* [*hi_num* [*inc*]]

Defines the range of **numeric** values to be generated.
The default for *lo_num* is *1.*

The default for *hi_num* follows the same principle as decribed for the RANGE option.

If *lo_num* is higher than *hi_num* then the default increment value (*inc*) is **-1**, otherwise it is **+1**. e.g.

```
rand SALARY seq 500   600.30   +0.05

      /* Will produce "500.00"     on 1st record
      |             "500.05"     on 2nd record
      |             "500.10" etc
      |         up to "600.30"     ... after which the sequence will repeat.
      */
```

`SEQ lo_ts [hi_ts [inc] ] [ DAYS|HOURS|MINUTES|SECS ]`

Defines the range of **date** or **date/time** (timestamp) values to be generated.
The default for *lo_ts* is *1.*
The default for *hi_ts* follows the same principle as decribed for the RANGE option.

If *lo_ts* is higher than *hi_ts* then the default increment value (*inc*) is **-1**, otherwise it is **+1**. e.g.

Specify **DAYS, HOURS, MINUTES** or **SECS** to indicate the unit of increment/decrement for **date** and **date/time** (timestamp) fields. Default is DAYS. Therefore, to increment a **date+time** field by a number of seconds you must use the **SECS** keyword. e.g.

```
rand ORDER-DATE  date seq 2029/12/31 2010/01/01 -30   days

rand LAST-CHG    date seq 2023/06/13 2023/07/13 +3600 secs  /* + 1 hour */
```

`SEQ lo_ti [hi_ti [inc] ] [ HOURS|MINUTES|SECS ]`

Defines the range of **time** values to be generated.
The default for *lo_ti* is *1.*
The default for *hi_ti* follows the same principle as decribed for the RANGE option.

If *lo_ti* is higher than *hi_ti* then the default increment value (*inc*) is **-1**, otherwise it is **+1**. e.g.

Specify **HOURS, MINUTES** or **SECS** to indicate the unit of increment/decrement for **time** fields. Default is SECS.

```
rand DURATION    time seq 10:00      16:00      +30               /* +30 secs */
```

`SEQLR | SEQRL`

If a field value is generated from multiple concatenated components (either via a **PATTERN *"pattern_string"*** or using the **CHAIN** option described earlier) then the combination may well involve several **"value sequences"**.

A "sequence" is typically an incrementing/decrementing number, but could just as easily be a single character selected from an array, or a character string selected from a list.

Rather than produce a new sequential value for every component at once, it's often useful to treat the whole thing as as a combined sequence. This is a way of guaranteeing that you produce a sample of **every possible combination**.

The SeqLR (**Sequence Left to Right**) and SeqRL (**Sequence Right to Left**) options activate this feature.

The following examples illustrate the feature

◊ The 1st example combines 3 fully **independent** sequence values.
◊ The 2nd example combines 3 sequence values, sequenced **right-left**.
◊ The 3nd example combines 3 sequence values, sequenced **left-right**.

```
rand PART  seq pattern "A(A,T)[-]#(101-103)[-]#(501-503)"

  /* Output         "A-101-501"
  |                 "T-102-502"
  |                 "A-103-503"
  |                 "T-101-501"
  |                 "A-102-502"
  |                 "T-103-503"
  |                 ... then series repeats
  */


rand PART  seq pattern "A(A,T)[-]#(101-103)[-]#(501-503)"  seqRL

  /* Output         "A-101-501"
  |                 "A-101-502"
  |                 "A-101-503"
  |                 "A-102-501"
  |                 "A-102-502"
  |                 "A-102-503"
  |                 "A-103-501"
  |                 "A-103-502"
  |                 "A-103-503"
  |                 "T-101-501"
  |                 "T-101-502"
  |                 "T-101-503"
  |                 "T-102-501"
  |                 "T-102-502"
  |                 "T-102-503"
  |                 "T-103-501"
  |                 "T-103-502"
  |                 "T-103-503"
  |                 ... then series repeats
  */


rand PART  seq pattern "A(A,T)[-]#(101-103)[-]#(501-503)"  seqLR

  /* Output         "A-101-501"
  |                 "T-101-501"
  |                 "A-102-501"
  |                 "T-102-501"
  |                 "A-103-501"
  |                 "T-103-501"
  |                 "A-101-502"
  |                 "T-101-502"
  |                 "A-102-502"
  |                 "T-102-502"
  |                 "A-103-502"
  |                 "T-103-502"
  |                 "A-101-503"
  |                 "T-101-503"
  |                 "A-102-503"
  |                 "T-102-503"
  |                 "A-103-503"
  |                 "T-103-503"
  |                 ... then series repeats
  */
```

STRIPboth | STRIPLeading | STRIPTrailing

Strips leading (STRIPL) blanks, trailing (STRIPT) blanks or both (STRIP) from the generated value.

```
rand DIAL-CODE  list MY.DIAL.CODES.F80   stript  /* File is RECFM=F L=80 */
```

TIME

Use the **TIME** keyword to identify the named field as a time field.

If no time format string is supplied, and the field is not defined using one of FileKit's built-in time formats, then for character fields the default is **"hh:mm:ss.ttt"**. For fields defined with a numeric data-type (e.g. binary, packed decimal etc) the default format is **"hhmmsstt"**.

```
rand START-TIME  time   range 03:30 07:30
```

TIME *"str_ti"*

A character string that defines the format of the time field for which test data should be generated or adjusted.

The following (case-sensitive) format codes are supported, with all other characters treated as literals and transferred directly to the output.

Code	*Description*	*Examples*
hh	2-digit Hour of the day	"00" to "24"
mm	2-digit Minute of the hour	"00" to "59"
ss	2-digit Second of the minute	"00" to "59"
ttt	3-digit Thousandth of the second	"000" to "999"

e.g.

```
rand START-TIME  date "hh-mm-ss.ttt" /* e.g. "13-45-04.782" */
```

TIME ... PAST

Ensures that any time value produced will be earlier than the current time.

Equivalent to coding **"RANGE '00:00:00' '*hh:mm:ss*'"**, where *hh:mm:ss* is the current date/time.

TIME ... PAST *int* HOURS|MINUTES|SECS

Ensures that any time value produced will be earlier than the current time, but no earlier than *int* number of **HOURS/MINUTES/SECS**.

TIME ... FUTURE

Ensures that any time value produced will be later than the current time.

Equivalent to coding **"RANGE '*hh:mm:ss*' '23:53:47' "**, where *hh:mm:ss* is the current time.

TIME ... FUTURE *int* HOURS|MINUTES|SECS

Ensures that any time value produced will be later than the current time, but no later than *int* number of **HOURS/MINUTES/SECS**.

VALLOCATION(*pos* [,*len*])

With the option to pick list lines at random, in sequence or via keyed lookup, use **VALLOC** to define the **position** and **length** within each list line of the value to be generated.

If a keyed lookup is required then the position and length of the key should also be supplied using the **KEYLOC** option.

```
rand FIRST-NAME  KeyLoc(1,10) ValLoc(11,10)  StripTrailing
                 keyList(

                       /*  Before    After   */
                       /*  ------    -----   */

                          "Annabel   Alison    "
                          "Edward    David     "
                          "Heidi     Etta      "
                          "Jack      James     "
                          "Laurence  John      "
                          "Paul      Nicholas  "
                          "Peter     Paul      "
                          "Pasqual   Peter     "
                          "Simon     Ricky     "
                        )
```

VOCAB | VOC

Use **VOCAB** to supply a list of words or phrases that will be used to fill a character field.

Using the **LIST** option the list may be supplied from **in-line** values or as a **separate file**. You may omit the list altogether in order to use the product's default built-in vocabulary list that is supplied in file ***%SitePfx%*.SZZSSAM2(ZZSVOCAB)**.

Items will be repeatedly selected from the list, and concatenated with an intervening blank, to build up a **"sentence"**. The process ends when the next selected word won't fit in the remaining space.

To get realistic looking sentences you may wish to improve the chance that commonly used words, such as **"a", "an", "the", "and"** etc, have of being selected, by including them in the vocabulary list multiple times.

Use the **CAP1** option to cause the first character of the first word to be **uppercased**.

Use the **PERIOD** option to cause a period (full-stop) to be added to the end of the sentence. e.g.

For variable length fields (e.g. VARCHAR) the length allocated to build each sentence will randomly vary according to the field's defined min/maximum length.

For short (len<=20) fixed length fields (e.g. CHAR) the length allocated to build each sentence is fixed.

For longer (len>20) fixed length fields (e.g. CHAR) the length allocated to build each sentence will also randomly vary from 20 up to the length of the field.

Your vocabulary list may include some case-sensitive **special codes**:

Special Code	*Description*	*Example*
@l?	Abutt "?" to next word (no intervening blank)	Use "@l(" to start a "(xxx ...)" fragment
@L?	Abutt "?" to next word (no intervening blank) and upper-case 1st char of next word	Use "@L(" to start a "(Xxx ...)" fragment
@t?	Abutt "?" to previous word (no intervening blank)	Use "@t)" to end a "(xxx ...)" fragment.
@T?	Abutt "?" to previous word (no intervening blank) and upper-case 1st char of next word	Use "@t." to end a "xxx." fragment.

`VOCAB LIST(`*`"item" [, ...]`*`)`

The list may be supplied in-line as a series of blank or comma separated strings enclosed in brackets.

Each string must be quoted if it contains blanks or special characters.

```
                        /* Note the "LIST" keyword may be omitted */
rand FIRST-NAME  vocab list( "a"
                             "an"
                             "the"
                             "have"
                             "that"
                           )
```

`VOCAB LIST` *`list_file`*

The list may be supplied as a separate file.

```
rand FIRST-NAME  vocab list MY.RAND.LIST(VOCAB1)
rand FIRST-NAME  vocab      MY.RAND.LIST(VOCAB2) /* "LIST" may be omitted */
```

`ZEROS`

Causes numeric values generated for character fields to have leading zeros instead of leading blanks.

```
rand DIAL-CODE  seq len=4  range 1 1000 +10

  /* Output         "   1"
  |                 "  11"
  |                 "  21"
  |                 etc
  */

rand DIAL-CODE  seq len=4  range 1 1000 +10 zeros

  /* Output         "0001"
  |                 "0011"
  |                 "0021"
  |                 etc
  */
```

Miscellaneous_Operations:

```
|------ CMD ---- command_text ---------------------------------------------|
```

CMD *command_text*

In theory the CMD operation may be used to execute any Data-Edit primary command, although practically speaking, only a few will actually be of any benefit.

Some examples:

```
"fileio MYFILE cmd ascii on CUST_NAME"
```

Sets the ASCII translation option for the field named **CUST_NAME**.
See also: Option ASCII

```
"fileio MYFILE cmd bwz on NUM_OF_CHILDREN for CUST_DETAIL"
```

Sets the "blank when zero" option for the field named **NUM_OF_CHILDREN** in the record type named **CUST_DETAIL**.
See also: Option BLANKWHENZERO

```
"fileio MYFILE cmd zeros on X_COORD for DIMS"
```

Sets the "leading zeros" option for the field named **X_COORD** in the record type named **DIMS**.
See also: Option ZEROS

```
"fileio MY.GDG(-5) open name MYGDG"
"fileio MYGDG cmd extract /dsn/"
```

Extracts the actual GDS dataset name (MY.GDG.GnnnnVnn) to REXX variable DSN.1

Options:

```
                  +------ , --------+
                  v                 |
|--+-- OPTions ---+- ALIAS ---------+----------+----------------------------|
   |              +- ASCII ---------+          |
   |              +- BLANKWHENZERO -+          |
   |              +- CREATED -------+          |
   |              +- CURSIZE -------+          |
   |              +- DSORG   -------+          |
   |              +- DIRectory -----+          |
   |              +- GENeration ----+          |
   |              +- IOP -----------+          |
   |              +- NOIOP ---------+          |
   |              +- KEY -----------+          |
   |              +- KEYLEN --------+          |
   |              +- KEYPOS --------+          |
   |              +- LASTMOD -------+          |
   |              +- LIBNAME -------+          |
   |              +- MEMBER --------+          |
   |              +- MEMLIST -------+          |
   |              +- RANDomize -----+          |
   |              +- RECID ---------+          |
   |              +- RECno ---------+          |
   |              +- RECTYPe -------+          |
   |              +- RESET ---------+          |
   |              +- SEGTYPe -------+          |
   |              +- SPEED ---------+          |
   |              +- USER  ---------+          |
   |              +- ZEROS ---------+          |
   |                                           |
   |              +------ , --------+          |
   |              v                 |          |
   +-- MODRPL ----+- BWD -----------+----------+
                  +- FWD -----------+
                  +- FULLKEY -------+
                  +- GENKEY --------+
                  +- KEYEQ ---------+
                  +- KEYGE ---------+
                  +- LASTREC -------+
                  +- XRBA ----------+
```

`ALIAS`

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the ALIAS option may be used to determine if the **current member** is an **alias of another member**, and if so, the name of that member.

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.ALIAS**	Set to the name of the aliased member or NULL if the current member is not an alias.

Batch JCL Example:

```
//FIOALIAS  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SDEIN     DD *
  macro FIOTMPDD SYS1.MACLIB

//FIOTMPDD  DD *
 address "CBLSDATA" /* REXX */

 arg Lib
 Lib=strip(Lib)   /* e.g. Lib='SYS1.MACLIB' */


 'fileio' Lib 'open    pds   name MYLIB'
 'fileio MYLIB options member alias   noiop'

 MemberCount=0
 AliasCount=0
 do forever
   'fileio MYLIB read -'    ; if rc <> 0  then leave
   MemberCount=MemberCount+1

   if MYLIB.1.alias <> '' then
     do; AliasCount=AliasCount+1
         say ' 'MYLIB.1.member 'is an alias of:' MYLIB.1.alias
     end

   'fileio MYLIB nextmember'; if rc <> 0  then leave
 end

 say ' --------------------------------------------------------'
 say ' 'Lib 'has' AliasCount 'aliases out of' MemberCOunt 'members'
 say ' --------------------------------------------------------'

 'fileio MYLIB close'

/* Sample Output:
/* IKJEFLRT is an alias of: IKJEBERT
/* IKJEFLSV is an alias of: IKJEBESV
/* IWM4EBLK is an alias of: IWMEBLK
/* IWM4ESTO is an alias of: IWMESTOP
/* IWM4ESTR is an alias of: IWMESTRT
/* IWM4EUBL is an alias of: IWMEUBLK
/* IWM4GCOF is an alias of: IWMGCORF
/* IWM4SCOF is an alias of: IWMSCORF
/* PICTURE  is an alias of: ERBPICT
/* --------------------------------------------------------
/* SYS1.MACLIB has 9 aliases out of 2005 members
/* --------------------------------------------------------
```

`ASCII`

If the OPEN option USING *copybook_or_mapping_file* **has not** been specified, then this option will trigger:

◊ ASCII to EBCDIC translation for the whole record on **input**
◊ EBCDIC to ASCII translation for the whole record on **output**

If the OPEN option USING *copybook_or_mapping_file* **has** been specified, then this option will trigger ASCII to EBCDIC translation for the extracted value (see FVALUE/FVALUEQP/FVALUEQF) of any **character** (AN) field.

If more selective ASCII translation of individual fields is required then the CMD ASCII *field_name* operation maybe used instead. e.g.

```
  "fileio CUSTFILE  cmd ascii on CUST_NAME"
```

`BLANKWHENZERO`

Used in conjunction with the OPEN option USING *copybook_or_mapping_file*, this option will cause the extracted value (see FVALUE/FVALUEQP/FVALUEQF) of any **numeric** field with a **value of zero** to return a blank value instead.

The following are all synonyms/abbreviations for BLANKWHENZERO.

◊ BWZ
◊ BLANKIFZERO
◊ BIZ

If BLANKWHENZERO needs to be set more selectively for individual fields then the CMD BLANKWHENZERO *field_name* operation maybe used instead. e.g.

```
  "fileio CUSTFILE  cmd BlankWhenZero on NUMBER_OF_CHILDREN"
```

`CREATED`

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the CREATED option may be used to determine the **first created date** of the **current member**.

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.CREATED**	Set to the created date in **yyyy/mm/dd** format. Value is set to **null** if the member's directory entry does not contain this information.

Related OPTIONS: LASTMOD

Batch JCL Example:

```
//FIODATES  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//MYJCL     DD DISP=SHR,DSN=MY.BATCH.JCLLIB
//SDEIN     DD *
  macro FIOSAMP

//FIOSAMP   DD *
address "CBLSDATA" /* REXX */

'fileio MYJCL open    library'
'fileio MYJCL options member created lastmod recno noiop'

 do forever
   'fileio MYJCL read -'; if rc >= 4  then leave

    if MYJCL.1.created >  '2015/06/01' ,
     & MYJCL.1.lastmod <= '2018/06/01'   then
      do
          'fileio MYJCL member' MYJCL.1.member /* Need rec 1 again */

          'fileio MYJCL where (record  << "PGM="          ',
                           '&  record  << "REGION="       ',
                           '&  record \<< "REGION=0M"   )'
          'fileio MYJCL sread MYREC'
          if MYREC.0 > 0 then
            do; say ' '
                say ' 'MYJCL.1.member MYJCL.1.created MYJCL.1.lastmod
                say ' ------------------------------------'

                do i = 1 to MYREC.0
                  say ' 'right(MYJCL.i.recno,5,'0') MYREC.i
                end
            end
          'fileio MYJCL where ()'  /* Reset the WHERE */
      end

   'fileio MYJCL nextmember' ; if rc <> 0  then leave
 end

'fileio MYJCL close'

/* Sample Output:
/*
/* ARCSTRST 2016/12/14 2016/12/14 10:37
/* ------------------------------------
/* 00367 //DFSMSHSM  EXEC PGM=ARCCTL,DYNAMNBR=&DDD,REGION=&SIZE,TIME=1440,       11693800
/* 02217 //STEP1 EXEC PGM=IDCAMS,REGION=512K                                     65225600
/* 02275 //IDCAMS EXEC PGM=IDCAMS,REGION=512K                                    66704000
/*
/* IMSPSBG1 2016/09/22 2018/04/12 11:39
/* ------------------------------------
/* 00010 //C      EXEC PGM=ASMA90,REGION=&RGN,
/*
/* IQ004645 2016/05/11 2018/04/13 15:44
/* ------------------------------------
/* 00043 //G      EXEC PGM=DFSRRC00,REGION=&RGN,
```

CURSIZE

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the CURSIZE option may be used to determine the **number of records** in the **current member**.

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.CURSIZE**	Set to the number of records in the current member according to its directory entry. Value is set to **null** if the member's directory entry does not contain this information.

DSORG

For both input and output files, the DSORG option may be used to determine its **dataset organisation**.

The following REXX variable(s) are set:

***<filename>*.1.DSORG**	Possible values are "PDS", "SEQ", "HFS", "KSDS", "ESDS", "RRDS" and "LDS".

DIRECTORY | DIR

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the DIRECTORY (min abbrev DIR) option may be used to extract the **complete directory entry** for the **current member**.

FILEIO also supports separate options to extract many of most commonly required details from the directory, e.g. ALIAS, CREATED, LASTMOD, GENERATION, MEMBER and USER.

If the information you require is known to be in the directory, but is not one of the above options, then you may extract it yourself from the raw directory entry (provided you know the directory layout).

The directory entry is most commonly maintained by either:

◊ The **ISPF Editor** for text type files.
◊ The **Linkage Editor (Binder**) for program objects (load modules).

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.DIR**	Set to the **complete raw directory entry** for the current member.

GENERATION | GEN

For input only, when reading multiple members of a **PDSE version 2 library** using open option LIBRARY the GENERATION (min abbrev GEN) option may be used to extract the **member generation number** for the **current member**.

Generation number is only available when a **member generation mask** is explicitly coded on the OPEN operation. See the LIBRARY open option for more information. e.g.

To select all member generations whose member name begins with *CALL* and whose absolute generation number is greater than or equal to *12*:

```
MY.JCLLIB(CALL*.>=12)
```

To select all generations except the base (i.e. relative generation 0):

```
MY.JCLLIB(NBJX.<0)
```

To select all member generations whose member name begins with *SS* and whose relative generation number is *-1*:

```
MY.JCLLIB(SS*.-1)
```

To select all member generations whose entries match one of the member and generation masks:

```
MY.JCLLIB(SS*.>=-3,ADA%%%%.>=-5,CBL*.0)
```

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.GEN**	Set to the **relative generation number** followed by the **absolute generation number**. e.g. "-3 4079"

Related OPTIONS: LASTMOD MEMBER

IOP | NOIOP

If FILEIO is called from a REXX macro running within an online FileKit session (TSO/ISPF foreground) then option IOP may be used to display an **"I/O Progress" window** for *filename*,

As well as communicating progress, in terms of records read or written, its primary purpose is to give the user an oppunity to **interrupt the process** by pressing the **"Attention" key** (once the keyboard has been unlocked).

The **IOP** and **NOIOP** options allow the user to switch this feature on and off as desired. For example, while testing it's good idea to switch on the IOP, but if your process uses REXX's **"say"** feature to display information on your TSO terminal, then having FILEIO IOP causing a screen refresh every second or so would be very annoying.

For input files, the IOP window is displayed at the top right of the screen.

For output files, the IOP window is displayed halfway down on right of the screen.

Once the **"Attention" key** interrupt has been registered all input and output operations for *filename* will set **return code 4** (RC=4).

IOP may also be specified directly as an option on the OPENREAD/OPENWRITE/OPENUPDATE/OPENINSERT operation.

KEYLEN | KEYPOS

For **VSAM KSDS** data sets only, the KEYPOS (KEYLEN is a synonym) option may be used to determine its **key position and length**.

The following REXX variable(s) are set:

***<filename>*.1.KEYPOS**	Set to the **KSDS key position**.
***<filename>*.1.KEYLEN**	Set to the **KSDS key length**.

Note that **both** REXX variable(s) are set regardless of whether you specify option **KEYLEN** or **KEYPOS**.

KEY

For **VSAM KSDS** input data sets only, the KEY option may be used to extract the **key value** of the current input record.

Each time an input record is **selected** the following REXX variable(s) are set:

***<filename>*.1.KEY**	Set to the raw **KSDS key** of the current record. For the *SREAD* (stem read) operation, which inputs multiple records, REXX variable ***<filename>*.2.KEY** is set for the 2nd record, and so on.

LASTMOD

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the LASTMOD option may be used to determine the **last modified timestamp** of the **current member**.

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.LASTMOD**	Set to the last modified timestamp in **yyyy/mm/dd hh:mm** format. Value is set to **null** if the member's directory entry does not contain this information.

Related OPTIONS: CREATED

Batch JCL Example:

```
//FIODATES  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//MYJCL     DD DISP=SHR,DSN=MY.BATCH.JCLLIB
//SDEIN     DD *
  macro FIOSAMP

//FIOSAMP   DD *
  address "CBLSDATA" /* REXX */

  'fileio MYJCL open    library'
  'fileio MYJCL options member created lastmod recno noiop'

   do forever
     'fileio MYJCL read -'; if rc >= 4  then leave

      if MYJCL.1.created >  '2015/06/01' ,
       & MYJCL.1.lastmod <= '2018/06/01'   then
        do
            'fileio MYJCL member' MYJCL.1.member /* Need rec 1 again */

            'fileio MYJCL where (record  << "PGM="         ',
                             '&  record  << "REGION="      ',
                             '&  record \<< "REGION=0M"   )'
            'fileio MYJCL sread MYREC'
            if MYREC.0 > 0 then
              do; say ' '
                  say ' 'MYJCL.1.member MYJCL.1.created MYJCL.1.lastmod
                  say ' -------------------------------------'

                  do i = 1 to MYREC.0
                    say ' 'right(MYJCL.i.recno,5,'0') MYREC.i
                  end
              end
            'fileio MYJCL where ()'  /* Reset the WHERE */
        end

     'fileio MYJCL nextmember' ; if rc <> 0  then leave
   end
  'fileio MYJCL close'

/* Sample Output:
/*
/* ARCSTRST 2016/12/14 2016/12/14 10:37
/* -------------------------------------
/* 00367 //DFSMSHSM  EXEC PGM=ARCCTL,DYNAMNBR=&DDD,REGION=&SIZE,TIME=1440,       11693800
/* 02217 //STEP1 EXEC PGM=IDCAMS,REGION=512K                                     65225600
/* 02275 //IDCAMS EXEC PGM=IDCAMS,REGION=512K                                    66704000
/*
/* IMSPSBG1 2016/09/22 2018/04/12 11:39
/* -------------------------------------
/* 00010 //C      EXEC PGM=ASMA90,REGION=&RGN,
/*
/* IQ004645 2016/05/11 2018/04/13 15:44
/* -------------------------------------
/* 00043 //G      EXEC PGM=DFSRRC00,REGION=&RGN,
```

LIBNAME

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory (where the library name is likely to be different for each member), the LIBNAME option may be used to determine the **library dataset name** that the **current member** was read from.

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.LIBNAME**	Set to the library dataset name for the current member.

Related OPTIONS: GENERATION MEMBER

Batch JCL Example:

```
//PROCUSER  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSPROC   DD DISP=SHR,DSN=USER.Z23A.CLIST
//          DD DISP=SHR,DSN=FEU.Z23A.CLIST
//          DD DISP=SHR,DSN=ADCD.Z23A.CLIST
//          DD DISP=SHR,DSN=ISP.SISPCLIB
//          DD DISP=SHR,DSN=SYS1.DGTCLIB
//          DD DISP=SHR,DSN=SYS1.HRFCLST
//          DD DISP=SHR,DSN=ICQ.ICQCCLIB
//          DD DISP=SHR,DSN=SYS1.SBLSCLI0
//          DD DISP=SHR,DSN=SYS1.SBPXEXEC
//          DD DISP=SHR,DSN=SYS1.SCBDCLST
//          DD DISP=SHR,DSN=SYS1.SEDGEXE1
//          DD DISP=SHR,DSN=GIM.SGIMCLS0
//          DD DISP=SHR,DSN=SYS1.SERBCLS
//          DD DISP=SHR,DSN=CSQ800.SCSQCLST
//          DD DISP=SHR,DSN=CSQ901.SCSQCLST
//          DD DISP=SHR,DSN=EOY.SEOYCLIB
//          DD DISP=SHR,DSN=ADBC10.SADBEXEC
//          DD DISP=SHR,DSN=AUT350.SINGIREX
//SDEIN     DD *
  macro FIOSAMP

//FIOSAMP   DD *
  address "CBLSDATA" /* REXX */
       /* Process SYSPROC as a library search path, printing
       |  first (up to) 30 records for all members that have a
       |  non-blank modified by user-id, skipping all aliases.
       */

  'fileio SYSPROC open  CLD'; if rc <> 0 then exit rc
  'fileio SYSPROC options libname member lastmod user alias'

   do forever
     'fileio SYSPROC sread MYREC 30'; if rc >= 4  then leave

     if SYSPROC.1.alias = '' & SYSPROC.1.user <> '' then
       do; say ' 'left(strip(    SYSPROC.1.libname),
                 || '('strip(    SYSPROC.1.member) ,
                 || ')',54)                        ,
                  'USER='    || SYSPROC.1.user     ,
                  'LASTMOD=' || SYSPROC.1.lastmod
           say left(' -',94,'-')
           say ' '

           do i = 1 to MYREC.0
             say right(i,5,'0') MYREC.i
           end
           say ' '; say ' '; say ' '
           say ' '; say ' '; say ' '
       end


     'fileio SYSPROC nextmember' ; if rc <> 0  then leave
   end

  'fileio SYSPROC close'
/*
/* Sample output:
/*
/* USER.Z23A.PROCLIB(DJHDISPF)                            USER=IBMUSER  LASTMOD=2018/03/29 15:22
/* ---------------------------------------------------------------------------------------------
/*
/* 00001 //* USER.PROCLIB(DJHDISPF) *-*   L=021 --- 2016/03/16 16:02:04  (NBJ)
/* 00002 //*----------------------------------------------------------------
/* 00003 //*
/* 00004 //*                 ISPF FULL-FUNCTION LOGON PROC
/* 00005 //*
/* 00006 //*
/* 00007 //*
/* 00008 //*  L=022 2018/03/29 -nbj- z/OS 2.3.
/* 00009 //*  L=021 2016/03/16 -nbj- Replace DSN910 with DSNB10 libs.
/* 00010 //*  L=019 2014/10/12 -nbj- z/OS 2.1 libraries. Based on DBSPROCA.
```

MEMBER

Typically used for input only, when processing multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the MEMBER operation may be used to switch directly to the current member.

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.MEMBER**	Set to the member name for the current member.

Related OPTIONS: GENERATION LIBNAME

Batch JCL Example:

```
//FIOASM    EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//ASM       DD DISP=SHR,DSN=CBL.CBLI360.ASM
//          DD DISP=SHR,DSN=CBL.CBLI350.ASM
//          DD DISP=SHR,DSN=CBL.CBLI340.ASM
//          DD DISP=SHR,DSN=CBL.CBLI330.ASM
//          DD DISP=SHR,DSN=CBL.CBLI320.ASM
//          DD DISP=SHR,DSN=CBL.CBLI310.ASM
//SDEIN     DD *
  macro FIOSAMP

//FIOSAMP   DD *
  address "CBLSDATA" /* REXX */

  'fileio ASM     open  CLD'; if rc <> 0 then exit rc
  'fileio ASM     options libname member recno'

  'fileio ASM     filter                     ',
                  '(include record           ',
                   'where                    ',
                   '(   record << "ISTDNIB"  ',
                   ' or record << "IKJTCB"   ',
                   ' or record << "IEFTIOT1" ',
                   ' or record << "IHADECB"  ',
                   ' or record << "IEZDEB"   ',
                   ' or record << "IEZIOB"   ',
                   ')                        ',
                  ')                         '

   do forever
     'fileio ASM read MYREC';
     if rc >= 4      then leave
     if rc  = 2      then iterate


     say ' 'strip(ASM.1.libname),
           || '('strip(ASM.1.member) ,
           || ')'                  ,
                 right(ASM.1.recno,5,'0') MYREC

     'fileio ASM nextmember';
     if rc <> 0      then leave
   end

  'fileio ASM close'
/*
/* CBL.CBLI360.ASM(APEEINIT) 01590  ISTDNIB             ,      VTAM NIB.
/* CBL.CBLI360.ASM(CBLATRAC) 04249  ISTDNIB             ,      VTAM NIB.                          IQ004843
/* CBL.CBLI340.ASM(CBLVIMSS) 00535  IKJTCB  LIST=YES           Include TCB DSECT.
/* CBL.CBLI310.ASM(DSN3SATH) 00253 *    MVS         TCB           IKJTCB        TASK CONTROL BLOCK       * 0
/* CBL.CBLI350.ASM(EDTFMAC1) 01388          IEFTIOT1           Include TIOT DSECT                 IQ004573
/* CBL.CBLI360.ASM(IOSFFIO4) 01028  IEZDEB LIST=YES            MVS Data Extent Block.             IQ003759
```

MEMLIST

Typically used for input only when processing multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory MEMBER operation may be used to switch directly to the current member. Immediately after open, the MEMLIST option may be used to access the list of members that will be processed, and for each member its:

◊ Library and member name
◊ Last modified timestamp
◊ Current size (number of records)
◊ Last modification user name

For example:

```
CBL.INST.CBL21042.SZZSDIST.TLIB(ZZSDB2SS) 2020/04/23 15:43          7 JGE
CBL.INST.CBL21042.SZZSDIST.TLIB(ZZSIMSSL) 2017/06/14 17:34          9 NBJ
CBL.INST.CBL21042.SZZSDIST.TLIB(ZZSIMSSS) 2017/08/29 16:58        107 JGE
```

As soon as the MEMLIST option is requested the following REXX variable(s) are set:

***<filename>*.1.MEMLIST.0**	Set to the number of members (*n*) about to be processed.
<filename>*.1.MEMLIST.*n	Set to **5** blank delimited tokens detailing the ***n*th member**: 1. The library name and bracketed member name (plus generation number) 2. The member's last modified date in yyyy/mm/dd format 3. The member's last modified time in hh:mm format 4. The member's size (number of records) 5. The name of the user that last modified the member

Please note that last modified timestamp, size and user name are not necessarily maintained in the library's directory record, and where a detail is unavailable a place holding dot (".") will be substituted.

Related OPTIONS: CURSIZE GENERATION LASTMOD LIBNAME MEMBER USER

```
address "CBLSDATA" /* REXX */
  /* Display 1st 5 recs from each member of PDS library search-path */
  /* processing member names in reverse order.                      */

 arg MyLib      /* e.g. SYSEXEC, SYSPROC, SYSHELP etc */

 "fileio" MyLib "openread  cld   name InLib"

 "fileio InLib options  memlist  libname  member  recno  noiop"

   /* Switch directly to the last member */
   /* before proceeding in reverse order */
 i          = InLib.memlist.0
 parse var InLib.memlist.i Lib'('LastMember')' .

 "fileio InLib member" LastMember

 do forever
   "fileio InLib sread MYREC 5"; rrc=rc

   say 'MEMBER='InLib.1.libname'('InLib.1.Member')'
   do i=1 to MYREC.0
     say right(InLib.i.recno,5,'0') MYREC.i
   end
   say '----------------------------------------------'
   say ' '
   say ' '
   say ' '

   "fileio InLib prevmember"; rrc=rc
   if rrc > 0 then leave
 end
 "fileio InLib close"
```

RANDOMIZE

The RANDOMIZE option affects subsequent output operations and causes a new test data value to be generated for all fields that already have an existing RANDOMIZER object defined.

The FINSERT/FUPDATE/FWRITE operations are not affected by this option, as they each have individual control over test data value generation.

```
address "CBLSDATA" /* REXX */
  /* Copy a file but disguise personal detail fields with generated test data */

 "fileio PROD.CUST.DETAILS read  name PROD  using cobol PROD.COBOL(CUSTDET)"
 "fileio TEST.CUST.DETAILS write name TEST  using cobol PROD.COBOL(CUSTDET)"

 "fileio TEST  rand  FIRSTNAME  person(ANY)"
 "fileio TEST  rand  LASTNAME   person(LAST)"
 "fileio TEST  rand  EMAIL      pattern "aaaaaaa##[@gmail.com"
 "fileio TEST  rand  TEL-NUM    pattern "[07]###[-]### ###"

 "fileio TEST  option  randomize"

 Copied  = 0
 do forever
   "fileio PROD  sread   MYREC 500"; rci = rc
   "fileio TEST  swrite  MYREC"    ; rco = rc
   Copied=Copied+MYREC.0

   if rci > 0 | rco > 0 then leave
 end
 "fileio PROD close"
 "fileio TEST close"

 'msg' Copied 'records copied'

 if rci > 4 then
   do; 'msg RC='rci 'reading PROD.CUST.DETAILS'
       exit rci
   end

 if rco <> 0 then
   do; 'msg RC='rco 'writing TEST.CUST.DETAILS'
       exit rco
   end
```

RECID

For each input record, the RECID option may be used to obtain one of the following:

◊ **RBA** (Relative Byte Address) for **VSAM ESDS**, **VSAM KSDS** and **z/OS Unix (HFS/ZFS)** files.
◊ **TTR** (volume, track, block and block offset combination) for z/OS **standard sequential files** and **PDS/PDSE Library Members**.

Each time an input record is **selected** the following REXX variable(s) are set:

***<filename>*.1.RECID**	Set to the RBA or TTR for the current record. For the *SREAD* (stem read) operation, which inputs multiple records, REXX variable ***<filename>*.2.RECID** is set for the 2nd record, and so on.

Having, at some point, recorded a record's unique RBA/TTR using the RECID option, it's subsequently possible to a reposition a file's input by performing a **direct read** on that record.

Some examples of direct reads are provided with the related read options: RBA and TTR.

RECNO

For each input record, the RECNO option may be used to determine the **record number**.

Each time an input record is **selected** the following REXX variable(s) are set:

***<filename>*.1.RECNO**	Set to the record number for the current record. For the *SREAD* (stem read) operation, for which inputs multiple records, REXX variable ***<filename>*.2.RECNO** is set for the 2nd record, and so on.

Batch JCL Example:

```
//FIOASM    EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//ASM       DD DISP=SHR,DSN=CBL.CBLI360.ASM
//          DD DISP=SHR,DSN=CBL.CBLI350.ASM
//          DD DISP=SHR,DSN=CBL.CBLI340.ASM
//          DD DISP=SHR,DSN=CBL.CBLI330.ASM
//          DD DISP=SHR,DSN=CBL.CBLI320.ASM
//          DD DISP=SHR,DSN=CBL.CBLI310.ASM
//SDEIN     DD *
  macro FIOSAMP

//FIOSAMP   DD *
  address "CBLSDATA" /* REXX */

  'fileio ASM     open  CLD'; if rc <> 0 then exit rc
  'fileio ASM     options libname member recno'

  'fileio ASM     filter                     ',
                  '(include record           ',
                   'where                    ',
                   '(   record << "ISTDNIB"  ',
                   ' or record << "IKJTCB"   ',
                   ' or record << "IEFTIOT1" ',
                   ' or record << "IHADECB"  ',
                   ' or record << "IEZDEB"   ',
                   ' or record << "IEZIOB"   ',
                   ')                        ',
                  ')                         '

   do forever
     'fileio ASM read MYREC';
     if rc >= 4      then leave
     if rc  = 2      then iterate


     say ' 'strip(ASM.1.libname),
           || '('strip(ASM.1.member) ,
           || ')'                    ,
                 right(ASM.1.recno,5,'0') MYREC

     'fileio ASM nextmember';
     if rc <> 0      then leave
   end

  'fileio ASM close'
/*
/* CBL.CBLI360.ASM(APEEINIT) 01590  ISTDNIB             ,      VTAM NIB.
/* CBL.CBLI360.ASM(CBLATRAC) 04249  ISTDNIB             ,      VTAM NIB.                          IQ004843
/* CBL.CBLI340.ASM(CBLVIMSS) 00535  IKJTCB  LIST=YES           Include TCB DSECT.
/* CBL.CBLI310.ASM(DSN3SATH) 00253 *    MVS         TCB           IKJTCB        TASK CONTROL BLOCK       * 0
/* CBL.CBLI350.ASM(EDTFMAC1) 01388          IEFTIOT1           Include TIOT DSECT                 IQ004573
/* CBL.CBLI360.ASM(IOSFFIO4) 01028  IEZDEB LIST=YES            MVS Data Extent Block.             IQ003759
```

RECTYPE

For each input record, the RECNO option may be used to determine the **record**

Used in conjunction with the USING *copybook_or_mapping_file* option for each input record, the RECTYPE option may be used to determine the **record-type** name.

Each time an input record is **selected** the following REXX variable(s) are set:

***<filename>*.1.RECTYPE**	Set to the record-type for the current record. For the *SREAD* (stem read) operation, for which inputs multiple records, REXX variable ***<filename>*.2.RECTYPE** is set for the 2nd record, and so on.

Batch JCL Example:

```
//FIOSAMP  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SDEIN     DD *
  macro FIOTMPDD

//FIOTMPDD  DD *
  address "CBLSDATA" /* REXX */

  'fileio  CBL.SMF.T030  openread   name SMF30',
        'using CBL.INST.CBL21042.SZZSDIST.SDO(T030)'

  'fileio SMF30  view SMF030_Common_Address_Space_Work',
                    ',SMF030_Identification'

  'fileio SMF30  select zTME',
                'from SMF030_Common_Address_Space_Work'

  'fileio SMF30  select zJOBNAME,zUSERID,zRST,zGRP',
                'from SMF030_Identification'

  'fileio SMF30  option rectype'

  do forever
    'fileio SMF30  fvalue read'
    if rc <> 0 then leave

    if SMF30.1.rectype = 'SMF030_COMMON_ADDRESS_SPACE_WORK'
      then iterate

    msg 'SMF Date/Time :' SMF30.zTME
    msg 'RDR Date/Time :' SMF30.zRST
    msg 'Job Name      :' SMF30.zJOBNAME
    msg 'UserId        :' SMF30.zUSERID
    msg 'Group         :' SMF30.zGRP
    msg '       --------------------------------'
    msg ' '

  end
  'fileio SMF30  close'
/*
```

RESET

The RESET option simply **switches off** all options that are currently set on.

SEGTYPE

If the USING *copybook_or_mapping_file* option specified on the OPEN operation for *filename* refers to a *segmented* file mapping, then for each input segment, the SEGTYPE option may be used to determine if the segment is a **primary** or **secondary**, and for a primary, the total number of segments comprising the complete record.

So, for **a primary** followed by **25 secondaries**, this number will be **26**.

If the file mapping is not segmented this number will always be **1**.

Each time an input segment is **selected** the following REXX variable(s) are set:

***<filename>*.1.SEGTYPE**	For a primary segment, set to "PRI" followed by the total number of segments e.g. "PRI 26" For a secondary segment, set to "SEC". For the *SREAD* (stem read) operation, which inputs multiple records, REXX variable ***<filename>*.2.RECTYPE** is set for the 2nd record, and so on.

Please note that the RECTYPE option may be used to determine the **record-type name** for both primary and secondary segments.

Batch JCL Example:

```
//FIOALLF  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//MYDATA    DD DISP=SHR,DSN=USER123.SMF.TYPE30
//MYMAP     DD DISP=SHR,DSN=CBL.INST.CBL21042.SZZSDIST.SDO(T030)
//SDEIN     DD *
  macro FIOALLF 10  MYDATA using MYMAP    /* First 10 SMF Type 30 recs */

//FIOALLF   DD *
   address "CBLSDATA" /* REXX */

          /* Print every field in every record (or secondary segment).
          |  If the field has a description defined then it is
          |  printed underneath (great for exploring SMF records).
          */

   arg RecsMax FileId 'USING' Mapping NameWidth


   'fileio' FileId 'openr  name MYFILE  using' Mapping
   'fileio MYFILE options rectype segtype recno'

   if NameWidth='' then NameWidth = 50

   call SetRemarks

   FNamePos=length('MYFILE')+2
   RecsDone=0
   do forever

     'sd fileio MYFILE FVALUEQP READ'; if rc <> 0 then leave

     if MYFILE.1.segtype <> 'SEC' then   /* If it's a PRIMARY */
       do; RecsDone=RecsDone+1
           Segs=word(MYFILE.1.segtype,2)
           if RecsMax <> '*' & RecsDone > RecsMax then leave

           say '                      '
           say '                      '
           say '                      '
           pri = '           *-* pri:' MYFILE.1.rectype '*-*'
           if Segs > 1
             then   pri = pri '(+' Segs-1 'segs)'
                    pri = left(pri,71) right(MYFILE.1.recno,8,'0')
           say pri
       end
      else say '            -- sec:' MYFILE.1.rectype '--'

       /* Print field names+values+descriptions */
     do i = 1 to MYFILE.FVALUEQP.0
       FName = substr(MYFILE.FVALUEQP.i,FNamePos)

       if length(FName) > NameWidth
         then FName = right(FName,NameWidth)

       IsArrayElem=0
       Suffix = FName
       do forever
         p=lastpos('.',Suffix); if p = 0 then leave
         Prefix = left(Suffix,p-1)
         Suffix = substr(Suffix,p+1)

         if datatype(Suffix) <> 'NUM'   then leave
         IsArrayElem=1
         Suffix=Prefix
       end
       Val = value(MYFILE.FVALUEQP.i)
```

```
         /* Try to omit non-existant array elements - set to "------" */
       if IsArrayElem=1 & Val = copies('-',length(Val)) then iterate

       Descript = value('Descript.'strip(MYFILE.1.rectype)'.'Suffix)


       say ' ' || left(FName,NameWidth) '|' Val
       if Descript <> '' then call ShowRemarks

     end
     say ' '

   end
   'sd fileio MYFILE close'
   exit 0


SetRemarks:                         /* *-* */
  'list struct /'Mapping'/ columns strip stem SDO',
      'subset /select rectype,name,rem where name <> "" /'

  do i = 1 to SDO.0

    interpret 'Descript.'SDO.i.rectype'.'SDO.i.name '= SDO.i.rem'

  end
  Descript.0 = SDO.0
  SDO. = ''
return 0


ShowRemarks:                       /* *-* */
  WrapWidth=75

  say ' '
  do while Descript <> ''

    p=pos('\\n',Descript)          /* Explicit line-break? */
    if      p = 1            then
      do; CommentLine = ''
          Descript = strip(substr(Descript,4 ))
      end
    else if p > 1 & p < WrapWidth then
      do; CommentLine = strip( left(Descript,p-1))
          Descript = strip(substr(Descript,p+3))
      end
     else
      do; p=lastpos(' ',left(Descript,WrapWidth))
          if p = 0 then p = WrapWidth
          CommentLine = strip( left(Descript,p))
          Descript = strip(substr(Descript,p))
      end

    say right('Desc:'                  ,NameWidth+2) '  ' CommentLine
  end
  say ' ';  say ' ';  say ' '
return 0

 Sample output:

           *-* pri: SMF030_COMMON_ADDRESS_SPACE_WORK *-* (+ 2 segs)     00000001
 Header_Self_Defining_Section.zFLG                  | DE

                                               Desc:    (IBM name: SMF30FLG)
                                               Desc:    System indicator: Bit Meaning when set
                                               Desc:    0 Subsystem identification follows system identification
                                               Desc:    1 Subtypes used
                                               Desc:    2 Reserved
                                               Desc:    3-6 Version indicators
                                               Desc:    7 Reserved.



 Header_Self_Defining_Section.zRTY                  |  30

                                               Desc:    (IBM name: SMF30RTY)
                                               Desc:    Record type 30 (X'1E').



 Header_Self_Defining_Section.zTME                  | 2018/08/31 22:03:18.30

                                               Desc:    (IBM name: SMF30TME)
                                               Desc:    Date/Time that the record was moved to the SMF buffer.



 Header_Self_Defining_Section.zSID                  | S0W1

                                               Desc:    (IBM name: SMF30SID)
                                               Desc:    System identification (from the SID parameter).
/*
```

SPEED

The SPEED option only affects the multi-record/segment input operations SREAD and SREADSEGMENT.

Each time one of these operations is executed it will potentially cause hundreds (possibly thousands) of REXX variables to be set.

If the **SPEED** option is set then all of the variables are set in one call to REXX using a **chain of SHV blocks**.

The SPEED option therefore will likely **reduce cpu time**, but **storage requirements** will **dramatically increase**, since the names and values of **all** the REXX variables being set must be available at once.

USER

For input only, when reading multiple members of a PDS/PDSE library using open options LIBRARY or CONCATENATEDLIBRARYDIRectory, the USER option may be used to determine the **last modified user name** for the **current member**.

Each time a new member is encountered the following REXX variable(s) are set:

***<filename>*.1.USER**	Set to the **userid** that most recently modified the **current member** according to its directory entry. Value is set to **null** if the member's directory entry does not contain this information.

ZEROS

Used in conjunction with the OPEN option USING *copybook_or_mapping_file*, this option will cause the extracted value (see FVALUE/FVALUEQP/FVALUEQF) of any **numeric** field to be returned with **leading zeros** included.
e.g. **"0000005289"**

If ZEROS needs to be set more selectively for individual fields then the CMD ZEROS *field_name* operation maybe used instead. e.g.

```
  "fileio CUSTFILE  cmd zeros on NUMBER_OF_CHILDREN"
```

BWD

For **VSAM** input data sets only, the BWD option of the **MODRPL operation** may be used to set a **backward direction** of input for any subsequent reads.

See also MODRPL LASTREC which will set the file position to the last record, ready for subsequent backward reads. Option BWD is implied by LASTREC, so strictly speaking BWD is unnecessary when LASTREC is specified.

Batch JCL Example:

```
//FIODATES  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//MYKSDS    DD DISP=SHR,DSN=MY.KSDS.DATASET
//SDEIN     DD *
  macro FIOSAMP

//FIOSAMP   DD *
  address "CBLSDATA" /* REXX */

     /* Print the 5 highest keys, then the 5 lowest keys for a KSDS */

  'fileio MYKSDS openread'
  'fileio MYKSDS option key'

      /* BWD is actually unnecessary when LASTREC is specified */
  'fileio MYKSDS modrpl lastrec bwd'
  say '   Hi-Keys:'
  do 5
    'fileio MYKSDS read'
    say ' 'MYKSDS.1.key
  end

  'fileio MYKSDS modrpl fwd'
  say ' '
  say '   Lo-Keys:'
  "fileio MYKSDS read - key x'00'"
  say ' 'MYKSDS.1.key
  do 4
    'fileio MYKSDS read'
    say ' 'MYKSDS.1.key
  end


  'fileio MYKSDS close'
  exit
/*
```

FWD

For **VSAM** input data sets only, the FWD option of the **MODRPL operation** may be used to set a **forward direction** of input (default) for any subsequent reads.

See also MODRPL BWD which will set a backward direction.

FULLKEY

For **VSAM KSDS** input data sets only, the FULLKEY option of the **MODRPL operation** may be used to indicate that for subsequent keyed reads you will be supplying a **full key** (which can match only one record), as apposed to a generic key (which can match several records).

See also MODRPL GENKEY which will set generic key processing.

GENKEY

For **VSAM KSDS** input data sets only, the GENKEY option of the **MODRPL operation** may be used to indicate that for subsequent keyed reads you will be supplying a **generic key** (which can match several records), as apposed to a full key (which can match only one record).

See also MODRPL FULLKEY which will set full key processing.

GENKEY is the **default** processing option.

KEYEQ

For **VSAM KSDS** input data sets only, the KEYEQ option of the **MODRPL operation** may be used to indicate that for subsequent keyed reads to succeed, a record must exist with an **exact match** for your supplied (full or partial) key, otherwise a non-zero return code will be set.

See also MODRPL KEYGE which will set the **"key greater than or equal to"** processing mode.

KEYGE

For **VSAM KSDS** input data sets only, the KEYGE option of the **MODRPL operation** may be used to indicate that for subsequent keyed reads to succeed, a record must exist with a **key greater than or equal to** your supplied (full or partial) key, otherwise a non-zero return code will be set.

See also MODRPL KEYEQ which will set the **"exact key"** processing mode.

KEYGE is the **default** processing option.

LASTREC

For **VSAM** input data sets only, the LASTREC option of the **MODRPL operation** may be used to locate the last record in the file (and set a **backward direction** of input) ready for any subsequent reads.

Batch JCL Example:

```
//FIODATES  EXEC PGM=FILEKITB
//SDEPRINT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//MYVSAM    DD DISP=SHR,DSN=MY.VSAM.DATASET
//SDEIN     DD *
  macro FIOSAMP

//FIOSAMP   DD *
  address "CBLSDATA" /* REXX */

   /* Print the 1st 50-bytes of the last record containing "BLUES" */

  'fileio MYVSAM openread'
  'fileio MYVSAM modrpl lastrec'
  'fileio MYVSAM where (record << "BLUES")'  /* Will read backwards */
  'fileio MYVSAM read MYREC'
  'fileio MYVSAM close'
  say left(MYREC,50)
  exit
/*
```

XRBA

For **VSAM** input data sets only, the XRBA option of the **MODRPL operation** may be used to indicate that for subsequent direct reads by relative byte address you will be supplying an **extended RBA**, which is necessary to handle numbers larger than 4GB.

FILEKIT

Description:

SDE CLI command, FILEKIT, performs the same operation as the CBLe CLI command FILEKIT. See FILEKIT in CBLe Text Edit documentation.

FILTER

Syntax:

```
>>-- FILTer ------------------------------------------------------------------><
```

Description:

Starts the FILTER dialog which may be used to filter the display of records by executing a WHERE (ALL), MORE or LESS command that is generated using a panel in which the user enters selection criteria against a table of fields that comprise the focus record-type.

Parameters:

FILTER has no parameters.

See Also:

LOCATE dialog

FIND

Syntax:

```
                 +- EQ -+             (1) +- ANY ---+
                 |      |                 |         |
>>-+- Find -+--+-+------+- string --+-----+---------+------------------------->
   |        |  | |      |           |     |         |
   +- / ----+  | +- op -+                 +- FOCus -+
               |                    |
               +- VALID ------------+
               |                    |
               +- INVALID ----------+


   +- NEXT --+  +- CHARs --+
   |         |  |          |
>>-+---------+--+----------+--+------+---------------------------------------->
   |         |  |          |  |      |
   +- ALL ---+  +- PREfix -+  +- EX -+
   +- FIRST -+  +- SUFfix -+  +- NX -+
   +- LAST --+  +- WORD ---+  +- X --+
   +- PREV --+


   +-- #ALL -------------------------------------+ +- .ZFIRST ---- .ZLAST -+
   |                                             | |                       |
 >-+---------------------------------------------+-+-----------------------+--><
   |                                             | |                       |
   +-- pos1 ---+---------+-----------------------+ +- .name1 --+-----------+
   |           |         |                       |             |           |
   |           +- pos2 --+                       |             +- .name2 --+
   |                                             |
   |      +----+---------+-----------------+     |
   |      |    |         |                 |     |
   |      |    +-- , ----+                 |     |
   |      v                                |     |
   +-- ( -+-- field_col -------------------+- ) -+
          |                                |
          +-- field_col1:field_col2 -------+
```

(1) Default (ANY or FOCUS) set by the RTSCOPE option.

Description:

Search DB2 table rows or data records in the current edit or browse view for the specified character string or numeric value.

If the **FOCUS** option is specified then only records/segments assigned the default record type (visible and EXCLUDED records) are included in the search. Otherwise all records types are searched.

If the specified occurrence (ALL, FIRST, LAST, NEXT or PREV) of the search string or numeric value is found within a record, then:

1. If the record is EXCLUDED, it becomes unexcluded.
2. The cursor is positioned at the beginning of the string or numeric field.
3. If necessary, scrolling occurs to display the found data.

All occurrences of the search string or numeric value are highlighted in the text. (Enter the RESET FIND command to turn off the highlighting.)

When the duration of a FIND or RFIND execution exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the oportunity to interrupt the operation using the Attn key.

Use of FIND with no parameters is equivalent to RFIND (assigned to function key **PF5** by default) and repeats the last find command executed, including all its specified parameters.

The FORMAT of the SDE display affects the execution of FIND.

Unformatted Multi or Single Record Display (CHAR or UNFMT):

By default, a character compare for the supplied search string is performed against the entire length of unformatted data records.

The prevailing BOUNDS left and right column values define the area of the record within which the search occurs. i.e. the matched data must begin at or after the left bound and not exceed the right bound.

The BOUNDS columns may be overridden using *pos1* and *pos2* positional parameters.

field_col and #ALL parameters are not applicable to these display formats and are ignored.

Formatted Multi or Single Record Display (VFMT or FMT):

For formatted records, the search string is compared against **individual fields** that have been selected for display in the formatted data record. (See SELECT)

Fields are searched from left to right in the order that they appear in the display. This is true regardless of the order in which field columns are specified on the FIND command, or the order in which fields are encountered within the unformatted record.

The prevailing BOUNDS left and right column values define which of the fields within the formatted records are eligible to be searched. i.e. Fields are eligible only if they exist within the left and right bounds when applied to the expanded record data (which is not necessarily the same as the unformatted record data.)

Where a BOUNDS column occurs within a field definition, then the following rules apply:

1. For character data type fields, only the area of the field that falls within the BOUNDS columns is eligible for the search.

2. For numeric data type fields, the field is not eligible to be included in the search.

If one or more field columns are specified on the FIND command (using *field_col* or *field_col1*:*field_col2*) that do not reference at least one field defined by the BOUNDS columns as being eligible for search, then the following error message is returned:

```
ZZSD280E No fields selected within the current bounds for the FIND command.
```

If a field column is specified on the FIND command that is not part of the display (e.g. a group field or a field that has been removed from display by a SELECT command), then the following error message is returned:

```
ZZSD179E Data element field_col is not selected in the current view
    of record type record-type of structure struct_name.
```

For character data type fields, a string compare is performed. For numeric data type fields (binary, packed decimal, floating point, zoned, etc.), then the following will occur:

1. If *pos1*, *pos2* positional parameters are **not** specified, the search string is interpreted as a signed numeric value and an arithmetic compare is performed against the field's formatted numeric value.

   The length and data type of the numeric field, and the number of digits in the search value are not significant. e.g.

   ```
   FIND  67
   ```

   Finds numeric fields with value "67" (e.g. "0067", "67.00", "0.0670E+03") and character fields containing the string "67" (e.g. "167 Baker Street").

If the search string is non-numeric, then numeric data type fields are not searched. Therefore, to bypass searching numeric data type fields, explicitly set the search string to be character or hex using 'string', C'string' or X'string' formats respectively. e.g.

```
FIND   '67'
FIND  C'67'
FIND  X'F6F7'
```

Finds only character fields containing the string "67".

2. If *pos1*, *pos2* positional parameters are specified, the search string is interpreted as being a character string and so a string compare is performed against the unformatted representation of the field data for fields falling entirely or partly within the range of record positions.

   Although the range of positions may span a number of fields, the search is still performed against individual fields within the range. i.e. a match for the search string will not occur for data that spans a field boundary.

   A match for the search string may occur on just part of the unformatted data representation of a numeric field. e.g.

   ```
   FIND   476    21  100
   ```

   This will find a match in any numeric field where unformatted representation of the data contains the string "476". (e.g. a zoned decimal field with value "14760" or "-4762")

Any match of the search string on data in a numeric field will highlight the entire formatted display of the field. If the numeric field is also the current, identified occurrence of the search string, the cursor is positioned at the start of the formatted numeric field display.

Parameters:

op

A relational operator used in the compare operation which determines the relationship that the data must have with the search string in order for it to be identified as a successful match.

Valid values for *op* are as follow:

Operator	Description
EQ	Data must be equal to *string*. (Default)
NE	Data must be not equal to *string*.
GT	Data must be greater than *string*.
GE	Data must be greater than or equal to *string*.
LT	Data must be less than *string*.
LE	Data must be less than or equal to *string*.

If a character string compare is performed, the EBCDIC values assigned to characters in the search and data strings determine the relationship (equal to, greater than or less than) between the two strings.

string

The FIND search string. The search string may be any of the following:

◊ An unquoted numeric value. The search string is treated as a numeric value when a numeric field is searched in a formatted record view. In all other cases a numeric search string is treated as a character string.

◊ An unquoted character string containing no commas or blanks. The search for the character string will be case-insensitive so that uppercase and lowercase characters are treated as being the same.

◊ A character string enclosed in single (') or double (") quotation marks. The search string may contain embedded commas and blanks and the character string will be case-insensitive.

Two adjacent quotation mark characters that are embedded in a search string which is enclosed by the same quotation mark characters, will be treated as a single occurrence of the character. e.g.

```
FIND  'Jim O''Brien'
```

Find the character string "Jim O'Brien".

◊ A character string enclosed in single (') or double (") quotation marks with the prefix C. This is equivalent to specifying a quoted search string but that the string search will be case-sensitive. (e.g. C'Book')

◊ A hexadecimal string enclosed in single (') or double (") quotation marks with the prefix X.

◊ A picture string enclosed in single (') or double (") quotation marks with the prefix P.

Picture strings use special characters to represent a generic group of characters as described below. Any character in a picture string that is not one of these special characters is untranslated.

String	Description
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.

◊ A regular expression string enclosed in single (') or double (") quotation marks with the prefix R. For example:

```
R'C[0-9]^2'
```

is a regular expression which would match the upper case character **C** followed by 2 digits. This expression would find **C00 C91 C22** etc. but not **c99 C 99** etc.

Regular expressions enable powerful string pattern matching at the cost of rather complex syntax and potentially extended command processing time. For syntax and usage see Regular Expressions in Text Editor documentation.

If a no search string is specified and there are no other parameters, then the command is treated as RFIND.

However, if a no search string is specified but one or more *field_col* is supplied then it is treated as a special case that means accept any value in the specified field column(s). For FIND, this is a way of scrolling and placing the cursor on the next/prev occurrence of a record that includes the specified field column(s).

`VALID`

Intended for use with formatted records, VALID will search fields for valid data. i.e. data that satisfies the field's assigned data type.

`INVALID`

Intended for use with formatted records, INVALID will search fields for invalid data. i.e. data that does not satisfy the field's assigned data type.

`ANY`

All record-types are included in the search provided they are not suppressed. See VIEW , VBASE , and V/V+/V-line-commands to suppress and unsuppress record-types.

`FOCUS`

Only the default record type is included in the search. This is normally the type of record at the top of the screen or at the cursor location. This option was the default in earlier versions of FileKit and can be made so again using the RTSCOPE option or via the settings panel (=0.4).

`ALL`

If none of the records to be scanned are EXCLUDED or NX is specified, then FIND ALL is the same as FIND FIRST except that a message is displayed providing the number of occurrences of the string/value found in the data. Unlike FIND FIRST, **all** excluded records that contain an occurrence of the string/value are made visible if NX is not specified. FIND ALL with no other parameters is equivalent to RESET FIND EXCLUDED.

`FIRST`

Search forwards from the top of the file data (i.e. the first position of the first data record) to find the first occurrence of the string.

`LAST`

Search backwards from the bottom of the file data (i.e. the last position of the last data record) to find the last occurrence of the string.

`NEXT`

Search forwards from the current cursor location to find the next occurrence of the string. If the cursor is not within the window's data display area, the search begins at the first position of the first visible or excluded record within the display area that is of the default record type.

`PREV`

Search backwards from the current cursor location to find the previous occurrence of the string. If the cursor is not within the window's data display area, the backwards search begins at the first position of the first visible or excluded record within the display area that is of the default record type.

`CHARS`

For non-numeric search strings only, CHARS indicates that a successful match occurs if the search string is found anywhere within the data being searched.

`PREFIX`

For non-numeric search strings only, PREFIX indicates that a successful match only occurs if the search string is found at the start of a word within the data being searched. i.e. the matched text must precede an alphanumeric character and either be preceded by a non-alphanumeric character or be at the start of a line or field.

`SUFFIX`

For non-numeric search strings only, SUFFIX indicates that a successful match only occurs if the search string is found at the end of a word within the data being searched. i.e. the matched text must be preceded by an alphanumeric character and either precede a non-alphanumeric character or be at the end of a line or field.

`WORD`

For non-numeric search strings only, WORD indicates that a successful match only occurs if the search string is found to be a complete word within the data being searched. i.e. the matched text must either be preceded by a non-alphanumeric character or be at the start of a line or field, and either precede a non-alphanumeric character or be at the end of a line or field.

`EX`
`X`

Search EXCLUDED data records only.
FIND does not search records that are NOTSELECTED or SUPPRESSED.

`NX`

Search only visible data records (i.e. not EXCLUDED).
FIND does not search records that are NOTSELECTED or SUPPRESSED.

pos1

The first position of a range of positions within the data record to be searched.

For formatted records, this is a position in the expanded record . Only those fields, or parts of fields, that fall within the position range will be searched. Fields will be searched in the order that they occur within the display area.

pos1 may be a positive or negative integer value (not zero) and must be a value that is less than or equal to the maximum length of the data records or, for formatted record data, the length of the expanded record.

A negative value represents a position in the record relative to the end of the record. Therefore, where position 1 references the 1st character in the record, position -1 references the last character.

For all display formats, the string is treated as being non-numeric and the search is actioned on the character representation of the record data.

pos2

The last position of a range of positions within the data record to be searched.

Like *pos1*, *pos2* may be a positive or negative integer value (not zero). If *pos1* references a position within the record data which is higher than that referenced by *pos2*, then the *pos1* and *pos2* values are swapped.

If *pos2* is greater than the maximum length of the data records or, for formatted record data, greater than the length of the expanded record, then *pos2* is set equal to the maximum (or expanded) record length.

Default is *pos1* plus the length of the search string minus 1.

`#ALL`

Search all eligible field columns in the current formatted display.

field_col

An individual field column to be searched within a formatted display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID). If a field name is used, enclosing parentheses are **mandatory** Field names are automatically converted to a field reference and Referencing the same field column more than once will not cause an error.

If the field is an array, then all elements of the array are searched. To search an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array. An entry must exist for each dimension of the array. e.g. RoomSize(6,2,4) - a three dimensional array.

Specification of multiple field columns must either be enclosed in parentheses and/or separated by commas. The field columns may be specified in any order, however, the data is always searched from the first column in the display to the last.

Field column search specifications may be a combination of individual field columns and columns ranges. e.g. (JobID #6 Tax_Reference #12 #15:#20).

A search is only performed on field columns that are selected for display. Therefore, any field column specified on the FIND command that has not been selected for display, will be ignored.

field_col1:*field_col2*

The first and last fields of a range of field columns to be searched within a formatted record display display.

field_col1 and *field_col2* must be separated by ":" (colon) and if *field_col1* occurs after *field_col2* in the data record, then the values are swapped. *field_col1* and *field_col2* have the same specifications as *field_col*.

Specification of multiple field column ranges must either be enclosed in parentheses and/or separated by commas. Field column search specifications may be a combination of individual field columns and field columns ranges. e.g. (FirstName:LastName, #2, Salary:Bonus, EmpNo, #10:#15). If field names are used, enclosing parentheses are **mandatory**.

.name1

A label name identifying the first record of a range of data records to be searched. The preceding "." (dot) is mandatory. Default is .ZFIRST.

.name2

A label name identifying the last record of a range of data records to be searched. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then the order is reversed.
If FIND PREV is executed and *.name1* is specified, the default is .ZFIRST. Otherwise the default is .ZLAST.

Examples:

`find  22.3`

Search all fields in the display belonging to the default record type for the next occurrence of the search string/value 22.3. Numeric fields are tested for numeric value 22.3, character fields are tested for character string "22.3" anywhere within the field.

`find all 'ask' suffix  ex  (DsName:ProcLib, JobStep)`

Search selected character fields in the display belonging to excluded records of the default record type for all occurrences of the word suffix string "ask".

See Also:

CHANGE   EXCLUDE   LOCATE   ONLY   RFIND   SELECT

FLIP

Syntax:

```
>>-- FLIP -------------------------------------------------------------------><
```

Description:

Flip records that are of the default record type so that visible data records become EXCLUDED and EXCLUDED records are made visible.

See Also:

LESS   MORE   MORE

FORMAT

Syntax:

```
>>-- FORmat --+-- Character ------+-------------------------------------------><
              |                   |
              +-- Hex ------------+
              |                   |
              +-- HEXDump --------+
              |                   |
              +-- Single ---------+
              |                   |
              +-- Sngl -----------+
              |                   |
              +-- Tabl -----------+
```

Description:

Sets the display format for the current SDE BROWSE or EDIT window.

See also SDE command ZOOM to display unformatted data (FORMAT CHARACTER or FORMAT HEX) in single record display.

Parameters:

`Character`

Multi record view with all records or record segments mapped as a single, character field with field name "UnMapped" or "UnMappedSeg". No data conversion is performed.

`Hex`

Same as CHARACTER with the addition that the data is also displayed in Hex below the character display. Note that the Hex display occupies an additional 2 lines of data.

`HEXDump`

Display the focus record or record segment in single record, unformatted hex dump view. See command HEXDUMP for more information.

`Single`
`Sngl`

Single record format with data types formatted according to the record structure. Each field occupies a separate line. Vertical scrolling transfers focus between fields. Horizontal scrolling transfers focus between records.

By default, **PF2** is assigned to distributed CBLe edit macro SDEZOOMW which opens a new view of the record data and executes FORMAT SINGLE to display the record occupying the focus line in single format.

`Table`

Multi-record table format (Default) with data types formatted according to the record structure. Each field occupies a separate column. Vertical scrolling transfers focus between records. Horizontal scrolling transfers focus between fields.

GEN

GEN command is as supported by the Text Editor. See GEN in Text Editor documentation.

GENCOMP

GENCOMP command is as supported by the Text Editor. See GENCOMP in Text Editor documentation.

GENORPH

GENORPH command is as supported by the Text Editor. See GENORPH in Text Editor documentation.

GETXML

Syntax:

```
>>-- GETXML --- INTO --+--------+- fileid ---------------------------------->
                       |        |
                       +- FILE -+

 >------ FROM --+---------+- table_name ------+---------+- column_name ----->
                |         |                   |         |
                +- TABle -+                   +- COLumn -+

 >--------------+---------+- DOCid -+-----+- document_id -+----------------->
                |         |         |     |               |
                +- WHERE -+         +- = -+               |
                |                                         |
                +- WHERE ( where_clause ) ----------------+

 >--------------+-----------------------+---+---------+---------------------><
                |                       |   |         |
                +- DB2 -+--+-- ssn --+      +- AUDit -+
                |       |  |                |
                +- SSN -+  +- (ssn) -+
```

Description:

Applicable to DB2 base tables only, GETXML copies an XML document from an XML column within a specific DB2 table row to a data set, library member or HFS/ZFS file.

If a connection to the required DB2 sub-system does not already exist, then GETXML will first start a new connection before fetching the XML column data from selected table row and disconnect again afterwards.

Having retrieved the XML document to the specified *fileid*, the XML text is formatted and displayed in a text edit view. Formatting of the XML document starts a new line after each tag element not followed by text data and after each end tag. Nested tag elements are also indented to illustrate the tag hierarchy. A SAVE must be executed if the formatting is to be preserved.

A WHERE filter must be specified which uniquely identifies the table row from which the XML column data will be copied. If the filter identifies more than one table row, then the operation fails with error ZZSD711E.

Parameters:

`INTO FILE` *`fileid`*

Specifies the name of the target XML document data set, library member or HFS/ZFS file into which the DB2 XML column data will be copied.

If *fileid* references an uncataloged sequential data set, DB2 will catalog the new data set as RECFM=VB, LRECL=27994 an BLKSIZE=27998. However, GETXML will fail if *fileid* references a member of an uncataloged PDS/PDSE library.

If *fileid* references an existing sequential data set or a member of an existing PDS/PDSE library, then it must have been cataloged as RECFM=VB with an LRECL value which is 4 less than the BLKSIZE.

`FROM TABLE` *`table_name`*

Specifies the name of the source DB2 table from which the XML column text will be fetched.

table_name may be specified with 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default location is the local DB2 server for the specified sub-system (*ssn*) and the default schema is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID).

`COLUMN` *`column_name`*

Identifies the name of the source DB2 column of data type XML.

If *column_name* specifies a column which is not type XML, then error ZZSD691E is returned.

`WHERE DOCID =` *`document_id`*

For any DB2 base table in which one or more XML columns have been created, there exists an implicit DOCID column (DB2_GENERATED_DOCID_FOR_XML) which stores a unique, integer document identifier for the XML column entries in a row.

A unique document id (*document_id*) may be specified to identify the table row from which data will be copied. Using this syntax, specification of the WHERE keyword and/or equal to (=) operator are optional.

`WHERE (` *`where_clause`* `)`

Specifies a DB2 SQL WHERE clause which identifies a single row in the DB2 table from which the XML column data will be copied. See "*DB2 SQL Reference*" for syntax of the *where-clause*.

`DB2 | SSN   ssn | (ssn)`

Specifies the name of the local DB2 sub-system (*ssn*) to which a connection will be made in order to fetch data from the base table. Before FileKit can successfully connect to a DB2 sub-system, a BIND of the FileKit DB2 plan must have been performed for that sub-system.

Default is the users default DB2 sub-system name as last entered in the DB2 Primary Options menu.

`AUDIT`

Sets DB2 auditing on for the GETXML operation.

A new FileKit DB2 audit data set is allocated to record the results of the SQL FETCH statement executed. See Audit Trail Functions for details.

Default is not to perform edit auditing.

Examples:

```
getxml  into  SYSA.XSPORT.D2016034.XML                               \
                   from  CBL.SPORTS_SAMPLE  column STATS_D2016034_XML  \
                   where DOCID = 2      db2 CBLA
        \
```

Copy an XML document from DB2 table CBL.SPORTS_SAMPLE to the cataloged data set SYSA.XSPORT.D2016034.XML. The required XML document exists in the DB2 table XML column STATS_D2016034_XML, in the row identified by document id 2. Connection will be made to DB2 sub-system CBLA.

```
getxml  into  SYSA.XBOOK.XML(SMITHW01)                               \
                   from  CBL.ZZSBOOKS       column BOOK_REF_XML        \
                   where(AUTHOR LIKE 'Smith, Wilbur%')                 \
                   ssn(LTBX)
```

Copy an XML document from a DB2 table XML column entry in the row uniquely referenced by an SQL WHERE clause, to a new PDS member of an already cataloged library.

See Also:

PUTXML

GO

Syntax:

```
>>-- GO -----+-- Browse --+--------------------------------------------------><
             |            |
             +-- Edit ----+
             |            |
             +-- SE ------+
             |            |
             +-- SU ------+
             |            |
             +-- View ----+
```

Description:

GO closes the current view and redisplays the file data in either a text edit or data edit view using the specified edit/browse type.

When an SDE data edit view is opened to display the text, no attempt is made to format the record data unless AUTOSTRUCTURE APPLY is in effect. If the current view contains unsaved alterations to the text, the GO operation will stop and an error message displayed.

Parameters:

`BROWSE`

Browse the text in a SDE data edit window view.

`EDIT`

Display the text in a CBLe text edit window view for edit.

`SE`

Display the text in a SDE data edit window view with full edit capabilities.

`SU`

Display the text in a SDE data edit window view with update-in-place edit capabilities.

`VIEW`

Display the text in a CBLe text edit window view for read-only edit (view).

GRPC

Syntax:

```
>>-- GRPC -------------+---------------+------------------------------------->
                       |               |
                       +-- field_col --+

 >-+-----------------------------------------------------------------------+-><
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

Description:

GRPC is equivalent to SET GROUPASCHARACTER ON and so displays the specified group (structure) field, defined within a specified record type mapping, as a fixed length character field.

Parameters:

field_col

The individual column identifying the group field to which the option will apply. This may be the group field itself or a field contained within. The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

Default is the focus column.

FOR [RECORD] *record_type*

Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

IN [STRUCTURE] *struct_name*

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

See Also:

GRPX ARRC ARRX

GRPX

Syntax:

```
>>-- GRPX -------------+---------------+------------------------------------->
                       |               |
                       +-- field_col --+

 >-+-----------------------------------------------------------------------+-><
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

Description:

GRPX is equivalent to SET GROUPASCHARACTER OFF and so redisplays the specified group (structure) field, which has been displayed as a fixed length character field (GRPC), as a formatted group of its component fields.

Parameters:

field_col

The individual column identifying the group field to which the option will apply. This may be the group field itself or a field contained within. The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

Default is the focus column.

FOR [RECORD] *record_type*

Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

IN [STRUCTURE] *struct_name*

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

See Also:

GRPC ARRC ARRX

HELP

Description:

SDE CLI command, HELP, performs the same operation as the CBLe CLI command HELP. See HELP in CBLe Text Edit documentation.

HEX

Syntax:

```
>>-- HEX --+-- ON ----+------------------------------------------------------><
           |          |
           +-- OFf ---+
```

Description:

Sets the hexadecimal display format on or off. Issued without the **ON** or **OFF** parameter the display format is toggled.

If the current display format has been set using **FORMAT CHAR**, or **FORMAT HEX**, then **HEX ON** is the equivalent of issuing **FORMAT HEX**.

If the current display format has been set using **FORMAT SINGLE**, or **FORMAT TABLE**, then for each line displayed **HEX ON** will add two further lines to show the hexadecimal representation of the raw (unformatted) data below each formatted field. e.g.

```
 SRCRecOff  SRCRecLen SRCMbr
       #44        #45 #47
  BN 121:4   BN 125:4 AN 137:8
<---+----> <---+----> <---+-->
       152         80 APEZLINK
      0009       0005 CDCEDCDD
      0008       0000 17593952

       144         80 SDEFCMD1
      0009       0005 ECCCCDCF
      0000       0000 24563441
```

Parameters:

ON
OFF

HEX display format is set ON or OFF.

HEXDUMP

Syntax:

```
>>-- HEXDump --+----------+--------------------------------------------------><
               |          |
               +-- New ---+
```

Description:

Display the focus record or record segment in single record, unformatted hex dump view. HEXDUMP may also be executed for an individual record/record segment using prefix command HEX or HEXD.

The hex dump view is a storage display window which displays a number of fullwords of data in both long hexadecimal and character representation. The hex dump display supports the storage window options popup menu, opened by positioning the cursor in the character or hex representation of the data and pressing <F16> or executing primary command SPM (SHOWPOPUPMENU).

```
  Record type: REC-CARD   Fixed(66) Offset=0 Data elements=13

  Record> 00000008   Flags: f        Length:     66

       1 00000000 00000008 0000061C 0001C3C4        .  .. .CD
      17 C5D3E340 40400675 48756764 74656CD4 ELT   .ÍçÍÅÀÈÁ%M
      33 D940C340 E2D3C1C9 D5404040 40404040 R C SLAIN
      49 40404040 40404040 0200409C 0200501C         .  æ. &.
      65 719C                                Éæ
```

Unlike other storage display windows, altering the number of fullwords displayed per line and/or including display of the data's address in storage will not attempt to resize the window display.

For EDIT, both the hexadecimal and character representations of the data may be updated whereby changes to one representation of the data are automatically reflected in the other.

Beware, however, that overtyping the character representation of the data will update **all** fullwords in the updated line so they reflect the character data. In the character representation, a null character (x'00') is represented by blank (EBCDIC x'40') and other non-printable characters are represented by dots (EBCDIC x'4B'). Therefore, updating the character representation will result in these characters replacing the original hexadecimal values.

If parameter NEW is specified, the hex dump display is opened in a new window view allowing the user to preserve the current data display format in the original window view.

Parameters:

`NEW`

Open the hex dump display in a new SDE EDIT/BROWSE window view.

See Also:

CHAR MAP (FMT) UNMAP (UNFMT) VFMT ZOOM

HIDE

Syntax:

```
>>-- HIDE -------------------------------------------------------------------><
```

Description:

Hide all shadow lines. HIDE is equivalent to SET SHADOW OFF ALL.

RESET HIDE will redisplay all shadow lines.

IDENTIFY

Syntax:

```
                  +- 1 -------------------------+
                  |                             |
>>-- IDentify ----+-----------------------------+-------+------+------------><
                  |                             |       |      |
                  +- n_lines -------------------+       +- EX -+
                  |                             |       |      |
                  +- ALL -----------------------+       +- X --+
                  |                             |       |      |
                  +- * -------------------------+       +- NX -+
                  |                             |
                  +-  .name1 -----+-------------+
                                  |             |
                                  +-- .name2 ---+
```

Description:

IDENTIFY is used to remap records (re-assign record-type RTO) in the current SDE edit view. For segmented record edit, if any segment within a record is selected for remap by the IDENTIFY command, then all segments within the record will be remapped.

By default, IDENTIFY will not attempt to select and remap records that have **not** been altered since the file was loaded for edit.

See *"Record Type Assignment"* for description of the processing performed on each record/record segment selected by IDENTIFY for remap.

ID fields are identified by a record type (RTO) definition containing USE WHEN criteria. Values in these fields determine whether that RTO is suitable to be assigned to an individual record or record segment.

FileKit SDE is sensitive to changes made to ID fields that have been identified explicity by field name or field reference number in the USE WHEN expresssion. It is also sensitive to changes to data in when record-type formatting is disabled (i.e. field name "Unmapped" or field reference number #1).

Where one of these ID field changes is detected, or where the length of the record/segment data is altered (via CHANGE or RECLEN updates), the ID flag (**==ID>**) is set on for the changed record/segment. This is intended to notify the user that the record/segment data may no longer satisfy the USE WHEN criteria for its assigned record type.

In order to allow the user the opportunity to complete any changes being made, FileKit does not automatically remap records with the ID flag set on.

The records selected for remap by IDENTIFY depend on the current value of the IDSCOPE option. By default, IDSCOPE is set to CHANGED indicating that all records flagged as having been changed may be selected for remap. Alternatively, IDSCOPE FLAGGED indicates that only records with the ID flag set on may be selected for remap.

The remap of records, performed on execution of IDENTIFY, may be individually undone and subsequently redone using the UNDO (<PF22>) and REDO (<PF23>) commands.

Parameters:

Note that in single-record view the SDE window view displays the focus line (record or record segment) of the equivalent multi-record view.

`n_lines`

Specifies the number of lines (records or segments), including the focus line, to be scanned searching for records/segments to be selected for remap.
The default is 1 line.

`ALL | *`

Specifies that all lines in the file are to be scanned, searching for records/segments to be selected for remap.

`.name1`

A label name identifying the first line of a range of lines to be scanned, searching for records/segments to be selected for remap. The preceding "." (dot) is mandatory.

`.name2`

A label name identifying the last line of a range of lines to be scanned, searching for records/segments to be selected for remap. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then the order is reversed.
If *.name1* is specified without *.name2*, then .ZLAST is the default.

`EX | X`

Search for records/segments occupying EXCLUDED lines only.

`NX`

Search for records/segments occupying visible (non-EXCLUDED) lines only.
Default is to search for records/segments in both EXCLUDED and non-EXCLUDED lines.

See Also:

SET/QUERY/EXTRACT Option: IDSCOPE

IMMEDIATE

Description:

SDE CLI command, IMMEDIATE, performs the same operation as the CBLe CLI command IMMEDIATE. See IMMEDIATE in CBLe Text Edit documentation.

INFORMATION

Syntax:

```
>>-- INFOrmation -----+------------------------------------------+-----------><
                      |                                          |
                      +------------------------- fileid  --------+
                      |                                          |
                      +-- DB2 --+---------+----- table_name -----+
                                |         |
                                +- (ssn) -+
```

Description:

INFORMATION has the same operation as the FileKit primary command DSINFORMATION.

INSERT

Syntax:

```
             +--- 1 -----+
             |           |
>>- Insert --+-----------+---+---------------+------------------------------->
             |           |   |               |
             +- n_lines -+   +- record_type -+

 >---+-----------------------+-----+------------------------------+---------->
     |                       |     |                              |
     |    +- , --------+     |     |          +- , ----------+    |
     |    v            |     |     |          v              |    |
     +- (-- field_col -+- ) -+     +- Values(-- field_value -+-) -+

 >---+-------------------------------------+---+--------------+--------------><
     |                                     |   |              |
     +- RANDomize -+-----------------------+   +- SKIPerrors -+
                   |                       |
                   |    +- , --------+     |
                   |    v            |     |
                   +- (-- field_col -+- ) -+
                   |                       |
                   +- ( ) -----------------+
```

Description:

Insert one or more new records following the focus line, optionally providing explicit values to be inserted into specified fields.

Insert of new records is not allowed for Update-in-place editing unless the data set is an RRDS or VRDS. For these types of files, records that are empty slots may be activated using the INSERT command where, starting from the focus line, *n_lines* represents the number of lines to be searched for empty slots. All empty slots found within this range of lines will be activated.

Fields within the inserted records do not require explicitly assigned values. This occurs either when no list of field values is specified or when the default record type contains more fields than there are values in the specified list of field values.

If fields are not explicitly assigned values (either because no list of field values is specified or because the default record type contains more fields than there are values in the specified list of field values) then they are assigned **default values** as follows:

- A "Test Data" value is **generated** if either
    1. **RANDOMIZE(*field_col*)** is explicitly requested, or
    2. A RANDOMIZER object already exists for *field_col*.
- 0 (zero) for numeric fields.
- Nulls for bit and hex fields.
- Blanks for all other field types.

If a USE WHEN expression has been defined for the chosed record type, then an attempt is made to populate any fields specified in that expression with values which satisfy the expression.

INSERT with no parameters inserts one record of the current default record type with default values in all fields.

If no structure (SDO) or copybook has been applied to data records of variable length (i.e. unformatted variable length records), then the user is prompted to specify a record length for the inserted record. This is because the SDE editor preserves record length when the records are saved. See Changing Record Lengths.

Following execution of INSERT, the DRECTYPE setting is updated to be the record type specified by the *record_type* parameter.

Also see the "I(n)" insert prefix area command.

Parameters:

n_lines

Number of records to be inserted or, for RRDS/VRDS Update-in-place edit, the number of lines to be searched for Empty Slots.
Default is 1.

record_type

The record type of the record(s) to be inserted. If *record_type* is not specified, then the default record type is used.

(*field_col1*, *field_col2*, ...)

An individual field column within a list of field columns for which a corresponding *field_value* is to be inserted.

The field column may be identified by its reference number (e.g. #1) or its name (e.g. SeqNUM). Specification of multiple field columns must be separated by commas and enclosed in parentheses.

If the field is a **struct** or a **union**, then an error message is returned. If the field is an array, then an individual array element must be specified in parentheses as a subscript to the field. e.g. #13(3) for the 3rd element of a single dimension array. A subscript entry must exist for each dimension of the array. e.g. RoomSize(6,2,4) - an element within a three dimensional array.

If a list of field columns is not specified then as many of the set of selected fields in the current view as there are values in the list of field values, are used as the default. In this case, the order of the field columns is equal to the order in which fields are displayed in the current view.

VALUES (*field_value1*, *field_value2*, ...)

An individual field value within a list of field values to be inserted in a corresponding *field_col* entry in the field column list.

If one or more *field_col* entries are specified (i.e. a field list is specified), then there must be the same number of *field_value* values to correspond with each *field_col* entry.

If a character string field value contains special characters, blanks or commas, then it must be delimited by quotation mark (") or apostrope (') characters. If the value contains characters which match the delimiting characters, then each occurrence of this character must be escaped by prefixing it with the escape character. The escape character is the same as the delimiting character. e.g. VALUES('He said "It''s John''s bike."')

RANDOMIZE

Specified without an explicit list of columns, this option causes a new test data value to be generated for all fields.

If any field does not already have a RANDOMIZER object defined, then one will be automatically created using default characteristics based on the **data-type** of the field.

RANDOMIZE (*field_col1*, *field_col2*, ...)

Specified with an explicit list of columns, this option causes a new test data value to be generated for the **specified fields only**.

If a specified field does not already have a RANDOMIZER object defined, then one will be automatically created using default characteristics based on the **data-type** of the field.

RANDOMIZE ( )

Since the **default** action is that a new test data value will be generated for all fields that already have an existing RANDOMIZER object defined, in order to deactivate it you must specify **RANDOMIZE()**.

SKIPERRORS

SKIPERRORS will allow insert processing to continue even if an error has occurred whilst attempting to insert a field value.

Examples:

```
insert  (SeqNum, FirstName, LastName, JobTitle, Salary)   Val(283,Jacob,Smith,'Systems Engineer',35,950.00)
```

Insert a record of the default record type populating the specified list of fields with the explicit values in the corresponding list of field values. All other fields are populated with default values.

```
insert 2  Val(18.3,"Michael O'Leary",-12.333)
```

Insert 2 records of the default record type populating the first 3 selected fields in the display with the explicit values specified in the list of field values. All other fields are populated with default values.

```
i 10000 track rand
```

Insert 10000 records of record type "TRACK" generating test data values for all fields.

See Also:

REPLACELINE

JSONGEN

Syntax:

```
>>--- JSOngen ----------------------------------------------------------------><
```

Description:

JSONGEN with no parameters will open the SDE JSON Generation Panel to generate JSON output for the contents of the focus SDE data edit or browse display.

If the focus window is not an SDE window view, the general JSON Generation dialog panel is displayed.

If parameters are specified, then the general FileKit JSONGEN primary command is executed instead.

See Also:

CSVGEN XMLGEN PRINT

LAYOUT

Syntax:

```
>>-- LAYout --+------------------------------------------------+-------------->
              |                                                |
              +-+---------+--- sfile_fileid -------------------+
              | |         |                                    |
              | +- SDO ---+                                    |
              |                                                |
              +-+- COBol -+--- copybook_library(member_name)  -+
              | |         |                                    |
              | +- PL1 ---+                                    |
              |                                                |
              +--- ADAta ----- adata_fileid -------------------+

              +- NOEXPand -+     +- NUMWidth 5 -------+
              |            |     |                    |
 >------------+------------+-----+--------------------+-----------------------><
              |            |     |                    |
              +- EXpand ---+     +- NUMWidth fwidth --+
```

Description:

Display the record structure layout list window detailing all record-types in the specified structure file (SDE structure (SDO), COBOL/PL1 copy book or COBOL/PL1 ADATA file).

If executed from an SDE browse/edit view without specifying the name of a structure file, the layout window is displayed for all record-types in the structure used to may records in the current display.

If executed from any other window with no parameters the Display Record Layout panel (=9.3) is opened.

Parameters:

SDO *sfile_fileid*
Identifies an existing FileKit SDE structure (SDO) file, *sfile_fileid*

COBOL|PL1 *copybook_library(member_name)*
Identifies a COBOL or PL1 copy book PDS/PDSE library and member name, *copybook_library(member_name).*

ADATA *adata_fileid*
Identifies a COBOL or PL1 ADATA output file, *adata_fileid*.

NUMWIDTH *fwidth*
Specifies the displayed width of numeric columns: **RefNo**, **Start**, **End** and **Length** in the layout list output.

The default (and minimum) width of these fields is 5 characters.

EXPAND | NOEXPAND
Specifies whether or not array (OCCURS) fields are to be expanded to display every repeating instance of a field within that array.
Default is NOEXPAND.

LEFT

Syntax:

```
>>- Left ---+-------------------------------------------+--------------------><
            |                                           |
            +----------------------+-- Cursor ----------+
            |                      +-- CSR -------------+
            |                                           |
            +---------------+-----+-- Data ------------+
            |               |     |                    |
            +-- ALL ---------+     +-- Half ------------+
                                   |                    |
                                   +-- Max -------------+
                                   |                    |
                                   +-- Page ------------+
                                   |                    |
                                   +-- n_cols ----------+
```

Description:

In multi record view and unless parameter **ALL** is specified, scroll to the left the display of field and header lines belonging to records that are of the default record type. The display of all other visible records, header lines and shadow lines remains unchanged.

In single record view, LEFT will display the fields belonging to a visible data record that has a lower line number than the record (or record segment) that is in the current view. For display of segmented records, LEFT and RIGHT scroll through each segment in the file, regardless of the record to which it belongs. NEXT and PREV may be used to scroll between secondary segments of the current record only. LEFT CURSOR/DATA/HALF/PAGE are not applicable in single record view.

LEFT is assigned to **PF10** by default. Any characters specified on the command line when the PFKey is hit will be concatenated to the command and treated as a parameter string.
Where no parameter is specified, the scroll amount will be the value specified in the "**Scroll>**" field.

Multi Record View Scrolling:

Columns may have the HOLD flag set on by the SELECT command. These columns are static in the display and, like the prefix area and Lrecl column, are unaffected by LEFT and RIGHT scrolling. Columns that have the HOLD flag set off are referred to as **floating** columns.

The following interpretation of cursor location applies so that LEFT CURSOR can operate successfully on the appropriate data:

- Cursor is located at an offset within a column header line.
  The cursor position is interpreted as being at the same offset within the data record that immediately follows the group of header lines.

- Cursor is located within a column of type character (AN) but beyond the width of the character data.
  The cursor position is interpreted as being the last position of the displayed character data.

- Cursor is located immediately to the left of a column of any type.
  The cursor position is interpreted as being the the first position of the data displayed within that column.

Where the last column to be displayed is not of type character (AN), the magnitude by which the display is scrolled to the left is always reduced to display all field data belonging to the last column. In addition, the display area must contain all column header text belonging to the first column and, where the first column is not of type character (AN), it must also contain all field data belonging to the first column. Therefore, to accomodate this, the magnitude by which the display is scrolled to the left may be further reduced.

Because of these adjustments, it is possible that execution of a LEFT command may result in no change to the displayed data, in which case LEFT PAGE is executed instead.

No adjustment is made for scrolling left into an offset within a character data column. i.e. where the last column of the display is of type character, the last position of the column display need not be the last character of the character data. If the last column to be displayed is of type character and the display width is too small to accomodate all the column's data and header text, the column will be truncated. However, due to the other adjustments made for scrolling left, it may be impossible for some characters within the character data to occupy the last position of the column display. In order to overcome this, the user would have to alter the display area dimensions.

Attempting to scroll left beyond the start of the record will make the first data column the first column of the display.

Parameters:

`CURSOR`
`CSR`

The focus column becomes the last column of the scrolled display.
In addition to this, if the focus column is type character (AN) and the cursor is positioned on a character that is at an offset into the focus column data, then an attempt is made to make that character occupy the last position of the scrolled display.

If any of the following conditions are true, then LEFT PAGE is executed instead:

◊ Cursor is positioned anywhere other than within a floating column in a visible data record.
◊ Focus column is **not** type AN and is already the last column of the display.
◊ Focus column is type AN, is already the last column of the display and cursor position is at the last position of the displayed column data.

`ALL`

Valid only in multi-record view, ALL indicates that scrolling is to apply to **all** records in the display and not only those records that are assigned the default record type.

`DATA`

Scroll left so that the first floating column in the current display area becomes the last column of the scrolled display.
If this column is type character (AN) and the first position of the display falls on a character that is at an offset into the column, then an attempt is made to make that character occupy the last position of the scrolled display.

`HALF`

Scroll a number of columns so that the column situated half way along the width of the current display of floating columns, becomes the last column of the scrolled display.
If this column is type character (AN) and the half display position falls on a character that is at an offset into the column, then an attempt is made to make that character occupy the last position of the scrolled display.

`MAX`

Scroll left to display the first floating column of data in a multi record view.

In single record view, the first visible data record is displayed.

`PAGE`

Scroll left so that the floating column of data to the left of the first floating column in the current display, becomes the last column of the scrolled display.
If this column is type character (AN) and the first position of the display minus one falls on a character that is at an offset into the column, then an attempt is made to make that character occupy the last position of the scrolled display.

`n_cols`

In a multi record view, scroll left a specified number of floating columns.
The floating column of data that is *n_cols* to the left of the first floating column becomes the new first floating column of the scrolled display.

In single record view, the visible data record that is *n_cols* records before the record currently in view, gets displayed.

Examples:

`left page`

Scroll records of the default record type in a multi record view display area, left by a full display area width minus the width of any columns flagged with HOLD.

See Also:

DOWN RIGHT UP

LESS

Syntax:

```
>>--- LESS ---+-- where_clause --+-------------------------------------------><
              |                  |
              +-- line_flag -----+
```

Description:

Exclude any visible records that are of the default record type and also satisfy the specified *where_clause* criteria. Records that are already EXCLUDED are unaffected by the LESS command. The current line is also unaffected unless the SHADOW option for EXCLUDED lines is set off and the current line is one which is excluded by the LESS command. In this case the line following the current line becomes the new current line.

If LESS is executed with no parameters, then all visible records that are of the default record type are EXCLUDED.

The record type used on a LESS command becomes the new value of the DRECTYPE option.

Parameters:

`where_clause`
where_clause syntax is described fully in documentation for the WHERE command.

`line_flag`
line_flag may be specified to locate a record or record segment that has been flagged with the requested line flag.

Each valid *line_flag* keyword detailed below corresponds to a built-in function.

A description of each *line_flag* keyword may be found in its equivalent function description.

***line_flag* Keyword**	**Built-in Function**
ERRor	CHANGEERROR()
CHAnge | Chg	CHANGEOK()
ALTered | UPDated	CHANGED()
DATaerror	DATATYPEERROR()
DUPkey	DUPLICATEKEY()
EMPTY	EMPTYSLOT()
EXcluded | X	EXCLUDED()
LABel	HASPOINT()
COMmand | CMd	HASPREFIXCMD()
IDentify | IDrequired	IDREQUIRED()
NEW	INSERTED()
KEYChanged | KEYChg	KEYCHANGED()
LENgtherror	LENGTHERROR()
EOL | NOEOL	NOEOL()
SAVE	SAVEREQUIRED()
SQLError	SQLERROR()
TRNC | TRUNCated	TRUNCATED()
VALERRor	VALUEERROR()

Examples:

`less     #14 >= 255`
Exclude any records that are of the default record type where numeric field reference number 14 is greater than or equal to 255.

`less     OrderDate <= PaymentReceivedDate`
Exclude any records that are of the default record type where the contents of field "OrderDate" is less than or equal to the contents of field "PaymentReceivedDate".

See Also:

FLIP MORE WHERE

LIST

Description:

The Data Editor command, LIST, performs the same operation as the Text Editor command LIST provided the first parameter is not one of the keywords MODULES, RPOS, SDO, STORAGE or UKRS. See LIST in CBLe Text Editor documentation.

LIST MODULES

Syntax:

```
                           +-- SDELIB ---+
                           |             |
>>--+-- LIst MODules --+---+-------------+-----------------------------------><
    |                  |   |             |
    +-- LM ------------+   +-- libname --+
```

Description:

Use the LIST MODULES command to display a list window containing details of modules in the specified FileKit function library.

LIST MODULES may be executed to determine the level of any currently installed FileKit function module. CBL may request that this command is executed for diagnostic purposes in problem determination.

Parameters:

libname

The name of a FileKit library. Valid entries include VCILIB, WINLIB, EDTLIB, SDELIB, TABLIB, etc.

Default is SDELIB.

LIST RPOS

Syntax:

```
>>----- LIst RPOs -----------------------------------------------------------><
```

Description:

Use the LIST RPOS command to display a list window containing details of RPOs (Record Position Objects) for the current SDE edit/browse view.

CBL may request that this command is executed for diagnostic purposes in problem determination.

LIST SDO

Syntax:

```
                     +-- user.**.SDO ---+
                     |                  |
>>----- LIst SDO ----+------------------+-----------------------------------><
                     |                  |
                     +-- library_mask --+
```

Description:

Use the LIST SDO command to display the Structure Definition Members list window containing details of each SDE structure (SDO) library member selected by the nominated PDS/PDSE library and member masks.

Parameters:

library_mask

Specifies a PDS/PDSE library DSN mask and optionally one or more member name masks. If no member mask is specified, all members in selected libraries will be listed.

A library DSN mask may contain the following wild card characters:

`*` A single asterisk represents a DSN qualifier or zero or more characters within a DSN qualifier.
e.g. `DEV.*.SDO`, `DEV.C*.*.SDO`

`**` Double asterisk represents zero or more qualifiers within a DSN. Double asterisk must be preceded or followed by either a "." (dot/period) or a blank. It cannot precede or follow an alphanumeric character.
e.g. `DEV.**.SDO`, `DEV.SDO.**`

`%` A single percent sign represents exactly one character other than "." (dot/period) within a DSN qualifier.
e.g. `DEV.C%%.SDO`

One or more member masks may be specified between a single pair of "( )" (parentheses). Multiple PDS/PDSE member masks must be separated by a "," (commma) and/or one or more intervening blanks.

A member mask may contain the following wild card characters:

`*` A single asterisk represents an entire member name or zero or more characters within a member name.

`%` A single percent sign represents exactly one character within a member name mask.

Default is library mask is "*user*.**.SDO" where *user* is the user's RACF logon id.

Examples:

```
LIST SDO   DEV.%BL*.**.SDO(CC*, *MAN, XM*J*)
```

List SDO members that match this library and member masks.

LIST STORAGE

Syntax:

```
>>----- LIst STOrage -------------------------------------------------------><
```

Description:

Use the LIST STORAGE command to open a list window containing details of bit mapped storage used for the current SDE edit/browse view.

CBL may request that this command is executed for diagnostic purposes in problem determination.

LIST UKRS

Syntax:

```
>>----- LIst UKRs ----------------------------------------------------------><
```

Description:

For SDE edit of a VSAM KSDS data set, use the LIST UKRS command to open a list window containing details of updated KSDS keys for the current SDE edit view.

CBL may request that this command is executed for diagnostic purposes in problem determination.

LOCATE

Syntax:

```
   +- Locate --+
   |           |
>>-+-----------+-------+---------------------------------------+-------------><
                       |                                       |
                       +--| Record Locate Options |------------+
                       |                                       |
                       +--| Formatted Column Locate Options |--+
```

Record Locate Options:

```
 |---+----------------------------- .name ------------------------------+----|
     |                                                                  |
     | +- : -+  (1) +- RECord -+                                        |
     | |     |      |          |                                        |
     +-+-----+------+----------+---- record_num ------------------------+
     |                                                                  |
     |          (1) +- KEY ----+                                        |
     |              |          |                                        |
     +--- : --------+----------+---- key_string ------------------------+
     |                                                                  |
     |             (1)                                                  |
     +--- : ---------- RBA --------- byte_offset -----------------------+
     |                                                                  |
     +-----------+- + (plus) --+---- relative_line ---------------------+
     |           |             |                                        |
     |           +- - (minus) -+                                        |
     |                                                                  |
     |                                                                  |
     |                       +- NEXT -----------  1 ----+               |
     |                       |                          |               |
     +-+-- expression ---+---+--------------------------+-----+------+--+
       |                 |   |                          |     |      |
       +-- line_flag ----+   +- NEXT----+   +---- 1 ----+     +- EX -+
                             |          |   |           |     |      |
                             +- FIRST --+---+-----------+     +- NX -+
                             |          |   |           |     |      |
                             +- LAST ---+   +- n_times -+     +- X --+
                             |          |               |
                             +- PREV ---+               |
                             |                          |
                             +- ALL  -------------------+
```

Formatted Column Locate Options:

```
                             +- NEXT  -+
                             |         |
 |---+-- field_name --+------+---------+-------------------------------------|
     |                |      |         |
     +-- field_ref ---+      +- FIRST -+
                             |         |
                             +- LAST --+
                             |         |
                             +- PREV --+
```

Notes:

1. Parameters RECORD, KEY or RBA may be specified before or after their associated *record_num*, *key_string* or *byte_offset* value.

Description:

Locate and scroll to a data record or the formatted record column field that matches the specified criteria. Where no match is found, the display is unchanged.

LOCATE with no parameters will open the SDE LOCATE Records Panel.

Field column location is supported for formatted record data (i.e mapped by an SDO structure record type) and applies to the default record. only.

In single record view, record locate will scroll left or right to display the record that matches the locate criteria, whereas formatted column locate will scroll up or down so that the specified field appears first in the display.

The LOCATE command verb may be omitted altogether if the first token in the command can be identified as being a valid locate parameter and ambiguity exists with defined FileKit command verbs.

If *expression* is specified for record location and no field name nor field reference is included within *expression*, then the LOCATE is performed only for records that are of the default record type. Records that are not of the default record type are bypassed.

When the duration of a LOCATE execution exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the opportunity to interrupt the operation using the Attn key.

Parameters:

Record Locate Options

`.name`

Label name assigned to a line within the data display.
The preceding "." (dot) is mandatory.

`:`

A ":" (colon) identifies the locate target to be an absolute record or record segment number, KSDS data set key value or VSAM RBA (relative byte address.)

Required target record is identified via one of the following:

`(RECORD) record_num`

For any data set organisation, locate the record number specified by *record_num*.

For edit techniques where record numbers have not yet established (e.g. KSDS edit following a LOCATE by KEY, or DOWN MAX), then locating by record number will establish the record numbers and update the prefix area display accordingly.

For segmented record edit only, LOCATE is sensitive to the current setting (PHYSICAL or LOGICAL) of the PREFIX option. If PHYSICAL is in effect, *record_num* corresponds to a record number whereas, if LOGICAL is in effect, *record_num* corresponds to a record segment number within the file.

Specification of parameter keyword RECORD and leading ":" (colon) is optional if *record_num* is an unquoted numeric value. e.g. `5`, `212`

`(KEY) key_string`

For KSDS data sets only, locate the record with a key string equal to *key_string*. If not found, the record with the next key string greater than *key_string* is located.

key_string may be specified as:

1. An unquoted character string. The string must contain no commas or blanks and will be upper cased.
2. A character string enclosed in apostrophes (') or quotation (") marks. The string will be upper cased and any two adjacent characters embedded in the string that are the same as the enclosing symbols, are treated as a single occurrence of the character.
3. A character string enclosed in apostrophes (') or quotation (") marks with the prefix (or suffix) C. This has characteristics equivalent to key strings specified in the previous item except that no upper casing will occur.
4. A hexadecimal string enclosed in apostrophes (') or quotation (") marks with the prefix (or suffix) X.

Specification of parameter keyword KEY is optional if *key_string* is **not** an unquoted numeric value. e.g. `:ABC`, `:12UF00`, `:'123456'`, `:X'003D67A6'`, `:C'abc_00'`

`RBA byte_offset`

For KSDS and ESDS data sets only, locate the record starting at or following the relative byte address specified by *byte_offset*.

By default, VSAM returns a point error if the specified RBA does not point at the start of a record. SDE detects this condition so that, if the RBA does not point at the start of a record, then the record with the next higher RBA becomes the current line of the display. For KSDS data sets, a best effort attempt will be made to locate the record with the next higher RBA though this may not be accurate.

RBA address for each record is displayed as part of the record information columns within an SDE window view. Display of these columns is controlled using the SDE SET RECINFO CLI command.

byte_offset may be specified as a decimal or hexadecimal numeric value. e.g. `:RBA X'01D5'` is equivalent to `:RBA 469`

Specification of parameter keyword RBA is mandatory.

Parameter keywords RECORD, KEY and RBA may be specified before or after their associated locate line target values. e.g. `:KEY C'APE110X5'` is equivalent to `:C'APE110X5' KEY`

If the target line contains a record that is flagged as being EXCLUDED, SUPPRESSED or NOTSELECTED, then the following message is returned:

```
ZZSD177E Requested line record_num|key_string|byte_offset is not visible.
```

relative_line

An integer number representing the number of lines through which the display is to be scrolled. The lines scrolled include data records and shadow line groups.

relative_line may be preceded by "+" (plus) or "-" (minus) which specifies the direction, down or up respectively, in which the display will be scrolled. If omitted, "+" is assumed. (e.g. +18, -2, 3)

Specifying *relative_line* is equivalent to executing the DOWN *n_lines* or UP *n_lines* commands.

expression

expression is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of *expression* must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

When an expression is applied to the record, it must test "true" in order to be selected as a target of the LOCATE command.

For LOCATE, *expression* may not be specified as a single field value term with no operators. An expression of this sort will be treated as a formatted column locate specification. Furthermore, if *expression* involves a function call to a built-in function name that matches a subscripted field name in the default record, then the function call will be interpreted as a field value. To force the function call, the internal function name should be used instead (i.e. BIF_*function_name*).

By default, data records of the selected record type that have been EXCLUDED are included in the locate scan and are made visible if they satisfy *expression*.

line_flag

line_flag may be specified to locate a record or record segment that has been flagged with the requested line flag.

Each valid *line_flag* keyword detailed below corresponds to a built-in function. Because field column name location will take precedence, if the *line_flag* keyword conflicts with a *field_name* in the default record type, then the equivalent built-in function name should be used instead.

A description of each *line_flag* keyword may be found in its equivalent function description.

***line_flag* Keyword**	**Built-in Function**
ERRor	CHANGEERROR()
CHAnge | Chg	CHANGEOK()
ALTered | UPDated	CHANGED()
DATaerror	DATATYPEERROR()
DUPkey	DUPLICATEKEY()
EMPTY	EMPTYSLOT()
EXcluded | X	EXCLUDED()
LABel	HASPOINT()
COMmand | CMd	HASPREFIXCMD()
IDentify | IDrequired	IDREQUIRED()
NEW	INSERTED()
KEYChanged | KEYChg	KEYCHANGED()
LENgtherror	LENGTHERROR()
EOL | NOEOL	NOEOL()
SAVE	SAVEREQUIRED()
SQLError	SQLERROR()
TRNC | TRUNCated	TRUNCATED()
VALERRor	VALUEERROR()

ALL

Locate all records that match the specified *expression* expression and position the display at the first matching record. All records for which *expression* test "true" are made visible if NX is not specified.

FIRST

Search forwards from the first record in the file with the selected record type to locate the first record that satisfies the specified *expression*.

LAST

Search backwards from the last record in the file with the selected record type to locate the last record that satisfies the specified *expression*.

NEXT

Search forwards from the focus line to locate the next record that satisfies the specified *expression*.

`PREV`
Search backwards from the focus line to locate the previous record that satisfies the specified *expression*.

`EX`
`X`
Locate EXCLUDED data records only.
LOCATE does not search records that are NOTSELECTED or SUPPRESSED.

`NX`
Locate only visible data records (i.e. not EXCLUDED).
LOCATE does not search records that are NOTSELECTED or SUPPRESSED.

Formatted Column Locate Options

Formatted column locate is applicable to formatted records only and will scroll the display to a nominated field within the default record only.

If the field is an array of repeating elements, then a field subscript must be supplied in parentheses "( )" identifying the individual array field element to be located. Array field co-ordinates must be specified for each dimension in the array where each co-ordinate separated is separated from the next by a comma (,). e.g. #12(1,2) identifies an element of a 2-dimensional array.

`field_name`
The name of a field in the default record. *field_name* may be fully qualified, partially qualified or unqualified (see Field Value expression terms), and may include a subscript array element reference. e.g. Member(3)

Regardless of whether the ABBREVIATION option has been set on, the group name or field item designators that constitute *field_name* may be abbreviated, starting with the first letter of the designator. If *field_name* identifies more than one field, then only the first occurrence of a field that matches *field_name* will be located.

`field_ref`
A field reference number in the default record. *field_ref* may include a subscript array element reference. e.g. #22(3)

`FIRST`
Search forwards from the first field in the default record to locate the first field that matches the specified field name or reference.

`LAST`
Search backwards from the last field in the default record to locate the last field that matches the specified field name or reference.

`NEXT`
Search forwards from the field that is currently displayed first in the visible display, to locate the next field that matches the specified field name or reference.

`PREV`
Search backwards from the field that is currently displayed first in the visible display, to locate the previous field that matches the specified field name or reference.

Examples:

`locate 3`
Scroll the display area so that the line that is 3 lines below the focus line becomes the new current line.

`-18`
Scroll the display area so that the line that is 18 lines above the focus line becomes the new current line.

`:18`
Scroll the display area so that the visible record at line number 18 becomes the new current line.

```
locate  (LastName = 'Jones' &  Salary > '21950') | (Dept \>> 'Tech')  prev
```
Scroll the display area so that the first record before the focus line that is of the default record type and also satisfies the specified expression, becomes the new current line.

```
locate  #15
```
Scroll to field number 15 in the default record.

See Also:

DOWN FIND UP WHERE

MACRO

Syntax:

```
>>--- MACRO ---- macro_name ---+---------------+----------------------------><
                               |               |
                               +- parm_string -+
```

Description:

Force SDE to execute the REXX macro specified by *macro_name* as opposed to an SDE command of the same name.

By default, SDE first checks a user supplied token for a recognised SDE command. If the token is not recognised as a command, an attempt is made to find a macro with the same name. Therefore, when invoking a macro, the command verb MACRO may not be necessary.

Any text specified following the macroname is passed to the macro as a parameter string. The specified macro is read into memory from disk, executed and removed from memory on completion.

If the macro cannot be located, an error message is issued and the MACRO command fails.

Parameters:

macro_name
Name of the macro to be executed.

If *macro_name* is a full fileid containing file name, path, etc., then the macro is loaded from the specified location. If only a file name is specified, each directory in the macro path is searched for a matching macro name.

parm_string
Text to be passed to the macro as a parameter string.

Examples:

`macro    cbl.dist.filekit.site.cble(xarc)    XXFLD  22 33`
Execute the macro XARC in library CBL.DIST.FILEKIT.SITE.CBLE with parameters "XXFLD 22 33".

`macro    select`
Execute the user macro SELECT from a library in the macro path, not the SDE command SELECT.

MAP

Syntax:

```
>>---+-- MAP --+------------------------------------------------------------><
     |         |
     +-- FMT --+
```

Description:

Display records or record segments in single record, formatted view.

If the record-type (RTO) assigned to each record or record segment has been disabled (record data is unformatted), then MAP will re-enable the assigned record-types to reformat the record data.

Compare with FORMAT SINGLE which does not attempt to re-enable the assigned record-types.

See Also:

FORMAT   CHAR   UNFMT   VFMT   ZOOM

MAXIMUM

Syntax:

```
               +-------------------------- , -----------------------------+
               v                                                          |
>>-- MAXimum --+- field_col ----------------------------------------------+-->
               |                                                          |
               |                                        +- CHaracter --+  |
               |                                        |              |  |
               +- field_pos -- , -- field_length -- , --+--------------+--+
                                                        |              |
                                                        +- Fixed ------+
                                                        +- Binary -----+
                                                        |              |
                                                        +- PD ---------+
                                                        +- DECimal ----+
                                                        +- PACKed -----+
                                                        |              |
                                                        +- ZD ---------+
                                                        +- Zoned ------+
                                                        |              |
                                                        +- HFP --------+
                                                        |              |
                                                        +- BFP --------+
                                                        |              |
                                                        +- DFP --------+
```

```
                                  +- .ZFIRST -- .ZLAST -+
                                  |                     |
 >--+- WHere --- expression ---+--+---------------------+-------------------->
    |                          |  |                     |
    |                          |  +- .name1 -+----------+
    | +- NX -----------+       |             |          |
    | +- NOTEXcluded --+       |             +- .name2 -+
    | |                |       |
    +-+----------------+-------+
      |                |
      +- EXcluded -----+
      +- X ------------+
      |                |
      +- ALL ----------+
```

```
 >--+--------------------------------------------+--+----------------+-------><
    |                                            |  |                |
    +-------+- RECord -+--------+- record_type --+  +- STEM stemvar -+
    |       |          |        |
    +- FOR -+          +- TYPE -+
```

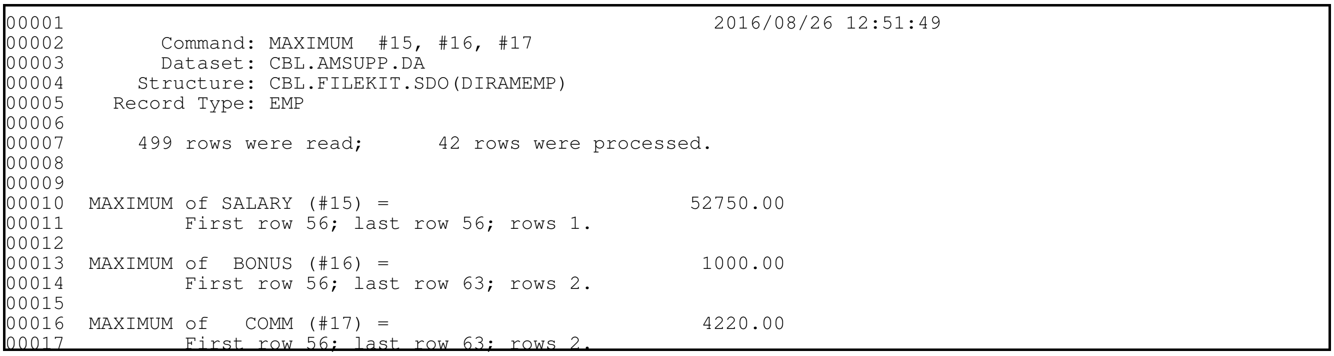
Description:

For browse or edit of formatted or unformatted data in the focus Data Editor view, MAXIMUM will display or extract the maximum value belonging to columns in unformatted records or, if formatting is applied, records of a specific record type.

Multiple columns may be specified as a mixture of formatted record column ids (names and/or reference numbers) and field positions and length. The maximum value will be obtained for each column specification.

Records may be included or excluded from the MAXIMUM operation via a WHERE filter expression or using the current excluded status of lines within the display.

Unless STEM is specified for use in REXX procedures (edit macros), the MAXIMUM values are reported in a temporary Text Edit view. e.g.



```
00001                                                      2016/08/26 12:51:49
00002        Command: MAXIMUM  #15, #16, #17
00003        Dataset: CBL.AMSUPP.DA
00004      Structure: CBL.FILEKIT.SDO(DIRAMEMP)
00005    Record Type: EMP
00006
00007      499 rows were read;      42 rows were processed.
00008
00009
00010  MAXIMUM of SALARY (#15) =                         52750.00
00011          First row 56; last row 56; rows 1.
00012
00013  MAXIMUM of  BONUS (#16) =                          1000.00
00014          First row 56; last row 63; rows 2.
00015
00016  MAXIMUM of   COMM (#17) =                          4220.00
00017          First row 56; last row 63; rows 2.
```

Following each reported maximum value, is a report line which details the first, last and total number of lines on which the maximum value was found. Additionally, for each reported column maximum, report lines may follow which detail the number of column values that were excluded from the process for the following reasons:

1. The value is null.
2. The value is invalid (not in the data type format).
3. For decimal floating point data type, the value is QNAN, SNAN or Infinity.

When the duration of a MAXIMUM execution exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the oportunity to interrupt the operation using the Attn key.

Parameters:

field_col

An individual field column id within a formatted record display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. SALARY).

If *field_col* identifies an array, then MAXIMUM treats each element of the array as a separate maximum value. To perform MAXIMUM on an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array.

field_pos

The start position of a field within the record data. For formatted records, this is a position in the expanded record.

field_length

Following *field_pos*, *field_length* defines the length of the field in the record data. The value of *field_length* must be valid for the corresponding *field_type*.

field_type

The data type of the data in the field identified by *field_pos* and *field_length*. *field_type* may be one of the following:

CHARACTER	Character - the default data type. *field_len* must be less than 32768. Numeric interpretation of character data supports use of the following: 1. Currency symbol (x'5B'). 2. A dot/period (".") representing decimal point. 3. Commas (","). 4. Character "E" (or "e") representing start of an exponent. 5. Unary plus ("+") or minus ("-") preceding the value and/or exponent value.
FIXED BINARY	Fixed Point Binary. *field_len* must be less or equal to 8.
PD DECIMAL PACKED	Packed Decimal. *field_len* must be less or equal to 16.
ZD ZONED	Zoned Decimal. *field_len* must be less or equal to 31.
HFP	Hexadecimal Floating Point. *field_len* must be 4, 8 or 16.
BFP	Binary Floating Point. *field_len* must be 4, 8 or 16.
DFP	Decimal Floating Point. *field_len* must be 4, 8 or 16.

WHERE *expression*

Only records of the specified record type *record_type* that satisfy the WHERE expression are included in the MAXIMUM operation.

expression is a valid SDE expression which supports function calls, *record_type* field names/references, sub-expressions, arithmetic, relational and logical operators. The result of *expression* must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

When an expression is applied to the record, it must test "true" in order to be selected as a target of the MAXIMUM operation.

By default, all data records in the selected range of lines are subject to the WHERE *expression* test. This includes data records that are of the selected record type and have EXCLUDED status at the time the MAXIMUM command is executed. If one of these excluded records satisfies the *expression*, it will be included in the MAXIMUM calculation.

EXCLUDED
X

Only records assigned the specified record type *record_type* and have EXCLUDED status are included in the MAXIMUM operation. By default, only non-EXCLUDED records are selected.
MAXIMUM does not include records that have the status NOTSELECTED or SUPPRESSED.

NOTEXCLUDED
NX

Only records assigned the specified record type *record_type* and have non-EXCLUDED status are included in the MAXIMUM operation. This is the default action.
MAXIMUM does not include records that have the status NOTSELECTED or SUPPRESSED.

`ALL`

All records (EXCLUDED and non-EXCLUDED) assigned the specified record type *record_type* are included in the MAXIMUM operation. By default, only non-EXCLUDED records are selected.
MAXIMUM does not include records that have the status NOTSELECTED or SUPPRESSED.

.name1

A label name identifying the first line of a range of lines on which the MAXIMUM operation will operate. The preceding "." (dot) is mandatory.
Default is .ZFIRST.

.name2

A label name identifying the last line of a range of lines on which the MAXIMUM operation will operate. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then *.name2* and *.name1* identify the first and last rows respectively.
Default is .ZLAST.

`[FOR] RECORD [TYPE]` *record_type*

Identifies the record type mapping *record_type* used by the MAXIMUM operation. Only records assigned this record type are eligible to be included in the MAXIMUM calculation.

If a *field_col* is specified, it must reference a column field in this record type definition.

Default is the default record type.

`STEM` *rexx_stemvar*

Applicable only to execution of MAXIMUM within a REXX procedure edit macro, STEM indicates that the MAXIMUM value for each of the specified columns be assigned to a REXX compound variable.

The mandatory *rexx_stemvar* argument specifies the character string to be used as the stem part of the compound variable name.

The following REXX compuond variables are set:

rexx_stemvar`.MAX.0`

The number of MAXIMUM values returned (i.e. number of columns specified).

rexx_stemvar`.MAX.i`

The ith MAXIMUM value corresponding to the ith column specification.

See Also:

AVERAGE MINIMUM SUM TOTALS

MINIMUM

Syntax:

```
               +-------------------------- , -----------------------------+
               v                                                          |
>>-- MINimum --+- field_col ----------------------------------------------+-->
               |                                                          |
               |                                        +- CHaracter --+  |
               |                                        |              |  |
               +- field_pos -- , -- field_length -- , --+--------------+--+
                                                        |              |
                                                        +- Fixed ------+
                                                        +- Binary -----+
                                                        |              |
                                                        +- PD ---------+
                                                        +- DECimal ----+
                                                        +- PACKed -----+
                                                        |              |
                                                        +- ZD ---------+
                                                        +- Zoned ------+
                                                        |              |
                                                        +- HFP --------+
                                                        |              |
                                                        +- BFP --------+
                                                        |              |
                                                        +- DFP --------+


                                  +- .ZFIRST -- .ZLAST -+
                                  |                     |
 >--+- WHere --- expression ---+--+---------------------+-------------------->
    |                          |  |                     |
    |                          |  +- .name1 -+----------+
    | +- NX -----------+       |             |          |
    | +- NOTEXcluded --+       |             +- .name2 -+
    | |                |       |
    +-+----------------+-------+
      |                |
      +- EXcluded -----+
      +- X ------------+
      |                |
      +- ALL ----------+


 >--+--------------------------------------------+--+----------------+-------><
    |                                            |  |                |
    +-------+- RECord -+--------+- record_type --+  +- STEM stemvar -+
    |       |          |        |
    +- FOR -+          +- TYPE -+
```

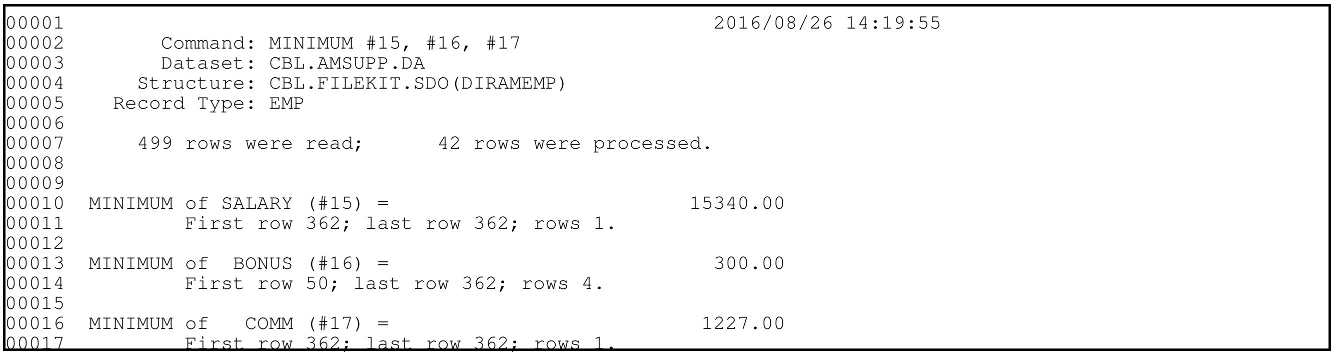
Description:

For browse or edit of formatted or unformatted data in the focus Data Editor view, MINIMUM will display or extract the minimum value belonging to columns in unformatted records or, if formatting is applied, records of a specific record type.

Multiple columns may be specified as a mixture of formatted record column ids (names and/or reference numbers) and field positions and length. The minimum value will be obtained for each column specification.

Records may be included or excluded from the MINIMUM operation via a WHERE filter expression or using the current excluded status of lines within the display.

Unless STEM is specified for use in REXX procedures (edit macros), the MINIMUM values are reported in a temporary Text Edit view. e.g.



```
00001                                                      2016/08/26 14:19:55
00002        Command: MINIMUM #15, #16, #17
00003        Dataset: CBL.AMSUPP.DA
00004      Structure: CBL.FILEKIT.SDO(DIRAMEMP)
00005    Record Type: EMP
00006
00007      499 rows were read;      42 rows were processed.
00008
00009
00010  MINIMUM of SALARY (#15) =                         15340.00
00011          First row 362; last row 362; rows 1.
00012
00013  MINIMUM of  BONUS (#16) =                           300.00
00014          First row 50; last row 362; rows 4.
00015
00016  MINIMUM of   COMM (#17) =                          1227.00
00017          First row 362; last row 362; rows 1.
```

Following each reported minimum value, is a report line which details the first, last and total number of lines on which the minimum value was found. Additionally, for each reported column minimum, report lines may follow which detail the number of column values that were excluded from the process for the following reasons:

1. The value is null.
2. The value is invalid (not in the data type format).
3. For decimal floating point data type, the value is QNAN, SNAN or Infinity.

When the duration of a MINIMUM execution exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the oportunity to interrupt the operation using the Attn key.

Parameters:

field_col

An individual field column id within a formatted record display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. SALARY).

If *field_col* identifies an array, then MINIMUM treats each element of the array as a separate minimum value. To perform MINIMUM on an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array.

field_pos

The start position of a field within the record data. For formatted records, this is a position in the expanded record.

field_length

Following *field_pos*, *field_length* defines the length of the field in the record data. The value of *field_length* must be valid for the corresponding *field_type*.

field_type

The data type of the data in the field identified by *field_pos* and *field_length*. *field_type* may be one of the following:

CHARACTER	Character - the default data type. *field_len* must be less than 32768. Numeric interpretation of character data supports use of the following: 1. Currency symbol (x'5B'). 2. A dot/period (".") representing decimal point. 3. Commas (","). 4. Character "E" (or "e") representing start of an exponent. 5. Unary plus ("+") or minus ("-") preceding the value and/or exponent value.
FIXED BINARY	Fixed Point Binary. *field_len* must be less or equal to 8.
PD DECIMAL PACKED	Packed Decimal. *field_len* must be less or equal to 16.
ZD ZONED	Zoned Decimal. *field_len* must be less or equal to 31.
HFP	Hexadecimal Floating Point. *field_len* must be 4, 8 or 16.
BFP	Binary Floating Point. *field_len* must be 4, 8 or 16.
DFP	Decimal Floating Point. *field_len* must be 4, 8 or 16.

WHERE *expression*

Only records of the specified record type *record_type* that satisfy the WHERE expression are included in the MINIMUM operation.

expression is a valid SDE expression which supports function calls, *record_type* field names/references, sub-expressions, arithmetic, relational and logical operators. The result of *expression* must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

When an expression is applied to the record, it must test "true" in order to be selected as a target of the MINIMUM operation.

By default, all data records in the selected range of lines are subject to the WHERE *expression* test. This includes data records that are of the selected record type and have EXCLUDED status at the time the MINIMUM command is executed. If one of these excluded records satisfies the *expression*, it will be included in the MINIMUM calculation.

EXCLUDED
X

Only records assigned the specified record type *record_type* and have EXCLUDED status are included in the MINIMUM operation. By default, only non-EXCLUDED records are selected.
MINIMUM does not include records that have the status NOTSELECTED or SUPPRESSED.

NOTEXCLUDED
NX

Only records assigned the specified record type *record_type* and have non-EXCLUDED status are included in the MINIMUM operation. This is the default action.
MINIMUM does not include records that have the status NOTSELECTED or SUPPRESSED.

`ALL`

All records (EXCLUDED and non-EXCLUDED) assigned the specified record type *record_type* are included in the MINIMUM operation. By default, only non-EXCLUDED records are selected.
MINIMUM does not include records that have the status NOTSELECTED or SUPPRESSED.

.name1

A label name identifying the first line of a range of lines on which the MINIMUM operation will operate. The preceding "." (dot) is mandatory.
Default is .ZFIRST.

.name2

A label name identifying the last line of a range of lines on which the MINIMUM operation will operate. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then *.name2* and *.name1* identify the first and last rows respectively.
Default is .ZLAST.

`[FOR] RECORD [TYPE]` *record_type*

Identifies the record type mapping *record_type* used by the MINIMUM operation. Only records assigned this record type are eligible to be included in the MINIMUM calculation.

If a *field_col* is specified, it must reference a column field in this record type definition.

Default is the default record type.

`STEM` *rexx_stemvar*

Applicable only to execution of MINIMUM within a REXX procedure edit macro, STEM indicates that the MINIMUM value for each of the specified columns be assigned to a REXX compound variable.

The mandatory *rexx_stemvar* argument specifies the character string to be used as the stem part of the compound variable name.

The following REXX compuond variables are set:

rexx_stemvar`.MIN.0`

The number of MINIMUM values returned (i.e. number of columns specified).

rexx_stemvar`.MIN.i`

The ith MINIMUM value corresponding to the ith column specification.

See Also:

AVERAGE MAXIMUM SUM TOTALS

MORE

Syntax:

```
>>--- MORE ---+-- where_clause --+------------------------------------------><
              |                  |
              +-- line_flag -----+
```

Description:

Make visible any EXCLUDED records that are of the default record type and also satisfy the specified *where_clause* criteria. The current line of the display and any records that are already visible are unaffected by the MORE command.

If MORE is executed with no parameters, then all EXCLUDED records that are of the default record type are made visible.

The record type used on a MORE command becomes the new value of the DRECTYPE option.

Parameters:

where_clause

where_clause syntax is described fully in documentation for the WHERE command.

line_flag

line_flag may be specified to locate a record or record segment that has been flagged with the requested line flag.

Each valid *line_flag* keyword detailed below corresponds to a built-in function.

A description of each *line_flag* keyword may be found in its equivalent function description.

line flag Keyword	Built-in Function
ERRor	CHANGEERROR()
CHAnge | Chg	CHANGEOK()
ALTered | UPDated	CHANGED()
DATaerror	DATATYPEERROR()
DUPkey	DUPLICATEKEY()
EMPTY	EMPTYSLOT()
EXcluded | X	EXCLUDED()
LABel	HASPOINT()
COMmand | CMd	HASPREFIXCMD()
IDentify | IDrequired	IDREQUIRED()
NEW	INSERTED()
KEYChanged | KEYChg	KEYCHANGED()
LENgtherror	LENGTHERROR()
EOL | NOEOL	NOEOL()
SAVE	SAVEREQUIRED()
SQLError	SQLERROR()
TRNC | TRUNCated	TRUNCATED()
VALERRor	VALUEERROR()

Examples:

```
more     #5 >> X'41'
```
Make visible any EXCLUDED records that are of the default record type where field reference number 5 begins with X'41' (ASCII character 'A').

```
more     LastName = C'Evans'
```
Make visible any EXCLUDED records that are of the default record type where the contents of field "LastName" is equal "Evans".

See Also:

FLIP LESS WHERE

MSG

Description:

SDE CLI command, MSG, performs the same operation as the CBLe CLI command MSG. See MSG in CBLe Text Edit documentation.

NEXT

Syntax:

Segmented Record Edit:

```
              +-- Segment --+  +- 1 -----+  +- * -----------+
              |             |  |         |  |               |
>>- Next --+--+-------------+--+---------+--+---------------+--+-------------><
           |  |             |  |         |  |               |  |
           |  +-- Base -----+  +- n_seg -+  +- record_type -+  |
           |                                |               |  |
           |                                +- / -----------+  |
           |                                +- \ -----------+  |
           |                                |               |  |
           |                                +- ? -----------+  |
           |                   +- 1 -----+                     |
           |                   |         |                     |
           +----- Unmapped ----+---------+---------------------+
                               |         |
                               +- n_seg -+
```

Non-Segmented Record Edit:

```
              +-- Base -----+  +- 1 -----+  +- * -----------+
              |             |  |         |  |               |
>>- Next --+--+-------------+--+---------+--+---------------+--+-------------><
           |                   |         |  |               |  |
           |                   +- n_seg -+  +- record_type -+  |
           |                                |               |  |
           |                                +- / -----------+  |
           |                                +- \ -----------+  |
           |                   +- 1 -----+                     |
           |                   |         |                     |
           +----- Unmapped ----+---------+---------------------+
                               |         |
                               +- n_seg -+
```

Description:

When used in an SDE BROWSE/EDIT view of segmented records, NEXT scrolls to a segment that follows the current segment, making it the new current segment. For non-segmented record BROWSE/EDIT, every record is considered to be a primary (base) segment with no secondary segments. Therefore, using NEXT SEGMENT to scroll to secondary segments is invalid and will always return ZZSD441E error message.

In multi-record view, the view of the data within the SDE window is scrolled down to the next selected segment. In single record view, NEXT will display the next selected segment.

NEXT scrolls only to segments of a specific type (primary, secondary or unmapped) and, optionally, segments assigned a specifically named record type (RTO). In contrast, DOWN (in multi-record view) and RIGHT (in single-record view) scroll to records or record segments that follow, regardless of segment type and assigned RTO.

NEXT SEGMENT (the default for segmented record display) restricts scrolling to only those secondary segments belonging to the current segmented record. If the specified secondary segment cannot be found within the current record (e.g. NEXT 6 for a record containing only 5 secondary segments), then the last segment to match the NEXT parameter criteria becomes the current segment and the following message is returned:

```
  ZZSD441E Base record exhausted. Final match (#nn of mm) displayed.
```

Where *nn* is the last occurrence of the segment assigned record type *record_type* to be found following the current segment, and *mm* is the requested number of matching segments to be scrolled (*n_seg*).
If using the default parameter values to simply scroll to the next secondary segment, this message will contain `(#0 of 1)` where an attempt has been made to scroll to the first secondary segment following the last segment in the record.

Parameters:

`SEGMENT | BASE | UNMAPPED`

Specifies the type of segment to be selected as the current segment.

SEGMENT indicates a secondary segment, BASE indicates a primary (base) segment and UNMAPPED indicates an unformatted (primary or secondary) segment. Note that unformatted primary segments (i.e. records) have record type "Unmapped" and unformatted secondary segments have record type "UnmappedSeg".

Default is SEGMENT for segmented record browse/edit and BASE otherwise.

n_seg

Specifies the occurrence of a segment following the current segment that matches the optionally specified *record_type*. Default is 1.

`* | record_type | / | \ | ?`

Specifies the record type (RTO) of the segment that is to become the new current segment. Only segments assigned this record type will be included in the count of segments to be scrolled.

"*" (asterisk) indicates that segments may be of any record type. *record_type* nominates a specific record type. "/" or "\" indicates that segments are to be of the record type assigned to the current segment. "?" indicates that segments are to be of the record type **not** assigned to the current segment.

Default is *.

Examples:

`NEXT`

For segmented record browse/edit, scroll to the secondary segment within the same record that immediately follows the current segment. Fot non-segmented browse/edit, scroll to the next record.

`N B`

Scroll to the next primary segment (record).

`N B 2 /`

Scroll forwards to the 2nd occurrence of a primary segment (record) which is assigned the same record type as the current segment.

`N U`

Scroll forwards to the next unformatted primary or secondary segment.

See Also:

PREVIOUS LEFT RIGHT

NOMSG

Description:

SDE CLI command, NOMSG, performs the same operation as the CBLe CLI command NOMSG. See NOMSG in CBLe Text Edit documentation.

NOND

Syntax:

```
>>-- NOND ------------------------------------------------------------------><
```

Description:

Supplied as a text edit macro, NOND uses the SET COLOUR NONDISPLAY option to toggle the display of underscores on printable text in lines of data which include unprintable characters.

ONLY

Syntax:

```
                 +- EQ -+           (1) +- ANY ---+  +- CHARs --+
                 |      |               |         |  |          |
>>-+- Only -+--+-+------+- string --+---+---------+--+----------+--+------+-->
               | |      |           |   |         |  |          |  |      |
               | +- op -+               +- FOCus -+  +- PREfix -+  +- EX -+
               |                    |                |          |  |      |
               +- VALID ------------+                +- SUFfix -+  +- NX -+
               |                    |                |          |  |      |
               +- INVALID ----------+                +- WORD ---+  +- X --+

   +-- #ALL ------------------------------------+ +- .ZFIRST ---- .ZLAST -+
   |                                            | |                       |
 >-+--------------------------------------------+-+-----------------------+--><
   |                                            | |                       |
   +-- pos1 ---+---------+----------------------+ +- .name1 --+-----------+
   |           |         |                      |             |           |
   |           +- pos2 --+                      |             +- .name2 --+
   |                                            |
   |      +----+---------+----------------+     |
   |      |    |         |                |     |
   |      |    +-- , ----+                |     |
   |      v                               |     |
   +-- ( -+-- field_col ------------------+- ) -+
          |                               |
          +-- field_col1:field_col2 ------+
```

(1) Default (ANY or FOCUS) set by the RTSCOPE option.

Description:

Based on the FIND ALL command, ONLY displays all DB2 table rows, records or record segments that satisfy the specified search *string* criteria. All records or segments that do **not** satisfy the search *string* criteria are excluded.

If the **FOCUS** option is specified then only records/segments assigned the default record type (visible and EXCLUDED records) are included in the process. Otherwise all records types are processed.

All occurrences of the search string or numeric value are highlighted in the text. (Enter the RESET FIND command to turn off the highlighting.)

When the execution time of an ONLY command exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the oportunity to interrupt the operation using the Attn key.

The FORMAT of the SDE display affects the execution of ONLY.

Unformatted Multi or Single Record Display (CHAR or UNFMT):

By default, a character compare for the supplied search string is performed against the entire length of unformatted data records.

The prevailing BOUNDS left and right column values define the area of the record within which the search occurs. i.e. the matched data must begin at or after the left bound and not exceed the right bound.

The BOUNDS columns may be overridden using *pos1* and *pos2* positional parameters.

field_col and #ALL parameters are not applicable to these display formats and are ignored.

Formatted Multi or Single Record Display (VFMT or FMT):

For formatted records, the search string is compared against **individual fields** that have been selected for display in the formatted data record. (See SELECT)

Fields are searched from left to right in the order that they appear in the display. This is true regardless of the order in which field columns are specified on the ONLY command, or the order in which fields are encountered within the unformatted record.

The prevailing BOUNDS left and right column values define which of the fields within the formatted records are eligible to be searched. i.e. Fields are eligible only if they exist within the left and right bounds when applied to the expanded record data (which is not necessarily the same as the unformatted record data.)

Where a BOUNDS column occurs within a field definition, then the following rules apply:

1. For character data type fields, only the area of the field that falls within the BOUNDS columns is eligible for the search.
2. For numeric data type fields, the field is not eligible to be included in the search.

If one or more field columns are specified on the ONLY command (using *field_col* or *field_col1*:*field_col2*) that do not reference at least one field defined by the BOUNDS columns as being eligible for search, then the following error message is returned:

```
ZZSD280E No fields selected within the current bounds for the ONLY command.
```

If a field column is specified on the ONLY command that is not part of the display (e.g. a group field or a field that has been removed from display by a SELECT command), then the following error message is returned:

```
ZZSD179E Data element field_col is not selected in the current view
    of record type record-type of structure struct_name.
```

For character data type fields, a string compare is performed. For numeric data type fields (binary, packed decimal, floating point, zoned, etc.), then the following will occur:

1. If *pos1*, *pos2* positional parameters are **not** specified, the search string is interpreted as a signed numeric value and an arithmetic compare is performed against the field's formatted numeric value.

   The length and data type of the numeric field, and the number of digits in the search value are not significant. e.g.

   ```
   ONLY  67
   ```

   Finds numeric fields with value "67" (e.g. "0067", "67.00", "0.0670E+03") and character fields containing the string "67" (e.g. "167 Baker Street").

   If the search string is non-numeric, then numeric data type fields are not searched. Therefore, to bypass searching numeric data type fields, explicitly set the search string to be character or hex using 'string', C'string' or X'string' formats respectively. e.g.

   ```
   ONLY   '67'
   ONLY  C'67'
   ONLY  X'F6F7'
   ```

   Finds only character fields containing the string "67".

2. If *pos1*, *pos2* positional parameters are specified, the search string is interpreted as being a character string and so a string compare is performed against the unformatted representation of the field data for fields falling entirely or partly within the range of record positions.

   Although the range of positions may span a number of fields, the search is still performed against individual fields within the range. i.e. a match for the search string will not occur for data that spans a field boundary.

   A match for the search string may occur on just part of the unformatted data representation of a numeric field. e.g.

   ```
   ONLY    476    21  100
   ```

   This will find a match in any numeric field where unformatted representation of the data contains the string "476". (e.g. a zoned decimal field with value "14760" or "-4762")

Any match of the search string on data in a numeric field will highlight the entire formatted display of the field. If the numeric field is also the current, identified occurrence of the search string, the cursor is positioned at the start of the formatted numeric field display.

Parameters:

op

A relational operator used in the compare operation which determines the relationship that the data must have with the ONLY search string in order for it to be identified as a successful match.

Valid values for *op* are as follow:

Operator	Description
EQ	Data must be equal to *string*. (Default)
NE	Data must be not equal to *string*.
GT	Data must be greater than *string*.
GE	Data must be greater than or equal to *string*.
LT	Data must be less than *string*.
LE	Data must be less than or equal to *string*.

If a character string compare is performed, the EBCDIC values assigned to characters in the search and data strings determine the relationship (equal to, greater than or less than) between the two strings.

string

The ONLY search string. The search string may be any of the following:

- An unquoted numeric value. The search string is treated as a numeric value when a numeric field is searched in a formatted record view. In all other cases a numeric search string is treated as a character string.

◊ An unquoted character string containing no commas or blanks. The search for the character string will be case-insensitive so that uppercase and lowercase characters are treated as being the same.

◊ A character string enclosed in single (') or double (") quotation marks. The search string may contain embedded commas and blanks and the character string will be case-insensitive.

Two adjacent quotation mark characters that are embedded in a search string which is enclosed by the same quotation mark characters, will be treated as a single occurrence of the character. e.g.

```
ONLY  'Jim O''Brien'
```

Find the character string "Jim O'Brien".

◊ A character string enclosed in single (') or double (") quotation marks with the prefix C. This is equivalent to specifying a quoted search string but that the string search will be case-sensitive. (e.g. C'Book')

◊ A hexadecimal string enclosed in single (') or double (") quotation marks with the prefix X.

◊ A picture string enclosed in single (') or double (") quotation marks with the prefix P.

Picture strings use special characters to represent a generic group of characters as described below. Any character in a picture string that is not one of these special characters is untranslated.

String	Description
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.

◊ A regular expression enclosed in single (') or double (") quotation marks with the prefix R.

Regular expressions use special characters for complex pattern matching. See Regular Expressions in Text Editor documentation for detailed description.

If a no search string is specified and there are no other parameters, then the command is treated as RESET EXCLUDED.

However, if a no search string is specified but one or more *field_col* is supplied then it is treated as a special case that means accept any value in the specified field column(s). For ONLY, this is a way of displaying all occurrences of any record that includes the specified field column(s).

VALID

Intended for use with formatted records, VALID will search fields for valid data. i.e. data that satisfies the field's assigned data type.

INVALID

Intended for use with formatted records, INVALID will search fields for invalid data. i.e. data that does not satisfy the field's assigned data type.

ANY

All record-types are included in the search provided they are not suppressed. See VIEW , VBASE , and V/V+/V- line-commands to suppress and unsuppress record-types.

FOCUS

Only the default record type is included in the search. This is normally the type of record at the top of the screen or at the cursor location. This option was the default in earlier versions of FileKit and can be made so again using the RTSCOPE option or via the settings panel (=0.4).

CHARS

For non-numeric search strings only, CHARS indicates that a successful match occurs if the search string is found anywhere within the data being searched.

PREFIX

For non-numeric search strings only, PREFIX indicates that a successful match only occurs if the search string is found at the start of a word within the data being searched. i.e. the matched text must precede an alphanumeric character and either be preceded by a non-alphanumeric character or be at the start of a line or field.

SUFFIX

For non-numeric search strings only, SUFFIX indicates that a successful match only occurs if the search string is found at the end of a word within the data being searched. i.e. the matched text must be preceded by an alphanumeric character and either precede a non-alphanumeric character or be at the end of a line or field.

WORD

For non-numeric search strings only, WORD indicates that a successful match only occurs if the search string is found to be a complete word within the data being searched. i.e. the matched text must either be preceded by a non-alphanumeric character or be at the start of a line or field, and either precede a non-alphanumeric character or be at the end of a line or field.

`EX`
`X`

Search EXCLUDED data records only.
ONLY does not search records that are NOTSELECTED or SUPPRESSED.

`NX`

Search only visible data records (i.e. not EXCLUDED).
ONLY does not search records that are NOTSELECTED or SUPPRESSED.

pos1

The first position of a range of positions within the data record to be searched.

For formatted records, this is a position in the expanded record . Only those fields, or parts of fields, that fall within the position range will be searched. Fields will be searched in the order that they occur within the display area.

pos1 may be a positive or negative integer value (not zero) and must be a value that is less than or equal to the maximum length of the data records or, for formatted record data, the length of the expanded record.

A negative value represents a position in the record relative to the end of the record. Therefore, where position 1 references the 1st character in the record, position -1 references the last character.

For all display formats, the string is treated as being non-numeric and the search is actioned on the character representation of the record data.

pos2

The last position of a range of positions within the data record to be searched.

Like *pos1*, *pos2* may be a positive or negative integer value (not zero). If *pos1* references a position within the record data which is higher than that referenced by *pos2*, then the *pos1* and *pos2* values are swapped.

If *pos2* is greater than the maximum length of the data records or, for formatted record data, greater than the length of the expanded record, then *pos2* is set equal to the maximum (or expanded) record length.

Default is *pos1* plus the length of the search string minus 1.

`#ALL`

Search all eligible field columns in the current formatted display.

field_col

An individual field column to be searched within a formatted display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID). If a field name is used, enclosing parentheses are **mandatory** Field names are automatically converted to a field reference and Referencing the same field column more than once will not cause an error.

If the field is an array, then all elements of the array are searched. To search an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array. An entry must exist for each dimension of the array. e.g. RoomSize(6,2,4) - a three dimensional array.

Specification of multiple field columns must either be enclosed in parentheses and/or separated by commas. The field columns may be specified in any order, however, the data is always searched from the first column in the display to the last.

Field column search specifications may be a combination of individual field columns and columns ranges. e.g. (JobID #6 Tax_Reference #12 #15:#20).

A search is only performed on field columns that are selected for display. Therefore, any field column specified on the ONLY command that has not been selected for display, will be ignored.

field_col1:*field_col2*

The first and last fields of a range of field columns to be searched within a formatted record display display.

field_col1 and *field_col2* must be separated by ":" (colon) and if *field_col1* occurs after *field_col2* in the data record, then the values are swapped. *field_col1* and *field_col2* have the same specifications as *field_col*.

Specification of multiple field column ranges must either be enclosed in parentheses and/or separated by commas. Field column search specifications may be a combination of individual field columns and field columns ranges. e.g. (FirstName:LastName, #2, Salary:Bonus, EmpNo, #10:#15). If field names are used, enclosing parentheses are **mandatory**.

.*name1*

A label name identifying the first record of a range of data records to be searched. The preceding "." (dot) is mandatory. Default is .ZFIRST.

.*name2*

A label name identifying the last record of a range of data records to be searched. The preceding "." (dot) is mandatory. If *.name2* occurs before *.name1* in the display, then the order is reversed.

See Also:

EXCLUDE FIND SELECT

OPTIONS

Description:

OPTIONS displays the current setting of all SDE options in a message window.

OPTIONS is equivalent to executing QUERY *option* for every SET/QUERY/EXTRACT option supported by SDE.

PASTE

Syntax:

```
              +-- Keep ----+
              |            |
>>-- PASTE ---+------------+-------------------------------------------------><
              |            |
              +-- Delete --+
```

Description:

PASTE may be used instead of primary command CLIPBOARD PASTE to paste and optionally clear lines of data from the FileKit clipboard to the current data edit view.

Text from the clipboard is copied to a target line determined by the first occurrence in the text edit view of the line (prefix area) command "A" (after) or "B" (before). Otherwise, the text is copied following the focus line.

If records in the current data edit view are formatted, then the newly pasted records are assigned a suitable record type before being displayed.

Parameters:

`KEEP`

Text in the clipboard is preserved following the PASTE operation. This is default.

`DELETE`

Text in the clipboard is deleted following the PASTE operation.

PASTE DELETE is equivalent to excecuting CLIPBOARD PASTE followed by CLIPBOARD CLEAR.

See Also:

CLIPBOARD COPY CREATE CUT REPLACE

PERMANENT

Syntax:

```
>>-- PERManent -- command ---------------------------------------------------><
```

Description:

PERMANENT is a modal command that may be used as a prefix to SDE primary commands SELECT and SET COLWIDTH. PERMANENT will execute the SELECT or SET COLWIDTH command and save its settings in the FileKit SDO structure used to display the formatted data.

Parameters:

`command`

SDE primary command SELECT or SET COLWIDTH.

Example:

```
permanent   set colwidth #3 15
```

Set the column width for field reference number 3 to be 15 characters and save this setting the the prevailing SDO structure file.

See Also:

SELECT   SET COLWIDTH   TEMPORARY

POP

Syntax:

```
>>-- POP --------------------------------------------------------------------><
```

Description:

POP is used in SDE REXX macros to reset user configurable SDE SET options that have been saved by a previously executed PUSH operation. The reset includes all options that may be set at the Global, File and View levels.

See Also:

PUSH

PREVIOUS

Syntax:

Segmented Record Edit:

```
                  +-- Segment --+  +- 1 -----+  +- * -----------+
                  |             |  |         |  |               |
>>- Previous --+--+-------------+--+---------+--+---------------+--+---------><
               |  |             |  |         |  |               |  |
               |  +-- Base -----+  +- n_seg -+  +- record_type -+  |
               |                                |               |  |
               |                                +- / -----------+  |
               |                                +- \ -----------+  |
               |                   +- 1 -----+                     |
               |                   |         |                     |
               +----- Unmapped ----+---------+---------------------+
                                   |         |
                                   +- n_seg -+
```

Non-Segmented Record Edit:

```
                  +-- Base -----+  +- 1 -----+  +- * -----------+
                  |             |  |         |  |               |
>>- Previous --+--+-------------+--+---------+--+---------------+--+---------><
               |                   |         |  |               |  |
               |                   +- n_seg -+  +- record_type -+  |
               |                                |               |  |
               |                                +- / -----------+  |
               |                                +- \ -----------+  |
               |                   +- 1 -----+                     |
               |                   |         |                     |
               +----- Unmapped ----+---------+---------------------+
                                   |         |
                                   +- n_seg -+
```

Description:

When used in an SDE BROWSE/EDIT view of segmented records, PREVIOUS (or PREV) scrolls to a segment that occurs before the current segment, making it the new current segment. For non-segmented record BROWSE/EDIT, every record is considered to be a primary (base) segment with no secondary segments. Therefore, using PREV SEGMENT to scroll to secondary segments is invalid and will always return ZZSD441E error message.

In multi record view, the view of the data within the SDE window is scrolled upwards to a previous segment. In single record view, PREV will display a previous segment.

PREV scrolls only to segments of a specific type (primary, secondary or unmapped) and, optionally, segments assigned a specifically named record type (RTO). In contrast, UP (in multi-record view) and LEFT (in single-record view) scroll to preceding records or record segments, regardless of segment type and assigned RTO.

PREV SEGMENT (the default for segmented record display) restricts scrolling to only those secondary segments belonging to the current segmented record. If the specified secondary segment cannot be found within the current record (e.g. PREV 6 for a record containing only 5 secondary segments), then the last segment to match the PREV parameter criteria becomes the current segment and the following message is returned:

```
  ZZSD441E Base record exhausted. Final match (#nn of mm) displayed.
```

Where *nn* is the last occurrence of the segment assigned record type *record_type* to be found previous to the current segment, and *mm* is the requested number of matching segments to be scrolled (*n_seg*).
If using the default parameter values to simply scroll to the previous secondary segment, this message will contain `(#0 of 1)` where an attempt has been made to scroll to the first secondary segment prior to the first secondary segment in the record.

Parameters:

`SEGMENT | BASE | UNMAPPED`

Specifies the type of segment to be selected as the current segment.

SEGMENT indicates a secondary segment, BASE indicates a primary (base) segment and UNMAPPED indicates an unformatted (primary or secondary) segment. Note that unformatted primary segments (i.e. records) have record type "Unmapped" and unformatted secondary segments have record type "UnmappedSeg".

Default is SEGMENT for segmented record browse/edit and BASE otherwise.

n_seg

Specifies the occurrence of a segment preceding the current segment that matches the optionally specified *record_type*. Default is 1.

`* | record_type | / | \`

Specifies the record type (RTO) of the segment that is to become the new current segment. Only segments assigned this record type will be included in the count of segments to be scrolled.

"*" (asterisk) indicates that segments may be of any record type, *record_type* nominates a specific record type, and "/" or "\" indicates that segments are to be of the record type assigned to the current segment.

Default is *.

Examples:

`PREV`

For segmented record browse/edit, scroll to the segment within the same record that immediately precedes the current secondary segment. Fot non-segmented browse/edit, scroll to the previous record.

`P B`

Scroll to the first primary segment (record) that precedes the current segment.

`P B 2 /`

Scroll backwards to the 2nd occurrence of a primary segment (record) which is assigned the same record type as the current (primary) segment.

`P U`

Scroll backwards to the 1st unformatted primary or secondary segment that precedes the current segment.

See Also:

NEXT LEFT RIGHT

PRINT

Syntax:

```
>>--- PRINT -----------------------------------------------------------------><
```

Description:

PRINT with no parameters will open the SDE PRINT File Panel to print the contents of the focus SDE data edit or browse display.

If the focus window is not an SDE window view, the general Print File dialog panel is displayed.

If parameters are specified, then the general FileKit PRINT primary command is executed instead.

See Also:

CSVGEN XMLGEN

PRINTS

```
>>--- PRINTS ---- string ----------------------------------------------------><
```

Description:

Intended for use by FILEKITB (FileKit batch) execution, PRINTS will output a specified line of text to DD SDEPRINT, where the first character is a valid ASA print character.

PRINTS complements FILEKITB printed output (messages, etc.) which is also written to DD SDEPRINT.

Parameters:

string
Specifies the text string, with ASA character in position 1, to be written to DD SDEPRINT.

PUSH

Syntax:

```
>>-- PUSH -------------------------------------------------------------------><
```

Description:

PUSH is used in SDE REXX macros to save the current values of user configurable SDE SET options. All options that may be set at the Global, File and View levels are saved.

Saved option values may be subsequently restored using the POP operation.

See Also:

POP

PUTXML

Syntax:

```
>>-- PUTXML --- FROM --+--------+- fileid ----------------------------------->
                       |        |
                       +- FILE -+


 >------ INTO --+---------+- table_name ------+----------+- column_name ----->
                |         |                   |          |
                +- TABle -+                   +- COLumn -+


 >--------------+---------+- DOCid -+-----+- document_id -+------------------>
                |         |         |     |               |
                +- WHERE -+         +- = -+               |
                |                                         |
                +- WHERE ( where_clause ) ----------------+
 >--------------+-----------------------+---+---------+----------------------><
                |                       |   |         |
                +- DB2 -+--+-- ssn --+      +- AUDit -+
                |       |  |         |
                +- SSN -+  +- (ssn) -+
```

Description:

Applicable to DB2 base tables only, PUTXML copies an XML document from a data set, library member or HFS/ZFS file to an XML column within a specific DB2 table row.

Each execution of PUTXML will execute an immediate SQL UPDATE statement, to update the selected table row, and then COMMIT the change. If a connection to the required DB2 sub-system does not already exist, then PUTXML will start a new connection before attempting the UPDATE, and disconnect again following the COMMIT.

A data edit view of the DB2 table is not necessary. If a data edit view of the table has been opened using a non-zero value for BROWSE/EDIT option XMLLOBWIDTH, then the display of the XML column data is not updated as a result of an execution of PUTXML. However, execution of XMLBROWSE, XMLEDIT or XMLVIEW on the updated XML column entry will display the updated XML document text.

A WHERE filter must be specified which uniquely identifies the table row to which the XML document will be copied. If the filter identifies more than one table row, then the operation fails with error ZZSD711E.

Parameters:

`FROM FILE` *fileid*

Specifies the name of the source XML document data set, library member or HFS/ZFS file from which the DB2 XML column entry will be updated.

`INTO TABLE` *table_name*

Specifies the name of the target DB2 table to which the XML column belongs.

table_name may be specified with 1, 2 or 3 qualifiers representing *name*, *schema.name* or *location.schema.name* respectively. Default location is the local DB2 server for the specified sub-system (*ssn*) and the default schema is the value assigned to special register CURRENT SCHEMA (initially set to the user's SQLID).

`COLUMN` *column_name*

Identifies the name of the target DB2 column of data type XML.

If *column_name* specifies a column which is not type XML, then error ZZSD691E is returned.

`WHERE DOCID =` *document_id*

For any DB2 base table in which one or more XML columns have been created, there exists an implicit DOCID column (DB2_GENERATED_DOCID_FOR_XML) which stores a unique, integer document identifier for the XML column entries in a row.

A unique document id (*document_id*) may be specified to identify the table row to be updated. Using this syntax, specification of the WHERE keyword and/or equal to (=) operator are optional.

`WHERE (` *where_clause* `)`

Specifies a DB2 SQL WHERE clause which identifies a single row in the DB2 table into which the XML document will be copied. See "*DB2 SQL Reference*" for syntax of the *where-clause*.

`DB2 | SSN` *ssn* `|` (*ssn*)

Specifies the name of the local DB2 sub-system (*ssn*) to which a connection will be made in order to update the base table.

Before FileKit can successfully connect to a DB2 sub-system, a BIND of the FileKit DB2 plan must have been performed for that sub-system.

Default is the users default DB2 sub-system name as last entered in the DB2 Primary Options menu.

`AUDIT`

Sets DB2 auditing on for the PUTXML operation.

A new FileKit DB2 audit data set is allocated to record the results of the SQL UPDATE and COMMIT statements executed. See Audit Trail Functions for details.

Default is not to perform edit auditing.

Examples:

```
putxml  from  CBL.SOURCE.XML(XSTAT001)                      \
                into  CBL.ZZSSTATS        column ST_XML       \
                where DOCID = 12                              \
                ssn(CBLA)
```

Copy XML document from library member CBL.SOURCE.XML(XSTAT001) to the XML column, ST_XML, in row identified by document id 12 of DB2 table CBL.ZZSSTATS. Connection will be made to DB2 sub-system CBLA.

```
putxml  from  /tmp/input_books_xml                           \
                   into  CBL.ZZSBOOKS       column BOOK_REF_XML  \
                   where(AUTHOR LIKE 'Rowling, J.K.%')           \
                   ssn(LTBX)
```

Copy XML document from an HFS file into a DB2 table XML column entry for the row uniquely referenced by an SQL WHERE clause.

See Also:

GETXML

QQUIT

Syntax:

```
>>--- QQuit -----------------------------------------------------------------><
```

Description:

Close the current SDE window view only.

If additional SDE views exist for the data, then these will remain open.

If only one SDE edit view exists for the data, then any unsaved changes to the data will be discarded. However, if a non-temporary structure (SDO) is used to map the record data and changes have been made to that structure during the course of the edit session (e.g. USE WHEN), then the user will be prompted to save the changed structure to its structured data file (SDF). See command, SAVESTRUCTURE.

See Also:

END CANCEL

QUERY

Description:

See SET/QUERY/EXTRACT Options.

RCHANGE

Syntax:

```
>>-- RCHange ----------------------------------------------------------------><
```

Description:

Repeat the find and replace performed by the last CHANGE command. The RCHANGE search will be executed on records that are of the same record type as that used by the last CHANGE command.

Where the last CHANGE issued was CHANGE LAST or CHANGE PREV, RCHANGE will perform a CHANGE PREV operation, otherwise RCHANGE performs a CHANGE NEXT operation. i.e. Search forwards (NEXT) or backwards (PREV) from the current cursor location to find the next or previous occurrence of the string/value respectively.

If the cursor is not within the window's data display area, the forwards or backwards search begins at the first position of the first visible or excluded record within the display area that is of the record type used by the last CHANGE.

RCHANGE is assigned to function key **PF6** by default.

See Also:

CHANGE EXCLUDE FIND RFIND

RCOLOUR

Syntax:

```
>>-- RCOlour ---- record_type --+-----------------------------------------+-->
                                |                                         |
                                +-- IN -+---------------+-- struct_name --+
                                        |               |
                                        +-- STRUCTure --+

                        +- NONe ------+
                        |             |
 >--+--+- Blue ------+--+-------------+--- WHEN expression ---------+--------><
    |  |             |  |             |                             |
    |  +- Red -------+  +- BLInk -----+                             |
    |  |             |  |             |                             |
    |  +- Pink ------+  +- REVvideo --+                             |
    |  |             |  |             |                             |
    |  +- Green -----+  +- Uscore ----+                             |
    |  |             |                                              |
    |  +- Turquoise -+                                              |
    |  |             |                                              |
    |  +- Yellow ----+                                              |
    |  |             |                                              |
    |  +- White -----+                                              |
    |  |             |                                              |
    |  +- Default ---+                                              |
    |                                                               |
    +-- OFF --------------------------------------------------------+
```

Description:

RCOLOUR may be used to apply preferred colouring to records assigned specific record types (RTO) based on selection criteria.

Record colouring definitions specified by the RCOLOUR command are stored in the record type definition within the relevant structure (SDO). If not already loaded, the required SDO is loaded from its structured data file (SDF) and updated accordingly. If a non-temporary SDO, then this change to the SDO may be saved (written back to the SDF) by the user when the structure is dropped or on execution of the SAVESTRUCTURE command.

Any number of RCOLOUR specifications may exist for an individual record type based on different WHEN *expression* criteria. If records assigned this record type satisfy the WHEN *expression* criteria of more than one RCOLOUR definition, then the RCOLOUR definition used is the one that was defined first.

Parameters:

`record_type`
: The name of the record type (RTO) for which the colour definition will be applied.

`IN (STRUCTURE) struct_name`
: The name of the structure (SDO) in which *record_type* is defined.
Default is to the current (i.e. last referenced) structure. This is the structure used in the current SDE window view or explicitly referenced by an SDE command (e.g. USE WHEN)

`BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE | DEFAULT`
: Supported colours. If DEFAULT is specified, the default colour for record display is used.

`BLINK | REVERSE | USCORE | NONE`
: Extended highlighting. The colour may blink, be displayed in reverse video or be underlined. Default is NONE.

`WHEN expression`
: Specifies the criteria that must be satisfied before the defined colouring will occur.

expression is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of the WHEN expression must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

The WHEN expression is applied to each record assigned the record type *record_type* and, if the result is "true", the specified record colouring is applied. To apply record colouring for all records of record type *record_type*, simply specify "WHEN 1".

`OFF`
: Remove all specified record type (RTO) colouring from the SDO definition, reverting back to the default record colouring settings.

Examples:

```
<sd rcol  SRC  in CBL.SDO(ADATA)  red uscore  when (substr(#21,2) >> 'USING')
```

Define colouring of record data assigned to record type "SRC" in structure "CBL.SDO(ADATA)" when data starting in position 2 of field number 21 begins with "USING".

See Also:

DROP SAVESTRUCTURE USE and the SET/QUERY/EXTRACT Option: COLOUR

RECLEN

Syntax:

```
>>-- RECLength ---+------------+---------------------------------------------><
                  |            |
                  +-- ON ------+
                  |            |
                  +-- OFF -----+
```

Description:

RECLENGTH controls the display of the Record Information Length column.

If the records are variable length and Full Edit, KSDS Edit or Auxiliary Edit is in effect, then the length of each record displayed in this column may be altered simply by overtyping the existing value.

If the length of a record is increased, then pad characters are appended to the record. The pad character used is that defined by the current value for PAD. Reducing a record's length will truncate record data.

If a structure has been applied to the edited records, then the record type (RTO) associated with the updated record is re-evaluated and the contents of the SDE edit view updated accordingly.

Parameters:

```
ON
OFF
```

Set display of the record Length column on or off.
RECLENGTH ON and RECLENGTH OFF is equivalent to SET RECINFO ON LENGTH and SET RECINFO OFF LENGTH respectively.

Default is to toggle between RECLENGTH ON and RECLENGTH OFF.

See Also:

SET/QUERY/EXTRACT Options: LENGTH RECINFO PAD

REDIT

Syntax:

```
>>-- REdit ---+---------------------------------------------------+----------><
              |                                                   |
              +--- Parent ------+--+---------------------------+--+
              |                 |  |                           |
              +--- Dependent ---+  +- referential_constraint --+
              +--- Child -------+
```

Description:

The REDIT primary command applies only to browse and edit of DB2 tables and is used to list, edit or browse tables that are related to the DB2 table in the focus SDE view.

Relationships between tables are defined by referential constraints. A referential constraint specifies a set of columns as a foreign key in the dependent (or child) table which is a unique or primary key in the parent table.

REDIT opens an **Edit/Browse Related Table** view which is thereafter linked to the focus SDE DB2 edit/browse view from which it has originated. Closing the originating DB2 edit/browse view will close all Edit/Browse Related Table views that are linked to it.

The specified (or implied) referential constraint determines which of the related tables are to be edited or browsed by the REDIT operation. Furthermore, the focus row of the originating DB2 edit/browse view identifies the foreign or parent key value for which related rows are displayed. i.e. Only rows related to this parent/foreign key will be loaded into the related table view.

Note that the related table may be a parent table or a child table of the focus DB2 table.

REDIT with no parameters

When the REDIT command is issued with no parameters the behaviour depends on the context.

If the focus row has had an SQL relational error as a result of the last SAVE command then the relationship involved in the error will be used automatically to supply the REDIT parameters and the related table session will be created.

Otherwise the List Related Tables panel is displayed listing all the parent and dependent relationships for the current table. From this list a relationship can be chosen (with the RE prefix command) to supply the parameters to REDIT.

The RE prefix command

An RE prefix command can be used to issue the REDIT command with no parameters using the row on which it is issued as the focus.

Parameters:

PARENT
Indicates that the required related table to view or edit is a parent of the current table.

DEPENDENT
Indicates that the required related table to view or edit is a dependent of the current table.

CHILD
This is a synonym for **DEPENDENT**.

referential_constraint
The name of the referential constraint. This is the value of the RELNAME column in SYSIBM.SYSRELS identifying the relationship between parent and dependent child tables. If not supplied then the first PARENT or CHILD constraint will be used as appropriate.

REDO

Syntax:

```
>>-- REDO -------------------------------------------------------------------><
```

Description:

REDO re-applies one level of change made to the current file that was previous undone by the last execution of the UNDO command. Where UNDO has not previously been executed, REDO has no affect.

Each change level corresponds to an update of a data field within a record.

REDO is assigned to PF23 by default and may be executed repeatedly to re-apply multiple levels of changes.

Following one or more executions of UNDO, REDO will recover the changes even after the following have occured:

- The cursor position changes.
- The file is saved.
- Changes are made to other files in the file edit ring.

However, REDO is not able to recover changes following an UNDO if further changes are made to record data. (e.g. fields updated, records deleted or inserted.)

The third number following "Alt=n,n;m" on the status line indicates the number of change levels. If this number is followed by an "*" (asterisk), then it is possible to REDO previous UNDO commands.

See Also:

UNDO and the SET/QUERY/EXTRACT Option: UNDOING

REFRESH

Description:

SDE CLI command, REFRESH, performs the same operation as the CBLe CLI command REFRESH. See REFRESH in CBLe Text Edit documentation.

REPLACE

Syntax:

```
>>-- REPlace ----+----------+------+---------------------+-------------------><
                 |          |      |                     |
                 +- fileid -+      +- .name1 -- .name2 --+
                                    (1)
```

Note:

1. If a group of lines are not specified using line label names, then a group of lines must be specified using one of the following line (prefix area) commands:
   - **C** or **CC** (Copy), if the group of lines in the current file is to be preserved following successful execution of REPLACE.
   - **M** or **MM** (Move), if the group of lines in the current file is to be deleted following successful execution of REPLACE.

Description:

Replace the contents of a sequential or VSAM data set, PDS/PDSE library member or HFS file, with a group of lines extracted from the current SDE edit or browse view. The target sequential or library data set may be uncataloged.

If the target file does not already exist, then one of the following will occur:

1. If output is to a member of an existing PDS/PDSE library or to an HFS file, the target file will be created automatically created.

2. If output is to a non-existant data set or to a member of a non-existant PDS/PDSE library, the Allocate Non-VSAM panel is displayed in order to allocate the new data set.

Specify REPLACE with no parameters to open the SDE CREATE/REPLACE file panel.

Parameters:

fileid

Specifies the fileid of the target file whose records are to be replaced.

If *fileid* is a less than 8 characters in length and is a valid member name, the target file will be a member of member name *fileid* belonging to the same PDS/PDSE library referenced in the focus SDE view. If the focus SDE view does not display a library member, the *fileid* will be treated as an HFS file within the current HFS working directory.

If *fileid* includes a volume id, then the target file may be a cataloged or uncataloged data set or PDS/PDSE library which exists in that volume's VTOC. e.g. VOLWKA:DEV.UNCATLG.FILE.

.name1

A label name identifying the first line of the group of lines to be copied to the target file.
If not specified, then the group of lines must marked using "C" or "M" line (prefix area) commands.

.name2

A label name identifying the last line of the group of lines to be copied to the target file. If *.name1* has been specified, *.name2* is mandatory.

Examples:

```
replace   DEV.USER223.TESTDATA.D2012324   .CDBEG .CDEND
```

Replace the contents of data set "DEV.USER223.TESTDATA.D2012324" with records in the current SDE view which fall between defined line label names .CDBEG and .CDEND inclusively.

See Also:

COPY CREATE

REPLACELINE

Syntax:

```
>>- REPLACELine --+---------------------+---+-----------------------------+->
                  |                     |   |                             |
                  |   +- , --------+    |   |         +- , ----------+    |
                  |   v            |    |   |         v              |    |
                  +- (- field_col -+-) -+   +- Values(- field_value -+-) -+

 >---+------------------------------------+---+--------------+--------------><
     |                                   |   |              |
     +- RANDomize -+---------------------+   +- SKIPerrors -+
                   |                     |
                   |   +- , --------+    |
                   |   v            |    |
                   +- (- field_col-+- ) -+
```

Description:

Replace data in fields within the record occupying the focus line.

The record occupying the focus line must not be a shadow line and must contain any specified fields references.

REPLACELINE is primarily used within SDE Edit REXX macros.

Parameters:

`( field_col,... )`

A list of field columns into which corresponding *field_value* values are to be inserted.

The field column may be identified by its reference number (e.g. #1) or its name (e.g. SeqNUM). Specification of multiple field columns must be separated by commas and enclosed in parentheses.

If the *field_col* reference is a **struct** or a **union**, then an error message is returned. If the *field_col* reference is an array, then an individual array element must be specified in parentheses as a subscript to the field. e.g. #13(3) for the 3rd element of a single dimension array. A subscript entry must exist for each dimension of the array. e.g. RoomSize(6,2,4) - an element within a three dimensional array.

If a list of field columns is not specified then as many of the set of selected fields in the current view as there are values in the list of field values, are used as the default. In this case, the order of the field columns is equal to the order in which fields are displayed in the current view.

`VALUES ( field_value,... )`

A list of field values to be inserted into corresponding *field_col* entries in the field column list.

If one or more *field_col* entries are specified (i.e. a field list is specified), then there must be the same number of *field_value* values to correspond with each *field_col* entry.

If a character string field value contains special characters, blanks or commas, then it must be delimited by quotation mark (") or apostrope (') characters. If the value contains characters which match the delimiting characters, then each occurrence of this character must be escaped by prefixing it with the escape character. The escape character is the same as the delimiting character. e.g. VALUES('He said "It''s John''s bike."')

`RANDOMIZE`

Specified without an explicit list of columns, this option causes a new test data value to be generated for all fields that already have an existing RANDOMIZER object defined.

`RANDOMIZE ( field_col1, field_col2, ... )`

Specified with an explicit list of columns, this option causes a new test data value to be generated for the **specified fields only**.

If a specified field does not already have a RANDOMIZER object defined, then one will be automatically created using default characteristics based on the **data-type** of the field.

`SKIPERRORS`

SKIPERRORS will allow replaceline processing to continue even if an error has occurred whilst attempting to replace a field value.

Examples:

```
replaceline (NAME,#5,BONUS)    values ("Tom Jones", 62, 200.00)
```

Replace existing values in the specified fields with those supplied. Numerical data will be converted to the appropriate data type for the field.

```
replacel    rand(FIRSTNAME,LASTNAME,EMAIL,TEL-NUM)
```

Replace existing values in the specified fields with generated test data.

See Also:

INSERT

RESET

Syntax:

```
                        +- CHange -+  +- COmmand -+  +- Duplicates -+
                        |          |  |           |  |              |
>>-- RESet --+-------+--+----------+--+-----------+--+--------------+-------->
             |       |
             +- All -+

                     +- X --------+
                     |            |
        +- ERror -+  +- EXcluded -+  +- Find -+
        |         |  |            |  |        |
 >------+---------+--+------------+--+--------+--+--------+--+---------+-----><
                                                 |        |  |         |
                                                 +- HIDE -+  +- Label -+
```

Description:

Reset individual flags set on for records or record segments in the current SDE BROWSE or EDIT view.

Parameters:

`ALL`

Equivalent to: `RESET  COMMAND DUPLICATES EXCLUDED FIND LABEL CHANGE ERROR`

`CHANGE`

Remove "==CHG>" flags from the prefix area.

`COMMAND`

Remove any pending prefix area commands.

`DUPLICATES`

Reset the "=DUPE>" and "=DUPF>" line flags introduced with support for KSDS data set editing. These flags are set when an attempt is made to save changes to the data and cause the save operation to fail.
"=DUPE>" indicates that a duplicate key exists between 2 or more, new or changed records within the edit session.
"=DUPF>" indicates that a duplicate key exists between a new or changed record in the edit session and an existing file record.

`ERROR`

Remove "==ERR>" and "=ERRV>" line flags from the prefix area.
"=ERRV>" line flag indicates that the expanded record has a variable length or variable array size value error.

`EXCLUDED`
`X`

Re-include (make visible) any data records that have been previously excluded from view.

`FIND`

Remove all highlighting of search strings and numerical fields that have been found by the FIND command.

`HIDE`

Redisplay all shadow lines that have beed removed from the display by the HIDE command.
RESET HIDE is equivalent to SET SHADOW ON ALL.

`LABEL`

Remove all line label names set via the SET POINT command or entered in the prefix area directly.

RFIND

Syntax:

```
             +---- 1 ---+
             |          |
>>-- RFInd --+----------+----------------------------------------------------><
             |          |
             +- n_rpts -+
```

Description:

Repeat the operation performed by the last FIND or EXCLUDE command a number of times. Additionally, if RFIND is executed following a CHANGE operation, then FIND is performed using the search string specified on the CHANGE command.

The RFIND search will be executed on records that are of the same record type as that used by the last FIND, EXCLUDE or CHANGE command. An optional repetition factor may be specified to repeat teh operation a number of times. e.g. Following "EXCLUDE ALL" and "FIND ABC", "RFIND 20" will locate and unexclude a further 20 occurrences of the string "ABC".

Where the last FIND issued was FIND LAST or FIND PREV, RFIND will perform a FIND PREV operation, otherwise RFIND performs a FIND NEXT operation. i.e. Search forwards (NEXT) or backwards (PREV) from the current cursor location to find the next or previous occurrence of the string/value respectively.

If the cursor is not within the window's data display area, the forwards or backwards search begins at the first position of the first visible or excluded record within the display area that is of the record type used by the last FIND.

RFIND is assigned to function key **PF5** by default.

Parameters:

n_rpts
The number of times that the previous operation is to be repeated, defaulting to 1.

See Also:

RCHANGE CHANGE EXCLUDE FIND

RIGHT

Syntax:

```
>>- Right --+-------------------------------------------+--------------------><
            |                                           |
            +----------------------+-- Cursor ----------+
            |                      +-- CSR -------------+
            |                                           |
            +----------------+-----+-- Data ------------+
            |                |     |                    |
            +-- ALL ---------+     +-- Half ------------+
                                   |                    |
                                   +-- Max -------------+
                                   |                    |
                                   +-- Page ------------+
                                   |                    |
                                   +-- n_cols ----------+
```

Description:

In multi record view and unless parameter **ALL** is specified, scroll towards the right the display of field and header lines belonging to records that are of the default record type. The display of all other visible records, header lines and shadow lines remains unchanged.

In single record view, RIGHT will display the fields belonging to a visible data record that has a higher line number than the record (or record segment) that is in the current view. For display of segmented records, LEFT and RIGHT scroll through each segment in the file, regardless of the record to which it belongs. NEXT and PREV may be used to scroll between secondary segments of the current record only. RIGHT CURSOR/DATA/HALF/PAGE are not applicable in single record view.

RIGHT is assigned to **PF11** by default. Any characters specified on the command line when the PFKey is hit will be concatenated to the command and treated as a parameter string.
Where no parameter is specified, the scroll amount will be the value specified in the "**Scroll>**" field.

Multi Record View Scrolling:

Columns may have the HOLD flag set on by the SELECT command. These columns are static in the display and, like the prefix area and Lrecl column, are unaffected by LEFT and RIGHT scrolling. Columns that have the HOLD flag set off are referred to as **floating** columns.

The following interpretation of cursor location applies so that RIGHT CURSOR can operate successfully on the appropriate data:

- Cursor is located at an offset within a column header line.
  The cursor position is interpreted as being at the same offset within the data record that immediately follows the group of header lines.

- Cursor is located within a column of type character (AN) but beyond the width of the character data.
  The cursor position is interpreted as being the last position of the displayed character data.

- Cursor is located immediately to the left of a column of any type.
  The cursor position is interpreted as being the the first position of the data displayed within that column.

Where the first column to be displayed is not of type character (AN), the magnitude by which the display is scrolled to the right is always reduced to display all field data belonging to the first column. Similaraly, if the last column to be displayed is not of type character and the display width is too small to accomodate all the column's data and header text, the column will not be displayed.

No adjustment is made when scrolling right into an offset within a character data column. i.e. where the first column of the display is of type character, the first position of the column display need not be the first character of the character data. If the last column to be displayed is of type character and the display width is too small to accomodate all the column's data and header text, the column will be truncated.

Attempting to scroll right beyond the last data column will make the last data column the first column of the display. Furthermore, if the last data column is of type character (AN), then scrolling beyond this column will make the last character of the data the first character of the display.

Parameters:

`CURSOR`
`CSR`

The focus column becomes the first column of the scrolled display.
In addition to this, if the focus column is type character (AN) and the cursor is positioned at an offset into the focus column, then the character data at the cursor offset will occupy the first position within the first column of the scrolled display.

If any of the following conditions are true, then RIGHT PAGE is executed instead:

- ◊ Cursor is positioned anywhere other than within a floating column in a visible data record.
- ◊ Focus column is **not** type AN and is already the first floating column of the display.
- ◊ Focus column is type AN, is already the first floating column of the display and cursor position is at the first position of the displayed column data.

`ALL`

Valid only in multi-record view, ALL indicates that scrolling is to apply to **all** records in the display and not only those records that are assigned the default record type.

`DATA`

Scroll right so that the last floating column of the current display area becomes the first floating column of the scrolled display.
If this column is type character (AN) and the last position of the display falls on a character that is at an offset into the column data, then the character data at that offset will occupy the first position within the first column of the scrolled display.

`HALF`

Scroll a number of columns so that the column situated half way along the width of the current display of floating columns, becomes the first floating column of the scrolled display.
If this column is type character (AN) and the half display position falls on a character that is at an offset into the column data, then the character data at that offset will occupy the first position within the first floating column of the scrolled display.

`MAX`

Scroll right to display the last column of data in a multi record view. Where the display area is able to contain all data columns in the data record, the first floating column becomes the first floating column of the scrolled display. Otherwise, the last column of data becomes the last column of the scrolled display.

In single record view, the last visible data record is displayed.

`PAGE`

Scroll right so that the column of data to the right of the last floating column of the current display area, becomes the first floating column of the scrolled display.
If this column is type character (AN) and the last position of the display plus one falls on a character that is at an offset into the column data, then the character data at that offset will occupy the first position within the first floating column of the scrolled display.

n_cols

In a multi record view, scroll right a specified number of floating columns.
The floating column of data that is *n_cols* to the right of the first floating column becomes the new first floating column of the scrolled display.

In single record view, the visible data record that is *n_cols* records after the record currently in view, gets displayed.

Examples:

`right half`

Scroll records of the default record type in a multi record view display area, right by half a display area width minus the width of any columns flagged with HOLD.

See Also:

DOWN LEFT UP

RUNSELCOPY

Description:

The Data Editor command, RUNSELCOPY, performs the same operation as the Text Editor command RUNSELCOPY. See RUNSELCOPY in CBLe Text Editor documentation.

RUNSLC

Description:

The Data Editor command, RUNSLC, performs the same operation as the Text Editor command RUNSLC. See RUNSLC in CBLe Text Editor documentation.

SAVE

Syntax:

```
>>-- Save --------+-------------+-------------------------------------------><
                  |             |
                  +-- NEWGEN ---+
                  |             |
                  +-- NOGEN ----+
```

Description:

Save changes to the current data.

Saving changes to DB2 tables

There are important differences in the way the SAVE command works when editing DB2 tables compared with editing z/OS VSAM datasets, non VSAM datasets, PDS(E) members and HFS files. See DB2 table edit for a discussion of these differences and a more detailed description of how SAVE works for DB2 data.

Parameters:

`NEWGEN`

Applicable only if output is to a version 2 PDSE library member that supports member generations. Data will be saved to a new primary member generation.
This option overrides the default set by the GENSAVE option.

`NOGEN`

Applicable only if output is to a version 2 PDSE library member that supports member generations. Data will be saved to back to the same member generation referenced in the focus edit view.
This option overrides the default set by the GENSAVE option.

See Also:

FILE SAVEAS and SET/QUERY/EXTRACT Option: GENSAVE

SAVEAS

Syntax:

```
>>-- SAVEAS -----------------------------------------------------------------><
```

Description:

Applicable to structured data edit only (not DB2 table edit), SAVEAS prompts the user for a new fileid to be assigned to the data in the focus SDE edit view, and then saves the data to this file.

The fileid entered by the user may be an MVS DSN, PDS/PDSE library and member or HFS file path of a new file. If the file already exists, error ZZSD229E is returned. A PDS/PDSE library and member may only be specified if the currently assigned fileid is a PDS/PDSE library member.

If necessary SAVEAS will invoke the appropriate Allocate NonVSAM or Define VSAM data set dialog window to create the new data set with the same data set organisation as the currently assigned fileid.

SAVEAS is not valid for SDE edit techniques KSDS Edit and Update-in-place Edit where not all records are loaded into storage.

See Also:

SAVE

SAVESTRUCTURE

Syntax:

```
>>-- SAVEStructure ---+-----------------+--+--------------------+------------><
                      |                 |  |                    |
                      +-- struct_name --+  +-- newstruct_name --+
```

Description:

Save an in storage Structure Definition Object (SDO) to a Structure Definition File (SDF) on disk.

If changes have been made to the structure (e.g. USE WHEN or RCOLOUR criteria have been applied) and not saved, then the user will be automatically prompted to save the SDO following execution of the DROP command or when the structure is automatically dropped on exit from the parent CBLe frame window.

Parameters:

struct_name

The name of the in-storage structure to be saved.
Default is the name of the current (i.e. last referenced) structure. This is the structure used in the current SDE window view or explicitly referenced by an SDE command.

If *struct_name* is the name of a temporary structure (created dynamically for edit/browse of a DB2 table or when a COBOL, PL1 or Assembler copybook or ADATA file is used) and *newstruct_name* is **not** specified, then a panel is opened that prompts for the name of the SDF.

newstruct_name

The name of the SDF to which the structure will be saved.
Default is *struct_name*.

SDATA

Description:

SDE CLI command, SDATA, performs the same operation as the CBLe CLI command SDATA. See SDATA in CBLe Text Edit documentation.

SDATA command is supported so that it may be tolerated in edit macros when the focus CBLe child window is an SDE edit/browse view.

SEGTYPE

Syntax:

```
>>-- SEGtype -----+-- Secondary ----------------+----------------------------><
                  |                             |
                  +-- Primary ------------------+
                  +-- Base ---------------------+
```

Description:

Applicable to full or auxiliary edit of segmented records only, SEGTYPE is used to change the type (primary or secondary) of the focus segment. Note that Update-in-place edit does not support SEGTYPE.

SEGTYPE PRIMARY or SEGTYPE BASE changes a secondary segment to be a primary (base) segment so splitting the record into two. The record generated by the new primary segment is remapped so that all its segments are assigned an appropriate record type definition.

SEGTYPE SECONDARY changes a primary segment to be a secondary segment so joining the record with the record before. The record is remapped so that all its segments are assigned an appropriate record type definition.

Note that no action is taken if SEGTYPE does not change the segment type of the focus segment.

SEGTYPE may be undone and subsequently redone using the UNDO (<PF22>) and REDO (<PF23>) commands.

Parameters:

`SECONDARY`
Sets the focus segment to be a secondary segment.

`PRIMARY | BASE`
Sets the focus segment to be a primary (base) segment.

See Also:

SET/QUERY/EXTRACT Option: FOCUS

SELECT

SELECT is also a Text Edit CBLe option used to set the selection level of a group of edited records, and also a List Window primary command used to select columns for display.

Syntax:

```
>>-- SELect --+----------------------------------------+---------------------><
              |                                        |
              +-- | Field Column Selection Options | --+
```

Field Column Selection Options:

```
 |------------+--------------------------------+---------------------------->
              |                                |
              |  +-- ALL  --------------------+  |
              |  |                          |  |
              +--+-- * -----------------------+--+
              |                                |
              |  +----------- , -------------+  |
              |  v                          |  |
              +--+-- field_col --+-------+--+--+
                                 |       |
                                 +- Hold -+
```

```
 >---+-----------------------+--+---------------------+----------------------|
     |                       |  |                     |
     +-- FROM record_type ---+  +-- IN struct_name ---+
```

Description:

SELECT may be used with no parameters to open the SELECT Columns panel. Alternatively, SELECT may directly specify the column names of fields belonging to a particular record type and the order in which they are to be displayed in the current window view.

The selected field names must belong to the specified record type. If *record_type* is not specified, then the default record type is assumed.

If *struct_name* is not specified, then the SELECT command operates on the data in the current Data Editor view.

SELECT may be preceeded by modal primary command PERMANENT or TEMPORARY (default) to determine whether the columns selected for display is saved in the the FileKit structure (SDO). If so, subsequent display of data using the same structure will initially display the saved selection of columns.

SELECT operates at the Data Editor view level and so will not apply to the same data displayed in other Data Editor views.

Parameters:

If no parameters are given then the SELECT Columns panel is displayed.

`field_col`
: Any field reference number or name defined for this record type. If *field_col* refers to a **struct**, then all elements of that structure are considered to be selected.

  For BROWSE processing only, if the current Data Editor view contains segmented records (where a record is comprised of a primary segment followed by zero or more secondary segments and each segment is mapped by a specific record-type definition), then secondary segment fields may be selected for display on the primary segment. To do this, *field_col* must identify the record-type name of the secondary segment followed by the required filed name separated by a single "." (dot/period). e.g. `SMF020_Job_Initiation.zJobname`

`Hold`
: Indicates that this field (and any others preceding it) is to be held visible when scrolling occurs.

`ALL`
: Shorthand to define all fields in their default order. Must be supplied as the last parameter.

`* (asterisk)`
: Shorthand to define all fields other than those explicitly defined already (i.e. the rest) in their default order. Must be supplied as the last parameter.

`FROM record_type`
: Defines the record type to which the selection applies. If *record_type* is not specified, then the default record type is used.

`IN struct_name`
: Defines the name of the structure definition object (SDO) to which the selection applies. The Data Editor view containing data mapped by this SDO will be the target window for this SELECT command. Default is the current Data Editor view.

Examples:

```
select from Employees  EmpNumber,Dept,LastName,FirstName
select EmpNumber,Dept,LastName,FirstName H,*
```

SET

Description:

See SET/QUERY/EXTRACT Options.

SHIFT

Syntax:

```
>>-- SHIft --+- Left -+-- n_bytes --+--------------------------------------+->
             |        |             |                                      |
             +- Right-+             +- BOUNds -+- left_col -+- right_col --+
                                    |          |            |              |
                                    +- BNDs ---+            +- * ----------+

           +--- 1 -----+                    +- .ZCSR --+   +- .ZLAST -+
           |           |                    |          |   |          |
 >---------+-----------+------+------+------+----------+---+----------+------><
           |           |      |      |      |          |   |          |
           +- n_lines -+      +- EX -+      +- .name1 -+   +- .name2 -+
           |           |      |      |
           +- ALL -----+      +- NX -+
                              |      |
                              +- X --+
```

Description:

Supported for unformatted record view only, SHIFT moves record data within the left and right bounds columns, left or right without altering relative spacing. (See SDE BOUNDS option.)

Record data which is shifted left beyond the left bounds column or right beyond the right bounds column gets truncated.

Shift left and shift right may also be executed selected lines using the "("/"((" and ")"/"))" line (prefix area) commands respectively.

Variable Length Records:

Shifting variable length record data left or right has different effects on each record depending on location of BOUNDS columns relative to the individual record's record length.

1. If the left bounds column number (and therefore the right bounds column number) is greater than the record length, then shifting data left or right will have no effect on the record.
2. If the left bounds column number is less than or equal to the record length, then shifting data left will have the following affect on that record:
   - ♦ If the right bounds column number is less than or equal to the record length, data is truncated at the left bounds column, the record length is preserved and a pad character is inserted at the right bounds column for each character truncated at the left. (See SDE PAD option.)
   - ♦ If the right bounds column number is greater than the record length, data is truncated at the left bounds column and the record length is reduced by the number of truncated characters.
3. Shifting data right will have the following affect on a record:
   - ♦ If the right bounds column number is less than or equal to the record length, data is truncated at the right bounds column, the record length is preserved and a pad character is inserted at the left bounds column for each character truncated at the right.
   - ♦ If the right bounds column number is greater than the record length, a pad character is inserted at the left bounds column for each shifted column. and the record length is increased to a maximum defined by the right bounds column. Record data shifted beyond this column will be truncated.

Parameters:

`LEFT | RIGHT`
Specifies the direction in which record data is to be shifted.

`n_bytes`
Specifies the number of columns by which the record data will be shifted.

`BOUNDS|BNDS` *`left_col right_col`*
BOUNDS may be specified to temporarily override the prevailing bounds column settings. See SDE BOUNDS option for details on setting left and right bounds columns.

`n_lines` `| ALL`
Specifies *n_lines*, the number of lines, or ALL, indicating all subsequent lines, starting at the focus line, for which the shift operation will apply.

Unless EX or X is specified, groups of excluded lines are treated as 1 line in the *n_lines* count of lines affected by SHIFT. This is true whether or not display of shadow lines, which represent groups of excluded lines, is on or off. (See SDE SHADOW option or primary command HIDE).

Default is 1 line, i.e. the focus line.

`EX`
`X`
The shift operation will apply only to lines flagged as being EXCLUDED.
If neither EX nor NX is specified, shift will apply to both excluded and non-excluded lines.

`NX`
The shift operation will apply only to lines that are **not** flagged as being EXCLUDED.
If neither EX nor NX is specified, shift will apply to both excluded and non-excluded lines.

`.name1`
A label name identifying the first line of a range of lines eligible for inclusion in the SHIFT operation. The preceding "." (dot/period) is mandatory.
Default is .ZCSR, the focus line.

`.name2`
A label name identifying the last line of a range of lines eligible for inclusion in the SHIFT operation. The preceding "." (dot/period) is mandatory.
If *.name2* occurs before *.name1* in the display, then the order is reversed.
Default is .ZLAST.

See Also:

BOUNDS CHANGE PAD

SHOW

Syntax:

```
>>-- Show -------------- --+-- Level ----+-----------------------------------><
                           |             |
                           +-- Number ---+
                           |             |
                           +-- Format ---+
                           |             |
                           +-- Offset ---+
                           |             |
                           +-- Picture --+
                           |             |
                           +-- Type -----+
```

Description:

The *SHOW* command controls display of various types of information for SDE EDIT and BROWSE window views in either mapped table view (VFMT) or mapped single record view (MAP). The format and content is controlled by the SHOW option specified.

See also the *SET* options *TYPE* and *OFFSET*, various combinations of which are controlled using the SHOW command.

Parameters:

`LEVEL`
Equivalent to LEVEL ON , REFERENCE OFF .

NUMBER
Equivalent to LEVEL OFF , REFERENCE ON .

FORMAT
Equivalent to TYPE FORMAT .

OFFSET
Equivalent to TYPE OFFSET .

PICTURE
Equivalent to TYPE PICTURE .

TYPE
Equivalent to TYPE DEFAULT .

See Also:

SET LEVEL OFFSET REFERENCE TYPE

SORT

Syntax:

```
              +- .ZFIRST -+ +- .ZLAST -+  +----------- , -----------+
              |           | |          |  v                          |
>>-- SORT --+-+-----------+-+----------+--+-- | Sort Field Spec | --+--+-----><
            | |           | |          |                               |
            | +- .name1 --+ +- .name2 -+                               |
            |                                                          |
            +-- KEY ---------------------------------------------------+
                   (1)
```

Sort Field Spec:

```
                                                            +- Ascending --+
                                                            |              |
 |-+- field_col ----------------------------------------+---+--------------+-|
   |                                                    |   |              |
   |                                 +- CHaracter --+   |   +- Descending -+
   |                                 |              |   |
   +- field_pos -- field_length -----+--------------+---+
                                     |              |
                                     +- Fixed ------+
                                     |              |
                                     +- PD ---------+
                                     +- DECimal ----+
                                     +- PACKed -----+
                                     |              |
                                     +- ZD ---------+
                                     +- Zoned ------+
                                     |              |
                                     +- HFP --------+
                                     |              |
                                     +- BFP --------+
                                     |              |
                                     +- DFP --------+
```

Notes:

1. Only for a KSDS data set that has never been loaded.

Description:

Sort in-storage data records or DB2 table rows within the current edit view.

Sort is supported only if **all** edited records or table rows are loaded in available storage, otherwise error ZZSD643E is returned.

Sort fields are identified by a record position, length and data type. Alternatively, if the edited data is mapped by a single record type (incudes DB2 table rows), a sort field may be identified by a column field reference. If the edited data is mapped by more than one record type, sort fields must be specified using position, length and data type.

Multiple sort fields may be specified. The order in which sort fields are specified define the hierarchy in which the record or DB2 table row data will be sorted.

When sorting variable length records, it is possible for a specified sort field to reference positions that fall outside the length of a record, but not outside the defined maximum record length. In this case, the sort field is padded with nulls (x'00').

If *field_pos* syntax is used to identify a sort field that exists entirely within the fixed portion of **formatted** records, then the sort field references positions in the unexpanded records. If the sort field overlaps variable areas of the formatted records (i.e. in areas mapped by or following fields of variable length data types) then, the following will occur:

- For edited data mapped by a single record type, the sort field will references positions in expanded record data.
- For edited data mapped by multiple record types, the sort field can only reference positions in the unexpanded record and so the edited record data must be displayed in character format. If not, error ZZSD666E will be returned.

SORT KEY is a special form of the command applicable only when records are added to an initially empty VSAM KSDS data set. When records are saved to a KSDS data set, records get inserted in correct order of key sequence and so do not need to be in index key order within the edit view. However, if the KSDS data set is initially empty, saving newly added records will load them to the data set in the order they occur within the edit view. Therefore, to ensure a successful load, the records should first be sorted in ascending order of key using SORT KEY. Use of parameter KEY in a non-empty KSDS data set will be treated as named field (*field_name*.)

Parameters:

`.name1`

A label name identifying the first record of a range of data records to be sorted. The preceding "." (dot) is mandatory. Default is .ZFIRST.

`.name2`

A label name identifying the last record of a range of data records to be sorted. The preceding "." (dot) is mandatory. If *.name2* occurs before *.name1* in the display, then *.name2* and *.name1* identify the first and last records respectively. Default is .ZLAST.

`field_pos`

The first position of a sort field within the record data.

`field_length`

The length of the sort field. The value of *field_length* must be valid for the corresponding *field_type*.

`field_type`

The data type of the data within the sort field. This may be one of the following:

CHARACTER	Character - the default data type. *field_len* must be less than 32768.
FIXED	Fixed Point Binary. *field_len* must be less or equal to 8.
PD DECIMAL PACKED	Packed Decimal. *field_len* must be less or equal to 16.
ZD ZONED	Zoned Decimal. *field_len* must be less or equal to 31.
HFP	Hexadecimal Floating Point. *field_len* must be 4, 8 or 16.
BFP	Binary Floating Point. *field_len* must be 4, 8 or 16.
DFP	Decimal Floating Point. *field_len* must be 4, 8 or 16.

`field_col`

An individual field column within a formatted display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

If the field is an array, then all elements of the array constitute the sort field. To sort on an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array. An entry must exist for each dimension of the array. e.g. RoomSize(6,2,4) - a three dimensional array.

`ASCENDING | DESCENDING`

Sort data in the sort field in ascending or descending order
Default is ASCENDING.

`KEY`

Supported for VSAM KSDS data sets only, sorts records into ascending key sequence.

All records in the display (visible, EXCLUDED, NOT SELECTED and SUPPRESSED) are included in a SORT KEY operation.

SORTINDEX

Syntax:

```
                 +- .ZFIRST -+ +- .ZLAST -+
                 |           | |          |
>>-- SORTIndex --+-----------+-+----------+---+--------------+---------------><
```

```
                  |            | |          |    |              |
                  +- .name1 --+ +- .name2 -+    +- index_name -+
                                                 |              |
                                                 +- Prime ------+
```

Description:

SORTINDEX is supported for DB2 table edit only and may be used to sort in-storage table rows by key columns/expressions identified by a DB2 Index previously defined for the table.

If neither PRIME nor *index_name* are specified, the DB2 Select Table Index window is displayed, allowing selection of one of the indexes defined for the table.

Parameters:

.name1
A label name identifying the first table row of a range of rows to be sorted. The preceding "." (dot) is mandatory.
Default is .ZFIRST.

.name2
A label name identifying the last table row of a range of rows to be sorted. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then *.name2* and *.name1* identify the first and last rows respectively.
Default is .ZLAST.

index_name
The name of a DB2 Index that has been created for the edited DB2 table.

PRIME
Sort the edited DB2 table rows using the primary index.

SUM

Syntax:

```
               +-------------------------- , ----------------------------+
               v                                                          |
>>-- SUM ------+- field_col ----------------------------------------------+-->
               |                                                          |
               |                                        +- CHaracter --+  |
               |                                        |              |  |
               +- field_pos -- , -- field_length -- , --+--------------+--+
                                                        |              |
                                                        +- Fixed ------+
                                                        +- Binary -----+
                                                        |              |
                                                        +- PD ---------+
                                                        +- DECimal ----+
                                                        +- PACKed -----+
                                                        |              |
                                                        +- ZD ---------+
                                                        +- Zoned ------+
                                                        |              |
                                                        +- HFP --------+
                                                        |              |
                                                        +- BFP --------+
                                                        |              |
                                                        +- DFP --------+


                                  +- .ZFIRST -- .ZLAST -+
                                  |                     |
 >--+- WHere --- expression ---+--+---------------------+-------------------->
    |                          |  |                     |
    |                          |  +- .name1 -+----------+
    | +- NX -----------+       |             |          |
    | +- NOTEXcluded --+       |             +- .name2 -+
    | |                |       |
    +-+----------------+-------+
      |                |
      +- EXcluded -----+
      +- X ------------+
      |                |
      +- ALL ----------+


 >--+--------------------------------------------+--+----------------+-------><
    |                                            |  |                |
    +-------+- RECord -+--------+- record_type --+  +- STEM stemvar -+
    |       |          |        |
    +- FOR -+          +- TYPE -+
```

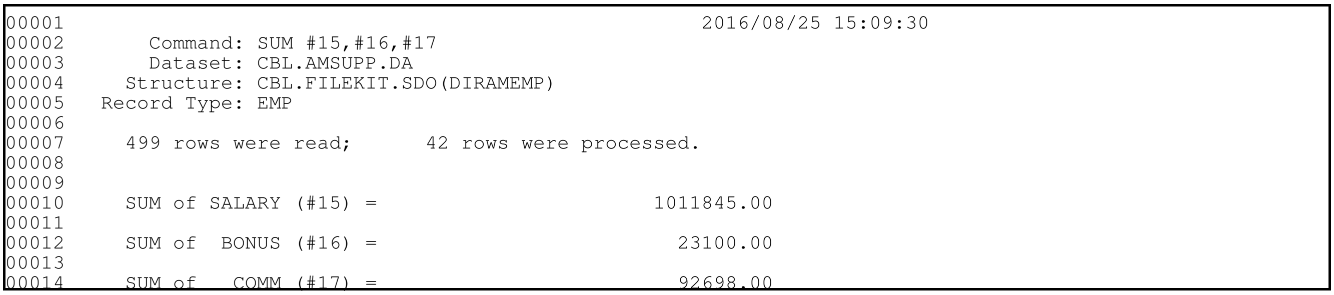
Description:

For browse or edit of formatted or unformatted data in the focus Data Editor view, SUM will display or extract the sum of values belonging to columns in unformatted records or, if formatting is applied, records of a specific record type.

Multiple columns may be specified as a mixture of formatted record column ids (names and/or reference numbers) and field positions and length. A sum value will be generated for each column specification.

Records may be included or excluded from the SUM operation via a WHERE filter expression or using the current excluded status of lines within the display.

Unless STEM is specified for use in REXX procedures (edit macros), the SUM values are reported in a temporary Text Edit view. e.g.



```
00001                                                     2016/08/25 15:09:30
00002       Command: SUM #15,#16,#17
00003       Dataset: CBL.AMSUPP.DA
00004     Structure: CBL.FILEKIT.SDO(DIRAMEMP)
00005   Record Type: EMP
00006
00007     499 rows were read;      42 rows were processed.
00008
00009
00010     SUM of SALARY (#15) =                       1011845.00
00011
00012     SUM of  BONUS (#16) =                         23100.00
00013
00014     SUM of   COMM (#17) =                         92698.00
```

For each reported column sum, report lines may follow which detail the number of column values that were excluded from the process for the following reasons:

1. The value is null.

2. The value is invalid (not in the data type format).
3. For decimal floating point data type, the value is QNAN, SNAN or Infinity.

When the duration of a SUM execution exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the oportunity to interrupt the operation using the Attn key.

Parameters:

field_col

An individual field column id within a formatted record display.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. SALARY).

If *field_col* identifies an array, then SUM treats each element of the array as a separate sum value. To perform SUM on an individual element of an array, the element occurrence should be specified in parentheses as a subscript to the field reference/name. e.g. #13(3) for the 3rd element of a single dimension array.

field_pos

The start position of a field within the record data. For formatted records, this is a position in the expanded record.

field_length

Following *field_pos*, *field_length* defines the length of the field in the record data. The value of *field_length* must be valid for the corresponding *field_type*.

field_type

The data type of the data in the field identified by *field_pos* and *field_length*. *field_type* may be one of the following:

CHARACTER	Character - the default data type. *field_len* must be less than 32768. Numeric interpretation of character data supports use of the following: 1. Currency symbol (x'5B'). 2. A dot/period (".") representing decimal point. 3. Commas (","). 4. Character "E" (or "e") representing start of an exponent. 5. Unary plus ("+") or minus ("-") preceding the value and/or exponent value.
FIXED BINARY	Fixed Point Binary. *field_len* must be less or equal to 8.
PD DECIMAL PACKED	Packed Decimal. *field_len* must be less or equal to 16.
ZD ZONED	Zoned Decimal. *field_len* must be less or equal to 31.
HFP	Hexadecimal Floating Point. *field_len* must be 4, 8 or 16.
BFP	Binary Floating Point. *field_len* must be 4, 8 or 16.
DFP	Decimal Floating Point. *field_len* must be 4, 8 or 16.

WHERE *expression*

Only records of the specified record type *record_type* that satisfy the WHERE expression are included in the SUM operation.

expression is a valid SDE expression which supports function calls, *record_type* field names/references, sub-expressions, arithmetic, relational and logical operators. The result of *expression* must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

When an expression is applied to the record, it must test "true" in order to be selected as a target of the SUM operation.

By default, all data records in the selected range of lines are subject to the WHERE *expression* test. This includes data records that are of the selected record type and have EXCLUDED status at the time the SUM command is executed. If one of these excluded records satisfies the *expression*, it will be included in the SUM calculation.

EXCLUDED
X

Only records assigned the specified record type *record_type* and have EXCLUDED status are included in the SUM operation. By default, only non-EXCLUDED records are selected.
SUM does not include records that have the status NOTSELECTED or SUPPRESSED.

NOTEXCLUDED
NX

Only records assigned the specified record type *record_type* and have non-EXCLUDED status are included in the SUM operation. This is the default action.
SUM does not include records that have the status NOTSELECTED or SUPPRESSED.

ALL

All records (EXCLUDED and non-EXCLUDED) assigned the specified record type *record_type* are included in the SUM operation. By default, only non-EXCLUDED records are selected.

SUM does not include records that have the status NOTSELECTED or SUPPRESSED.

`.name1`
A label name identifying the first line of a range of lines on which the SUM operation will operate. The preceding "." (dot) is mandatory.
Default is .ZFIRST.

`.name2`
A label name identifying the last line of a range of lines on which the SUM operation will operate. The preceding "." (dot) is mandatory.
If *.name2* occurs before *.name1* in the display, then *.name2* and *.name1* identify the first and last rows respectively.
Default is .ZLAST.

`[FOR] RECORD [TYPE] record_type`
Identifies the record type mapping *record_type* used by the SUM operation. Only records assigned this record type are eligible to be included in the SUM calculation.

If a *field_col* is specified, it must reference a column field in this record type definition.

Default is the default record type.

`STEM rexx_stemvar`
Applicable only to execution of SUM within a REXX procedure edit macro, STEM indicates that the SUM value for each of the specified columns be assigned to a REXX compound variable.

The mandatory *rexx_stemvar* argument specifies the character string to be used as the stem part of the compound variable name.

The following REXX compuond variables are set:

`rexx_stemvar.SUM.0`
The number of SUM values returned (i.e. number of columns specified).

`rexx_stemvar.SUM.i`
The ith SUM value corresponding to the ith column specification.

See Also:

AVERAGE MAXIMUM MINIMUM TOTALS

SYSCOMMAND, TSO, CMS

Description:

SDE primary command, SYSCOMMAND, performs the same operation as the Text Editor command SYSCOMMAND. See SYSCOMMAND in Text Editor documentation.

TEDIT

Syntax:

```
>>-- TEdit -- cble_command ---------------------------------------------------><
```

Description:

Direct a command to the Text Editor environment.

The TEDIT command allows Text Editor commands to be issued from a Data Editor view command line or Data Editor REXX macro. Compare this with the Text Editor command SDATA which allows Data Editor commands to be executed from any type of window so long as a Text Editor frame window exists.

Parameters:

`cble_command`
Any Text Editor primary command.
The command is passed to the current Text Editor view (the Text Editor window view on which focus was last placed.)

Examples:

```
tedit    edit   DEV.OEM.CBL.JCL(CBLINS01)
```
Open a data set for Text Editor edit.

```
tedit    insert <sdata use IMP_REC when #12 >= 100
```
Insert a line of text into the current Text Editor view. The inserted line of text is itself a Text Editor command (SDATA) which may be executed using the ACTION key (shift-F4).

TEMPORARY

Syntax:

```
>>-- TEMPorary -- command ---------------------------------------------------><
```

Description:

TEMPORARY is a modal command that may be used as a prefix to SDE primary commands SELECT and SET COLWIDTH. TEMPORARY will execute the SELECT or SET COLWIDTH command and suppress any save of the selected columns or new column width value in the FileKit SDO structure used to display the formatted data.

TEMPORARY is default for both SELECT and SET COLWIDTH.

Parameters:

command
SDE primary command SELECT or SET COLWIDTH.

See Also:

SELECT   SET COLWIDTH   PERMANENT

TOP

Syntax:

```
>>-- Top --------------------------------------------------------------------><
```

Description:

Display the first page of data.
Equivalent to the UP MAX command.

See Also:

BOTTOM   UP

TOTALS

Syntax:

```
               +-------------------------- , -----------------------------+
               v                                                          |
>>-- TOTals ---+- field_col ----------------------------------------------+-->
               |                                                          |
               |                                        +- CHaracter --+  |
               |                                        |              |  |
               +- field_pos -- , -- field_length -- , --+--------------+--+
                                                        |              |
                                                        +- Fixed ------+
                                                        +- Binary -----+
                                                        |              |
                                                        +- PD ---------+
                                                        +- DECimal ----+
                                                        +- PACKed -----+
                                                        |              |
                                                        +- ZD ---------+
                                                        +- Zoned ------+
                                                        |              |
                                                        +- HFP --------+
                                                        |              |
                                                        +- BFP --------+
                                                        |              |
                                                        +- DFP --------+
```

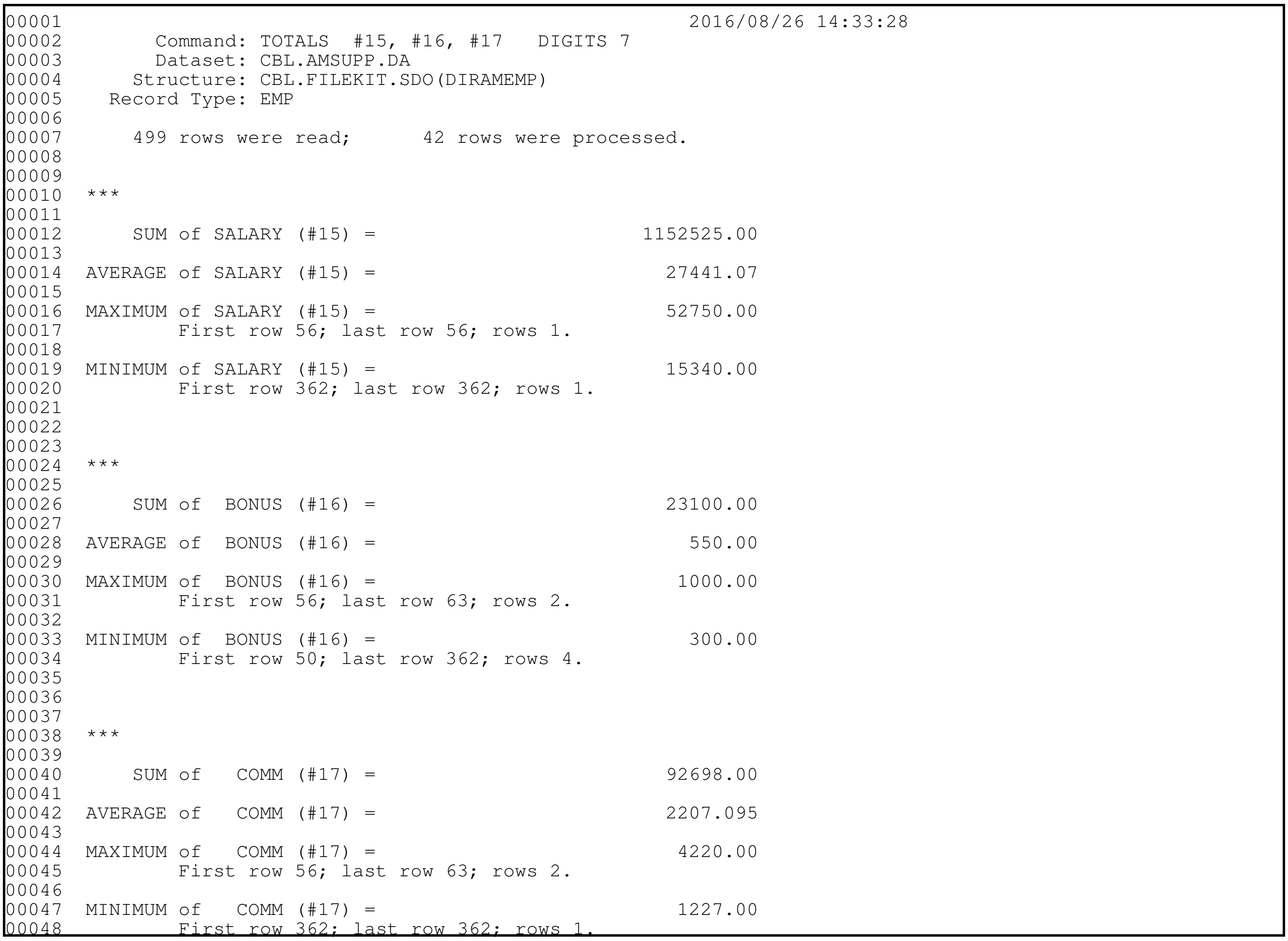
```
                                  +- .ZFIRST -- .ZLAST -+
                                  |                     |
 >--+- WHere --- expression ---+--+---------------------+--+---------------+->
    |                          |  |                     |  |               |
    |                          |  +- .name1 -+----------+  +- DIgits int --+
    | +- NX -----------+       |             |          |
    | +- NOTEXcluded --+       |             +- .name2 -+
    | |                |       |
    +-+----------------+-------+
      |                |
      +- EXcluded -----+
      +- X ------------+
      |                |
      +- ALL ----------+
```

```
 >--+--------------------------------------------+--+----------------+-------><
    |                                            |  |                |
    +-------+- RECord -+--------+- record_type --+  +- STEM stemvar -+
    |       |          |        |
    +- FOR -+          +- TYPE -+
```

Description:

TOTALS performs all the aggregate operations (AVERAGE, MAXIMUM, MINIMUM and SUM) for specific columns within unformatted records or, if formatting is applied, within records of a specific record type.

Unless STEM is specified for use in REXX procedures (edit macros), the aggregate values are reported in a temporary Text Edit view. e.g.



```
                                                    2016/08/26 14:33:28
      Command: TOTALS  #15, #16, #17   DIGITS 7
      Dataset: CBL.AMSUPP.DA
    Structure: CBL.FILEKIT.SDO(DIRAMEMP)
  Record Type: EMP

    499 rows were read;      42 rows were processed.


***

    SUM of SALARY (#15) =                       1152525.00

AVERAGE of SALARY (#15) =                         27441.07

MAXIMUM of SALARY (#15) =                         52750.00
        First row 56; last row 56; rows 1.

MINIMUM of SALARY (#15) =                         15340.00
        First row 362; last row 362; rows 1.



***

    SUM of  BONUS (#16) =                         23100.00

AVERAGE of  BONUS (#16) =                           550.00

MAXIMUM of  BONUS (#16) =                          1000.00
        First row 56; last row 63; rows 2.

MINIMUM of  BONUS (#16) =                           300.00
        First row 50; last row 362; rows 4.



***

    SUM of   COMM (#17) =                         92698.00

AVERAGE of   COMM (#17) =                         2207.095

MAXIMUM of   COMM (#17) =                          4220.00
        First row 56; last row 63; rows 2.

MINIMUM of   COMM (#17) =                          1227.00
        First row 362; last row 362; rows 1.
```

See individual commands for AVERAGE, MAXIMUM, MINIMUM and SUM for the parameter descriptions and additional report lines that follow each reported aggregate value.

When the duration of a TOTALS execution exceeds 1 second, a progress window is opened displaying the number of records processed so far. This window also provides the user with the oportunity to interrupt the operation using the Attn key.

UNDO

Syntax:

```
>>-- UNDO -------------------------------------------------------------------><
```

Description:

Undo one level of changes made to the current file.

Each change level corresponds to an update of a data field within a record.

UNDO is assigned to PF22 by default and may be executed repeatedly to undo multiple levels of changes.

The third number following "Alt=n,n;m" on the status line indicates the number of change levels. If this number is not zero, then changes may be undone using UNDO.

See Also:

REDO and the SET/QUERY/EXTRACT Option: UNDOING

UNFMT

Syntax:

```
>>--+-- UNFMT --+----------------------------------------------------------><
    |           |
    +-- UNMAP --+
```

Description:

Display records or record segments in single record, unformatted character view by temporarily disabling the record-type (RTO) assigned to each record or record segment.

Compare with ZOOM of an SDE view set as FORMAT CHAR which displays the records/segments in a single-record, unformatted view but does not disable the assigned record-type.

Note that unformatted records have record type "UnMapped", unformatted segments of a segmented record have record type "UnMappedSeg".

See Also:

FORMAT CHAR MAP (FMT) VFMT ZOOM

UP

Syntax:

```
>>- UP -----+-------------------+-------------------------------------------><
            |                   |
            +-- Cursor ---------+
            |                   |
            +-- CSR ------------+
            |                   |
            +-- Data -----------+
            |                   |
            +-- Half -----------+
            |                   |
            +-- Max ------------+
            |                   |
            +-- Page -----------+
            |                   |
            +-- n_lines --------+
```

Description:

Scroll the view of the data within the SDE window upwards towards the top of the file data.

Where no parameter is specified, the scroll amount will be the value specified in the **Scroll>** field in the top right corner of the window display.

Note that the first data record in any multi record view will **always** be preceded by its complete group of column header lines.

For single record views, the standard headers are always visible at the top of the display area and are not included as part of the scrollable text. Each non-header line within a single record view is identified as being a **field data line**.

UP and DOWN scroll commands will cause the window display area to be adjusted by a number of lines determined by the number of multi record view data records and shadow lines, or single record view field data lines. Other lines in the display area, i.e. Header lines and blank filler lines that precede a group of header lines or follow the "End of Data" record, are not included in this calculation.

Attempting to scroll up beyond the first entry (data record, shadow line or field data line) will make the first entry the first line of the display.

Multi Record View Scrolling:

Where the cursor is positioned at an offset within a column header line, the cursor is deemed to be located at the same offset within the data record that immediately follows the group of header lines. Therefore, UP CURSOR will operate successfully when the cursor is positioned within a header line.

The first data record in the display must be preceded by all of its header lines. To accomodate this, the magnitude by which the display is scrolled upwards may be reduced with the result that it may be impossible for the target line of the UP command to occupy the last line of the current display. The display will be scrolled upwards so that the target line is still in view though not the last line in the display. In order to overcome this, the user would have to alter the display area dimensions.

For the same reason, it is possible that execution of an UP command may result in no change to the displayed data, in which case UP PAGE is executed instead.

Parameters:

`CURSOR`
`CSR`

The data record, shadow line or field data line on which the cursor is positioned becomes the last line of the scrolled display.
If the cursor is positioned on a header line, the data record or field data line immediately following the header line becomes the last line in the display area.
If the cursor is positioned outside the display area or on the last line within the display area, then UP PAGE is executed instead.

`DATA`

Scroll up to display one page (display window depth) less one line of data.
The first data record, shadow line or field data line in the current display area becomes the last line of the scrolled display.

`HALF`

Scroll up half a page of data.
The data record, shadow line or field data line that is half way down the page of data in the current display area becomes the last line of the scrolled display.

`MAX`

Scroll up to display the first page of data.
The "Top of Data" line becomes the first line of the scrolled display.
Equivalent to the TOP command.

`PAGE`

Scroll up to display the next whole page of data.
The data record, shadow line or field data line before the first line of the current display area becomes the last line of the scrolled display.

`n_lines`

Scroll up a specified number of lines.
The data record, shadow line or field data line that is *n_lines* lines above the current line becomes the first line of the scrolled display.

Examples:

```
up  page
```

Scroll the display area up one page.

See Also:

DOWN LEFT RIGHT

USE

Syntax:

```
>>- USE -- record_type -+------------+-+----------------------------------+-->
                        |            | |                                  |
                        +- TEMPORARY + +- IN -+------------+- struct_name +
                                              |            |
                                              +- STRUCTURE +

                       +-- ON ---+
                       |         |
   +-- ALWAYS ---------+---------+------------------+
   |                   |         |                  |
   |                   +-- OFF --+                  |
   |                                                |
 >-+-- WHEN -----------+----------------+-----------+------------------------><
   |                   |                |           |
   |                   +-- expression --+           |
   |                                                |
   |                   +-- ON ---+                  |
   |                   |         |                  |
   +-- NEVER ----------+---------+------------------+
   |                   |         |                  |
   |                   +-- OFF --+                  |
   |                                                |
   |                                   +-- ON ---+  |
   |                                   |         |  |
   +-+-------+- FOCUS -+------------+--+---------+--+
     |       |         |            |  |         |
     +- FOR -+         +-- RECord --+  +-- OFF --+
```

Note:

1. A user interface to the USE command (allowing the user to list and set USE WHEN/ALWAYS/NEVER conditions) is available from the SDE Edit/Browse Utility Window menu accessed using <F16> in any SDE edit/browse session. Items 6 and 7, "Modify record-type Identification criteria" and "Override record-type for focus line" respectively, relate directly to the USE command.

2. USE does **not** affect data in the file, only the way in which the data is formatted. Unless FOR FOCUS or TEMPORARY is specified, USE may update the in-storage copy of the SDE structure (SDO) so that, when the structure is dropped, the user will be prompted to save the updated SDO to disk.

   Because no data is changed, USE is not included in the UNDO/REDO chain.

Description:

The USE command is not applicable to DB2 Table edit.

Where individual records are not automatically assigned the required record type (RTO) defined within the structure (SDO), then the **USE** command may be executed to express more sophisticated record type assignment criteria or force a specific record type assignment.
See "*Record Type Assignment*".

Incorrect record type assignment may occur if the total length of the field definitions within a record type matches that of another record type. Similarly, no record types may exist that exactly match the length of the data record, in which case the first record type in the SDO is assigned by default.

USE WHEN *expression* may be specified to one or more RTO definitions in order to sophisticate automatic assignment of record type, so matching records with their correct RTO.
USE NEVER ON will exclude an RTO definition from being eligible for assignment.
USE ALWAYS ON will force all records to be assigned a single RTO definition.
USE FOR FOCUS RECORD will force temporary assignment of a specific RTO definition to the record occupying the focus line.

Unless TEMPORARY or FOR FOCUS RECORD is specified, execution of the USE command will affect a change to the RTO record type definition, identified by *record_type*, in the specified SDO. If not already loaded, the required SDO is loaded from its structured data file (SDF) and updated accordingly. If the SDO containing this record type is a non-temporary structure, then the change may be saved to the structure definition file when the structure is dropped or on execution of the SAVESTRUCTURE command.

Any update to record type assignment criteria via the USE command, is **immediately** applied to in-storage records belonging to the current SDE window that are assigned record types from the updated structure. Except for USE FOR FOCUS RECORD, execution of the USE command will force record type assessment of **all** the in-storage records based on the standard RTO assignment precedence.

Records in an active SDE edit or browse session that use an updated structure but are not in the current SDE window, will be unaffected by the update until record type assessment is forced for that SDE window (i.e. via another USE command.) Any new SDE edit or browse session that uses an updated structure will obey the updated record selection criteria.

If either USE ALWAYS ON or USE NEVER ON are executed, then a USE WHEN *expression* defined in the RTO is preserved so that, following execution of USE ALWAYS OFF or USE NEVER OFF, records will be re-assigned their original RTOs based on the RTO defined WHEN expressions.

Parameters:

`record_type`

Identifies the record type to which assignment criteria will be applied.

`TEMPORARY`

Indicates that assignment of record types based on this USE command is to be temporary. Unless further non-temporary USE commands are executed, the user will not be prompted to save these changes to the in-storage structure (SDO).

`IN (STRUCTURE) struct_name`

Identifies the SDO that contains the specified record type. Defaults to the SDO of the current SDE window.

`ALWAYS ON|OFF`

ALWAYS ON forces the specified record type to be assigned to all records that use the specified structure so overriding any other record type assignment criteria. If the specified record type has been defined with USE NEVER ON, then USE NEVER OFF is set for that record type. Note that USE ALWAYS ON will not remove any USE WHEN *expression* defined to the record type.

ALWAYS OFF removes the ALWAYS ON definition for the specified record type and so all records that use the specified structure are assigned a record type based on other criteria.

ALWAYS ON is default if USE is executed without parameter ALWAYS, WHEN, NEVER or FOR FOCUS RECORD. Similarly, ALWAYS ON is default if ALWAYS is specified without ON or OFF.

`WHEN expression`

Specifies the criteria that must be satisfied before the record type can be assigned to a particular record.

expression is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of the WHEN expression must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

If a WHEN expression returns a "true" result for a data record, then the record type to which it is defined is assigned to that record.

If WHEN is specified with no *expression*, then any existing WHEN *expression* will be removed from the record type definition.

`NEVER ON|OFF`

NEVER ON indicates that the specified record type should never be assigned to a record and so this record type is removed from record type assignment processing. If the specified record type has been assigned to all records with USE ALWAYS ON, then USE ALWAYS OFF is set. Note that USE NEVER ON will not remove any USE WHEN *expression* defined to the record type.

NEVER OFF removes the NEVER ON definition for the specified record type and so reinstates the record type as being eligible for record type assignment processing.

NEVER ON is default if NEVER is specified without ON or OFF.

`(FOR) FOCUS (RECORD) < ON|OFF >`

FOR FOCUS RECORD assigns the specified record type to the record occupying the focus line only. If the specified record type has been defined with USE NEVER ON, then this flag is temporarily ignored for the focus line record only. Note that USE FOR FOCUS RECORD will not remove any USE WHEN *expression* or USE NEVER ON flag defined to the record type.

FOR FOCUS RECORD OFF removes the record type assignment for the record occupying the focus line and subsequently performs standard record type assignment processing for that record only.

Examples:

```
use   Type1Rec  when  col1 = 'A'  and  (col2 << 'CDE' or col3 > 42)
```

Define USE WHEN record type assignment criteria for record type Type1Rec in the SDO used by data records in the current SDE window. This *expression* would match records for which the value in field COL1 is equal to 'A' and either of the following are true:

◊ Value in field COL2 contains the literal string 'CDE'. (i.e. 'CDE' in any character case).
◊ Value in field COL3 is numeric and greater than 42.

```
use   Type1Rec  always
```

Assign record type Type1Rec to all records.

```
use   Type1Rec  never
```

Never allow record type Type1Rec to be assigned to a record. If executed following the previous USE ALWAYS example, then ALWAYS OFF is set and record type assignment processing is performed without Type1Rec.

See Also:

DROP RCOLOUR SAVESTRUCTURE

VFMT

Syntax:

```
>>-- VFMT -------------------------------------------------------------------><
```

Description:

Display records or record segments in multi-record, formatted view.

If the record-type (RTO) assigned to each record or record segment has been disabled (record data is unformatted), then VFMT will re-enable the assigned record-types to reformat the record data.

Compare with FORMAT TABLE which does not attempt to re-enable the assigned record-types.

See Also:

FORMAT CHAR MAP (FMT) UNFMT ZOOM

VIEW

Syntax:

```
>>-- View --+--------------------------------------+-------------------------><
            |                                      |
            +--------- * --------------------------+
            |                                      |
            |      +-- , --+                       |
            |      |       |                       |
            |  +---+-------+---------------------+ |
            |  |                                 | |
            |  V                                 | |
            +----+-------------+-- record_type --+-+
                 |             |
                 +- + (plus) --+
                 |             |
                 +- - (minus) -+
```

Description:

Select the record types associated with data records to be made visible within an SDE BROWSE or EDIT window. The VIEW command performs the same operation as the "V" prefix area command.

Records that not included in the display of records due to a VIEW CLI command or "V" prefix command, are flagged as being SUPPRESSED. Suppressed records can be identified if shadow lines are set on for SUPPRESSED records.

Where a non-generic record type is specified, the DRECTYPE setting is updated to be the first *record_type* parameter specified on the VIEW command.

If no parameters are specified, then the default record type is used.

Where more than one record type is selected in a multi record view, the appropriate column headings are displayed at each change of record type.

In a single-record view, horizontal scrolling occurs beteween visible records only.

VIEW is assigned to **F16** by default.

Parameters:

`+ (plus)`
Add records of the specified record type to the display of existing visible record types.

`- (minus)`

Remove records of the specified record type from the display of existing visible record types.

`record_type`

The record type of any RTO defined within the current structure definition object (SDO) or the internal record type, "Record" (or "UnMapped").

The *record_type* specification may end with an "*" (asterisk) symbol so that it is treated as a name mask. All RTO record-type names that start with the string preceding the "*" will be identified by the VIEW operation.

If no "+" (plus) or "-" (minus) qualifier is specified, then all other record types will be suppressed unless they are also named as parameters on the same VIEW command.
If no *record_type* is specified, the default record type is used.

`* (asterisk)`

Generic record type indicating that records of all record types within the SDO are to be made visible including the internal record type "Record" ("UnMapped") which is used to map data records that have no associated RTO within the SDO. If supplied, this must be the only parameter.

Examples:

`view CompUnit,Source,Instruction`

View records of the specified record types only. All other records will be suppressed.

`view  *`

View records of all record types.

`v   +REC_CARD,  -REC_CUST`

Add records of record type REC_CARD to the current display of records and suppress records of record type REC_CUST.

WHERE

Syntax:

```
                  +- ANY -+    +-- 1 -----------+
                  |       |    |                |
>>--+- WHere -+---+-------+----+----------------+----------------------------><
    |         |   |       |    |                |
    +- ALL ---+   +- NX --+    +-- expression --+
                  |       |    |                |
                  +- EX --+    +-- line_flag ---+
                  +- X ---+
```

Description:

Display only those records (data set records, record segments or DB2 table rows) that are of the default record type and satisfy the specified *expression* or *line_flag* criteria.

Records not assigned the default record type are not tested and so are unaffected by the WHERE operation. i.e. The current (excluded or non-excluded) status of records that are not of the default record type remain unchanged.

By default, the WHERE operation tests each record assigned the default record type, starting at the first record selected for EDIT/BROWSE. Records that satisfy the specified criteria are included for display whereas records that do not satisfy the specified criteria are excluded.

If NX is specified, the test criteria is performed only against non-excluded records of the default record type. Any previously excluded records remain excluded. If EX or X is specified, the test criteria is performed only against excluded records of the default record type. However, any non-excluded records of the same record type are automatically excluded. Note that, if non-excluded records are to remain unaffected when using expression search critaria against excluded records, then the MORE command must be used instead.

The first record that satisfies the specified WHERE criteria becomes the current line, otherwise, if no records satisfy the criteria, the Top of Data line becomes the current line.

If WHERE is executed with no parameters, then all excluded records of the default record type become non-excluded and so are included in the display.

In single record view, the display area will be scrolled to the first data record that satisfies the specified criteria.

Note that, WHERE uses an expression to filter records that have already been loaded or are eligible to be loaded by the Data Editor browse or edit session. However, the FILTER parameter of the EDIT and BROWSE primary commands use expressions to select records that are to be eligible for load.

Parameters:

`ANY`

Search all EXCLUDED and non-excluded data records of the default record type. Records that satisfy the *expression* or *line_flag* search criteria are included and those that do not are excluded.

`EX`
`X`

Exclude all non-excluded records of the default record type and search only EXCLUDED records of the default record type.

`NX`

Search only non-excluded records of the default record type.

`expression`

expression is a valid SDE expression which supports function calls, *record_type* field names and references, sub-expressions, arithmetic, relational and logical operators. The result of *expression* must be numeric and is treated as being Boolean in nature with a zero value indicating a "false" condition and any non-zero value indicating a "true" condition.

If the result of *expression* is "true" then the record being tested is selected for display.

Default *expression* is "1" indicating that all non-suppressed records are selected for display.

`line_flag`

line_flag may be specified to locate a record or record segment that has been flagged with the requested line flag.

Each valid *line_flag* keyword detailed below corresponds to a built-in function.

A description of each *line_flag* keyword may be found in its equivalent function description.

line_flag Keyword	Built-in Function
ERRor	CHANGEERROR()
CHAnge | Chg	CHANGEOK()
ALTered | UPDated	CHANGED()
DATaerror	DATATYPEERROR()
DUPkey	DUPLICATEKEY()
EMPTY	EMPTYSLOT()
EXcluded | X	EXCLUDED()
LABel	HASPOINT()
COMmand | CMd	HASPREFIXCMD()
IDentify | IDrequired	IDREQUIRED()
NEW	INSERTED()
KEYChanged | KEYChg	KEYCHANGED()
LENgtherror	LENGTHERROR()
EOL | NOEOL	NOEOL()
SAVE	SAVEREQUIRED()
SQLError	SQLERROR()
TRNC | TRUNCated	TRUNCATED()
VALERRor	VALUEERROR()

Examples:

```
where   LastName = C'Jones'
```

For records that are of the default record type, display only those records for which the field "LastName" is equal to "Jones" in mixed case.

```
all     LastName = C'Jones' & Salary > 26000
```

As above but filter the data records further so that the numeric field "Salary" is greater than 26,000.

```
where   #12(2,7) \<< 'E77'
```

Display records that are of the default record type and where the two dimensional character array element belonging to field reference number 12 does not contain the string "E77".

```
where  PostCode >> "SW1"  &  ( DispatchDate = OrderDate | ( Quantity > 30  &  Cost < 2155.53 ) )
```

Display records that are of the default record type where character field "PostCode" begins with string "SW1" **AND** one of the following conditions are true:

1. The contents of fields "DispatchDate" and "OrderDate" are equal.
2. The numeric field "Quantity" is greater than 30 **AND** the numeric field "Cost" is less than 2155.53.

See Also:

FLIP LESS LOCATE MORE

WINDOW

Syntax:

```
              +----- + (plus) -----------------------+
              |                                      |
>>-- Window --+--------------------------------------+----------------------><
              |                                      |
              +----- - (minus) ----------------------+
              |                                      |
              +----- CAScade ------------------------+
              |                                      |
              |                    +-- HOr -------+  |
              |                    |              |  |
              +----- TILE ---------+--------------+--+
              |                    |              |  |
              |                    +-- Vert ------+  |
              |                                      |
              +----- NEWwindow --+-------------------+
              |                  |                   |
              |                  +-- Character ------+
              |                  |                   |
              |                  +-- Hex ------------+
              |                  |                   |
              |                  +-- HEXDump --------+
              |                  |                   |
              |                  +-- Single ---------+
              |                  |                   |
              |                  +-- Sngl -----------+
              |                  |                   |
              |                  +-- Tabl -----------+
              |                                      |
              |                    +-- DOCument --+  |
              |                    |              |  |
              +--+-- RESTore ---+--+--------------+--+
                 |              |  |              |
                 +-- MINimise --+  +-- FRAme -----+
                 +-- MINimize --+
                 |              |
                 +-- MAXimise --+
                 +-- MAXimize --+
```

Description:

Perform window focusing, positioning and sizing operations on the current SDE (document) window view or CBLe MDI parent (frame) window.

Parameters:

`+ (plus)`
Place focus on the next MDI child window.

`- (minus)`
Place focus on the previous MDI child window.

`CASCADE`
Cascade all MDI child windows within the CBLe MDI parent window.

`TILE`
Tile all MDI child windows within the CBLe MDI parent window horizontally (HOR), the default, or vertically (VERT).

`NEWWINDOW`
Open an new SDE window view for the data in the current SDE window view.

NEWWINDOW may optionally be followed by a parameter to define the display format for the new window. If not specified then the new window inherits the format of original.

`Character`
Multi record view with all records or record segments mapped as a single, character field with field name "UnMapped" or "UnMappedSeg". No data conversion is performed.

`Hex`
Same as CHARACTER with the addition that the data is also displayed in Hex below the character display. Note that the Hex display occupies an additional 2 lines of data.

`HEXDump`
Display the focus record or record segment in single record, unformatted hex dump view. See command HEXDUMP for more information.

`Single`
`Sngl`

Single record format with data types formatted according to the record structure. Each field occupies a separate line. Vertical scrolling transfers focus between fields. Horizontal scrolling transfers focus between records.

By default, **PF2** is assigned to distributed CBLe edit macro SDEZOOMW which opens a new view of the record data and executes FORMAT SINGLE to display the record occupying the focus line in single format.

`Table`

Multi-record table format (Default) with data types formatted according to the record structure. Each field occupies a separate column. Vertical scrolling transfers focus between records. Horizontal scrolling transfers focus between fields.

`MAXIMISE`
`MAXIMIZE`

Maximise the current SDE window view (DOCUMENT), the default, or the CBLe MDI parent (FRAME) window.

`MINIMISE`
`MINIMIZE`

Minimise the current SDE window view (DOCUMENT), the default, or the CBLe MDI parent (FRAME) window.

`RESTORE`

Restore the current SDE window view (DOCUMENT), the default, or the CBLe MDI parent (FRAME) window back to its original state, prior to being maximised or minimised.

XMLBROWSE

Syntax:

```
>>--+- XMLBrowse -+---+-----------------+--+-----------------+---------------><
    |             |   |                 |  |                 |
    +- XMLBrws ---+   +-- row_number ---+  +-- #column_ref --+
                      |                                      |
                      +-- #column_ref -----+-----------------+
                                           |                 |
                                           +-- row_number ---+
```

Description:

Applicable to DB2 table edit only. XMLBROWSE opens a new **data editor** view in order to display the XML document text located within a specific row and XML column of the current DB2 data editor view.

Unlike XMLEDIT and XMLVIEW, the display of the XML text is not formatted.

XMLBROWSE is sensitive to the cursor position. By default, if a *row_number* value is omitted and a multi-record data edit view is current, then the row on which the cursor is positioned will be used. Alternatively, for a single record data edit view, the focus table row will be used. If *#column_ref* is omitted, the DB2 table column on which the cursor is positioned will be used.

Error ZZSD691E is returned if the column referenced is not of XML data type.

Parameters:

`#column_ref`

Identifies a single DB2 column, specified as a column number with a hash (#) character prefix, which is of XML data type (e.g. #4).

Default is the column on which the cursor is positioned.

`row_number`

Identifies the row number within the current DB2 table browse/edit view contining the XML document to be browsed.

Default in a multi-record view is the row on which the cursor is positioned. Otherwise, it is the currently displayed row in a single record view.

See Also:

GETXML   PUTXML   XMLEDIT   XMLLENGTH   XMLVIEW

XMLEDIT

Syntax:

```
>>-- XMLEdit --+-----------------+--+-----------------+----------------------><
               |                 |  |                 |
               +-- row_number ---+  +-- #column_ref --+
               |                                      |
               +-- #column_ref -----+-----------------+
                                    |                 |
                                    +-- row_number ---+
```

Description:

Applicable to DB2 table edit only. XMLEDIT opens a new **text editor** view in order to edit and update XML document text located within a specific row and XML column of the current DB2 data editor view.

The text is formatted and displayed in the edit view so that a new line is started after each tag element not followed by text data and after each end tag. Nested tag elements are also indented to illustrate the tag hierarchy.

The text editor view containing the XML data is assigned a temporary data set name. When edit updates to the XML document are saved, FileKit does not attempt to write the data to DASD using the assigned DSN as it would for standard text edit. Instead, the count of alterations made since the last save is reset to zero and, when the text edit view is subsequently closed, the corresponding DB2 row in the data edit view is flagged as changed. When save is executed in the DB2 table edit view, a DB2 SQL UPDATE statement is executed which parses the updated XML document and verifies that it is well-formed before being saved to the DB2 table.

XMLEDIT is sensitive to the cursor position. By default, if a *row_number* value is omitted and a multi-record data edit view is current, then the row on which the cursor is positioned will be used. Alternatively, for a single record data edit view, the focus table row will be used. If *#column_ref* is omitted, the DB2 table column on which the cursor is positioned will be used.

Error ZZSD691E is returned if the column referenced is not of XML data type.

Parameters:

#column_ref

Identifies a single DB2 column, specified as a column number with a hash (#) character prefix, which is of XML data type (e.g. #4).

Default is the column on which the cursor is positioned.

row_number

Identifies the row number within the current DB2 table browse/edit view contining the XML document to be edited.

Default in a multi-record view is the row on which the cursor is positioned. Otherwise, it is the currently displayed row in a single record view.

See Also:

GETXML PUTXML XMLBROWSE XMLLENGTH XMLVIEW

XMLGEN

Syntax:

```
>>--- XMLgen ----------------------------------------------------------------><
```

Description:

XMLGEN with no parameters will open the SDE XML Generation Panel to generate XML for the contents of the focus SDE data edit or browse display.

If the focus window is not an SDE window view, the general XML Generation dialog panel is displayed.

If parameters are specified, then the general FileKit XMLGEN primary command is executed instead.

See Also:

CSVGEN JSONGEN PRINT

XMLIMPort

Syntax:

```
>>--- XMLIMPort ---- XmlFile ---- StructuredFile ---------------------------><
```

Description:

XMLIMPORT reads the Extensible Markup Language (XML) dataset *XmlFile* and converts it into a **"Tree"** formatted structured dataset *StructuredFile* which may be mapped using supplied mapping structure "*<SystemHLQ>*.**SZZSDIST.SDO(TREE)**".

<SystemHLQ>.**SZZSDIST.SDO(TREE)** formats two record-types:

1. **TAG** comprising the following fields:

Field Name	*Description*
Id	T=Tag C=Content
Indent	Indentation (Level relative to previous)
Lev	Level (root=1)
NewL	nl=First item on the source line
Clos	Self= self-closing tag
LineNo	XML source line number
Tag	Tag name
AttrN	Number of attributes.
Attr	An array of up to 32 attribute name/value pairs
Name(1)	1st Attribute Name
Value(1)	1st Attribute Value
Name(n)	nth Attribute Name
Value(n)	nth Attribute Value

2. **CONTENT** comprising the following fields:

Field Name	*Description*
Id	T=Tag C=Content
Indent	Indentation (Level relative to previous)
Lev	Level (root=1)
NewL	nl=First item on the source line
Clos	Self= self-closing tag
LineNo	XML source line number
ParentTag	Name of Parent Tag
Length	Length of Content Text
Text	Content Text

Parameters:

XmlFile
Specifies the fileid of the source file containing Extensible Markup Language (XML).

StructuredFile
Specifies the fileid of the output file that you may subsequently browse using the supplied mapping structure "*<SystemHLQ>*.**SZZSDIST.SDO(TREE)**".

Examples:

```
  xmlimp   <SystemHLQ>.SZZSDIST.IPO(ZZSCOMPH)   USER123.TREE(ZZSCOMPH)


  browse  USER123.TREE(ZZSCOMPH)   using <SystemHLQ>.SZZSDIST.SDO(TREE)
```

XMLLENGTH

Syntax:

```
>>-- XMLLength --+-----------------+--+-----------------+--------------------><
                 |                 |  |                 |
                 +-- row_number ---+  +-- #column_ref --+
                 |                                      |
                 +-- #column_ref -----+-----------------+
                                      |                 |
                                      +-- row_number ---+
```

Description:

Applicable to DB2 table edit and browse only. XMLLENGTH displays the length of text belonging to an XML document located within a specific row and XML column of the current DB2 data editor view.

The message format is:

```
  ZZSD697I XML document length of column "col_name" in row row_number
           (DocId=document_id) is length.
```

Where:

- *col_name* is the name of the DB2 XML column.
- *row_number* is the row number within the data edit display.
- *document_id* is the integer value found in the implicit DOCID column which uniquely identifies the row containing the XML document.
- *length* is the length of the XML document text.

XMLLENGTH is sensitive to the cursor position. By default, if a *row_number* value is omitted and a multi-record data edit view is current, then the row on which the cursor is positioned will be used. Alternatively, for a single record data edit view, the focus table row will be used. If #*column_ref* is omitted, the DB2 table column on which the cursor is positioned will be used.

Error ZZSD691E is returned if the column referenced is not of XML data type.

Parameters:

`#column_ref`

Identifies a single DB2 column, specified as a column number with a hash (#) character prefix, which is of XML data type (e.g. #4).

Default is the column on which the cursor is positioned.

`row_number`

Identifies the row number within the current DB2 table browse/edit view contining the XML document whose length is to be displayed.

Default in a multi-record view is the row on which the cursor is positioned. Otherwise, it is the currently displayed row in a single record view.

See Also:

GETXML PUTXML XMLBROWSE XMLEDIT XMLVIEW

XMLVIEW

Syntax:

```
>>-- XMLView --+-----------------+--+-----------------+----------------------><
               |                 |  |                 |
               +-- row_number ---+  +-- #column_ref --+
               |                                      |
               +-- #column_ref -----+-----------------+
                                    |                 |
                                    +-- row_number ---+
```

Description:

Applicable to DB2 table edit and browse only. XMLVIEW opens a new **text editor** view in order to perform a read-only edit of the XML document text located within a specific row and XML column of the current DB2 data editor view.

The text is formatted and displayed in the edit view so that a new line is started after each tag element not followed by text data and after each end tag. Nested tag elements are also indented to illustrate the tag hierarchy.

The text editor view containing the XML data is assigned a temporary data set name and, although alterations to the text are possible, its contents cannot be used to update the XML column data in the DB2 table browse/edit view. However, alterations may be saved to a data set, library member or HFS/ZFS file using the text edit primary command SAVE or SSAVE with parameter *fileid*.

XMLVIEW is sensitive to the cursor position. By default, if a *row_number* value is omitted and a multi-record data edit view is current, then the row on which the cursor is positioned will be used. Alternatively, for a single record data edit view, the focus table row will be used. If *#column_ref* is omitted, the DB2 table column on which the cursor is positioned will be used.

Error ZZSD691E is returned if the column referenced is not of XML data type.

Parameters:

#column_ref

Identifies a single DB2 column, specified as a column number with a hash (#) character prefix, which is of XML data type (e.g. #4).

Default is the column on which the cursor is positioned.

row_number

Identifies the row number within the current DB2 table browse/edit view contining the XML document to be viewed.

Default in a multi-record view is the row on which the cursor is positioned. Otherwise, it is the currently displayed row in a single record view.

See Also:

GETXML PUTXML XMLBROWSE XMLEDIT XMLLENGTH

XREF

Syntax:

```
>>-- XREF --+---------------------------------------------------------+------><
            |                                                         |
            +- xref_file -+-------------------------------------------+
                          |                                           |
                          |            +- Foreground -+               |
                          |            |              |               |
                          +- sdo_file -+--------------+--+----------+-+
                                       |              |  |          |
                                       +- Batch ------+  +- max_rc -+
```

Description:

Supplied as an edit macro, the XREF utility may be used to generate a FileKit structure (SDO) from an XREF file (sequential data set or PDS/PDSE member).

If executed with no *sdo_file* parameter, the Create Structure from XREF File (=9.2) panel will be displayed. In this case, if *xref_file* has been specified or XREF has been executed against an entry in a file list, the XREF file fields of the panel will be populated with the *xref_file* name.

If *sdo_file* has been specified, then the utility will generate a CREATE STRUCTURE primary command with parameters that match that of the input XREF syntax.

Parameters:

xref_file

Identifies a single XREF file (sequential data set or PDS/PDSE library member).

sdo_file

Name of the output FileKit structure definition file (sequential data set or PDS/PDSE library member.)

FOREGROUND | BATCH

FOREGROUND (default) will execute the generated CREATE STRUCTURE command in the foreground. BATCH will display a temporary data set in a text edit window which contains JCL to execute the FILEKITB (FileKit data edit batch program). The job will include a single job step that executes the FileKit CREATE STRUCTURE command.

max_rc

The maximum acceptable return code that may be set by the COBOL or PL1 compiler for which structure generation will continue without error.

CREATE STRUCTURE will invoke the relevant compiler for the source copy book identified by the XREF file. This parameter allows the user to perform a one-time override of the maximum compiler return code value set by the COBOL Compiler Options (=0.4.1) or PL/1 Compiler Options (=0.4.2) panel.

Examples:

```
xref  OEM.FAID.XREF(XZ01556)  OEM.SELC320.SDO(XZ01556)  batch  8
```

Generate and display JCL to create a FileKit structure (SDO) member from an XREF member allowing possible return code 8 set by the COBOL compiler.

See Also:

XREFLIB   CREATE STRUCTURE

XREFLIB

Syntax:

```
>>-- XREFLIB -- xref_library+---------------------------+--------------------><
                            |                           |
                            |   +------ , ------+       |
                            |   V               |       |
                            +( -+- member_mask -+- ) ---+
```

Description:

Supplied as an edit macro, XREFLIB executes XREF, once for each selected member of an XREF PDS or PDSE library, in order to create a batch job to convert multiple XREF library members to their corresponding FileKit structures (SDO).

On execution of this utility, a temporary data set is displayed in a text edit window containing JCL to execute the FILEKITB (FileKit data edit batch program). The job will include one job step for each selected XREF member which executes the FileKit CREATE STRUCTURE command.

Parameters:

xref_library

Identifies a single PDS or PDSE library DSN containing XREF members.

member_mask

A member name mask which identifies one or more members within the selected library. The mask supports the following wild cards:

- `*` A single asterisk represents an entire member name or zero or more characters within a member name mask.
- `%` A single percent sign represents exactly one character within a member name mask. Up to 8 percent signs can be specified in each member name mask.

Multiple member name masks may be specified within the single set of parentheses. These must be separated by one or more blanks and/or a "," (comma).

ZOOM

Syntax:

```
>>-- ZOOm ---+----------+----------------------------------------------------><
             |          |
             +-- In ----+
             |          |
             +-- Out ---+
```

Description:

Switches the display format between single record view and multi record view. When switching to single record view the ZOOM command operates on the default record.

If no parameter is specified, then the display format toggles between the single and multi record view.

Parameters:

`IN`

Switch to single record view mode. With FORMAT TABLE in effect this is equivalent to issuing the command FORMAT SINGLE.

`OUT`

Switch to multi record view mode. With FORMAT SINGLE in effect this is equivalent to issuing the command FORMAT TABLE.

SET/QUERY/EXTRACT

Syntax:

```
>>-+-----------+----- option_name ----- value ------------------------------><
   |           |
   +- SET -----+


>>--- Query --------- option_name ------------------------------------------><


                 +-------------------+
                 v                   |
>>--- EXTract ---+-- /option_name ---+---- / ------------------------------><
```

Description:

Structured Data Environment options may set, and their current values queried or extracted into stem-variables for use in REXX macros using the SET, QUERY and EXTRACT commands repectively. Additional operations supported by EXTRACT, allow the user to extract information about individual records and also extract record text.

SDE options may be initialised via the following methods:

1. The (SData) section of the INI file which is processed at FileKit startup.
2. The SDEPROF macro which is executed at the start of every SDE session.
3. The SDE CLI command SET, executed from the command line or an SDE macro.
4. The "Edit Options" entry of the "Options" drop down menu on the parent MDI window menu bar.

SET options take effect at the following different levels:

Global	The option affects all SDE edited files.
File	The option may be set differently for each SDE edited file.
View	The option may be set differently for each SDE window view of the same file.

Parameters:

option_name
: The SDE environment option(s). For EXTRACT, multiple options maybe requested at once by separating each with a blank or "/" (forward slash).

value
: For SET, the new value to be assigned for *option_name*.

Supported Options:

ABBREVIATION
ARRAYALL
ARRAYASCHARACTER
ASCII
AUTOSAVE
AUTOSTRUCTURE
AUXDSNPREFIX
AUXDATACLASS
AUXMGMTCLASS
AUXSTORCLASS
AUXUNIT
BLANKWHENZERO/BWZ/BIZ
BOUNDS
CAPS
CCSID
CLIPBOARD
COLATTRIBUTES
COLOUR, COLOR
COLWIDTH
COLWIDTHAUTO
COMMIT
COMPILER
COMPILERDDSIZE
COPYBOOKPROC
DESCRIPTION
DFPSCALE
DRECTYPE
DSN
DSORG
EDITPRIMEKEY
EOLIN
EOLOUT
FIELD/QP/QF
FILEID
FMODE
FNAME, MBR
FOCUS
FORMAT
FPATH
FTYPE
FVALUE/QP/QF
GENSAVE
GROUP
GROUPASCHARACTER
HEXPUNCTUATION
HEXVIEW
IDSCOPE
IDWARNING
KEY
LASTMSG
LENGTH
LEVEL
LOADWARNING
LRECL
MACROPATH
MAPPING
MAXCOBOLRC
MAXHLASMRC
MAXPL1RC
MAXSTOR
MSGLINE
MSGMODE
MULTIPOINT
NULLCHAR
NULLIFBLANK
OFFSET
PAD
PAGEDEPTH
PAGEWIDTH
PFKEY
POINT
PREFIX
QSEPARATOR
RANDOMIZER
RECFM
RECINFO
RECTYPES
REFERENCE
REGION
RESERVED
RESERVEDLEVEL
RTSCOPE
SAVEOPTIONS
SCALE
SESSION
SHADOW
SIZE
STRUCTURE
TITLE
TYPE
UNDOING
UNNAMED
USEOFFSET
USERNAME
USING
VALUE
VBASE
VENDCHAR
VERSION
VBASE
VIEW
VSHOWEND
VSTRIP
WINNAME
WINPOS
WINSIZE
WRAP
ZEROS

ABBREVIATION - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- ABBREViation ----+-- ON ---+-----------------------------><
   |           |                    |         |
   +- SET -----+                    +-- OFF --+


>>--- Query ------ ABBREViation --------------------------------------------><


>>--- EXTract --- /ABBREViation/ ------------------------------------------><
```

Description:

This option controls whether the group name and field item designators that constitute a field name may be abbreviated, starting with the first letter of the designator, in SDE commands that reference record fields by name. (e.g. FIND, CHANGE, WHERE)

For example, in formatted records that contain the field name 'SPROCKET-SIZE' in a group field 'MARCX', the field may be referenced as M.SPR if this name is unique to that field.

Note that abbreviated field names may **always** be specified for a LOCATE command, regardless of the setting of the ABBREVIATION option.

SET ABBREVIATION takes effect at the Global level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON|OFF`
Controls whether use of abbreviated field names is set ON or OFF.

QUERY Response:

The current setting of the ABBREVIATION option.

EXTRACT Rexx variables:

abbreviation.0	1
abbreviation.1	The current setting of the ABBREVIATION option, **ON** or **OFF**.

ALT - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option ALT, controls the same options in a Data Editor view as those supported by a Text Editor view. See SET ALT in Text Editor documentation.

ARRAYALL - SET/QUERY Option

Syntax:

```
>>-+-----------+-- ARRAYALL ----+-- ON ---+---------------------------------><
   |           |                |         |
   +- SET -----+                +-- OFF --+


>>--- Query ------ ARRAYALL ------------------------------------------------><


>>--- EXTract --- /ARRAYALL/ ----------------------------------------------><
```

Description:

By default, FileKit data edit of formatted data containing variable length array (OCCURS DEPENDING) elements will display the maximum possible number of array elements that may exist within the array.

This option determines whether null elements of the array that are **not** referenced by the "depending on" field, are displayed in formatted, single-record (MAP) view.

SET ARRAYALL takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON | OFF`
: Determine whether unreferenced array fields in formatted single view are displayed (ON) or suppressed (OFF). Default is ON.

QUERY Response:

The current setting of the ARRAYALL option (ON or OFF).

EXTRACT Rexx variables:

arrayall.0	1
arrayall.1	The current setting of ARRAYALL, **ON** or **OFF**.

ARRAYASCHARACTER - SET/QUERY Option

Syntax:

```
                                        +- ON  --+
                                        |        |
>>-+-----------+-- ARRAYAscharacter ----+--------+---+---------------+------->
   |           |                        |        |   |               |
   +- SET -----+                        +- OFF --+   +-- field_col --+

 >-+------------------------------------------------------------------------+-><
   |                                                                        |
   +- FOR -+----------+- record_type -+-------------------------------------+
           |          |               |                                     |
           +- RECord -+               +- IN -+--------------+- struct_name -+
                                             |              |
                                             +- STRUCTure -+
```

Description:

Applicable only to formatted, single dimension array (OCCURS) fields comprising single character (PIC X) elements with the number of array elements defined by a value in a separate numeric field (DEPENDING). ARRAYASCHARACTER option determines whether an array field of this type is displayed as a single, updateable variable length character field (XVARCHAR) or as a number of single character array element fields.

SET ARRAYASCHARACTER takes effect at the View level.

SET Value:

`ON | OFF`
: Display the array field as variable character (ON) or display its elements as individual, single character fields (OFF). Default is ON.

field_col
: The individual column identifying the array field to which the option will apply.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

If *field_col* does not exist in the specified *record_type*, the following error message is returned:

```
ZZSD148E Data element field_col is not defined in
       record type record_type of structure struct_name.
```

Default is the focus column.

`FOR [RECORD]` *record_type*
: Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

`IN [STRUCTURE] struct_name`

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

ASCII - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- ASCii -------+-- ON ---+---------------------------------><
   |           |                |         |
   +- SET -----+                +-- OFF --+


>>--- Query ------ ASCii ---------------------------------------------------><


>>--- EXTract --- /ASCii/ --------------------------------------------------><
```

Description:

This option causes data in all character (AN) fields to be interpreted in ASCII format.

SET ASCII takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON|OFF`

Controls whether ASCII interpretation is in effect (ON) or not (OFF).

Note that setting ASCII OFF does not interpret as EBCDIC data fields defined as being ASCII in the record-type definition. i.e. these types of character field are always interpreted as being ASCII.

QUERY Response:

The current setting of the ASCII option (ON or OFF).

EXTRACT Rexx variables:

ascii.0	1
ascii.1	The current setting of ASCII, **ON** or **OFF**.

AUTOSAVE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- AUTOSave ----+-- ON ---+---+--------------+---------------><
   |           |                |         |   |              |
   +- SET -----+                +-- OFF --+   +-- PROMPT ----+
                                              |              |
                                              +-- NOPROMPT --+


>>--- Query ------ AUTOSave -------------------------------------------------><


>>--- EXTract --- /AUTOSave/ ------------------------------------------------><
```

Description:

This option controls the action taken if unsaved changes exist in edited data and the END (or QUIT) command is executed. Actions are as follow:

AUTOSAVE	Action Taken
ON (PROMPT or NOPROMPT)	Changes to the data are saved without prompting the user before doing so.
OFF PROMPT	END command is terminated with the following message: `ZZSD465I Data Changed - Use SAVE/CANCEL (AUTOSAVE OFF PROMPT is set).`
OFF NOPROMPT	User is prompted to save the changes, discard the changes or to terminate the END command and return to the SDE edit view.

The initial value of AUTOSAVE is set by default to AUTOSAVE OFF NOPROMPT.

SET AUTOSAVE takes effect at the Global level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON|OFF`
Controls whether changes to data will be automatically saved (ON) or not (OFF).

`PROMPT|NOPROMPT`
For AUTOSAVE OFF only, controls whether a prompt message or a popup window is opened containing choices to save the changes, discard them or cancel END and return to the edited data.

QUERY Response:

The current setting of the AUTOSAVE option.

EXTRACT Rexx variables:

autosave.0	2
autosave.1	The current setting of automatic SAVE, **ON** or **OFF**.
autosave.2	The current setting of the prompt window, **PROMPT** or **NOPROMPT**.

AUTOSTRUCTURE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-------+------ AUTOSTRUCTure ----+-- ON -----+---------------------------><
   |       |                         |           |
   +- SET -+                         +-- SAVE----+
                                     |           |
                                     +-- APPLY --+
                                     |           |
                                     +-- OFF ----+


>>--- Query ------ AUTOSTRUCTure --------------------------------------------><


>>--- EXTract -- / AUTOSTRUCTure / ------------------------------------------><
```

Description:

AUTOSTRUCTURE controls the level at which the Structure to Dataset Associations facility is implemented, and is normally controlled via the "Associations" panel in Settings for Data Edit (=0.4.6).

SET AUTOSTRUCTURE takes effect at the Global level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON | SAVE | APPLY | OFF`
Specifies whether or not the structure to dataset associations are automatically defined and/or applied.

SAVE will automatically save associations performed during a data edit or browse operation, **APPLY** will automatically apply saved structure associations to data files edited or browsed without reference to a structure, and **ON** will do both. **OFF** turns off all structure to dataset association processing.

QUERY Response:

The current setting of the AUTOSTRUCTURE option (ON, SAVE, APPLY or OFF).

EXTRACT Rexx variables:

autostructure.0	1
autostructure.1	The current setting of AUTOSTRUCTURE, **ON**, **SAVE**, **APPLY** or **OFF**.

See Also:

SET/QUERY/EXTRACT Option: STRUCTURE

AUXDSNPREFIX - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- AUXdsnprefix --- dsn_prefix ------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ AUXdsnprefix ---------------------------------------------><


>>--- EXTract --- /AUXdsnprefix/ -------------------------------------------><
```

Description:

This option controls the default data set name HLQ prefix used to allocate the auxiliary data set when the Auxiliary Edit technique is required. This option may also be set via the Auxiliary Dataset Settings panel.

The initial value of AUXDSNPREFIX is set by the INI variable *SDE.AuxDSNPrefix* or, where this INI option has not been set, defaults to **%USER%.FILEKIT.SDEAUX**.
When SET AUXDSNPREFIX is executed, the *SDE.AuxDSNPrefix* option is set or updated in the User INI file when the FileKit session is closed. i.e. the auxiliary data set name will be set across FileKit sessions.

SET AUXDSNPREFIX takes effect at the Global level.

SET Value:

dsn_prefix

Valid data set name high level qualifiers, conforming to your system's standards, to be used as the DSN prefix for the generated auxiliary data set.

Qualifiers of the format **.Dyyyyddd.Thhmmssx**, representing the current local date and time, are appended to *dsn_prefix* to complete the auxiliary data set name. If *dsn_prefix* exceeds 26 characters, then the DSN is truncated.

QUERY Response:

The auxiliary DSN prefix that is the current setting of the AUXDSNPREFIX option.

EXTRACT Rexx variables:

auxdsnprefix.0	1
auxdsnprefix.1	The auxiliary DSN prefix that is the current setting of the AUXDSNPREFIX option.

AUXDATACLASS - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- AUXDataclass --- sms_dataclass ---------------------------><
   |           |
   +- SET -----+


>>--- Query ------ AUXDataclass ---------------------------------------------><


>>--- EXTract --- /AUXDataclass/ -------------------------------------------><
```

Description:

This option controls the Storage Management System (SMS) data class name to be used for allocation of a temporary, auxiliary data set when the Auxiliary Edit technique is required. This option may also be set via the ZZSGSETX settings panel.

The initial value of AUXDATACLASS is set by the INI variable *SDE.AuxDataClass*. If this INI option has not been set, no SMS data class will be used in the allocation of the auxiliary data set.
When SET AUXDATACLASS is executed, the *SDE.AuxDataClass* option is set or updated in the User INI file when the FileKit session is closed. i.e. the auxiliary data set data class will be set across FileKit sessions.

SET AUXDATACLASS takes effect at the Global level.

SET Value:

sms_dataclass
The name of a data class defined in the SMS installation.

QUERY Response:

The auxiliary data set data class that is the current setting of the AUXDATACLASS option.

EXTRACT Rexx variables:

auxdataclass.0	1
auxdataclass.1	The auxiliary data set SMS data class that is the current setting of the AUXDATACLASS option.

AUXMGMTCLASS - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- AUXMgmtclass --- sms_mgmtclass ---------------------------><
   |           |
   +- SET -----+


>>--- Query ------ AUXMgmtclass ---------------------------------------------><


>>--- EXTract --- /AUXMgmtclass/ -------------------------------------------><
```

Description:

This option controls the Storage Management System (SMS) management class name to be used for allocation of a temporary, auxiliary data set when the Auxiliary Edit technique is required. This option may also be set via the ZZSGSETX settings panel.

The initial value of AUXMGMTCLASS is set by the INI variable *SDE.AuxMgmtClass*. If this INI option has not been set, no SMS management class will be used in the allocation of the auxiliary data set.
When SET AUXMGMTCLASS is executed, the *SDE.AuxMgmtClass* option is set or updated in the User INI file when the FileKit session is closed. i.e. the auxiliary data set management class will be set across FileKit sessions.

SET AUXMGMTCLASS takes effect at the Global level.

SET Value:

sms_mgmtclass
The name of a management class defined in the SMS installation.

QUERY Response:

The auxiliary data set management class that is the current setting of the AUXMGMTCLASS option.

EXTRACT Rexx variables:

auxmgmtclass.0	1
auxmgmtclass.1	The auxiliary data set SMS management class that is the current setting of the AUXMGMTCLASS option.

AUXSTORCLASS - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- AUXStorclass --- sms_mgmtclass --------------------------><
   |           |
   +- SET -----+

>>--- Query ------ AUXStorclass -------------------------------------------><

>>--- EXTract --- /AUXStorclass/ ------------------------------------------><
```

Description:

This option controls the Storage Management System (SMS) storage class name to be used for allocation of a temporary, auxiliary data set when the Auxiliary Edit technique is required. This option may also be set via the ZZSGSETX settings panel.

The initial value of AUXSTORCLASS is set by the INI variable *SDE.AuxStorClass*. If this INI option has not been set, no SMS storage class will be used in the allocation of the auxiliary data set.
When SET AUXSTORCLASS is executed, the *SDE.AuxStorClass* option is set or updated in the User INI file when the FileKit session is closed. i.e. the auxiliary data set storage class will be set across FileKit sessions.

SET AUXSTORCLASS takes effect at the Global level.

SET Value:

sms_storclass
The name of a storage class defined in the SMS installation.

QUERY Response:

The auxiliary data set storage class that is the current setting of the AUXSTORCLASS option.

EXTRACT Rexx variables:

auxstorclass.0	1
auxstorclass.1	The auxiliary data set SMS storage class that is the current setting of the AUXSTORCLASS option.

AUXUNIT - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- AUXUnit -------- unit_name -------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ AUXUnit --------------------------------------------------><


>>--- EXTract --- /AUXUnit/ -------------------------------------------------><
```

Description:

This option controls the DASD UNIT generic device type or group name used to allocate a non-SMS managed auxiliary data set when the Auxiliary Edit technique is required. This option may also be set via the ZZSGSETX settings panel.

The initial value of AUXUNIT is set by the INI variable *SDE.AuxUnit* or, where this INI option has not been set, defaults to **SYSALLDA**.
When SET AUXUNIT is executed, the *SDE.AuxUnit* option is set or updated in the User INI file when the FileKit session is closed. i.e. the auxiliary data set unit will be set across FileKit sessions.

SET AUXUNIT takes effect at the Global level.

SET Value:

unit_name
Any valid DASD device number, device type or group name.
Default is SYSALLDA.

QUERY Response:

The auxiliary data set UNIT that is the current setting of the AUXUNIT option.

EXTRACT Rexx variables:

auxunit.0	1
auxunit.1	The auxiliary data set UNIT that is the current setting of the AUXUNIT option.

BLANKWHENZERO - SET/QUERY Option

Syntax:

```
                                        +- ON  --+
                                        |        |
>>-+-----------+--+- BLANKWHENZERO -+---+--------+---+---------------+------->
   |           |  |                 |   |        |   |               |
   +- SET -----+  +- BWZ -----------+   +- OFF --+   +-- field_col --+
                  |                 |
                  +- BLANKIFZERO ---+
                  |                 |
                  +- BIZ -----------+

 >-+-----------------------------------------------------------------------+-><
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

Description:

For fields with a numeric datatype this option causes a value of zero to be displayed as blank instead. Typically this is required when the value is predominantly zero, in order to make non-zero values visually stand out.

SET BLANKWHENZERO takes effect at the structure level. Follow up this setting by issuing command SAVESTRUCTURE in order to make the change permanent.

SET Value:

ON | OFF

Display the numeric field as blank when the value is zero (ON), or revert to its standard representation (OFF). Default for this command is ON.

field_col

The individual column identifying the numeric field to which the option will apply.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. ReturnCode).

If *field_col* does not exist in the specified *record_type*, the following error message is returned:

```
  ZZSD148E Data element field_col is not defined in
         record type record_type of structure struct_name.
```

Default is the focus column.

FOR [RECORD] *record_type*

Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

IN [STRUCTURE] *struct_name*

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

BOUNDS - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+--+-- BOUNds --+---+--------------------------+--------------><
   |           |  |            |   |                          |
   +- SET -----+  +-- BNDS ----+   +- left_col -- right_col --+


>>--- Query ------+-- BOUNds --+----------------------------------------------><
                  |            |
                  +-- BNDS ----+


>>--- EXTract ----+- /BOUNds/ --+---------------------------------------------><
                  |             |
                  +- /BNDS/ ----+
```

Description:

Applicable to formatted and unformatted, multi and single record views but invalid for DB2 table edit, BOUNDS defines the leftmost and rightmost columns between which the CHANGE, EXCLUDE, FIND, ONLY and SHIFT commands will operate.

By default, the left bound is set to 1 and the right bound is set to the maximum defined record length plus 1 (LRECL+1) for the file being edited. Note that an error occurs if an attempt is made to set either bound to a value greater than the maximum record length.

Execute SET BOUNDS with no parameters to reset the left and right bounds columns to these defaults.

For variable length records, BOUNDS allows the record length of an individual variable length record to either be preserved or increased (up to the defined maximum) following execution of the SHIFT primary command or the ")/))" or "(/((" line (prefix area) commands. When the right bound is set at a column number which is less than or equal to a record's current record length, that record's length is preserved following a shift operation. When the right bound is set at a column number greater than a record's current record length, shifting data left or right will alter the length of that record.

SET BOUNDS takes effect at the View level.

SET Value:

left_col

An integer value identifying the left bound column. Record data to the left of this column will not be included within the scope of shift and string search operations.

Alternatively, "*" (asterisk) may be specified to represent the current left bound value.

right_col

An integer value identifying the right bound column. Record data to the right of this column will no be included within the scope of shift and string search operations.

Alternatively, "*" (asterisk) may be specified to represent the current right bound value.

If *left_col* is specified, *right_col* is mandatory.

QUERY Response:

The current left and right bound values for the BOUNDS option.

EXTRACT Rexx variables:

bounds.0 | bnds.0	2
bounds.1 | bnds.1	The current left bound column number.
bounds.2 | bnds.2	The current right bound column number.

CAPS - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- CAPs ----+-- ON ---+--------------------------------------><
   |           |            |         |
   +- SET -----+            +-- OFF --+

>>--- Query ------ CAPs -----------------------------------------------------><

>>--- EXTract --- /CAPs/ ----------------------------------------------------><
```

Description:

This option controls whether upper casing of alpha characters will occur for all data within fields that are changed.

If record or record segment data is unformatted (record type "UnMapped" or "UnMappedSeg" respectively), then all alpha characters in the unformatted data will be upper cased.

SET CAPS takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

ON|OFF

Controls whether automatic upper casing is set ON or OFF.

QUERY Response:

The current setting of the CAPS option.

EXTRACT Rexx variables:

caps.0	1
caps.1	The current setting of CAPS option, **ON** or **OFF**.

CCSID - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- CCSID ----- nnn ------------------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ CCSID ----------------------------------------------------><


>>--- EXTract --- /CCSID/ --------------------------------------------------><
```

Description:

This option controls the default Coded Character Set Identifier (CCSID) used in the XMLGEN command when executed in batch. For interactive execution the user's 3270 terminal CCSID is used as the default (and cannot be changed by this SET option).

SET CCSID takes effect at the global level.

SET Value:

nnn
A valid Coded Character Set Identifier. This must be the integer identifier of an EBCDIC single byte character set CCSID supported by the IBM z/OS UNICODE character conversion feature.

QUERY Response:

```
   ZZSD079I CCSID tccsid CHARACTER SET cs CODE PAGE cp BATCH CCSID bccsid
```

Where:

tccsid
is the CCSID associated with the 3270 terminal (0 in batch).

cs
is the character set identifier of the terminal CCSID (0 in batch).

cp
is the code page of the terminal CCSID (0 in batch).

bccsid
is the default batch input CCSID. This will have be set to the user's terminal CCSID during interactive execution of any of the FileKit structured data commands (if not already set) and can be changed with the SET CCSID command.

EXTRACT Rexx variables:

ccsid.0	4
ccsid.1	The terminal CCSID (0 in batch)
ccsid.2	The terminal character set (0 in batch)
ccsid.3	The terminal code page (0 in batch)
ccsid.4	The default batch input CCSID

CLIPBOARD - QUERY/EXTRACT

Syntax:

```
>>--- Query ------ CLIPboard ------------------------------------------------><

>>--- EXTract --- /CLIPboard/ -----------------------------------------------><
```

Description:

QUERY and EXTRACT CLIPBOARD reports the current content of the FileKit clipboard.

QUERY Response:

Displays the number of elements in the clipboard and, if there are elements, the type of elements, either whole lines (type *LINE*) or contiguous sets of columns (type *COLUMN*).

For example:

```
 ZZSD079I CLIPBOARD ELEMENTS 5 TYPE COLUMN
```

EXTRACT Rexx variables:

clipboard.0	2
clipboard.1	The number of elements in the clipboard.
clipboard.2	The type of elements in the clipboard, *LINE* or *COLUMN*. (Null if clipboard.1=0).

See Also:

CLIPBOARD

COLATTRIBUTES - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- COLAttributes ----+-- ON ---+----------------------------><
   |           |                     |         |
   +-- SET -----+                    +-- OFF --+

>>--- Query ------ COLAttributes -------------------------------------------><

>>--- EXTract --- /COLAttributes/ -----------------------------------------><
```

Description:

Applicable to DB2 table edit and browse only, this option controls the display of the DB2 results table column attributes when a row is presented in single format view. (See commands FORMAT and ZOOM)

Column attributes are displayed under header **PkUIFCND** and reflects information obtained from the DB2 catalog table SYSIBM.SYSCOLUMNS as follow.

Attribute	Description
Pk	A 2 byte field displaying the numeric position of the column within the Primary Key, otherwise blank.
U	Displays "u" which indicates that this column is part of a unique key, otherwise blank.
I	Displays "d" which indicates that this column is part of a non-unique (duplicate) key index, otherwise blank.
F	Displays "f" which indicates that this column is part of a referential constraint foreign key, otherwise blank.
C	Displays "c" which indicates that this column is involved in a check constraint, otherwise blank.
N	Displays "n" which indicates that this column supports a null value, otherwise blank.
D	Displays one of the following single character DEFAULT value indicators. See description of the DEFAULT column of table SYSIBM.SYSCOLUMNS in the *"IBM DB2 SQL Reference"* manual.

SET COLATTRIBUTES takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON|OFF`
Controls whether display of DB2 column attributes is set ON or OFF.

QUERY Response:

The current setting of the COLATTRIBUTES option.

EXTRACT Rexx variables:

colattributes.0	1
colattributes.1	The current setting of column attributes display, **ON** or **OFF**.

COLOUR, COLOR - SET/QUERY/EXTRACT Option

Syntax:

```
                                                            +- NONe -----+
                                                            |            |
>>-+-------+--+- COLOr --+-+- Block ------+-+- Blue ------+-+------------+---><
   |       |  |          | |              | |             | |            |
   +- SET -+  +- COLOur -+ +- CMDline ----+ +- Red -------+ +- BLInk ----+
                           |              | |             | |            |
                           +- Filearea ---+ +- Pink ------+ +- REVerse --+
                           |              | |             | |            |
                           +- HEADing ----+ +- Green -----+ +- Uscore ---+
                           |              | |             |
                           +- HELDData ---+ +- Turquoise -+
                           |              | |             |
                           +- HELDTitle --+ +- Yellow ----+
                           |              | |             |
                           +- HIghlight --+ +- White -----+
                           |              | |             |
                           +- KEY --------+ +- Default ---+
                           |              |
                           +- MODified ---+
                           |              |
                           +- Msgline ----+
                           |              |
                           +- NONDisplay -+
                           |              |
                           +- Pending ----+
                           |              |
                           +- PRefix -----+
                           |              |
                           +- PRImary ----+
                           |              |
                           +- PROmpt -----+
                           |              |
                           +- RECFlags ---+
                           |              |
                           +- RECId ------+
                           |              |
                           +- RECLength --+
                           |              |
                           +- RECordtype -+
                           |              |
                           +- Scale ------+
                           |              |
                           +- SEGment ----+
                           |              |
                           +- SHadow -----+
                           |              |
                           +- THIghlight -+
                           |              |
                           +- TOfeof -----+


>>--- Query ---------+- COLOr --+--------------------------------------------><
                     |          |
                     +- COLOur -+


>>--- EXTract --- / -+- COLOr --+- / ----------------------------------------><
                     |          |
                     +- COLOur -+
```

Description:

This option controls the colour display of areas within the SDE window view.

SET COLOUR takes effect at the File level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`BLOCK`
Text within a marked block field. Default colour is BLUE REVERSE.

`CMDLINE`
Text entered on the command line. Default colour is GREEN.

`FILEAREA`
Untagged, unmarked text within display area. Default colour is BLUE

`HEADING`
Field Name, Field Reference and Field Data Type header lines. Default colour is WHITE.

`HELDDATA`
Field data within the display area belonging to field columns that have been HELD using the SELECT command. Default colour is TURQUOISE.

`HELDTITLE`
Field header (and scale) lines within the display area belonging to field columns that have been HELD using the SELECT command. Default colour is YELLOW.

`HIGHLIGHT`
Highlighted (tagged) lines. Default colour is YELLOW.

`KEY`
Data field(s) occupying the KEY field of a KSDS data set. Default colour is BLUE USCORE.

`MODIFIED`
Supported for DB2 table edit, fields containing changed data. Default colour is YELLOW.

`MSGLINE`
Message lines (with MSGLINE ON). Default colour is RED.

`NONDISPLAY`
Fields of type Character or AlphaNumeric (AN) which contain non-printable characters. Default colour is DEFAULT USCORE.

`PENDING`
Pending commands in the prefix area. Default colour is RED.

`PREFIX`
Prefix area (with PREFIX ON). Default colour is GREEN.

`PRIMARY`
Untagged, unmarked primary record segment text within display area. Default colour is TURQUOISE.

`PROMPT`
The colour of the command line prompt. Default colour is BLUE.

`RECFLAGS`
Record Flags (with RECINFO ON FLAGS/ALL) Default colour is yellow.

`RECID`
Record TTR/OFFSET or RBA field (with RECINFO ON ID/ALL) Default colour is yellow.

`RECLENGTH`
Record Length (with RECINFO ON LENGTH/ALL) Default colour is yellow.

`RECORDTYPE`
Record type line. Default colour is PINK.

`SCALE`
Scale line (with SCALE ON). Default colour is WHITE.

`SEGMENT`
Record Segment name header line. Default colour is YELLOW.

`SHADOW`
Shadow lines (for excluded lines when SHADOW ON). Default colour is WHITE.

`THIGHLIGHT`
Highlighted targets. Default colour is GREEN REVERSE.

`TOFEOF`
Top of File and End of File lines. Default colour is YELLOW.

`BLUE | RED | PINK | GREEN | TURQUOISE | YELLOW | WHITE | DEFAULT`
Supported colours on each field. If DEFAULT is specified, the default colour for the field is set.

`BLINK | REVERSE | USCORE | NONE`
Extended highlighting of the specified field. The colour may blink, be displayed in reverse video or be underlined. Default is NONE.

QUERY Response:

For each coloured area within the SDE window display, the display area name, current colour setting and extended highlighting option is displayed on an individual message line.

EXTRACT Rexx variables:

color.0 colour.0	Number of areas within the display for which a colour option may be assigned.
color.i colour.i	One stem for each area within the display. The value of each compound variable is an upper case string containing the display area name, the current colour setting and extended highlighting option.

COLWIDTH - SET/QUERY Option

Syntax:

```
>>-+-----------+-- COLWidth --- field_col --- width ------------------------->
   |           |
   +- SET -----+

 >-+---------------------------------------------------------------------+-><
   |                                                                     |
   +- FOR -+----------+- record_type -+----------------------------------+
           |          |               |                                  |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+

>>--- Query ------ COLWidth --- field_col ----------------------------------><
```

Description:

This option controls the number of characters currently displayed (column width) in a specified column field of the specified record type mapping.

Column width cannot be increased beyond the column's defined maximum width.

For SDE DB2 table EDIT, SDE EDIT and SDE BROWSE, the default width of a column within the display of formatted records is the maximum defined for that column. For SDE DB2 table BROWSE only, the displayed width of variable length columns is reduced to be equal to the maximum length of the data occupying a field in the column.

SET COLWIDTH may be preceeded by modal primary command PERMANENT or TEMPORARY (default) to determine whether the column's definition in the FileKit structure (SDO) is updated with the new column width value. If either of these commands are used, the SET keyword becomes mandatory.

SET COLWIDTH values take effect at the File level.

SET Value:

field_col
The individual column field for which the column width will be set.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

If *field_col* does not exist in the specified *record_type*, the following error message is returned:

```
ZZSD148E Data element field_col is not defined in
       record type record_type of structure struct_name.
```

width
The number of characters to be set as the column width. If this value exceeds the column's maximum width, then the maximum width value is used.

`FOR [RECORD]` *record_type*
Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

`IN [STRUCTURE] struct_name`

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

QUERY Response:

The column field name referenced by *field_col* in the default record type followed by the current and maximum values of the column's width.

COLWIDTHAUTO - SET/QUERY Option

Syntax:

```
>>-+-----------+-- COLWIDTHAuto ----+-- ON ---+---+--------------+---------------><
   |           |                    |         |   |              |
   +- SET -----+                    +-- OFF --+   +-- Browse ----+
                                                  |              |
                                                  +-- Edit     --+
                                                  |              |
                                                  +-- All      --+

>>--- Query ------ COLWIDTHAuto ---------------------------------------------><

>>--- EXTract --- /COLWIDTHAuto/ --------------------------------------------><
```

Description:

This option controls whether for SDE DB2 tables only, the displayed width of variable length columns is reduced to be equal to the maximum length of the data occupying a field in the column. i.e. for a DB2 variable length columns COLWIDTH is automatically set to length of the longest value encountered on initial load of the DB2 result table.

This action may be set ON for either BROWSE only (the default), EDIT only or ALL (EDIT and BROWSE).

If COLWIDTHAUTO is set on for EDIT and the user needs to enter a new value that is longer than any existing value, then the (SET) COLWIDTH command may be entered to set a new column width value for any individual column. e.g.

```
COLW #3 15
```

Alternatively, enter the SELECT (SEL) command without parameters to display the column selection dialog, from which current column widths settings may be updated. To reset all columns width settings to their default value, enter the RESET WIDTH (RES W) command from the SELECT dialog.

SET COLWIDTHAUTO takes effect at the Global level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON|OFF`

Controls whether COLWIDTH will be automatically reduced (ON) or not (OFF).

`BROWSE|EDIT|ALL`

For COLWIDTHAUTO ON only, controls whether this action takes place for BROWSE only, EDIT only or ALL (EDIT and BROWSE).

QUERY Response:

The current setting of the COLWIDTHAUTO option.

EXTRACT Rexx variables:

`colwidthauto.0`	2
`colwidthauto.1`	The current setting of COLWIDTHAUTO, **ON** or **OFF**.
`colwidthauto.2`	The level at which COLWIDTHAUTO ON takes effect, **BROWSE**, **EDIT** or **ALL**.

COMMIT - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+- COMMIT -+-- ONEXit -------+-------------------------------><
   |           |          |                 |
   +- SET -----+          +-- ONSAve -------+
                          |                 |
                          +-- ONCLeansave --+

>>--- Query ------ COMMIT ---------------------------------------------------><

>>--- EXTract --- /COMMIT/ --------------------------------------------------><
```

Description:

COMMIT only applies to DB2 table edit.

The COMMIT option determines when the changes made in a DB2 table edit session are committed to the database (with the SQL COMMIT statement).

SET COMMIT takes effect at the file level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ONEXIT`
Any saved changes to the table are committed only when exiting the edit session.

`ONSAVE`
Changes are committed after each SAVE command even if SQL errors are reported for some edited rows.

`ONCLEANSAVE`
Changes are committed after each SAVE command only if no SQL errors are reported for any edited row.

QUERY Response:

The current setting of the COMMIT option, for example:

```
ZZSD079I COMMIT ONCLEANSAVE
```

EXTRACT Rexx variables:

commit.0	1
commit.1	The current setting of the COMMIT option, **ONEXIT**, **ONSAVE** or **ONCLEANSAVE**.

COMPILER - SET/QUERY/EXTRACT Option

Syntax:

```
                               +-------------------------------------+
                               V                                     |
>>-+-----------+-- COMPiler ----+-- COBOL --+---- compiler_module --+--------><
   |           |                 |           |
   +- SET -----+                 +-- PL1 ----+

>>--- Query ------ COMPiler -------------------------------------------------><

>>--- EXTract --- /COMPiler/ ------------------------------------------------><
```

Description:

This option controls the name of the COBOL and PL1 module to be called by SDE when compilation of copy books is required (for CREATE STRUCTURE.)

The initial COMPILER values are set by the INI variables SDE.COBOLCompiler and SDE.PL1Compiler or, where these INI options have not been set, defaults to IGYCRCTL for COBOL and IBMZPLI for PL1. Note that the SDE.COBOLCompiler and SDE.PL1Compiler options may be set to a fully qualified load library DSN and member name.
When SET COMPILER is executed, the SDE.COBOLCompiler and/or SDE.PL1Compiler option is set or updated in the User INI

file when the FileKit session is closed. This means that the compiler module names will be set across FileKit sessions.

SET COMPILER values take effect at the Global level.

SET Value:

`COBOL`
`PL1`
: Specifies that the following token is the COBOL or PL1 compiler module name.

compiler_module
: The 8 byte member name of the compiler module.

: SDE will invoke the compiler module via standard system library search chain. This replaces any fully qualified load library name that may have been included as part of the SDE.COBOLCompiler or SDE.PL1Compiler INI option specifications.

QUERY Response:

The string "COBOL", the COBOL compiler name followed by the string "PL1" and the PL1 compiler name which are the current settings of the COMPILER option.

EXTRACT Rexx variables:

compiler.0	2
compiler.1	The current setting for the COBOL compiler name.
compiler.2	The current setting for the PL1 compiler name.

COMPILERDDSIZE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+- COMPILERDdsize -+- SYSAdata -+-+- Blk -------+- pri - sec -><
   |           |                  |            | |             |
   +- SET -----+                  +- SYSIn ----+ +- Blocks ----+
                                  |            | |             |
                                  +- SYSMdeck -+ +- Cylinders -+
                                  |            | |             |
                                  +- SYSPrint -+ +- Trk -------+
                                  |            | |             |
                                  +- SYSUt ----+ +- Tracks ----+

>>--- Query ----- COMPILERDdsize --------------------------------------------><

>>--- EXTract -- /COMPILERDdsize/ -------------------------------------------><
```

Description:

This option controls the amount of disk space dynamically allocated to work datasets when the COBOL compiler, the PL/1 compiler, or the high level assembler is invoked by FileKit to create data structure definition objects (SDOs) from COBOL or PL/1 data definitions or assembler DSECTs.

The default values of these options are:

DD Name	Allocation Unit	Primary Allocation	Secondary Allocation
SYSADATA	CYL	1	5
SYSIN	TRK	1	5
SYSMDECK	TRK	1	5
SYSPRINT	CYL	1	5
SYSUT	CYL	1	5

These defaults should be adequate for most cases but may need adjusting if very large structures containing many record types are to be generated.

These settings can also be viewed and modified using the Compiler work file allocation settings panel.

SET COMPILERDDSIZE values take effect at the Global level.

Set value:

`SYSAdata`
Specify the size of the SYSADATA associated data file generated by the compilers and the assembler to describe attributes of the generated code.

`SYSIn`
Specify the size of the SYSIN source data set which FileKit generates as input to the compilers and assembler.

`SYSMdeck`
Specify the size of the SYSMDECK dataset used by the COBOL compiler.

`SYSPrint`
Specify the size of the SYSPRINT listing dataset produced by the compilers and assembler.

`SYSUt`
Specify the size of the SYSUTnn work datasets used by the compilers and assembler.

`Blk`
`Blocks`
Allocate space in blocks.

`Cylinders`
Allocate space in cylinders.

`Trk`
`Tracks`
Allocate space in tracks.

`pri`
Primary allocation quantity.

`sec`
Secondary allocation quantity.

Query Response:

The current settings of all the compiler work dataset size allocations. For example:

```
ZZSD079I COMPILERDDSIZE SYSADATA(CYL,5,5) SYSIN(TRK,1,5)
         SYSMDECK(TRK,1,5) SYSPRINT(CYL,1,5) SYSUT(CYL,1,5)
```

EXTRACT Rexx variables:

The variable extracted for the COMPILERDDSIZE option for each work dataset consists of four blank separated tokens:

1. The work dataset DD name, SYSADATA|SYSIN|SYSMDECK|SYSPRINT|SYSUT.
2. The space allocation unit, BLK|CYL|TRK.
3. The primary allocation quantity.
4. The secondary allocation quantity.

compilerddsize.0	5
compilerddsize.1	The current setting of the SYSADATA dataset allocation size. For example **SYSADATA CYL 1 5**
compilerddsize.2	The current setting of the SYSIN dataset allocation size. For example **SYSIN TRK 1 5**
compilerddsize.3	The current setting of the SYSMDECK dataset allocation size. For example **SYSMDECK TRK 1 5**
compilerddsize.4	The current setting of the SYSPRINT dataset allocation size. For example **SYSPRINT CYL 1 5**
compilerddsize.5	The current setting of the SYSUTnn datasets allocation size. For example **SYSUT CYL 1 5**

COPYBOOKPROC - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+- COPYBOOKProc ---+- COBOL ----+-+- INTERNAL --+-------------><
   |           |                  |            | |             |
   +- SET -----+                  +- PL1 ------+ +- COMPILER --+


>>--- Query ----- COPYBOOKProc ----------------------------------------------><


>>--- EXTract -- /COPYBOOKProc/ ---------------------------------------------><
```

Description:

This option controls how COBOL and PL1 copybooks are to be processed by the CREATE STRUCTURE command if the COPYBOOKPROC (COMPILER|INTERNAL) keyword is not specified in the command.

By default, where available, FileKit will use internal code to process copybooks.

SET COPYBOOKPROC values take effect at the Global level.

Set value:

`COBOL`
Specify the processor of COBOL copybooks.

`PL1`
Specify the processor of PL1 copybooks.

`COMPILER`
The language compiler is to be used to process copybooks.

`INTERNAL`
FileKit internal code is to be used to process copybooks.

Query Response:

The current settings of the COBOL and PL1 copybook processor. For example:

```
   ZZSD079I COPYBOOKPROC COBOL INTERNAL PL1 COMPILER
```

EXTRACT Rexx variables:

The variables extracted for the COPYBOOKPROC option are:

copybookproc.0	2
copybookproc.1	The current setting of the COBOL copybook processor. For example **COBOL INTERNAL**
copybookproc.2	The current setting of the PL1 copybook processor. For example **PL1 COMPILER**

DESCRIPTION - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- DESCription ---- long_description ------------------------><
   |           |
   +- SET -----+


>>--- Query ------ DESCription ----------------------------------------------><


>>--- EXTract --- /DESCription/ ---------------------------------------------><
```

Description:

This option displays and assigns the long description that may be defined as part of an SDE structure (SDO) file. See CREATE STRUCTURE.

SET DESCRIPTION values take effect at the File level.

SET Value:

`long_description`
Specifies a structure description text string which may be no longer than 512 characters. Enclosing quotation marks (") or apostrophes (') may be specified and are stripped when saved in the structure.

QUERY Response:

The string "DESCRIPTION", followed by the complete structure definition text.

EXTRACT Rexx variables:

description.0	1
description.1	The complete structure definition text.

DFPSCALE - SET/QUERY/EXTRACT Option

Syntax:

```
                            +- ON --+
                            |       |
>>-+-------+--- DFPScale ---+-------+-+---------+---| field identifier |-----><
   |       |                |       | |         |
   +- SET -+                +- OFF -+ +- scale -+


>>--- Query --- DFPScale ---| field identifier |-----------------------------><


>>--- EXTract / DFPScale ---| field identifier |--- / -----------------------><
```

```
field identifier:

 |--- field_id -------------------------------------------------------------->

 >-+-----------------------------------------------------------------------+-|
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

Description:

This option applies only to *decimal floating-point* data types. It controls how decimal floating-point numbers in the specified record field or table column are formatted for display.

If scaling is *OFF* then all normal non-zero numbers are displayed in the standard exponent form as a decimal number greater than zero and less than 10, and an exponent (a power of 10):

Precision	Byte size	Format width	Format
7	4	14	`±n.nnnnnnE±eee`
16	8	23	`±n.nnnnnnnnnnnnnnnE±eee`
34	16	42	`±n.nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnE±eeee`

If scaling is *ON* then an attempt is made to display the numbers in standard decimal form (without an exponent). This can be done when the width of the number when expressed without an exponent is less than or equal to the format width. If a non-zero value of *scale* is given then an attempt is made to align the decimal points of all numbers as if they all had *scale* decimal places (fraction digits).

When a structure is created either explicitly with the CREATE STRUCTURE command or implicitly as a temporary structure with the EDIT or BROWSE commands, all decimal floating-point fields or columns are set to:

```
   DFPSCALE ON 0
```

SET DFPSCALE may be preceded by modal primary command PERMANENT or TEMPORARY (default) to determine whether the record field or table column definition in the FileKit structure definition (SDO) is updated with the new scale setting. If either of these commands are used, the SET keyword is required.

SET DFPSCALE takes effect at the File level.

Field identification parameters:

This SET option applies to a particular decimal floating-point field or column which must be identified by supplying the following:

`field_id`

The identifier of the record field or table column for which the scale is to be set, queried or extracted.

This can be either the name (e.g MYDECFLOAT) or the reference number (e.g #6) of the field or column.

If this field is not in the current default record type then the following parameters may be used to uniquely identify it:

`FOR [RECORD] record_type`

The record-type in which the specified *field_id* is defined.

`IN [STRUCTURE] struct_name`

Identifies the name of the SDO structure definition in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

Set Value:

`ON`

Scaling is to be set on for the field or column. A number will be displayed in standard decimal format if possible.

`OFF`

Scaling is to be set off for the field or column. All numbers are displayed in standard exponent format.

If neither ON nor OFF keywords are given then ON is implied.

`scale`

The number of decimal places (fraction digits) to be allowed for in the scaled format. If this value is greater than zero, numbers with at most this number of decimal places will be aligned so that their decimal point is placed *scale+1* character positions in from the right of the display area.

If no scale value is given then the current value is unchanged.

QUERY Response:

The query response is the message:

```
ZZSD079I DFPSCALE on_or_off scale field_name
         FOR RECORD record_type
         IN STRUCTURE structure_name
```

Extract REXX variables:

`dfpscale.0`	5
`dfpscale.1`	ON or OFF.
`dfpscale.2`	The scale value.
`dfpscale.3`	The field or column name.
`dfpscale.4`	The record type name.
`dfpscale.5`	The structure name.

Examples:

In the following examples a long format (8 byte) decimal floating-point table column named *D1* reference *#1* is shown with different settings of DFPSCALE.

Command	Display
SET DFPSCALE OFF #1	D1 #1 DECFLOAT(16) <---+----1----+----2--> -2.220000000000000E+001 0 +1.345000000000000E+000 +6.330000000000000E+000 +1.036660000000000E+002
SET DFPSCALE ON 0 #1	D1 #1 DECFLOAT(16) <---+----1----+----2--> -22.2 0 1.345 6.33 103.666
SET DFPSCALE ON 3 #1	D1 #1 DECFLOAT(16) <---+----1----+----2--> -22.2 0 1.345 6.33 103.666

DRECTYPE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- DRECtype --- record_type ---------------------------------><
   |           |
   +- SET -----+

>>--- Query ------ DRECtype -------------------------------------------------><

>>--- EXTract --- /DRECtype/ ------------------------------------------------><
```

Description:

This option controls the record type to be used as the default record type if FileKit SDE is unable to determine the default record type from the focus line.

The initial value of DRECTYPE is set when the SDE window is opened via an EDIT or BROWSE command. In addition to the SET DRECTYPE command, DRECTYPE may be dynamically set following a VIEW, WHERE, MORE or LESS command or any command that requires a record type.

See section *Default Record Type* for further information.

SET DRECTYPE takes effect at the View level.

SET Value:

record_type
The record type of any RTO defined within the current structure definition object (SDO).

QUERY Response:

The current default record type followed by the record type that is the current setting of the DRECTYPE option.

EXTRACT Rexx variables:

drectype.0	2
drectype.1	The current default record type.
drectype.2	The record type that is the current setting of the DRECTYPE option.

DSN - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option DSN, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET DSN in Text Editor documentation.

DSORG - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option DSORG, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET DSORG in Text Editor documentation.

EDITPRIMEKEY - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-------+---+- EDITPRimekey -+---+-- ON ---+-----------------------------><
   |       |   |                |   |         |
   +- SET -+   +- EPK ----------+   +-- OFF --+

>>--- Query ---+- EDITPRimekey -+--------------------------------------------><
               |                |
               +- EPK ----------+

>>--- EXTract -+ /EDITPRimekey/ +--------------------------------------------><
               |                |
               + /EPK/ ---------+
```

Description:

EDITPRIMEKEY only applies to DB2 edit of tables with primary keys. EPK is an alternative name for this set variable.

When EDITPRIMEKEY is ON, the columns of the primary key are enterable.

When EDITPRIMEKEY is OFF, the columns of the primary key are protected.

SET EDITPRIMEKEY takes effect at the file level. The value is not saved across sessions and may be specified on the edit command with the presence or absence of the EDITPRIMEKEY parameter.

SET Value:

`ON | OFF`
Specifies whether columns of the table primary key are enterable (ON) or not (OFF).

QUERY Response:

The current setting of the EDITPRIMEKEY (EPK) option, **ON** or **OFF**.

EXTRACT Rexx variables:

If EDITPRIMEKEY was used as the name of the extracted variable:

editprimekey.0	1
editprimekey.1	The current setting of the EDITPRIMEKEY option, **ON** or **OFF.**

If EPK was used as the name of the extracted variable:

epk.0	1
epk.1	The current setting of the EDITPRIMEKEY option, **ON** or **OFF.**

EOLIN - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- EOLIn ------+------------+-------------------------------><
   |           |               |            |
   +- SET -----+               +-- CR ------+
                               |            |
                               +-- LF ------+
                               |            |
                               +-- NL ------+
                               |            |
                               +-- CRLF ----+
                               |            |
                               +-- LFCR ----+
                               |            |
                               +-- CRNL ----+
                               |            |
                               +-- string --+

>>--- Query ------ EOLIn ---------------------------------------------------><

>>--- EXTract --- /EOLIn/ --------------------------------------------------><
```

Description:

EOLIN alters the current input EOL (end-of-line) delimiter string used to interpret variable length records obtained from an HFS file for SDE EDIT and BROWSE CLI commands.

An EOLIN value is set for all Data Editor and Text Editor views including those containing non-HFS files.

When an edit view is opened and before the edit data is read, the default EOLIN is automatically set to be one of the following values, in the order of precedence:

1. The EOL parameter argument specified on the EDIT or BROWSE command.
2. For SDE EDIT/BROWSE only, the EOLIN value set in the SDE profile macro (using SET EOLIN).
3. The EOL format value defined in the directory entry.
4. EOLIN=NL (new line).

SET EOLIN takes effect at the File level.

SET Values:

`CR|LF|NL|CRLF|LFCR|CRNL|`*`string`*

Identifies the end-of-line delimiter. Delimiter elements are as follow:

NL	X'15'	New Line.
CR	X'0D'	Carriage Return.
LF	X'0A'	Line Feed.
string	-	A 2-byte user specified character or hex string.

QUERY Response:

The current setting of the EOLIN value.

EXTRACT Rexx variables:

`eolin.0`	1
`eolin.1`	The current setting of the EOLIN value.

EOLOUT - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- EOLOut------+------------+---------------------------------><
   |           |               |            |
   +- SET -----+               +-- CR ------+
                               |            |
                               +-- LF ------+
                               |            |
                               +-- NL ------+
                               |            |
                               +-- CRLF ----+
                               |            |
                               +-- LFCR ----+
                               |            |
                               +-- CRNL ----+
                               |            |
                               +-- string --+

>>--- Query ------ EOLOut ----------------------------------------------------><

>>--- EXTract --- /EOLOut/ ---------------------------------------------------><
```

Description:

EOLOUT determines the output HFS file EOL (end-of-line) delimiter string to be used when saving edited data to an HFS fileid which is not fixed format. i.e RECFM F is **not** the current setting for the edited data.

By default, the EOLOUT value is equal to the EOLIN value established when the Data Editor view is initially opened for edit or browse. When using Update-in-place Edit, EOLOUT cannot be changed and must equal the EOLIN value when the data was read.

An EOLOUT value is also set for non-HFS files allowing the user to subsequently save the data in the edit view to a new HFS file simply by using the SAVE *fileid* command where fileid is an HFS path name.

SET EOLOUT takes effect at the File level.

SET Values:

`CR|LF|NL|CRLF|LFCR|CRNL|`*`string`*

Identifies the end-of-line delimiter. Delimiter elements are as follow:

NL	X'15'	New Line.
CR	X'0D'	Carriage Return.
LF	X'0A'	Line Feed.
string	-	A 2-byte user specified character or hex string.

QUERY Response:

The current setting of the EOLOUT value.

EXTRACT Rexx variables:

`eolout.0`	1
`eolout.1`	The current setting of the EOLOUT value.

EXCLUSIONS - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-------+----- EXCLUSions --------+-- Logical ---+-------------------------><
   |       |                         |              |
   +- SET -+                         +-- Physical --+

>>--- Query ----- EXCLUSions -------------------------------------------------><

>>--- EXTract -- /EXCLUSions/ ------------------------------------------------><
```

Description:

The EXCLUSIONS option applies only to records that are segmented in the formatted display of the data.

For operations that include or exclude lines of data from the display (e.g. EXCLUDE, FIND, ONLY, MORE, LESS), the EXCLUSIONS option controls whether the include/exclude applies **only** to the display of a segment that matches the operation search criteria or **all** segments in the record to which it belongs.

EXCLUSIONS LOGICAL indicates that an individual segment may be included or excluded.

EXCLUSIONS PHYSICAL indicates that all segments in the record are included or excluded.

SET EXCLUSIONS takes effect at the file level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`LOGICAL | PHYSICAL`
Specifies whether individual segments (LOGICAL) or all segments belonging to the same record (PHYSICAL) are included/excluded.

QUERY Response:

The current setting of the EXCLUSIONS option, **LOGICAL** or **PHYSICAL**.

EXTRACT Rexx variables:

exclusions.0	1
exclusions.1	The current setting of the EXCLUSIONS option, **LOGICAL** or **PHYSICAL.**

FIDCHANGED - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option FIDCHANGED, controls the same option in an Data Editor view as supported by the same option in a Text Editor view. See SET FIDCHANGED in Text Editor documentation.

FIELD - EXTRACT Option

Syntax:

```
>>--- EXTract --- /-+-FIELD ---+- / ----------------------------------------><
                    |          |
                    +-FIELDQP -+
                    |          |
                    +-FIELDQF -+
```

Description:

For use in macros, EXTRACT FIELD obtains the characteristics (field reference number, field data type, field name, currently displayed field width and REXX var name) of each field column header belonging to the default record type.

For EXTRACT /**FIELD**/ the field name is the elementary field name with no leading record-type and/or parent structure (group) name(s) added.
e.g. RELEASE_YYYY

For EXTRACT /**FIELDQP**/ the field name is the partially qualified field name with only parent structure (group) name(s) added, each separated by a full stop, but no leading record-type.
e.g. RELEASE_DATE.RELEASE_MM

For EXTRACT /**FIELDQF**/ the field name is the fully qualified field name with both leading record-type and parent structure names added.
e.g. TRACK.RELEASE_DATE.RELEASE_DD

Field data type may be one of the following:

BINTEGER BIT	FIXED FLOATBIN	INTEGER STRUCTURE	XVARCHAR ZONED

CHARACTER	FLOATHEX	UNION	
DECIMAL	HEXADECIMAL	VARCHAR	

Values are obtained for each displayed field column in the order in which they appear in the display. Therefore, the SELECT command will influence the EXTRACT FIELD values.

EXTRACT Rexx variables:

`field.0`
`fieldqp.0`
`fieldfp.0`
	Number of currently selected fields belonging to the default record type.

`field.i`
`fieldqp.i`
`fieldfp.i`
	Up to 5 blank delimtted items relating to the *i*th field in the default record type:
	1. The field reference number (#nn)
	2. The field data type (see above).
	3. The field name (including any array element subscripts in parentheses) For unnamed fields this item will be blank (missing).
	4. The screen field width. This is the greater of the width of the heading and the data.
	5. The REXX variable name that will be assigned for this field should "EXTRACT /FVALUE/" be performed. This differs from the field name (item 3) in that array subscripts will use dotted notation e.g. MYARRAY.4, instead of bracketed notation e.g. MYARRAY(4), and any **minus signs** will be converted to **underscores**.

See Also:

QUERY/EXTRACT Options: COLWIDTH FOCUS FVALUE VALUE

FILEID - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option FILEID, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET FILEID in Text Editor documentation.

FMODE - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option FMODE, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET FMODE in Text Editor documentation.

FNAME, MBR - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option FNAME or MBR, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET FNAME in Text Editor documentation.

FOCUS - EXTRACT Option

Syntax:

```
>>--- EXTract --- /FOCUS/ ---------------------------------------------------><
```

Description:

For use in macros, EXTRACT FOCUS obtains information about the focus field.

EXTRACT Rexx variables:

focus.1	11
focus.1	Line number of the first logical record (segment) in the focus line record group.
focus.2	Line number of the last logical record (segment) in the focus line record group.
focus.3	Shadow state of the focus line. (VISIBLE, EXCLUDED, SUPPRESSED or NOTSELECTED)
focus.4	Description of the focus line record group. (DATA, TOF, EOF, SHADOW or BOUNDS)
focus.5	Record type of the focus line record group.
focus.6	Focus field name.
focus.7	Position of cursor within the focus field.
focus.8	Segment type (PRIMARY or SECONDARY)
focus.9	Focus field name fully qualified.
focus.10	Line number of the first physical record in the focus line record group. This is the same as focus.1 if records are not segmented.
focus.11	Line number of the last physical record in the focus line record group. This is the same as focus.2 if records are not segmented.
focus.12	Line identifier. i.e. the TTR, RBA etc of the current record as displayed by "SET RECID ON HEX"

See Also:

EXTRACT Options: FIELD FVALUE VALUE

FORMAT - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- FORmat ------+-- Character ------+------------------------><
   |           |                |                   |
   +- SET -----+                +-- Hex ------------+
                                |                   |
                                +-- Single ---------+
                                +-- Sngl -----------+
                                |                   |
                                +-- Table ----------+

>>--- Query ------ FORmat ---------------------------------------------------><

>>--- EXTract --- /FORmat/ --------------------------------------------------><
```

Description:

This option controls the display format for the current SDE BROWSE or EDIT window. SET FORMAT is equivalent to the FORMAT command.

SET FORMAT takes effect at the View level.

SET Value:

`Character`
Multi record view with all records (or record segments) mapped as a single, character field with field name "UnMapped" (or "UnMappedSeg"). No data conversion is performed.

`Hex`

Same as CHARACTER with the addition that the data is also displayed in Hex below the character display. Note that the Hex display occupies an additional 2 lines of data.

`Single`
`Sngl`

Single record format with data types formatted according to the record structure. Each field occupies a separate line. Vertical scrolling transfers focus between fields. Horizontal scrolling transfers focus between records.

By default, **PF2** is assigned to distributed CBLe edit macro SDEZOOMW which opens a new view of the record data and executes FORMAT SINGLE to display the record occupying the focus line in single format.

`Table`

Multi-record table format (Default) with data types formatted according to the record structure. Each field occupies a separate column. Vertical scrolling transfers focus between records. Horizontal scrolling transfers focus between fields.

QUERY Response:

The string "FORMAT" followed by the current setting of the FORMAT option (CHAR, HEX, SINGLE or TABLE).

EXTRACT Rexx variables:

`format.0`	1
`format.1`	The current setting of the FORMAT option.

FPATH - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option FPATH, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET FPATH in Text Editor documentation.

FTYPE - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option FTYPE, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET FTYPE in Text Editor documentation. iq

FVALUE - EXTRACT Option

Syntax:

```
>>--- EXTract --- /-+-FVALUE ---+---+-------------+- / ---------------------><
                    |           |   |             |
                    +-FVALUEQP -+   +- fieldname -+
                    |           |
                    +-FVALUEQF -+
```

Description:

For use in macros, EXTRACT FVALUE generates a REXX variable with an assigned value, one for each field column selected for display and belonging to the focus record type. Unnamed field columns are ignored.

Alternatively, if a specific field name (*fieldname*) is specified then a REXX variable with assigned value is generated for this field only. Note that *fieldname* must also have been selected for display and must belong to the focus record type.

The generated REXX variable names are the record field column names, and their assigned values are the character representation of the individual field's contents within the record occupying the focus line. No variables are generated if the focus line is an EXCLUDED, SUPPRESSED or NOT SELECTED shadow line.

For EXTRACT /**FVALUE**/ the name of the generated REXX variable is the elementary field name with no leading record-type and/or parent structure (group) name(s) added.
e.g. RELEASE_YYYY

For EXTRACT /**FVALUEQP**/ the name of the generated REXX variable is the partially qualified field name with only parent structure (group) name(s) added, each separated by a full stop, but no leading record-type.
e.g. RELEASE_DATE.RELEASE_MM

For EXTRACT /**FVALUEQF**/ the name of the generated REXX variable is the fully qualified field name with both leading record-type and parent structure names added.
e.g. TRACK.RELEASE_DATE.RELEASE_DD

Field names that are elements of an array, and so have array suffices, generate REXX compound variables where the field name is the variable name stem and each dimension of the array is represented within the variable name tail. e.g. Field name "AddrLine(6)" generates variable "AddrLine.6".

Any occurrence of the "-" (minus) character in a field column name (as supported by COBOL) is translated to "_" (underscore) in the generated variable name thus avoiding REXX "Bad Arithmetic Conversion" errors. e.g. Field name "XX-HRMN" generates variable "XX_HRMN".

Beware of using variables names within your REXX macro procedure that match field column names in your structured record data. REXX variable values generated by EXTRACT FVALUE will replace any existing value already assigned to a variable name that matches a record field column name.

Because only variables for displayed field columns of the default record type are generated, the SELECT command will influence the effect of EXTRACT FVALUE.

EXTRACT Rexx variables:

EXTRACT FVALUE assigns variables as described above. In addition the following stem variables are assigned.

`fvalue.0`
`fvalueqp.0`
`fvalueqf.0`
	Number of currently selected fields belonging to the default record type.

`fvalue.i`
`fvalueqp.i`
`fvalueqf.i`
	The name of the REXX variable set to the value of the *i*th field.

See Also:

EXTRACT Options: FIELD FOCUS VALUE

GENSAVE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- GENSave ----+- AUTO -------------------+-------------------><
   |           |               |                          |
   +- SET -----+               +- NEWGEN -----------------+
                               |                          |
                               +- NOGEN ------------------+
                               |                          |
                               +- PROMPT -----------------+


>>--- Query ------ GENSave ---------------------------------------------------><


>>--- EXTract --- /GENSave/ --------------------------------------------------><
```

Description:

This option controls the default action taken when a SAVE or FILE command is executed for a member generation in a PDSE version 2 library allocated with a MAXGENS value.

The default action set by GENSAVE may be temporarily overridden by specifying the NEWGEN or NOGEN option on the SAVE, SSAVE, FILE and FFILE primary commands.

The following table identifies the action taken for each of the GENSAVE values.

GENSAVE	Action Taken
AUTO	If the edited member text is for the primary generation (generation 0) then NEWGEN is used. Otherwise, NOGEN is used. This is the FileKit system default and matches ISPF Edit.
NEWGEN	The data is automatically saved to a new primary generation of the member. e.g. When editing generation -3 of a member, data will be saved as a new generation (generation 0) of the member and the data in generation -3 becomes generation -4. The absolute and relative generation values in the data edit window view title bar is updated to be that of the new generation 0.
NOGEN	The data is automatically saved to the same generation of the member. e.g. When editing generation -3 of a member, data will be saved back to generation -3. The generation numbers of all other generations of the same member are unchanged.
PROMPT	The Structured Data Edit (SDE) Save Generation popup window is displayed prompting the user to reply "Yes" to save a new generation, "No" to save to the same generation or "Cancel" to terminate the SAVE command and return to the edit view.

The initial value of GENSAVE is set by default to PROMPT.

SET GENSAVE takes effect at the File level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`AUTO`
: Same as NEWGEN for the primary generation, otherwise NOGEN.

`NEWGEN`
: Forces the save of member generation data automatically creates a new primary generation.

`NOGEN`
: Forces the save of member generation data automatically replaces data in the generation referenced in the current data edit view.

`PROMPT`
: Opens the Structured Data Edit (SDE) Save Generation popup window when a save operation is executed for any member of a PDSE version 2 library which supports multiple generations.

QUERY Response:

The current setting of the GENSAVE option: AUTO, NEWGEN, NOGEN or PROMPT.

EXTRACT Rexx variables:

autosave.0	1
autosave.1	The current setting of generation SAVE, **AUTO**, **NEWGEN**, **NOGEN** or **PROMPT**.

GROUP - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- Group -------+-- ON ---+----------------------------------><
   |           |                |         |
   +- SET -----+                +-- OFF --+


>>--- Query ------ Group ----------------------------------------------------><


>>--- EXTract --- /Group/ ---------------------------------------------------><
```

Description:

This option is applicable to singe-record view only and is used to control whether or not each occurrence of a group item is displayed. Group items correspond to structure, union and root array field names.

SET GROUP takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON|OFF`
: Controls whether GROUP field names are displayed (ON) or not (OFF).

QUERY Response:

The current setting of the GROUP option (ON or OFF).

EXTRACT Rexx variables:

group.0	1
group.1	The current setting of GROUP, **ON** or **OFF**.

GROUPASCHARACTER - SET Option

Syntax:

```
                                        +- ON  --+
                                        |        |
>>-+-----------+-- GROUPAscharacter ----+--------+---+---------------+------->
   |           |                        |        |   |               |
   +- SET -----+                        +- OFF --+   +-- field_col --+

 >-+-----------------------------------------------------------------------+-><
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

Description:

Applicable only to formatted data, this option determines whether a specified group (structure) field, defined within a specified record type mapping, is to be displayed as a fixed length character field or as a formatted group of its component fields.

When a group field which comprises variable length field elements is displayed as a single character field, the group is first expanded to its maximum length. As a safety mechanism and so avoiding destruction of the group field structure, the character display of this variable length, expanded group field is non-enterable and cannot be altered by a CHANGE operation.

If the specified field name *field_col* is not a group field, then the GROUPASCHARACTER option is applied to the next higher level group field within which the field is defined.

Note that the record-type is itself a group field, so specifying a field name whose next higher level group is the record-type group field will display as character all data in the record/segment mapped by its assigned record-type. This is not the same as displaying the record data in unmapped (CHAR or UNFMT) format.

SET GROUPASCHARACTER takes effect at the View level.

SET Value:

`ON | OFF`
: Display the group field as character (ON) or display its formatted component fields (OFF).
Default is ON.

`field_col`
: The individual column identifying the group field to which the option will apply. This may be the group field itself or a field contained within.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

If *field_col* does not exist in the specified *record_type*, the following error message is returned:

```
ZZSD148E Data element field_col is not defined in
       record type record_type of structure struct_name.
```

Default is the focus column.

`FOR [RECORD]` *`record_type`*
: Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

`IN [STRUCTURE] struct_name`

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

HEXPUNCTUATION - SET/QUERY/EXTRACT Option

Syntax:

```
                                     +-- DEFAULT -+
                                     |            |
>>-+-----------+-- HEXPunctuation ---+------------+--------------------------><
   |           |                     |            |
   +- SET -----+                     +-- NONE ----+


>>--- Query ------ HEXPunctuation -------------------------------------------><


>>--- EXTract --- /HEXPunctuation/ ------------------------------------------><
```

Description:

HEXPUNCTUATION controls whether or not data in formatted record fields displayed as printable hexadecimal digits contain "," (comma) and blank character punctuation.

Note that 2 printable hexadecimal digits are displayed for each byte of data within a field displayed as printable hex. If HEXPUNCTUATION is set to DEFAULT, the hex display will have a "," (comma) inserted at each half-word (2-byte) boundary and a blank character at each full-word (4-byte) boundary.

Printable hex display with HEXPUNCTUATION DEFAULT:

```
   <---,---- ----,---- ----,---- ----,---- 1---,---- ----,---- ----,--->
   D9A4,9496 A499,40C8 81A2,40C9 A340,4040 4040,4040 4040,4040 4040,4040
   E3A4,9995 8995,8740 E381,8293 85A2,4040 4040,4040 4040,4040 4040,4040
   C496,957D A340,E896 A440,D985 9485,9482 8599,4040 4040,4040 4040,4040
```

Printable hex display with HEXPUNCTUATION NONE:

```
   <---+----1----+----2----+----3----+----4----+----5----+----6--->
   D9A49496A49940C881A240C9A340404040404040404040404040404040404040
   E3A4999589958740E381829385A2404040404040404040404040404040404040
   C496957DA340E896A440D98594859482859940404040404040404040404040404040
```

SET HEXPUNCTUATION takes effect at the global level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Values:

`DEFAULT|NONE`

DEFAULT turns on printable hex display punctuation, NONE suppresses punctuation.
Default is DEFAULT.

QUERY Response:

The current setting of the HEXPUNCTUATION value.

EXTRACT Rexx variables:

hexpunctuation.0	1
hexpunctuation.1	The current setting of the HEXPUNCTUATION value.

HEXVIEW - SET Option

Syntax:

```
                                    (1) +- ON   --+
                                        |        |
>>-+-----------+-- HEXView -------------+--------+---+---------------+------->
   |           |                        |        |   |               |
   +- SET -----+                        +- OFF --+   +-- field_col --+

 >-+-----------------------------------------------------------------------+-><
   |                                                                       |
   +- FOR -+----------+- record_type -+------------------------------------+
           |          |               |                                    |
           +- RECord -+               +- IN -+-------------+- struct_name -+
                                             |             |
                                             +- STRUCTure -+
```

(1) If *field_col* is not specified, default is either ON or OFF depending on the current HEXVIEW status of the focus column.

Description:

Applicable only to formatted data, this option determines whether a specified field, defined within a specified record type mapping, is to be displayed in its default format (as dictated by the field's data type) or as printable hexadecimal digits.

Note that 2 printable hexadecimal digits are displayed for each byte of data in the field. Furthermore, if option HEXPUNCTUATION is set to DEFAULT, the hex display will have a "," (comma) inserted at each half-word (2-byte) boundary and a blank character at each full-word (4-byte) boundary.

SET HEXVIEW takes effect at the View level.

SET Value:

`ON | OFF`

Display the field as printable hex (ON) or display in its default formatted (OFF).
Default is ON if *field_col* is specified, otherwise the default is either ON or OFF depending on the current HEXVIEW status of the focus column. HEXVIEW will toggle the display of the focus column between its defualt format and printable hex.

field_col

The individual column identifying the field to which the option will apply.

The field column may be identified by its reference number (e.g. #6) or its name (e.g. JobID).

If *field_col* does not exist in the specified *record_type*, the following error message is returned:

```
  ZZSD148E Data element field_col is not defined in
         record type record_type of structure struct_name.
```

Default is the focus column.

`FOR [RECORD]` *record_type*

Identifies the record-type mapping in which the specified *field_col* is defined.

Default is the default record type.

`IN [STRUCTURE]` *struct_name*

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

IDSCOPE - SET/QUERY/EXTRACT Option

Syntax:

```
                                +-- ALL -------+
                                +-- CHANGED ---+
                                |              |
>>-+-----------+-- IDScope -----+--------------+----------------------------><
   |           |                |              |
   +- SET -----+                +-- FLAGGED ---+


>>--- Query ------ IDScope --------------------------------------------------><


>>--- EXTract --- /IDScope/ -------------------------------------------------><
```

Description:

This option controls which record flag must be set on in order for that record to be selected for remap by the IDENTIFY command.

Note that, for segmented record edit, a record may be selected for remap if the required flag is set on for any of its primary or secondary segments.

SET IDSCOPE takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ALL|CHANGED`
Records may be selected for remap if either the ID flag or CHANGED flag is set on.
Default is ALL which is a synonym for CHANGED.

`FLAGGED`
Records may be selected for remap only if the ID flag is set on.

QUERY Response:

The current setting of the IDSCOPE option (CHANGED or FLAGGED).

EXTRACT Rexx variables:

idscope.0	1
idscope.1	The current setting of IDSCOPE, **CHANGED** or **FLAGGED**.

IDWARNING - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- IDWarning ----+-- ON ---+---------------------------------><
   |           |                 |         |
   +- SET -----+                 +-- OFF --+


>>--- Query ------ IDWarning ------------------------------------------------><


>>--- EXTract --- /IDWarning/ -----------------------------------------------><
```

Description:

This option controls whether the prefix area ID warning symbol **==ID>** is displayed when data in an ID field, identified by USE WHEN criteria, is changed in any way.

The ID warning symbol indicates that the IDENTIFY command should be executed to possibly remap data in the record using a different record type definition.

SET IDWARNING takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON|OFF`
Controls whether the ID warning symbol is to be displayed.

QUERY Response:

The current setting of the IDWARNING option.

EXTRACT Rexx variables:

idwarning.0	1
idwarning.1	The current setting of IDWARNING option, **ON** or **OFF**.

KEY - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ KEY ------------------------------------------------------><

>>--- EXTract --- /KEY/ -----------------------------------------------------><
```

Description:

QUERY and EXTRACT KEY reports the data occupying the key field of the KSDS record occupying focus line of the displayed data.

QUERY Response:

The string "KEY" followed by the key field data in printable character and hexadecimal string representation.

EXTRACT Rexx variables:

key.0	1
key.1	The data in the key field of the record occupying the focus line.

LASTMSG - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ LASTmsg --------------------------------------------------><

>>--- EXTract --- /LASTmsg/ -------------------------------------------------><
```

Description:

Obtain the last message or error message generated for the current SDE window view.

Unlike QUERY/EXTRACT LASTMSG for CBLe window views, the recalled message string is automatically returned as is. i.e. No message number and "LASTMSG" prefix string.

QUERY Response:

Displays the last message or error message generated for the current window view. The message may not have been displayed if NOMSG was used or MSGMODE was OFF.

EXTRACT Rexx variables:

lastmsg.0	1
lastmsg.1	The last message issued.

See Also:

EXTRACT Option: MSGMODE

LENGTH - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- LENgth ------ n_bytes ------------------------------------><
   |           |
   +- SET -----+

>>--- Query ------ LENgth ---------------------------------------------------><

>>--- EXTract --- /LENgth/ --------------------------------------------------><
```

Description:

This option controls the record length of the record occupying the focus line.

If the length of a record is increased, then pad characters are appended to the record. The pad character used is that defined by the current value for PAD. Reducing a record's length will truncate record data.

The length of a record may also be updated by overtyping the new value in the Length field displayed in the Record Information area. (See SET/QUERY/EXTRACT RECINFO and RECLEN commands.)

If a structure has been applied to the edited records, then the record type (RTO) associated with the updated record is re-evaluated and the contents of the SDE edit view updated accordingly.

Record length changes are only applicable to variable length records where Full Edit, KSDS Edit or Auxiliary Edit is in effect.

Beware that changing record lengths incorrectly may result in length errors, indicated by =LGTH> or =ERRV> in the prefix area, and so the record data is no longer correctly mapped by the defined record structure.

SET LENGTH takes effect at the File level.

SET Value:

n_bytes
A positive integer value representing the new length assigned to the record.

QUERY Response:

The length of the record occupying the focus line.

EXTRACT Rexx variables:

length.0	1
length.1	Length of the record occupying the focus line.

LEVEL - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- LEVel ------+-- ON ---+----------------------------------><
   |           |               |         |
   +-- SET -----+               +-- OFF --+


>>--- Query ------ LEVel ----------------------------------------------------><


>>--- EXTract --- /LEVel/ ---------------------------------------------------><
```

Description:

For SDE EDIT and BROWSE window views in mapped single record view (MAP), this option controls display of the field's hierarchical level number as a prefix to the field name.

With *LEVEL ON* in effect the Field name value is indented for each successively higher level number.

The LEVEL option operates at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON`

Level number display is set ON.

`OFF`

Level number display is set OFF.

QUERY Response:

The current setting of the LEVEL option, **ON** or **OFF**.

EXTRACT Rexx variables:

level.0	1
level.1	The current setting of the LEVEL option, **ON** or **OFF**.

LOADWARNING - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- LOADWarning --- n_recs ----------------------------------><
   |           |
   +-- SET -----+


>>--- Query ------ LOADWarning ----------------------------------------------><


>>--- EXTract --- /LOADWarning/ ---------------------------------------------><
```

Description:

This option specifies the threshold of number of data set records to be loaded into storage before the Load Warning pop-up message window is displayed prompting the user to either continue or stop loading records for SDE display and edit. The message also provides the user with the option to respect or bypass the load warning when the load theshold is once again encountered on continuing to load records from the data set.

If the user chooses to interrupt load of the data set's records once the load threshold has been reached, then records that have already been loaded are displayed in the SDE window view. If the records were loaded for edit, then Update-in-place edit will be used overriding any EDIT REPLACE request for full edit capabilities.

The purpose of the Load Warning is to allow users to break out of possible time and resource consuming loads of large data sets.

Note the difference between SDE LOADWARNING and CBLe text edit LOADWARNING which specifies number of bytes loaded instead of number of records. SDE SET LOADWARNING has no affect on the CBLe text edit LOADWARNING threshold.

Load Warning applies only to edit techniques where an attempt is made to load all requested records into storage. e.g. no load warning threshold checks are made for KSDS and AUXILIARY edit techniques.

The initial LOADWARNING value is set by the INI variable SDE.LoadWarning or, where this INI option has not been set, defaults to 5000 records.
When SET LOADWARNING is executed, the SDE.LoadWarning option is set or updated in the User INI file when the FileKit session is closed. This means that the load warning threshold will be set across FileKit sessions.

SET LOADWARNING value take effect at the Global level for all SDE EDIT and BROWSE commands and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

n_recs
The number of records to be loaded after which the Load Warning message occurs.

QUERY Response:

The number of records to be loaded which is the current settings of the LOADWARNING option.

EXTRACT Rexx variables:

loadwarning.0	1
loadwarning.1	The current loadwarning threshold value in number of records.

LRECL - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option LRECL, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET LRECL in Text Editor documentation.

MACROPATH - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option MACROPATH, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET MACROPATH in Text Editor documentation.

MAPPING - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- MAPPing ----+-- ON ---+----------------------------------><
   |           |               |         |
   +- SET -----+               +-- OFF --+


>>--- Query ------ MAPPing --------------------------------------------------><


>>--- EXTract --- /MAPPing/ -------------------------------------------------><
```

Description:

This option controls whether formatted data is displayed in its mapped or unmapped format.

With MAPPING OFF, the record type assigned to formatted records or record segments persists but is not used to format the display of the record data. The record data is displayed in its unformatted (unmapped) format, instead.

With MAPPING ON, the record type assigned to formatted records or record segments is re-applied to the display of the record data.

SET MAPPING does not affect unformatted record data. i.e. SDE EDIT/BROWSE of a data set no USING structure (SDO) specified.

SET MAPPING takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON | OFF`
: Formatted record data mapping is set ON or OFF.

QUERY Response:

The current setting of the MAPPING option, **ON** or **OFF**.

EXTRACT Rexx variables:

mapping.0	1
mapping.1	The current setting of the MAPPING option, **ON** or **OFF**.

MAXCOBOLRC - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- MAXCObolrc ------ value ---------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ MAXCObolrc -----------------------------------------------><


>>--- EXTract --- /MAXCObolrc/  --------------------------------------------><
```

Description:

This option controls the maximum acceptable COBOL compiler return code for which an SDE structure will be successfully generated.

The MAXCOBOLRC value is only applicable to execution of the SDE CLI command, CREATE STRUCTURE, with parameter **FROM COBOL *copybook***.

Where a COBOL return code greater than the MAXCOBOLRC value occurs, the CREATE STRUCTURE operation fails with an error message.

The initial value of MAXCOBOLRC is set by the INI variable SDE.MaxCOBOLRC or, where this INI option has not been set, defaults to 4.
When SET MAXCOBOLRC is executed, the SDE.MaxCOBOLRC option is set or updated in the User INI file when the FileKit session is closed. This means that the maximum storage value will be set across FileKit sessions.

SET MAXCOBOLRC takes effect at the Global level.

SET Value:

value
: Maximum acceptable Return Code from the COBOL compiler.

QUERY Response:

"MAXCOBOLRC" followed by a single numeric indicating the current MAXCOBOLRC value.

EXTRACT Rexx variables:

maxcobolrc.0	1
maxcobolrc.1	The value that is the current setting of the MAXCOBOLRC option.

MAXHLASMRC - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- MAXHLasmrc ------ value ----------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ MAXHLasmrc -----------------------------------------------><


>>--- EXTract --- /MAXHLasmrc/  ---------------------------------------------><
```

Description:

This option controls the maximum acceptable HLASM assembler return code for which an SDE structure will be successfully generated.

The MAXHLASMRC value is only applicable to execution of the SDE CLI command, CREATE STRUCTURE, with parameter **FROM HLASM *copybook***.

Where a HLASM return code greater than the MAXHLASMRC value occurs, the CREATE STRUCTURE operation fails with an error message.

The initial value of MAXHLASMRC is set by the INI variable SDE.MaxHLASMRC or, where this INI option has not been set, defaults to 4.
When SET MAXHLASMRC is executed, the SDE.MaxHLASMRC option is set or updated in the User INI file when the FileKit session is closed. This means that the maximum storage value will be set across FileKit sessions.

SET MAXHLASMRC takes effect at the Global level.

SET Value:

value
Maximum acceptable Return Code from the HLASM assembler.

QUERY Response:

"MAXHLASMRC" followed by a single numeric indicating the current MAXHLASMRC value.

EXTRACT Rexx variables:

maxhlasmrc.0	1
maxhlasmrc.1	The value that is the current setting of the MAXHLASMRC option.

MAXPL1RC - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- MAXPL1rc -------- value ----------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ MAXPL1rc -------------------------------------------------><


>>--- EXTract --- /MAXPL1rc/ -----------------------------------------------><
```

Description:

This option controls the maximum acceptable PL/1 compiler return code for which an SDE structure will be successfully generated.

The MAXPL1RC value is only applicable to execution of the SDE CLI command, CREATE STRUCTURE, with parameter **FROM PL1 *copybook***.

Where a PL/1 return code greater than the MAXPL1RC value occurs, the CREATE STRUCTURE operation fails with an error message.

The initial value of MAXPL1RC is set by the INI variable SDE.MaxPL1RC or, where this INI option has not been set, defaults to 4. When SET MAXPL1RC is executed, the SDE.MaxPL1RC option is set or updated in the User INI file when the FileKit session is closed. This means that the maximum storage value will be set across FileKit sessions.

SET MAXPL1RC takes effect at the Global level.

SET Value:

value

Maximum acceptable Return Code from the PL/1 compiler.

QUERY Response:

"MAXPL1RC" followed by a single numeric indicating the current MAXPL1RC value.

EXTRACT Rexx variables:

maxpl1rc.0	1
maxpl1rc.1	The value that is the current setting of the MAXPL1RC option.

MAXSTOR - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- MAXSTOR ---- n_bytes --+-----+----------------------------><
   |           |                          |     |
   +- SET -----+                          +- K -+
                                          |     |
                                          +- M -+

>>--- Query ------ MAXSTOR --------------------------------------------------><

>>--- EXTract --- /MAXSTOR/ -------------------------------------------------><
```

Description:

This option limits the amount of storage available to handle edit of any single data set within the SDE environment.

An SDE edited data set is limited by the lesser of the prevailing MAXSTOR value and the amount of free private area storage above the 16MB line available within the region at the time of open. The edited data set is unaffected by any subsequent change to the MAXSTOR setting.

This limit will be used to determine the SDE edit technique and data record management used to edit the data set. e.g. SDE EDIT of a sequential data set may be of a size which is too large to fit comfortably within the defined area of storage resulting in use of Auxiliary Edit.

If the MAXSTOR value is set to 0 (zero), it is considered to be unset and no artificial limit is imposed.

The initial value of MAXSTOR is set by the INI variable SDE.MaxStor or, where this INI option has not been set, defaults to 0 (zero).
When SET MAXSTOR is executed, the SDE.MaxStor option is set or updated in the User INI file when the FileKit session is closed. This means that the maximum storage value will be set across FileKit sessions.

SET MAXSTOR takes effect at the Global level.

SET Value:

n_bytes
*n_bytes*K
*n_bytes*M

Amount of storage available for SDE edit of a single data set.

Setting this value to 0 (zero) unsets the MAXSTOR limitation.

This value may be specified as a number of bytes (*n_bytes*), number of kilobytes (*n_bytes*K) or a number of megabytes (*n_bytes*M).

QUERY Response:

Four numeric values indicating the current MAXSTOR value and, for the current SDE edited file, the MAXSTOR value at the time the file was opened, the amount of storage allocated for the file edit so far and, of that allocation, the amount of unused storage.

EXTRACT Rexx variables:

maxstor.0	4
maxstor.1	The value that is the current setting of the MAXSTOR option.
maxstor.2	The MAXSTOR value at the time the current file was opened for SDE edit.
maxstor.3	The amount of storage allocated so far for SDE edit of the current file.
maxstor.4	Of the storage allocated so far for SDE edit of the current file, the amount of storage that is unused.

MSGLINE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- MSGLine -- ON -- line_num ---+------------------------+--->< 
   |           |                                |                        |
   +- SET -----+                                +- lines --+-------------+
                                                           |             |
                                                           +-- OVERLAY --+


>>--- Query ------ MSGLine -------------------------------------------------><


>>--- EXTract --- /MSGLine/ -----------------------------------------------><
```

Description:

This option controls the location and number of lines used to display messages in the SDE display window and also display properties of the first message line. The line number is specified as being relative to the top, middle or bottom of the data display area.

SET MSGLINE takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

line_num

Line number of first message line relative to the top, middle or bottom of the document window.

Line numbers relative to the top of the document window are specified as positive integers. e.g. 2, 8

Line numbers relative to the middle of the document window are specified as offsets from M. e.g. M+1, M-3

Line numbers relative to the bottom of the document window are specified as negative integers. e.g. -6, -10

By default *line_num* is set to 1.

lines

The number of lines that message text may occupy within the display area.

If the message text exceeds the number of message lines then an SDE Message List window is opened to display the message text instead.

If *lines* is not specified, the number of message lines last defined in the current CBLe view, is unchanged.

By default *lines* is set to 5.

OVERLAY

Specifies that the first message line should overlay a line normally used to display a line of data. If omitted, the first message line will be reserved for message display only. Second and subsequent message lines always overlay data lines.

Initially OVERLAY is set on.

QUERY Response:

The current setting of the MSGLINE option, **ON** or **OFF** followed by location of the first message line, the number of message lines and whether OVERLAY is in effect.

EXTRACT Rexx variables:

msgline.0	4
msgline.1	The current setting of the MSGLINE option, **ON** or **OFF**.
msgline.2	The current position of the first message line.
msgline.3	The number of message lines.
msgline.4	"OVERLAY", if overlay is in effect.

MSGMODE - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option MSGMODE, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET MSGMODE in Text Editor documentation.

MULTIPOINT - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- MULTIPoint --+-- ON ---+---------------------------------><
   |           |                |         |
   +- SET -----+                +-- OFF --+
```

```
>>--- Query ------ MULTIPoint ----------------------------------------------><
```

```
>>--- EXTract --- /MULTIPoint/ ---------------------------------------------><
```

Description:

This option controls whether or not more than one label name may be assigned to a line in the file display via the SET POINT command or by overtyping a name in the line's prefix area.

SET MULTIPOINT takes effect at the File level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

ON

Allow multiple label names on a single line.

OFF

Do not allow multiple label names on a single line. (Default)

QUERY Response:

The current setting of the MULTIPOINT option, **ON** or **OFF**.

EXTRACT Rexx variables:

multipoint.0	1
multipoint.1	The current setting of the MULTIPOINT option, **ON** or **OFF**.

NAMECASE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-------+----- NAMECASE ---------+-- Mixed -----+-------------------------><
   |       |                        |              |
   +- SET -+                        +-- Upper -----+
>>--- Query ----- NAMECASE --------------------------------------------------><

>>--- EXTract -- /NAMECASE/ -------------------------------------------------><
```

Description:

Option NAMECASE defines whether or not XML and JSON tags are by default upper cased when generated by XMLGEN and JSONGEN utilities respectively.

XMLGEN and JSONGEN use the field names assigned to record structures as tag names in the generated output. NAMECASE MIXED indicates that the field name is used as is with no upper case translation. NAMECASE UPPER indicates upper case translation of the field (tag) name will occur. The defined NAMECASE action may be overridden by XMLGEN/JSONGEN parameter TAGUPPER or NOTAGUPPER.

SET NAMECASE takes effect at the global level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`MIXED | UPPER`
Specifies the default action for XMLGEN/JSONGEN tag generation.

QUERY Response:

The current setting of the NAMECASE option, **MIXED** or **UPPER**.

EXTRACT Rexx variables:

`namecase.0`	1
`namecase.1`	The current setting of the NAMECASE option, **MIXED** or **UPPER.**

NULLCHAR - SET/QUERY/EXTRACT Option

Syntax:

```
                            +-- @ --------------+  +-- _ ----------------+
                            |                   |  |                     |
>>-+-----------+- NULLChar -+- input_indicator -+--+- output_indicator -+----><
   |           |
   +- SET -----+

>>--- Query ------ NULLChar -------------------------------------------------><

>>--- EXTract --- /NULLChar/ ------------------------------------------------><
```

Description:

NULLCHAR only applies to DB2 edit or browse of table columns which can have NULL values.

Use NULLCHAR to specify two special characters which indicate when a column value is NULL.

SET NULLCHAR takes effect at the file level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`input_indicator`
The character which if specified sets the column value to NULL.

For character based fields, this value must be specified as the first and only non-blank value in the input area.

For non-character based fields, this value may specified anywhere in the input area.

The default is @ (the commercial at sign).

`output_indicator`
The character which is used on output to indicate that a column has a NULL value. The default is _ (the underscore).

QUERY Response:

The current NULLCHAR input and output indicator characters. For example:

```
ZZSD079I NULLCHAR @ _
```

EXTRACT Rexx variables:

`nullchar.0`	2
`nullchar.1`	The current NULLCHAR input indicator character.
`nullchar.2`	The current NULLCHAR output indicator character.

NULLIFBLANK - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+- NULLIfblank -+-- ON ---+-----------------------------------><
   |           |               |         |
   +- SET -----+               +-- OFF --+


>>--- Query ------ NULLIfblank ----------------------------------------------><


>>--- EXTract --- /NULLIfblank/ ---------------------------------------------><
```

Description:

NULLIFBLANK only applies to DB2 edit of table columns which can have NULL values.

When NULLIFBLANK is ON, setting a nullable column input field to blanks in the edit view will cause the column value (of whatever data type) to be set to NULL.

When NULLIFBLANK is OFF, there is no special effect when setting a column to blanks. Note that non-character fields may give an invalid data type message in this case.

SET NULLIFBLANK takes effect at the file level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON | OFF`

Specifies whether nullable columns are give the NULL value if entered as blanks (ON) or set to blanks (OFF).

QUERY Response:

The current setting of the NULLIFBLANK options, **ON** or **OFF**.

EXTRACT Rexx variables:

`nullifblank.0`	1
`nullifblank.1`	The current setting of the NULLIFBLANK option, **ON** or **OFF**.

OFFSET/OFST - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+--+- OFFSet --+---+-- Columns---+----------------------------><
   |           |  |           |   |             |
   +- SET -----+  +- OFST ----+   +-- Position--+
                                  |             |
                                  +-- Hex | X --+
                                  |             |
                                  +-- Relative -+
                                  |             |
                                  +-- Offset ---+

>>--- Query -------- OFFSet -------------------------------------------------><


>>--- EXTract ----- /OFFSet/ ------------------------------------------------><
```

Description:

This option controls format of the field offset for SDE EDIT and BROWSE window views in mapped single record view (MAP) when *SHOW OFFSET* is in effect.

SET OFFSET takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect. i.e. if multiple SDE views are open for the same file, then this option may be individually controlled for each view.

SET Value:

`Columns | Position`
: Displays the location of the start of each field as a decimal position.

`Hex | X`
: Displays the location of the start of each field as a hexadecimal offset.

`Relative | Offset`
: Displays the location of the start of each field as a decimal offset.

QUERY Response:

The current setting of the OFFSET option, **POSITION/HEX/OFFSET**.

EXTRACT Rexx variables:

offset.0	1
offset.1	The current setting of the OFFSET option, **POSITION/HEX/OFFSET**.

OPSYS - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ OPSYS ---------------------------------------------------><

>>--- EXTract --- /OPSYS/ --------------------------------------------------><
```

Description:

QUERY/EXTRACT OPSYS provides information about the current operating environment.

OPSYS is not an option for the SET command.

QUERY Response:

Displays operating system name and release. e.g.

```
   z/OS 1.11.0
```

EXTRACT Rexx variables:

opsys.0	3
opsys.1	Operating System name.
opsys.2	Operating System release.
opsys.3	ISPF | TSO | VTAM | BATCH | CMS

PAD - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- PAD ---+---------- BLank ---------------+----------------------><
   |           |          |                                |
   +-- SET ----+          +---------- NUll -----------------+
                          |                                |
                          +---------- char -----------------+


>>--- Query ------ PAD ------------------------------------------------------><


>>--- EXTract --- /PAD/ -----------------------------------------------------><
```

Description:

This option defines the PAD character to be used when a new record is inserted or when the length of a variable length record is increased by updating the Record information Length field. It also defines the character used to pad text at the end of a character field when a CHANGE DATA command changes a string within the text to a shorter length string.

The initial value for PAD is BLANK.

SET PAD takes effect at the File level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`BLANK`
: The blank character (x'40').

`NULL`
: The null character (x'00').

`char`
: A single byte value represented in character, hexadecimal or binary format.
Character format may be unquoted or enclosed in quotation marks (") or apostrophes (').
Hex format is a value 0-255, enclosed in apostrophes and prefixed by 'x' or 'X' (e.g. x'FE'.)
Binary format is a value 00000000-11111111, enclosed in apostrophes and prefixed by 'b' or 'B' (e.g. b'111111110'.)

QUERY Response:

The current PAD character value displayed in character (or NULL) and hexadecimal.

EXTRACT Rexx variables:

pad.0	1
pad.1	The current PAD character setting.

PAGEDEPTH - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- PAGEDepth ------- n_lines -------------------------------><
   |           |
   +-- SET ----+


>>--- Query ------ PAGEDepth ------------------------------------------------><


>>--- EXTract --- /PAGEDepth/ -----------------------------------------------><
```

Description:

Applicable to PRINT output only, PAGEDEPTH specifies the number of lines printed per page.

For batch processing, a PAGEDEPTH value may be specified any number of times within the same SYSIN input to change the page depth of pages printed by subsequent PRINT commands.

SET Value:

n_cols
: The number of lines to be printed per page including the page header line (minimum 10, default 60).

QUERY Response:

The current PAGEDEPTH value.

EXTRACT Rexx variables:

pagedepth.0	1
pagedepth.1	The current PAGEDEPTH value setting.

PAGEWIDTH - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- PAGEWidth ------- n_cols --------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ PAGEWidth -------------------------------------------------><


>>--- EXTract --- /PAGEWidth/ ---------------------------------------------><
```

Description:

Applicable to PRINT output only, PAGEWIDTH specifies the number of character columns printed per page.

For batch processing, a PAGEWIDTH value may be specified any number of times within the same SYSIN input to change the page width of pages printed by subsequent PRINT commands.

SET Value:

n_cols
: The number of character columns to be printed per page (minimum 50, default 133).

QUERY Response:

The current PAGEWIDTH value.

EXTRACT Rexx variables:

pagewidth.0	1
pagewidth.1	The current PAGEWIDTH value setting.

PFKEY - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option PFKEY, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET PFKEY in Text Editor documentation.

POINT - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+---- Point ---- .name ----+---------+-------------------------><
   |           |                          |         |
   +- SET -----+                          +- OFf ---+

>>--- Query ----+--- Point ----+----------------------------------------------><
                |              |
                +--- Point* ---+

>>--- EXTract --+-- /Point/ ---+----------------------------------------------><
                |              |
                +-- /Point*/ --+
```

Description:

Assign or unassign a label name to the focus line (data record or shadow line) for subsequent use as a line target. The assigned label name is displayed in the line's prefix area.

If MULTIPOINT is set on, then a line may be assigned more than one label name.

The same label name may not be assigned to more than one line in the current file. Where a label name is already assigned to a line in the currunt file, it is unassigned from that line and reassigned to the focus line.

A label name remains assigned to a line even if the line record number changes due to record inserts.

SET POINT takes effect at the File level.

SET Value:

`.name`

Label name of a new label to be assigned to the focus line or an existing label to be unassigned.
The preceding "." (dot) is mandatory.

`OFF`

Unassign the specified label name from any of the lines in the file display.

QUERY Response:

`Point`

Displays the focus line number and all label names currently allocated to the focus line. If no names are allocated to the focus line (line number n), the following message is returned:

```
ZZSD010I POINT No points assigned to line n
```

`Point*`

Displays the line number and associated label names of all named lines in the current file.

EXTRACT Rexx variables:

`Point`

point.0	0 if focus line is not a named line; otherwise, 1.
point.1	Line number and label name(s) assigned to the focus line.

`Point*`

point.0	Number of named lines.
point.i	Line number and associated label name(s) of the ith named line in the current file.

PREFIX - SET/QUERY/EXTRACT Option

Syntax:

```
                                       + Left --+ +--- 6 ----+ + LOGical --+
                                       |        | |          | |           |
>>-+-----------+-- PREFix --+- ON --+--+--------+-+----------+-+-----------+-><
   |           |            |       |  |        | |          | |           |
   +- SET -----+            +- OFf -+  + Right -+ + n_bytes -+ + Physical -+

>>--- Query ------ PREFix ---------------------------------------------------><

>>--- EXtract --- /PREFix/ --------------------------------------------------><
```

Description:

PREFIX defines whether the prefix area is displayed, whether it is displayed on the left of the window view or on the right and the number of columns to use.

The prefix displays the record number, and is also a space where line-sensitive prefix commands may be entered.

The PREFIX option operates at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON | OFF`
: The prefix area is set displayed (ON) or hidden (OFF). Default is ON.

`LEFT | RIGHT`
: Determines whether the prefix area is displayed on the left or right of the record data. Default is LEFT.

n_bytes
: The width of the prefix area. Minimum is 1, maximum is 8 and the default is 6.

`LOGICAL | PHYSICAL`
: Applicable to segmented record display only, determines whether the prefix area contains the physical record numbers or the segment numbers within the file.
Default is LOGICAL.

QUERY Response:

The current setting of the PREFIX option, **ON** or **OFF**, followed by **LEFT** or **RIGHT**, followed by the length **n_bytes**, followed by **PHYSICAL** or **LOGICAL**.

EXTRACT Rexx variables:

`prefix.0`	3
`prefix.1`	The current setting of the PREFIX option, **ON** or **OFF**.
`prefix.2`	The current position of the PREFIX option, **LEFT** or **RIGHT**.
`prefix.3`	The current length of the PREFIX area, a number from 1-8.
`prefix.4`	The current display of record or segment numbers, **PHYSICAL** or **LOGICAL**.

QSEPARATOR - QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- QSEParator --- char -------------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ QSEParator ----------------------------------------------><


>>--- EXTract --- /QSEParator/ ---------------------------------------------><
```

Description:

This option controls the separator character to be used in qualified field names. By default, this is "." (dot).

The same field name may be defined in more than one nested group with in the record structure. Therefore, to reference the correct field, it should be specified using a qualified field name. e.g. *group_name.field_name*

A field's fully qualified field name includes the record type name in which it is defined in addition to the name of every group level in which it is nested. e.g. *record_type.field_name*, *record_type.group_name.field_name*. Using the record type name qualifier is required for segmented record editing where USE WHEN criteria is based on a named formatted field in a previously formatted segment.

SET QSEPARATOR takes effect at the Global level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

char

Specifies a field qualifier separator character.
Valid characters are: "." (dot), ":" (colon), "\" (backslash), "@" (commercial at), "^" (carat) or "%" (percent).

QUERY Response:

Displays "QSEPARATOR" followed by the qualifier separator character.

EXTRACT Rexx variables:

qseparator.0	1
qseparator.1	The qualifier separator character.

RANDOMIZER - SET Option

Syntax:

```
    +- TEMPorary -+  +- SET -+
    |             |  |       |
>>--+-------------+--+-------+--- RANDomizer ---+-- field_name -+------------>
    |             |                             |               |
    +- PERManent -+                             +-- field_ref --+

 >-- | Randomizer Definition | --+--------------+--+-----------------+-------->
                                 |              |  |                 |
                                 +- LENGTH len -+  +- STRIPboth -----+
                                                   +- STRIPLeading --+
                                                   +- STRIPTrailing -+

 >--+-------------------------------------------------------------------+----><
    |                                                                   |
    +- FOR +----------+- rec_type -+------------------------------------+
           |          |            |                                    |
           +- RECord -+            +- IN -+-------------+- struct_name -+
                                          |             |
                                          +- STRUCTure -+
```

Randomizer Definition:

```
  (1) +--- | Character Values | ---+
      +---- | Numeric Values | ----+
      +--- | Date/Time Values | ---+
      +----- | Time Values | ------+
      |                            |
 |--+-+----------------------------+--+-----------------------+-------+------|
    | |                            |  |                       |       |
    | +----- | List Values | ------+  |         (2)           |       |
    | |                            |  |           +- SEQRL -+ |       |
    | +--- | Key List Values | ----+  |           |         | |       |
    | |                            |  +- CHAIN ---+---------+-+       |
    | +--- | Sentence Values | ----+              |         |         |
    | |                            |              +- SEQLR -+         |
    | +- | Personal Name Values | -+                                  |
    | |                            |                                  |
    | +-- LITeral -- "string" -----+                                  |
    |                                                                 |
    +------- | Pattern Value | ---------------------------------------+
    |                                                                 |
    +-- REPlacement --+--------------------------+--------------------+
                      |                          |
                      +- ( replace_expression ) -+
```

(1) Default value types matches that of the source field data type.

Numeric Values	Default for any numeric data type field.
Character Values	Default for any character or hex data type field.
Date/Time Values	Default for any DATE or TIMESTAMP data type field.
Time Values	Default for any TIME data type field.

(2) The last CHAIN element to have SEQRL or SEQLR specified will apply.

Character Values:

```
                     (2)
    +---- CHARS ------ "default_chars_list" --+
    | (1)                                     |
    |  +- CHars --+                           |
    |  |          |                           |
 |--+--+----------+-+-------------------------+---- | Index Values | --------|
                    |                         |
                    +- "chars_list" ----------+
                    +- ALPHA -----------------+
                    +- ALPHANumeric ----------+
                    +- HEX -------------------+
                    +- HEXEVEN ---------------+
                    +- NUMeric ---------------+
                    +- LALPHA ----------------+
                    +- LALPHANumeric ---------+
                    +- LHEX ------------------+
                    +- LHEXEVEN --------------+
                    +- MALPHA ----------------+
                    +- MALPHANumeric ---------+
```

(1) CHARS keyword is mandatory for *"chars_list"*

(2) "*default_chars_list*" characters are: "A-Z", "0-9", "+-=,./*()_!:@#$" and the blank character.

Numeric Values:

```
                  (1)        (2)                               (4)
   +- RANGE -------- * -------- * ----------------------------+
   |                                                          |
|--+----------------------------------------------------------+-+---------+-|
   |                                                          | |         |
   +- RANGE ------ lo_num -+----------+-----+---------------+-+ +- ZEROS -+
   |                       |          |     |               | |
   |                       +- hi_num -+     +- BASE string -+ |
   |                     (2)                                  |
   |                                                          |
   |                        (2)                               |
   |            +- 1 ---------- * ------ +1 ----+             |
   |            |                               |             |
   +- SEQuence -+-------------------------------+-------------+
   |            |                               |             |
   |            +- lo_num -+------------------+-+             |
   |                       |                  |               |
   |                   (2) +- hi_num -+-------+               |
   |                                  |       |               |
   |                              (3) +- inc -+               |
   |          (3)                                             |
   +- ADJust -- inc --+-----------+--+------------------+-----+
                      |           |  |                  |
                      +- PERCENT -+  +- (source_field) -+
```

(1) Default low number (*lo_num*) for a RANGE is "*", the minimum value supported for the numeric field.

(2) Default high number (*hi_num*) is "*", the maximum supported for the field. (Character => "9" for length #digits).

(3) Negative integer increment (*inc*) values are supported. Default increment for SEQUENCE is: +1
The low number (*lo_num*) may be **higher** than the high number (*hi_num*), in which case the default increment is -1.

(4) Numbers may be expressed with or without a decimal point and may include an exponent. The scale of the generated value will equal the largest scale supplied on *lo_num*, *hi_num* and *inc* values.

Date/Time Values:

```
(1)                                                           (3)
  +- DATE -+              +- RANGE -- 2001/01/01 00:00:00.00 -- hi_ts ----+
  |        |              |                                               |
|-+--------+-+----------+-+-----------------------------------------------+-|
             |          | |                                               |
             + "str_ts" + +--- NOW ---------------------------------------+
           (2)            |                                               |
                          +--- TODay -------------------------------------+
                          |                                               |
                          +-+- PAST ---+-+--------------------------------+
                          | |          | |                                |
                          | +- FUTure -+ |           +- DAYs --------+    |
                          |              |           |               |    |
                          |              +- int -----+---------------+----+
                          |                          |               |    |
                          |                          +- HOURs -------+    |
                          |                          +- MINutes -----+    |
                          |                          +- SECs --------+    |
                          |                                               |
                          +- RANGE -- lo_ts -+--------+-+---------------+-+
                          |                  |        | |               | |
                          |                  + hi_ts -+ +- BASE string -+ |
                          |                (3)                            |
                          |                                   + DAYs ---+ |
                          |                                   |         | |
                          +- SEQuence lo_ts -+--------------+-+---------+-+
                          |                  |              | |         | |
                          |              (3) + hi_ts -+-----+ + HOURs --+ |
                          |                           |     | + MINutes + |
                          |                       (4) + inc + + SECs ---+ |
                          |                                               |
                          |                + DAYs ---+                    |
                          |                |         |                    |
                          +- ADJust - inc -+---------+-+----------------+-+
                                     (4)   |         | |                |
                                           + HOURs --+ + (source_field) +
                                           + MINutes +
                                           + SECs ---+
```

(1) DATE keyword required if source field is **not** a FileKit DATE or TIMESTAMP data type. (e.g. a character field)

(2) Omit "*str_ts*" for a DATE or TIMESTAMP source field.
"*str_ts*" is a quoted date/time string containing any character, but the following are substituted with date/time elements:
"CC", "CI", "YYYY", "YY", "MM", "MMM", "Mmm", "DD", "DDD", "Ddd", "JJJ", "WW", "WWS", "hh", "mm", "ss", "ttt"
Default combination is: **"CCYY/MM/DD hh:mm:ss.ttt"**, or **"CCYYMMDDhhmmssttt"** for numeric source fields.

(3) Default high date/time (*hi_ts*) is: 2042/09/17 23:53:47.37

(4) Negative integer increment (*inc*) values are supported. Default increment for SEQUENCE is: +1 DAY
The low date/time (*lo_ts*) may be **higher** than the high date/time (*hi_ts*), in which case the default increment is -1 DAY.

Time Values:

```
(1)                                                            (3)
  +- TIME -+              +- RANGE ------------- 00:00:00.00 -- hi_ti -----+
  |        |              |                                                |
|-+--------+-+----------+-+------------------------------------------------+-|
             |          | |                                                |
             + "str_ti" + +--- NOW ----------------------------------------+
           (2)            |                                                |
                          +-+- PAST ---+-+---------------------------------+
                          | |          | |                                 |
                          | +- FUTure -+ |           +- SECs ---------+    |
                          |              |           |                |    |
                          |              +- int -----+----------------+----+
                          |                          |                |    |
                          |                          +- HOURs --------+    |
                          |                          +- MINutes ------+    |
                          |                                                |
                          +- RANGE -- lo_ti -+--------+-+----------------+-+
                          |                  |        | |                | |
                          |                  + hi_ti -+ +- BASE string --+ |
                          |                (3)                             |
                          |                                    + SECs ---+ |
                          |                                    |         | |
                          +- SEQuence lo_ti -+---------------+-+---------+-+
                          |                  |               | |         | |
                          |                  + hi_ti -+------+ + HOURs --+ |
                          |                (3)       |      | + MINutes + |
                          |                           + inc +              |
                          |                          (4)                   |
                          |                                                |
                          |                + SECs ---+                     |
                          |                |         |                     |
                          +- ADJust - inc -+---------+-+-----------------+-+
                                      (4)  |         | |                 |
                                           + HOURs --+ + (source_field)  +
                                           + MINutes +
```

(1) TIME keyword required if source field is **not** a FileKit TIME data type. (e.g. a character field)

(2) Omit "*str_ti*" for a TIME source field.
"*str_ti*" is a quoted time string containing any character, but the following are substituted with time elements:
"hh", "mm", "ss", "ttt"
Default combination is: **"hh:mm:ss.ttt"**, or **"hhmmssttt"** for numeric source fields.

(3) Default high time (*hi_ti*) is: 23:59:59.99

(4) Negative integer increment (*inc*) values are supported. Default increment for SEQUENCE is: +1 SEC
The low time (*lo_ti*) may be **higher** than the high time (*hi_ti*), in which case the default increment is -1 SEC.

List Values:

```
                  +--------+        +- VALLOCATION (1) -------------------+
                  v        |        |                                     |
|-- LIST --+-- ( -+- item -+- ) --+-+-------------------------------------+-->
           |                      | |                                     |
           +------ list_file -----+ +- VALLOCATION ( pos -+---------+- ) -+
                                                          |         |
                                                          +- ,len  -+

>---- | Index Values | ------------------------------------------------------|
```

Key List Values:

```
                   +--------+
                   v        |
|- KEYLIST -+-- ( -+- item -+- ) --+--- VALLOCATION ( pos -+--------+- ) --->
            |                      |                       |        |
            +------ list_file -----+                       +- ,len -+

>-- KEYLOCATION ( pos -+--------+- ) --+-------------------------+-----------|
                       |        |      |                         |
                       +- ,len -+      +- KEY( key_expression ) -+
```

Sentence Values:

```
  +- LIST -+       +--------+       +- VALLOCATION (1) -------------------+
  |        |       v        |       |                                     |
|-+--------+-+- ( -+- item -+- ) -+-+-------------------------------------+-->
             |                    | |                                     |
             +---- list_file -----+ +- VALLOCATION ( pos -+---------+- ) -+
                                                          |         |
                                                          +- ,len  -+

>---- | Index Values | ------ VOCab ----+--------+-+----------+--------------|
                                        |        | |          |
                                        +- CAP1 -+ +- PERIOD -+
```

Personal Name Values:

```
                    +----- ANY ---------+
                    |                   |
 |-- PERSon --- ( --+-------------------+-- ) ---- | Index Values | ---------|
                    |                   |
                    +----- BOY ---------+
                    +----- FULL --------+
                    +----- FULL1 -------+
                    +----- FULL2 -------+
                    +----- FULL3 -------+
                    +----- GIRL --------+
                    +----- LAST --------+
                    +----- TITLE -------+
                    +----- TITLE2 ------+
```

Pattern Value:

```
 |-- PATTERN -- "pattern_string" ---+------------------------+-+---------+---|
                                    |                        | |         |
                                    |            +- SEQRL -+ | +- ECHO --+
                                    |            |         | |
                                    +- SEQuence -+---------+-+
                                    |            |         | |
                                    |            +- SEQLR -+ |
                                    |                        |
                                    +- BASE string ----------+
```

Index Values:

```
                              (2)                               (4)
    +- RANGE -------- 1 -------- * ----------------------------+
(1) |                                                          |
 |--+----------------------------------------------------------+-------------|
    |                                                          |
    +- RANGE ------ lo_num -+----------+-----+---------------+-+
    |                       |          |     |               | |
    |                       +- hi_num -+     +- BASE string -+ |
    |                     (2)                                  |
    |                                                          |
    |                         (2)                              |
    |            +- 1 ---------- * ------ +1 ----+             |
    |            |                               |             |
    +- SEQuence -+-------------------------------+-------------+
                 |                               |
                 +- lo_num -+------------------+-+
                            |                  |
                        (2) +- hi_num -+-------+
                                       |       |
                                   (3) +- inc -+
```

(1) An **Index Value** is a numeric index into a string of characters or list items.

(2) Default high number (*hi_num*) is "*", the number of random characters, or the number of items in the random value list.

(3) Negative integer increment (*inc*) values are supported. Default increment is: +1
The low number (*lo_num*) may be **higher** than the high number (*hi_num*), in which case the default increment is -1.

(4) Numbers *lo_num*, *hi_num* and *inc* are integer values.

Description:

Use the SET RANDOMIZER command to define test data generation options for any individual field in your record layout structure. e.g.

```
set randomizer TRACK-NUM  range 1 30
randomizer     TRACK-ID   chars "0123456789ABCDEF"
rand           ARTIST     MAlphaNum     /* Mixed case alpha or numeric */
```

If no mapping structure is applied then the whole record may be treated as a single field named **"Record"** or **"UnMapped"**.

As well as "random" numbers and character data, the RANDOMIZER may be used to:

- Generate a **sequence** of numbers based on a supplied increment/decrement value. e.g.

```
rand SALARY seq 500   600.30   +0.05

      /* Will produce "500.00",    on 1st record
      |               "500.05",    on 2nd record
      |               "500.10" etc
      |         up to "600.30"     ... after which the sequence will repeat.
      */
```

- Generate **date/time** fields either randomly or in sequence. e.g.

```
rand START-DATE date "CCYY-MM-DD (DDD)"       range 1980/01/01 2030/12/31
             /* e.g. "2002-10-27 (SUN)" */

rand LASTUPDATE date "Mmm DD CCYY hh:mm:ss"
                                 seq  "2003/04/10 08:30" 2003/04/20 +5 SECS

             /* i.e. "Apr 10 2003 08:30:00"
             |       "Apr 10 2003 08:30:05"
             |       "Apr 10 2003 08:30:10"
             |       "Apr 10 2003 08:30:15"
             |         etc.
             */
```

- Adjust existing numeric or date/time values using a supplied increment/decrement value. e.g.

```
rand NAME-POS    adjust +8

rand ORDER-DATE  date "CCYY-MM-DD (DDD)"     adjust -30 days

rand SALARY      adjust +3.75 percent
```

- Perform a **calculation** based on current value(s) in this and/or other field(s). e.g.

```
rand BONUS       replacement(BONUS*2)      /* Double it! */

rand GAS-RATE    rep(  (GAS-COST-0.2848) / GAS-KWHs  )
```

- Pick from a **list** of values, either randomly or in sequence. Any list may be supplied as a **separate file** or from **in-line** values. e.g.

```
rand CALLER          list=MY.RAND.LIST(MODULES)   /* Take list from a file */

rand FIRST-NAME  seq list( "George", "Paul", "Ringo", "John" )
```

- **Substitute** values by performing a **keyed lookup** into a supplied list. e.g.

```
rand FIRST-NAME  KeyLocation(1,10) ValLocation(11,10)
             /*  keyList MY.RAND.LIST(FIRSTNAME)  */
                 keyList(
                          "Annabel   Alison    "
                          "Edward    David     "
                          "Heidi     Etta      "
                          "Jack      James     "
                          "Laurence  John      "
                          "Paul      Nicholas  "
                          "Peter     Paul      "
                          "Pasqual   Peter     "
                          "Simon     Ricky     "
                         )
```

- Fill character fields using a supplied list of **"vocabulary"**. The selected words will be blank separated. Optionally the first character of the first word may be uppercased, and a period (full-stop) added to the end. e.g.

```
rand DESC-TXT  vocab    cap1  period
             /*     List MY.RAND.LIST(WORDLIST)  */
                    List(
                          "a"     "an"    "the"   "and"
                          "have"  "that"  "for"   "you"
                          "with"  "say"   "this"  "they"
                          "but"   "his"   "from"  "not"
                          "ask"   "need"  "too"   "feel"
                          "three" "state" "never" "become"
                          "night" "high"  "real"  "each"
                          "most"  "other" "much"  "family"

                          "a"     "an"    "the"   "and"
                          "a"     "an"    "the"   "and"

                          "A complete sentance?"
                         )
```

- Generate data according to a supplied **"pattern"** string. e.g.

```
rand PART-ID   pattern "A(J-N)#[-]#(1001-1999)[-]A(JGE,DJG,NBJ)"

  /* Example output "K5-1758-NBJ"
  |                 "J8-1044-JGE"
  |                 "M1-1346-DJG"
  */
```

- To go beyond any limitations of the built-in "pattern" string syntax, multiple components may be manually **"chained"** together. e.g.

```
rand PART-ID            chars "JND"     len=1
rand PART-ID  chain     range 1,3       len=1
rand PART-ID  chain     lit    "-"
rand PART-ID  chain     range 901,999   len=3 zeros sequence
rand PART-ID  chain     lit    "-"
rand PART-ID  chain     range 1001,1999 len=4
rand PART-ID  chain     lit    "-"
rand PART-ID  chain     list ( "JGE", "DJG", "NBJ", "NGH", "CLS" )

  /* Example output "D3-901-1758-CLS"
  |                 "J1-902-1044-NBJ"
  |                 "N2-903-1346-DJG"
  */
```

Note that SET RANDOMIZER does not itself cause any test data to be generated, it merely creates a **"randomizer object"** that is associated with the specified field.

The following methods may be used to actually generate record data, each of them including options to generate field values according to their respective randomizer objects:

- Online, while editing a dataset using the Data-Editor, by entering the INSERT and REPLACELINE commands.
- In batch or online, the FSU Utility may be used to either **copy** or **update** an existing file, generating "randomized" test data for certain fields, while preserving existing values in others.
- In batch or online (typically under control of a REXX procedure), the FILEIO command may be used to read/write/update values from one or more files. **FILEIO** may be thought of as an extended version of TSO's standard **EXECIO** command, and should be used when the FSU Utility does not provide sufficient control, and/or to generate large amounts of test data from scratch.

Set Value:

ADJUST [+|-]*inc*

ADJUST (or ADJ) causes the existing (numeric or date/time) value to be incremented or decremented by the supplied number *inc*. e.g.

```
rand DESC-LEN    adjust -15

rand HOURLY-RATE adj +1.38

rand EXPIRY-DATE adj 365
```

ADJUST *inc* PERCENT

The increment/decrement as described above is expressed as a percentage. e.g.

```
rand HOURLY-RATE adj +4.75 PERCENT

rand HOURLY-RATE adj +4.75 %    /* Valid with blank before "%" */

rand HOURLY-RATE adj +4.75%     /* Not valid */
```

ADJUST *inc* DAYS|HOURS|MINUTES|SECS

Specify **DAYS** or **SECS** to indicate the unit of increment/decrement for **date** and **time** fields. For **date** and **date+time** fields the default increment/decrement unit is **days**, and for **time** fields the default unit is **seconds**, so to adjust a **date+time** field by a number of seconds you must use the **SECS** keyword.

```
rand ORDER-DATE  date "CCYY-MM-DD (DDD)"     adjust -30 days

rand LAST-CHG    date "MM/DD/CCYY hh:mm:ss"  adjust +60 secs

rand DURATION    time                        adjust +3600    /* Add one hour */
```

ADJUST *inc* ... (*source_field*)

The increment/decrement may be applied to a value that exists in a different field, for example **EXPIRY-DATE** should be an adjustment based on the existing value in **START-DATE**. e.g.

```
rand EXPIRY-DATE date "CCYY-MM-DD"     adj 365 days ( START-DATE )
```

BASE *string*

Specify a fixed "base" in order get a **repeatable** set of results from the randomizer.

string is a character string of up to 8 bytes that is used to seed the random number generation algorithm.

If BASE is not explicitly coded then a default is generated using the current TOD clock value combined with the field's unique reference number.

For **ADJUST, KEY, REPLACEMENT** and **SEQUENCE** options, BASE specification is not relevant as the process does not involve generation of a random number at any stage.

```
rand GAS-KWHS    range 00,100  base "ABCDEFGH"

rand ELC-KWHS    range 22,150  base X'1234'
```

CAP1

The **CAP1** option will cause the first letter of generated character strings to be upper-cased.

```
rand FirstName   cap1 alpha

rand Descrip     cap1 vocab list(every,good,boy,deserves,favour) period
```

CHAIN

The **CHAIN** option may be used to produce a concatenated result from multiple randomizers. Typically this would be used to generate a combination of numbers, alpha and special characters in a data "pattern".

For most straight forward data patterns the **PATTERN** *"pattern_string"* feature is recommended instead of coding multiple chained randomizers for the same field.

```
rand NAME       list=("Mr ", "Mrs ", "Miss ")
rand NAME chain list=MY.FIRST.NAMES.LIST
rand NAME chain lit ' '
rand NAME chain alpha  len=1              /* Get middle initial */
rand NAME chain list=(" "," "," ",". ")    /* Get occasional "." */
rand NAME chain list=MY.LAST.NAMES.LIST
```

CHARS

The CHARS (CH) option indicates that a character string is to be generated, and is the default for all fields that are not of a numeric data-type.

An optional quoted string may follow to supply the array of characters from which a value may be selected (either at random or in sequence).

Alternatively, one of a number of shortcut keywords may be supplied e.g. ALPHA, to indicate the eligible list of characters.

If no keyword or quoted string is supplied, the following characters are used:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 +-=,./*()_!:@#$
```

CHARS *"chars_list"*

A quoted string supplying the list of characters eligible for selection. e.g.

```
rand ARITH-OP    chars "+-/*"

rand PREFIX      chars 'EHPRXehprx' sequence
```

CHARS *chars_keyword*

A keyword that implies a list of eligible characters.

Note that the **CHARS** keyword itself may be omitted if any of the following are supplied.

Keyword	*Description*	*Implied Character List*
ALPHA	Upper case Alpha only	ABCDEFGHIJKLMNOPQRSTUVWXYZ
ALPHANumeric	Upper case Alpha + Num	ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789
HEX	Hexadecimal digits	0123456789ABCDEF 0123456789
HEXEVEN	Even value Hexadecimal digits	02468ACE 0123456789
NUMeric	Numeric only	0123456789
LALPHA	Lower case Alpha only	abcdefghijklmnopqrstuvwxyz
LALPHANumeric	Lower case Alpha + Num	0123456789 abcdefghijklmnopqrstuvwxyz

LHEX	Lower case Hexadecimal digits	0123456789abcdef 0123456789
LHEXEVEN	Lower case even value Hexadecimal digits	02468ace 0123456789
MALPHA	Mixed case Alpha only	ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
MALPHANumeric	Mixed case Alpha + Num	ABCDEFGHIJKLMNOPQRSTUVWXYZ 0123456789 abcdefghijklmnopqrstuvwxyz

e.g.

```
rand LASTNAME    chars lalpha cap1
rand PASSWORD    malphanum         /* CHARS is implied */
```

DATE

Use the **DATE** keyword to identify the named field as a date or date+time field.

If no date format string is supplied, and the field is not defined using one of FileKit's built-in date/time formats, then for character fields the default is **"CCYY/MM/DD hh:mm:ss.ttt"**. For fields defined with a numeric data-type (e.g. binary, packed decimal etc) the default format is **"CCYYMMDDhhmmssttt"**.

```
rand START-DATE  date   range 2001/01/01 2010/12/12
```

DATE *"str_ts"*

A character string that defines the format of the date/time (timestamp) field for which test data should be generated or adjusted.

The following (case-sensitive) format codes are supported, with all other characters treated as literals and transferred directly to the output.

Code	*Description*	*Examples*
CC	2-digit Century	"19" or "20"
CI	2-digit Century Indicator	"00" for 19xx "01" for 20xx
YYYY	4-digit Year (Same as coding "CCYY")	"1923"
YY	2-digit Year	"23"
MM	2-digit Month	"08"
MMM	3-char Month name in upper case	"JUL"
Mmm	3-char Month name in mixed case	"Dec"
DD	2-digit Day of the month	"31"
DDD	3-char Day name in upper case	"WED"
Ddd	3-char Day name in mixed case	"Sat"
JJJ	3-digit Julian Day of the year	"365"
WW	2-digit Week number (Monday start)	"52"
WWS	2-digit Week number (Sunday start)	"52"
hh	2-digit Hour of the day	"00" to "24"
mm	2-digit Minute of the hour	"00" to "59"
ss	2-digit Second of the minute	"00" to "59"
ttt	3-digit Thousandth of the second	"000" to "999"

e.g.

```
rand START-DATE  date "(Ddd)  DD-Mmm CCYY" /* e.g. "(Thu)  02-May 2002" */

rand PD_TSTMP    date "MMDDCCYYhhmmss"     /* e.g. For packed dec -> */
                                           /*       X'025122019154521C' */
                                           /*   meaning "15:45:21"      */
                                           /*        on "25th Dec 2019" */
```

DATE ... TODAY

Ensures that any date/time value produced will be for the current date.

Equivalent to coding **"RANGE '*yyyy/mm/dd* 00:00:00' '*yyyy/mm/dd* 23:59:59'"**, where *yyyy/mm/dd* is the current date.

DATE ... PAST

Ensures that any date/time value produced will be earlier than the current date/time.

Equivalent to coding **"RANGE '2001/01/01 00:00:00' '*yyyy/mm/dd hh:mm:ss*'"**, where *yyyy/mm/dd hh:mm:ss* is the current date/time.

DATE ... PAST *int* DAYS|HOURS|MINUTES|SECS

Ensures that any date/time value produced will be earlier than the current date/time, but no earlier than *int* number of **DAYS**/**HOURS**/**MINUTES**/**SECS**.

DATE ... FUTURE

Ensures that any date/time value produced will be later than the current date/time.

Equivalent to coding **"RANGE '*yyyy/mm/dd hh:mm:ss*' '2042/09/17 23:53:47' "**, where *yyyy/mm/dd hh:mm:ss* is the current date/time.

DATE ... FUTURE *int* DAYS|HOURS|MINUTES|SECS

Ensures that any date/time value produced will be later than the current date/time, but no later than *int* number of **DAYS**/**HOURS**/**MINUTES**/**SECS**.

KEY (*key_expression*)

When using **KEYLIST** to perform a translation via a keyed lookup, the default key is the value of the field for which you are defining the randomizer.

Use the **KEY** (*key_expression*) option if the key value should be derived from one or more different fields.

The *key_expression* must be a valid Data-Edit expression

```
rand FUNC keylist=MY.RAND.LIST(FUNCNAM1)
          KeyLoc(01,08) ValLoc(11,20)
          key=(MODULE)


rand FUNC keylist=MY.RAND.LIST(FUNCNAM2)
          KeyLoc(01,12) ValLoc(15,20)
          key=(  cat( MODULE,  '|',   right(strip(ext(PARM1),'L'),3,'0')  ))

                                /* e.g. "SDEFSQX9|028  SQXColWidth" */
                                /*      "SDEFSQX9|02C  SQXAutoSave" */
```

KEYLIST

Use **KEYLIST** to supply a list of keys and their corresponding substitution values in order to perform a translation via a keyed lookup.

The list may be supplied from **in-line** values or as a **separate file**.

Each line of the list should contain a value that will be referenced by keyed lookup and a corresponding substitution value, the position and length of which should be defined using the
KeyLoc(*pos,len*) and
ValLoc(*pos,len*) options.

The default key is the value of the field that you are defining the randomizer for, but the **KEY** *(key_expression)* option may be used if the key value should be derived from one or more different fields.

KEYLIST (*item* [, ...])

The list may be supplied in-line as a series of blank or comma separated strings enclosed in brackets.

Each string must be quoted if it contains blanks or special characters.

```
rand FIRST-NAME  KeyLocation(1,10) ValLocation(11,10)
                 keyList(

                       /*  From      To       */
                       /*  ----      --       */

                          "Annabel   Alison    "
                          "Edward    David     "
                          "Heidi     Etta      "
                          "Jack      James     "
                          "Laurence  John      "
                          "Paul      Nicholas  "
                          "Peter     Paul      "
                          "Pasqual   Peter     "
                          "Simon     Ricky     "
                        )
```

KEYLIST ***list_file***

The list may be supplied as a separate file.

```
rand FIRST-NAME  KeyLocation(1,10) ValLocation(11,10)
                 keyList MY.RAND.LIST(FIRSTNAME)
```

KEYLOCATION(*pos* [,*len*])

Use **KEYLOC** to define the **position** and **length** within each list line of the key to be "looked up", in order to perform a value substitution.

The position and length of the substitution value should also be supplied using the **VALLOC** option.

```
rand FIRST-NAME  KeyLoc(1,10) ValLoc(11,10)  StripTrailing
                 keyList(

                       /*  Before    After   */
                       /*  ------    -----   */

                          "Annabel   Alison    "
                          "Edward    David     "
                          "Heidi     Etta      "
                          "Jack      James     "
                          "Laurence  John      "
                          "Paul      Nicholas  "
                          "Peter     Paul      "
                          "Pasqual   Peter     "
                          "Simon     Ricky     "
                        )
```

LALPHA

The LAPLPHA option indicates that a character string is to be generated, containing **lower-case alphabetic** ("a" to "z") characters only.

Add the **CAP1** option to generate a field that is all lower case character, but with the first character upper cased. e.g.

```
rand LASTNAME    lalpha cap1
```

LALPHANUMERIC

The LAPLPHANUMERIC (LALPHAN) option indicates that a character string is to be generated, containing **lower-case alphabetic** ("a" to "z") and **numeric** ("0" to "9") characters only.

Note that if the **SEQUENCE** option is added, then numerics ("0" to "9") will be produced ahead of the lower-case alpha ("a" to "z").

```
rand LASTNAME    lalphan
```

LENGTH ***len***

When generating data for **character** fields, unless a limit is imposed by some other option, the default is to generate data to fill the **whole field**.

Specify **LENGTH *len*** to limit the amount of data generated e.g. to generate 16 bytes of data in 20-byte field.

LENGTH *len* is also commonly used with the **CHAIN** option when generating a pattern of data consisting of multiple components. e.g.

```
rand NAME       list=("Mr ", "Mrs ", "Miss ")
rand NAME chain list=MY.FIRST.NAMES.LIST
rand NAME chain lit ' '
rand NAME chain alpha  len=1               /* Get middle initial */
rand NAME chain list=(" "," "," ",". ")    /* Get occasional "." */
rand NAME chain list=MY.LAST.NAMES.LIST
```

LIST

Use **LIST** to supply a list of possible values to be generated.

The list may be supplied from **in-line** values or as a **separate file**.

Each line of the list should contain a value. If the whole line is not to be used you may supply the value's **position** and **length** using the
ValLoc(*pos,len*) option.

LIST (*item* [, ...])

The list may be supplied in-line as a series of blank or comma separated strings enclosed in brackets.

Each string must be quoted if it contains blanks or special characters.

```
rand FIRST-NAME  list( "James G. Evans"
                       "Nicholas B. Jones"
                       "Daniel Gribble"
                       "Laurence A. Cross"
                       "Douglas J. Hegarty"
                     )
```

LIST *list_file*

The list may be supplied as a separate file.

```
rand FIRST-NAME  list MY.RAND.LIST(FIRSTNAME) ValLoc(11,10)
```

LITERAL *"string"*

The generated value will be a fixed character or numeric literal.

For numeric fields the value will automatically be converted to the correct data-type (e.g. fixed point binary or packed decimal).

LIT *"string"* is also commonly used with the **CHAIN** option when generating a pattern of data consisting of multiple components. e.g.

```
rand PART-ID            chars "JND"     len=1
rand PART-ID  chain     range 1,3       len=1
rand PART-ID  chain     lit   "-"
rand PART-ID  chain     range 901,999   len=3 zeros sequence
rand PART-ID  chain     lit   "-"
rand PART-ID  chain     range 1001,1999 len=4
rand PART-ID  chain     lit   "-"
rand PART-ID  chain     list  ( "JGE", "DJG", "NBJ", "NGH", "CLS" )

  /* Example output "D3-901-1758-CLS"
  |                 "J1-902-1044-NBJ"
  |                 "N2-903-1346-DJG"
  */
```

MALPHA

The MAPLPHA (mixed-case alpha) option indicates that a character string is to be generated, containing **both upper- and lower-case alphabetic** ("A to "Z" and "a" to "z") characters only.

```
rand VERY-WEAK-PASSWORD    malpha
```

MALPHANUMERIC

The MAPLPHANUMERIC (MALPHAN) option indicates that a character string is to be generated, containing **upper and lower-case alphabetic** ("a" to "z"), and **numeric** ("0" to "9") characters only.

Note that if the **SEQUENCE** option is added, then upper-case alpha ("A" to "Z") are produced first, followed by numerics ("0" to "9"), then lower-case alpha ("a" to "z").

```
rand WEAK-PASSWORD    malphan
```

NUMERIC

The NUMERIC (NUM) option indicates that a character string is to be generated, containing **numeric** ("0" to "9") characters only.

For fields defined with a numeric data-type (e.g. binary, packed decimal etc), coding the "NUM" keyword is unnecessary (just code "RANGE n1 n2").

```
rand PIN    num  len=4      /* PIN is a char field */
```

PATTERN *"pattern_string"*

Data may be generated according to a fixed pattern consisting of upper-/lower-case characters, numbers and literals.

"pattern_string"

Defines the layout of the data to be generated.

The following (case-sensitive) format codes are supported.

Code	*Description*	*Examples*
A	Any Upper-case Alpha (A-Z)	
A(*a1-a2*)	Upper-case Alpha in range *a1* to *a2*	A(P-V) = "PQRSTUV"

A(*a1,a2,a3...*)	List of (case-sensitive char) literals	A("J","N", "D")
		A("Jim","Nick", "Dan")

a	Any lower-case alpha (a-z)	
a(*a1-a2*)	Lower-case alpha in range *a1* to *a2*	a(p-v) = "pqrstuv"

a(*a1,a2,a3...*)	List of (case-sensitive char) literals	a("j","n", "d")
		a("Jim","Nick", "Dan")

# or **N**	Any numeric digit (0-9)	
#(*nnn1-nnn2*)	Any number in range *nnn1* to *nnn2*	#(101-200)

#(*n1,n2,n3...*)	List of (numeric) literals	a("1","3", "5")
		a("32,768","32.768","32768.00",)

[*literal*]	Any literal	[>>]
X	Upper-case HEX digits (0-F)	X(8-F) = "89ABCDEF"
x	Lower-case HEX digits (0-f)	x = "0123456789abcdef"
H	Upper-case even HEX digits (0-E)	H(4-C) = "468AC"
h	Lower-case even HEX digits (0-e)	h = "02468ace"

e.g.

```
rand PART-ID   pattern "A(J-N)#[-]a#(1001-1999)[-]A(JGE,DJG,NBJ)"

  /* Example output "K5-g1758-NBJ"
  |                 "J8-e1044-JGE"
  |                 "M1-j1346-DJG"
  */

rand FNAM      pat "[<=-= ]Aaaa[ ]A[. ]Aaaaaaa[ =-=>]"

  /* Example output "<=-= Kuhi R. Wohudiu =-=>"
  |                 "<=-= Ijyt W. Pytsltm =-=>"
  |                 "<=-= Vkth S. Hyewjjs =-=>"
  */

rand NAME  seq pat "A('Thomas', 'Tom', 'T.S.')[ Evans]"

  /* Output         "Thomas Evans"
  |                 "Tom Evans"
  |                 "T.S. Evans"
  */

rand PART  seq pat "A(A,T,X)[-]#(1050-1001)"  seqLR

  /* Output         "A-1050"
  |                 "T-1050"
  |                 "X-1050"
  |                 "A-1049"
  |                 "T-1049"
  |                 "X-1049"
  |                 "A-1048"
  |                 "T-1048"
  |                 etc
  */
```

***"pattern_string"* ECHO**

Whenever a pattern string is used, FileKit turns each component into a separate RANDOMIZER, the results of which are concatenated using the **CHAIN** option described earlier.

The **ECHO** option causes FileKit to display the "chain" of generated "SET RANDOMIZER" commands on the message queue, instead of executing them.

The feature may be used as a helpful shortcut to coding the chain yourself, if and when limits of the pattern string are encountered.

e.g. Your pattern contains a sequence number that needs to be decremented by -5 instead of -1.

```
rand PART       chars "ATX"   seq  seqLR
rand PART chain lit "-"
rand PART chain len=4 zeros   seq 1050 1001 -5

  /* Output         "A-1050"
  |                 "T-1050"
  |                 "X-1050"
  |                 "A-1045"
  |                 "T-1045"
  |                 "X-1045"
  |                 "A-1040"
  |                 "T-1040"
  |                 etc
  */
```

PERCENT | %

Indicates that the increment/decrement value on an ADJUST (or ADJ) operation is expressed as a **percentage** e.g.

```
rand HOURLY-RATE adj +4.75 PERCENT

rand HOURLY-RATE adj +4.75 %    /* Valid with blank before "%" */

rand HOURLY-RATE adj +4.75%     /* Not valid */
```

PERIOD

Applicable to the **VOCAB** operation only, use **PERIOD** option to ensure a full-stop (".") is added at the end of the generated string.

If the generated string already ends in ".", "?", or "!' then a period will not be added.

```
rand DESC-TXT  vocab    cap1  period
             /*     List MY.RAND.LIST(WORDLIST)  */
                    List(
                          "a"     "an"    "the"   "and"
                          "have"  "that"  "for"   "you"
                          "with"  "say"   "this"  "they"
                          "but"   "his"   "from"  "not"
                          "ask"   "need"  "too"   "feel"
                          "three" "state" "never" "become"
                          "night" "high"  "real"  "each"
                          "most"  "other" "much"  "family"

                          "a"     "an"    "the"   "and"
                          "a"     "an"    "the"   "and"

                          "A complete sentance?"
```

PERSON

PERSON (PERS) provides options for generating the **name of a person**. e.g.

```
rand FIRST-NAME      person          /* e.g. "Jacob", "Emma"    etc */
rand FIRST-NAME      person(BOY)     /* e.g. "Jacob", "Michael" etc */
rand FIRST-NAME      person(LAST)    /* e.g. "Smith", "Johnson" etc */
rand CONTACT-NAME    person(FULL)    /* e.g. "Emma Smith"       etc *
rand CONTACT-NAME    person(FULL2)   /* e.g. "Mrs Erin Fields"  etc *
```

PERSON (*person_keyword*)

Keyword	*Description*	*Example*
ANY	First-name (Male/Female)	"Chloe"
BOY	First-name (Male)	"Mark"
FULL **FULL1**	First-name (Male/Female) **+** Last-name	"Mark Smith"
FULL2	Title (Male/Female) plus First-name (Male/Female) **+** Last-name	"Mrs Mark Smith" (can't guarantee compatibility!)
FULL3	Title (M/F - ext choice) **+** First-name (Male/Female) **+** Last-name	"Major General Mark Smith"
GIRL	First-name (Female)	"Chloe"
LAST	Last-name	"Smith"
TITLE **TITLE1**	Title (Male/Female)	"Miss"
TITLE2	Title (M/F - ext choice)	"Rear Admiral"

PERSON (***person_keyword*** **)** basically provides a shortcut to the product supplied list files **"*ProdHlq*.SZZSSAM2(ZZSPERxx)"**. e.g.

```
rand PERSON  name(FULL3)


  /* Above is equivalent to coding ... */

rand PERSON       list "ProdHlq.SZZSSAM2(ZZSPERT2)"  /* PERSON(TITLE2) */
rand PERSON chain literal " "
rand PERSON chain list "ProdHlq.SZZSSAM2(ZZSPERAN)"  /* PERSON(ANY)    */
rand PERSON chain literal " "
rand PERSON chain list "ProdHlq.SZZSSAM2(ZZSPERLA)"  /* PERSON(LAST)   */
```

RANGE *low_val high_val*

Defines the range of **numeric**, **date/time** or **time** values to be generated.

If *high_val* is omitted, then the default is the maximum value able to fit in the field (e.g. 32767 for a 2-byte signed binary field).

For date/time fields (although later dates may be generated) *high_val* defaults to **2042/09/17 23:53:47.37**, which is the highest timestamp supported by the standard TOD clock (STCK).

If **RANGE** is omitted altogether, then the default for *low_val* follows the same principle, except that date/time fields default to **2001/01/01 00:00:00.00**.

```
rand HOURLY-RATE range 8.51 1250

rand START-DATE  range 1980/01/01 2030/12/31  date "CCYY-MM-DD (DDD)"

rand END-TIME    range 15:00 17:30            time "hh-mm-ss"
```

REPLACEMENT (*replace_expression*)

Use the **REPLACEMENT** (or REP) option to generate a value that is calculated based on the existing value in the same and/or separate field(s).

The *replace_expression* must be a valid Data-Edit expression

```
rand BONUS       replacement(BONUS*2)      /* Double existing value!  */

rand PREV-UPD    replacement(LAST-UPD)     /* Just copy another field */

rand GAS-RATE    rep(  (GAS-COST-0.2848) / GAS-KWHs  ) /* Fancy calc */
```

SEQUENCE

Use the **SEQUENCE** (or SEQ) option to indicate that, instead of at random, values are to be generated in sequence, the next being a fixed increment/decrement on the previous.

The increment defaults to "+1". e.g.

```
rand REFNO seq                /* A 4-byte signed binary field */

      /* Will produce "1"     on 1st record
      |                "2"     on 2nd record
      |                "3"     on 3rd record
      |                "4"     on 4rd record etc
      |
      */

rand MIDIN seq chars "PRS"

      /* Will produce "P"     on 1st record
      |                "R"     on 2nd record
      |                "S"     on 3rd record
      |                "P"     on 4rd record etc
      |
      */

rand CNAME seq list( "Jim", "Dan", "Nick" )

      /* Will produce "Jim"   on 1st record
      |                "Dan"   on 2nd record
      |                "Nick"  on 3rd record
      |                "Jim"   on 4rd record etc
      |
      */
```

SEQ *lo_num* [*hi_num* [*inc*]]

Defines the range of **numeric** values to be generated.
The default for *lo_num* is *1.*

The default for *hi_num* follows the same principle as decribed for the RANGE option.

If *lo_num* is higher than *hi_num* then the default increment value (*inc*) is **-1**, otherwise it is **+1**. e.g.

```
rand SALARY seq 500   600.30   +0.05

      /* Will produce "500.00"     on 1st record
      |               "500.05"     on 2nd record
      |               "500.10" etc
      |         up to "600.30"     ... after which the sequence will repeat.
      */
```

SEQ *lo_ts* [*hi_ts* [*inc*]] [DAYS|HOURS|MINUTES|SECS]

Defines the range of **date** or **date/time** (timestamp) values to be generated.
The default for *lo_ts* is *1.*
The default for *hi_ts* follows the same principle as decribed for the RANGE option.

If *lo_ts* is higher than *hi_ts* then the default increment value (*inc*) is **-1**, otherwise it is **+1**. e.g.

Specify **DAYS, HOURS, MINUTES** or **SECS** to indicate the unit of increment/decrement for **date** and **date/time** (timestamp) fields. Default is DAYS. Therefore, to increment a **date+time** field by a number of seconds you must use the **SECS** keyword. e.g.

```
rand ORDER-DATE  date seq 2029/12/31 2010/01/01 -30   days

rand LAST-CHG    date seq 2023/06/13 2023/07/13 +3600 secs  /* + 1 hour */
```

SEQ *lo_ti* [*hi_ti* [*inc*]] [HOURS|MINUTES|SECS]

Defines the range of **time** values to be generated.
The default for *lo_ti* is *1.*
The default for *hi_ti* follows the same principle as decribed for the RANGE option.

If *lo_ti* is higher than *hi_ti* then the default increment value (*inc*) is **-1**, otherwise it is **+1**. e.g.

Specify **HOURS, MINUTES** or **SECS** to indicate the unit of increment/decrement for **time** fields. Default is SECS.

```
rand DURATION    time seq 10:00      16:00      +30                /* +30 secs */
```

SEQLR | SEQRL

If a field value is generated from multiple concatenated components (either via a **PATTERN *"pattern_string"*** or using the **CHAIN** option described earlier) then the combination may well involve several **"value sequences"**.

A "sequence" is typically an incrementing/decrementing number, but could just as easily be a single character selected from an array, or a character string selected from a list.

Rather than produce a new sequential value for every component at once, it's often useful to treat the whole thing as as a combined sequence. This is a way of guaranteeing that you produce a sample of **every possible combination**.

The SeqLR (**Sequence Left to Right**) and SeqRL (**Sequence Right to Left**) options activate this feature.

The following examples illustrate the feature

◊ The 1st example combines 3 fully **independent** sequence values.
◊ The 2nd example combines 3 sequence values, sequenced **right-left**.
◊ The 3nd example combines 3 sequence values, sequenced **left-right**.

```
rand PART  seq pattern "A(A,T)[-]#(101-103)[-]#(501-503)"

  /* Output         "A-101-501"
  |                 "T-102-502"
  |                 "A-103-503"
  |                 "T-101-501"
  |                 "A-102-502"
  |                 "T-103-503"
  |                 ... then series repeats
  */


rand PART  seq pattern "A(A,T)[-]#(101-103)[-]#(501-503)"  seqRL

  /* Output         "A-101-501"
  |                 "A-101-502"
  |                 "A-101-503"
  |                 "A-102-501"
  |                 "A-102-502"
  |                 "A-102-503"
  |                 "A-103-501"
  |                 "A-103-502"
  |                 "A-103-503"
  |                 "T-101-501"
  |                 "T-101-502"
  |                 "T-101-503"
  |                 "T-102-501"
  |                 "T-102-502"
  |                 "T-102-503"
  |                 "T-103-501"
  |                 "T-103-502"
  |                 "T-103-503"
  |                 ... then series repeats
  */


rand PART  seq pattern "A(A,T)[-]#(101-103)[-]#(501-503)"  seqLR

  /* Output         "A-101-501"
  |                 "T-101-501"
  |                 "A-102-501"
  |                 "T-102-501"
  |                 "A-103-501"
  |                 "T-103-501"
  |                 "A-101-502"
  |                 "T-101-502"
  |                 "A-102-502"
  |                 "T-102-502"
  |                 "A-103-502"
  |                 "T-103-502"
  |                 "A-101-503"
  |                 "T-101-503"
  |                 "A-102-503"
  |                 "T-102-503"
  |                 "A-103-503"
  |                 "T-103-503"
  |                 ... then series repeats
  */
```

STRIPboth | STRIPLeading | STRIPTrailing

Strips leading (STRIPL) blanks, trailing (STRIPT) blanks or both (STRIP) from the generated value.

```
rand DIAL-CODE  list MY.DIAL.CODES.F80   stript  /* File is RECFM=F L=80 */
```

TIME

Use the **TIME** keyword to identify the named field as a time field.

If no time format string is supplied, and the field is not defined using one of FileKit's built-in time formats, then for character fields the default is **"hh:mm:ss.ttt"**. For fields defined with a numeric data-type (e.g. binary, packed decimal etc) the default format is **"hhmmsstt"**.

```
rand START-TIME  time   range 03:30 07:30
```

TIME *"str_ti"*

A character string that defines the format of the time field for which test data should be generated or adjusted.

The following (case-sensitive) format codes are supported, with all other characters treated as literals and transferred directly to the output.

Code	*Description*	*Examples*
hh	2-digit Hour of the day	"00" to "24"
mm	2-digit Minute of the hour	"00" to "59"
ss	2-digit Second of the minute	"00" to "59"
ttt	3-digit Thousandth of the second	"000" to "999"

e.g.

```
rand START-TIME  date "hh-mm-ss.ttt" /* e.g. "13-45-04.782" */
```

TIME ... PAST

Ensures that any time value produced will be earlier than the current time.

Equivalent to coding **"RANGE '00:00:00' '*hh:mm:ss*'"**, where *hh:mm:ss* is the current date/time.

TIME ... PAST *int* HOURS|MINUTES|SECS

Ensures that any time value produced will be earlier than the current time, but no earlier than *int* number of **HOURS/MINUTES/SECS**.

TIME ... FUTURE

Ensures that any time value produced will be later than the current time.

Equivalent to coding **"RANGE '*hh:mm:ss*' '23:53:47' "**, where *hh:mm:ss* is the current time.

TIME ... FUTURE *int* HOURS|MINUTES|SECS

Ensures that any time value produced will be later than the current time, but no later than *int* number of **HOURS/MINUTES/SECS**.

VALLOCATION(*pos* [,*len*])

With the option to pick list lines at random, in sequence or via keyed lookup, use **VALLOC** to define the **position** and **length** within each list line of the value to be generated.

If a keyed lookup is required then the position and length of the key should also be supplied using the **KEYLOC** option.

```
rand FIRST-NAME  KeyLoc(1,10) ValLoc(11,10)  StripTrailing
                 keyList(

                       /*  Before    After   */
                       /*  ------    -----   */

                          "Annabel   Alison    "
                          "Edward    David     "
                          "Heidi     Etta      "
                          "Jack      James     "
                          "Laurence  John      "
                          "Paul      Nicholas  "
                          "Peter     Paul      "
                          "Pasqual   Peter     "
                          "Simon     Ricky     "
                        )
```

VOCAB | VOC

Use **VOCAB** to supply a list of words or phrases that will be used to fill a character field.

Using the **LIST** option the list may be supplied from **in-line** values or as a **separate file**. You may omit the list altogether in order to use the product's default built-in vocabulary list that is supplied in file ***%SitePfx%*.SZZSSAM2(ZZSVOCAB)**.

Items will be repeatedly selected from the list, and concatenated with an intervening blank, to build up a **"sentence"**. The process ends when the next selected word won't fit in the remaining space.

To get realistic looking sentences you may wish to improve the chance that commonly used words, such as **"a", "an", "the", "and"** etc, have of being selected, by including them in the vocabulary list multiple times.

Use the **CAP1** option to cause the first character of the first word to be **uppercased**.

Use the **PERIOD** option to cause a period (full-stop) to be added to the end of the sentence. e.g.

For variable length fields (e.g. VARCHAR) the length allocated to build each sentence will randomly vary according to the field's defined min/maximum length.

For short (len<=20) fixed length fields (e.g. CHAR) the length allocated to build each sentence is fixed.

For longer (len>20) fixed length fields (e.g. CHAR) the length allocated to build each sentence will also randomly vary from 20 up to the length of the field.

Your vocabulary list may include some case-sensitive **special codes**:

Special Code	*Description*	*Example*
@l?	Abutt "?" to next word (no intervening blank)	Use "@l(" to start a "(xxx ...)" fragment
@L?	Abutt "?" to next word (no intervening blank) and upper-case 1st char of next word	Use "@L(" to start a "(Xxx ...)" fragment
@t?	Abutt "?" to previous word (no intervening blank)	Use "@t)" to end a "(xxx ...)" fragment.
@T?	Abutt "?" to previous word (no intervening blank) and upper-case 1st char of next word	Use "@t." to end a "xxx." fragment.

VOCAB LIST(*"item" [, ...]*)

The list may be supplied in-line as a series of blank or comma separated strings enclosed in brackets.

Each string must be quoted if it contains blanks or special characters.

```
                        /* Note the "LIST" keyword may be omitted */
rand FIRST-NAME  vocab list( "a"
                             "an"
                             "the"
                             "have"
                             "that"
                           )
```

VOCAB LIST *list_file*

The list may be supplied as a separate file.

```
rand FIRST-NAME  vocab list MY.RAND.LIST(VOCAB1)
rand FIRST-NAME  vocab      MY.RAND.LIST(VOCAB2) /* "LIST" may be omitted */
```

ZEROS

Causes numeric values generated for character fields to have leading zeros instead of leading blanks.

```
rand DIAL-CODE  seq len=4  range 1 1000 +10

  /* Output         "   1"
  |                 "  11"
  |                 "  21"
  |                 etc
  */

rand DIAL-CODE  seq len=4  range 1 1000 +10 zeros

  /* Output         "0001"
  |                 "0011"
  |                 "0021"
  |                 etc
  */
```

FOR [RECORD] *rec_type*

Identifies the record-type mapping in which the specified *field_name*/*field_ref* is defined.

Default is the default record type.

IN [STRUCTURE] *struct_name*

Identifies the name of the SDO structure in which the specified *record_type* is defined. Required only if a distinction is to be made between multiple data sets displayed in SDE views, each using different SDO structure definitions but where the specified *record_type* is defined in more than one of the structures.

Default is the SDO structure used to map records in the current SDE view.

RECFM - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option RECFM, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET RECFM in Text Editor documentation.

RECINFO - SET/QUERY/EXTRACT Option

Syntax:

```
                                     +-- ALL --+
                                     |         |
>>-+---------+- RECInfo --+- ON --+--+---------+---+-----------+------------>
   |         |            |       |                |           |
   +- SET ---+            +- OFF -+                +- Flags ---+
                                                   |           |
                                                   +- NOFlags -+

                                                  +-- DECimal ------+
                                                  |                 |
 >--+------------+--+--------+--+-------------+---+-----------------+--------><
    |            |  |        |  |             |   |                 |
    +- Length ---+  +- Id ---+  +- SQLCode ---+   +-- HEXadecimal --+
    |            |  |        |  |             |
    +- NOLength -+  +- NOId -+  +- NOSQLCode -+

>>--- Query --- RECInfo -----------------------------------------------------><

>>--- EXTract -- /RECInfo/ --------------------------------------------------><
```

Description:

This option controls which of the standard record information columns, if any, are to be displayed for records in the current SDE window view.

For both SDE EDIT and BROWSE operations, display of record information is set off by default.

The record information columns are displayed to the left of the record data field columns. In order from left to right these are flags that have been set for the record, the length of the record and the location identification of the record within the file.

Note that, if records are of variable length and Full Edit, KSDS Edit or Auxiliary Edit is in effect, then values displayed in the record information Length column may be overtyped to update the length of a record. In all other circumstances, record information columns are non-enterable.

SET RECINFO values take effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON`
`OFF`

Record information column display is switched on or off.

`ALL`

Include display of all record information columns unless column display option is specified with the "NO" prefix..

`FLAGS`
`NOFLAGS`

Include or suppress display of the record flags column.

The record flags column has a width of 4 bytes and header "Flags". Flag bytes and their meanings are, from left to right:

f	Record/segment is original, loaded from the disk copy of file.
i	Record/segment has been inserted during the edit session.
c	Record/segment has been changed since being inserted or loaded.
s	Record/segment will be saved on execution of a SAVE, or equivalent, operation.
m	Record/segment requires possible remap due to ID field change.

`LENGTH`
`NOLENGTH`

Include or suppress display of the record length column.
This option is equivalent to the SDE command RECLEN ON/OFF.

The record length column contains a 5-byte decimal value and has header "Length". For non-DB2 table edit, the contents of this column may be overtyped to alter the length of a record if an appropriate SDE edit type is in use and the records are of variable length.

If the length of a record is increased, then pad characters are appended to the record. The pad character used is that defined by the current value for PAD. Reducing a record's length will truncate record data.

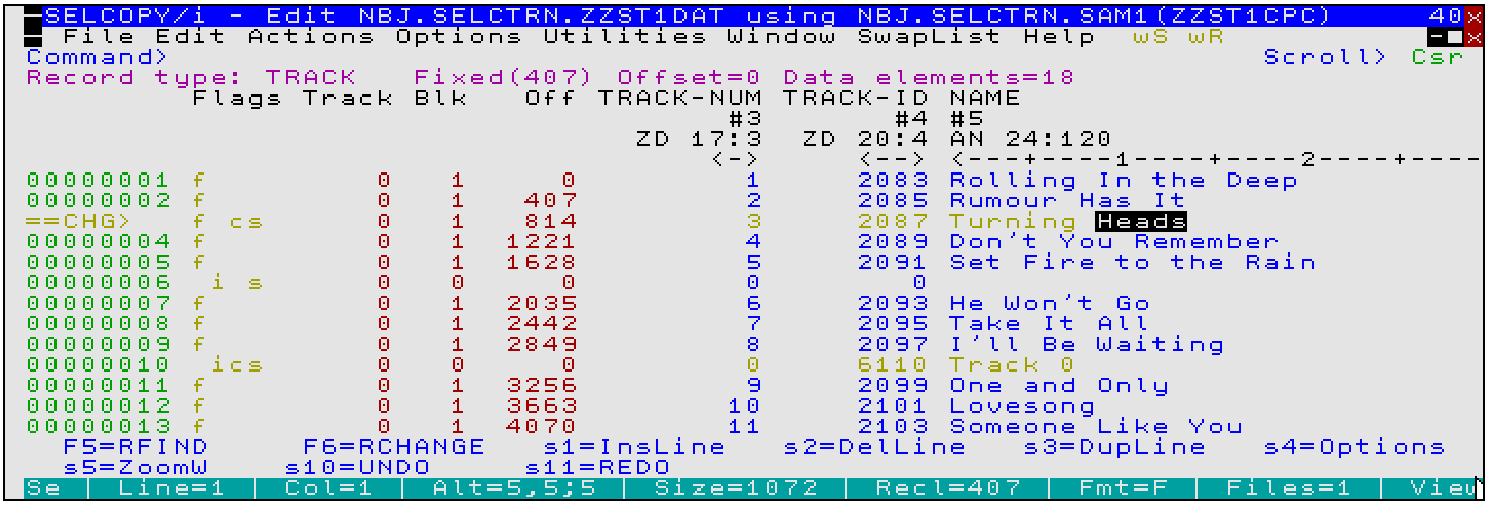
If a structure has been applied to the edited records, then the record type (RTO) associated with the updated record is re-evaluated and the contents of the SDE edit view updated accordingly.

ID
NOID

Applicable to non-DB2 table edit only, ID or NOID respectively includes or suppresses display of the record identification column.

For DB2 table edit, the ID column is not applicable and so is suppressed.

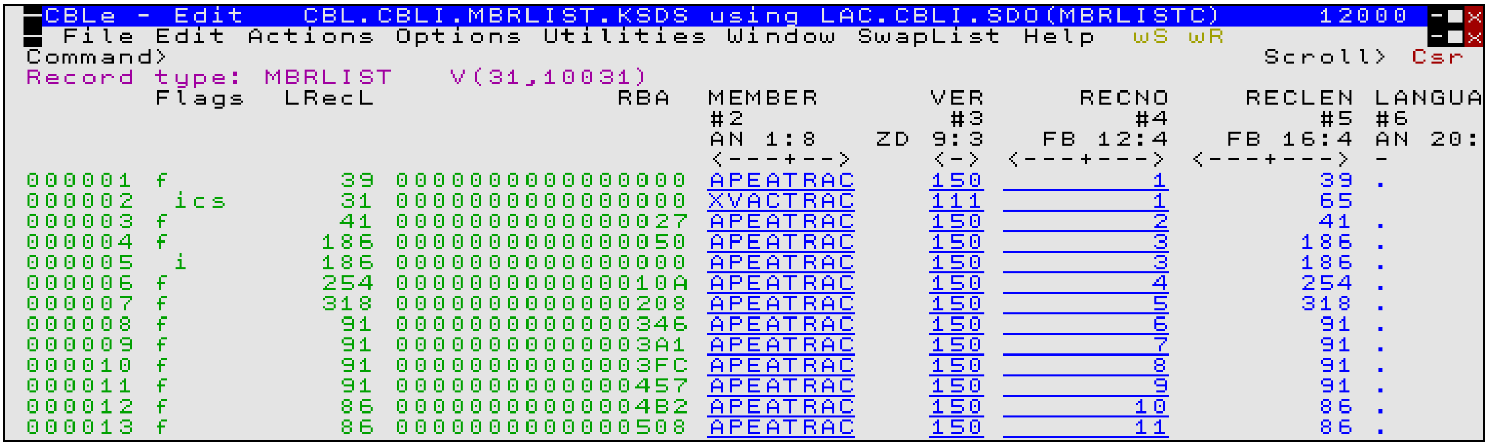
For non-VSAM data sets, the record identification is by Relative Volume, Track number, Physical Record (Block) number and offset within the physical record. The columns are displayed as decimal or hexadecimal values with headers "Vol" (if a multi-volume data set), "Track", "Blk" and "Off". If the data set is single volume, the "Vol" header is unnecessary and so omitted. The width of each of these fields are: 3 (Vol), 8 (Track), 3 (Blk) and 5 (Off).



```
 SELCOPY/i - Edit NBJ.SELCTRN.ZZST1DAT using NBJ.SELCTRN.SAM1(ZZST1CPC)    40
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                           Scroll> Csr
Record type: TRACK   Fixed(407) Offset=0 Data elements=18
         Flags Track Blk   Off TRACK-NUM TRACK-ID NAME
                                      #3       #4 #5
                                 ZD 17:3  ZD 20:4 AN 24:120
                                     <->     <--> <---+----1----+----2----+----
00000001 f         0   1     0         1     2083 Rolling In the Deep
00000002 f         0   1   407         2     2085 Rumour Has It
==CHG>   f cs      0   1   814         3     2087 Turning Heads
00000004 f         0   1  1221         4     2089 Don't You Remember
00000005 f         0   1  1628         5     2091 Set Fire to the Rain
00000006  i s      0   0     0         0        0
00000007 f         0   1  2035         6     2093 He Won't Go
00000008 f         0   1  2442         7     2095 Take It All
00000009 f         0   1  2849         8     2097 I'll Be Waiting
00000010  ics      0   0     0         0     6110 Track 0
00000011 f         0   1  3256         9     2099 One and Only
00000012 f         0   1  3663        10     2101 Lovesong
00000013 f         0   1  4070        11     2103 Someone Like You
  F5=RFIND    F6=RCHANGE   s1=InsLine   s2=DelLine   s3=DupLine   s4=Options
  s5=ZoomW    s10=UNDO      s11=REDO
Se | Line=1 | Col=1 | Alt=5,5;5 | Size=1072 | Recl=407 | Fmt=F | Files=1 | View
```

Figure 36. Record Info (Decimal) for single volume non-VSAM Data Set.

For VSAM data sets, the record identification is by Relative Byte Address (RBA). The column is displayed as a decimal or hexadecimal value with header "RBA".



```
 CBLe - Edit   CBL.CBLI.MBRLIST.KSDS using LAC.CBLI.SDO(MBRLISTC)     12000
  File Edit Actions Options Utilities Window SwapList Help  wS wR
Command>                                                           Scroll> Csr
Record type: MBRLIST   V(31,10031)
       Flags  LRecL              RBA MEMBER      VER     RECNO    RECLEN LANGUA
                                     #2           #3        #4        #5 #6
                                     AN  1:8  ZD 9:3   FB 12:4   FB 16:4 AN 20:
                                     <---+--->   <-> <---+---> <---+---> -
000001 f         39 0000000000000000 APEATRAC    150         1        39 .
000002  ics      31 0000000000000000 XVACTRAC    111         1        65
000003 f         41 0000000000000027 APEATRAC    150         2        41 .
000004 f        186 0000000000000050 APEATRAC    150         3       186 .
000005  i       186 0000000000000000 APEATRAC    150         3       186 .
000006 f        254 000000000000010A APEATRAC    150         4       254 .
000007 f        318 0000000000000208 APEATRAC    150         5       318 .
000008 f         91 0000000000000346 APEATRAC    150         6        91 .
000009 f         91 00000000000003A1 APEATRAC    150         7        91 .
000010 f         91 00000000000003FC APEATRAC    150         8        91 .
000011 f         91 0000000000000457 APEATRAC    150         9        91 .
000012 f         86 00000000000004B2 APEATRAC    150        10        86 .
000013 f         86 0000000000000508 APEATRAC    150        11        86 .
```

Figure 37. Record Info (Hexadecimal) for VSAM Data Set.

DECIMAL
HEXADECIMAL

Applicable to non-DB2 Table edit only, displays ID information as a decimal or hexadecimal value.

For **non-VSAM** data sets, the decimal display of Vol, TTR and Offset for the record occupies 4 columns in the following order:

1. Relative Volume number. (Displayed for multi-volume data sets only.)
2. Track number.
3. Physical record (block) number on the track.

4. Offset from the start of the physical record.

The hexadecimal display of Vol, TTR and Offset for the record is represented as 3 character fields of length 2, 6 and 8 bytes respectively, which represent a 1-byte relative volume number, 3-byte TTR and 4-byte offset hexadecimal value.

For **VSAM** data sets, the decimal display of Relative Byte Address (RBA) for the record is a single column containing a decimal value. The hexadecimal display of RBA is 16 bytes of character data representing a single 8-byte hexadecimal value.

`SQLCODE`
`NOSQLCODE`

Applicable to DB2 table edit only, SQLCODE or NOSQLCODE respectively includes or suppresses display of the SQL code column.

The SQL code column displays the last SQL code received from a an INSERT, DELETE or UPDATE statement performed on a row as a result of a SAVE operation.

For non-DB2 table edit, the SQLCODE column is not applicable and so is suppressed.

QUERY Response:

The current settings for RECINFO in order of display status ("ON" or "OFF"), column selections ("FLAGS" or "NOFLAGS" followed by "ID" or "NOID" followed by "LENGTH" or "NOLENGTH"), ID display format ("HEX" or "DEC") and SQL code display ("SQLCODE" or "NOSQLCODE").

EXTRACT Rexx variables:

`recinfo.0`	5
`recinfo.1`	The current setting of the record information column display, **ON** or **OFF**.
`recinfo.2`	The current value for selection of the record flags column, **FLAGS** or **NOFLAGS**.
`recinfo.3`	The current value for selection of the record identification column, **ID** or **NOID**.
`recinfo.4`	The current value for selection of the record length column, **LENGTH** or **NOLENGTH**.
`recinfo.5`	The current value for display of ID information, **HEX** or **DEC**.
`recinfo.6`	The current value for selection of the SQL code column, **SQLCODE** or **NOSQLCODE**.

RECTYPES - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ RECTypes -------------------------------------------------><

>>--- EXTract --- /RECTypes/ ------------------------------------------------><
```

Description:

Obtain all record type object (RTO) names in the current SDO.

QUERY Response:

Displays all the RTO names on a single message line.

EXTRACT Rexx variables:

`rectypes.0`	Number of RTOs in the current SDO.
`rectypes.i`	The record type name of the *i*th RTO in the SDO.

REFERENCE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- REFerence --+-- ON ---+---------------------------------><
   |           |               |         |
   +- SET -----+               +-- OFF --+


>>--- Query ------ REFerence -----------------------------------------------><


>>--- EXTract --- /REFerence/ ---------------------------------------------><
```

Description:

This option controls the display of the reference-numbers (**#nn**) occupying a row in the field column-headings for SD edit/browse.

The REFERENCE option operates at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON | OFF`
Reference number display is set ON or OFF.

QUERY Response:

The current setting of the REFERENCE option, **ON** or **OFF**.

EXTRACT Rexx variables:

reference.0	1
reference.1	The current setting of the REFERENCE option, **ON** or **OFF**.

REGION - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ REGion --------------------------------------------------><


>>--- EXTract --- /REGion/ ------------------------------------------------><
```

Description:

QUERY/EXTRACT REGION provides information about the private area region for the current MVS TCB.

REGION is not an option for the SET command.

QUERY Response:

Two pairs of values indicating the available private area storage region limit and high value both below and above the 16M line. e.g.

```
REGION Below 16M  Limit=9412608 High Value=9412608; Above 16M  Limit=33554432 High Value=33554432
```

These values are as found in the Virtual Storage Manager Local Data Area Control Block.

EXTRACT Rexx variables:

region.0	4
region.1	The below 16M line private area region limit.
region.2	The below 16M line private area region size (high value).
region.3	The above 16M line private area region limit.
region.4	The above 16M line private area region size (high value).

RESERVED - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option RESERVED, has the same effect in a Data Editor view as that supported by a Text Editor view. See SET RESERVED in Text Editor documentation.

See also RESERVEDLEVEL which controls the level (File or View) at which the RESERVED option takes effect.

RESERVEDLEVEL - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- RESERVEDLevel --+---------+-------------------------------><
   |           |                   |         |
   +- SET -----+                   +- File --+
                                   |         |
                                   +- View --+

>>--- Query ------ RESERVEDLevel---------------------------------------------><

>>--- EXTract --- /RESERVEDLevel/ -------------------------------------------><
```

Description:

This option controls the level (File or View) at which the RESERVED option takes effect. This option is primarily used in edit macros.

Option RESERVEDLEVEL takes effect at the File level.

SET Value:

`FILE | VIEW`

Reserved lines in SDE edit views, configured using the RESERVED option, apply to all edit views of the same data (FILE) or only the current view (VIEW.)

QUERY Response:

The current setting of the RESERVEDLEVEL option, **FILE** or **VIEW**.

EXTRACT Rexx variables:

reservedlevel.0	1
reservedlevel.1	The current setting of the RESERVEDLEVEL option, **FILE** or **VIEW**.

RTSCOPE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- RTSCOPE --------+- ANY ---+-------------------------------><
   |           |                   |         |
   +- SET -----+                   +- FOCus -+

>>--- Query ------ RTSCOPE --------------------------------------------------><

>>--- EXTract --- /RTSCOPE/ -------------------------------------------------><
```

Description:

Where records are assigned different record types, the RTSCOPE option determines whether a search value specified in operations FIND, CHANGE, EXCLUDE and ONLY can be found in records of ANY record type or only in records of the FOCUS record type (i.e. the default record type).

The default scope may be overridden on the individual FIND, CHANGE, EXCLUDE or ONLY primary command.

Option RTSCOPE takes effect at the Global level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ANY | FOCUS`

Default formatted record search scope is ANY record type or FOCUS record type.

QUERY Response:

The current setting of the RTSCOPE option, **ANY** or **FOCUS**.

EXTRACT Rexx variables:

rtscope.0	1
rtscope.1	The current setting of the RTSCOPE option, **ANY** or **FOCUS**.

SAVEOPTIONS - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- SAVEOPTions --+-- ON ---+--------------------------------><
   |           |                 |         |
   +- SET -----+                 +-- OFF --+

>>--- Query ------ SAVEOPTions ---------------------------------------------><

>>--- EXTract --- /SAVEOPTions/ --------------------------------------------><
```

Description:

This option controls whether or not the following SDE options are saved for use in subsequent SDE views opened in the current and subsequent FileKit sessions:

ABBREVIATION
ASCII
AUTOSAVE
AUXDSNPREFIX
CAPS
COLATTRIBUTES
COLOUR, COLOR
GENSAVE
GROUP
IDSCOPE
IDWARNING
LEVEL
LOADWARNING
MAPPING
MSGLINE
MULTIPOINT
OFFSET
PAD
PREFIX
QSEPARATOR
RECINFO
REFERENCE
SCALE
TYPE
UNDOING
UNNAMED
WRAP
ZEROS

Option SAVEOPTIONS takes effect at the Global level.

SET Value:

`ON | OFF`

Save of selected SDE options is set on or off. As distributed, the default is ON.

QUERY Response:

The current setting of the SAVEOPTIONS option, **ON** or **OFF**.

EXTRACT Rexx variables:

saveoptions.0	1
saveoptions.1	The current setting of the SAVEOPTIONS option, **ON** or **OFF**.

SCALE - SET/QUERY/EXTRACT Option

Syntax:

```
                                        +-- - (minus) --+  +-- + (plus) --+
                                        |               |  |              |
>>-+-----------+-- SCAle --+-- ON ---+--+---------------+--+--------------+-><
   |           |           |         |  |               |  |              |
   +- SET -----+           +-- OFF --+  +------ c1 -----+  +------ c2 ----+

>>--- Query ------ SCAle ---------------------------------------------------><

>>--- EXTract --- /SCAle/ --------------------------------------------------><
```

Description:

This option controls the display of the scale (**<----,----1**), occupying a row in the field column-headings for SD edit/browse.

The SCALE option operates at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

ON

Scale display is set ON.

OFF

Scale display is set OFF.

c1

The format character used for intervals of 1 byte.
Default is the minus-sign (-).
e.g. "SET SCALE ON . +" results in **'<....+....1.'**

c2

The format character used for intervals of 5 bytes.
Default is the plus-sign (+).
e.g. "SET SCALE ON - ," results in **'<----,----1-'**

QUERY Response:

The current setting of the SCALE option, **ON** or **OFF**, followed by **c1** and **c2**.

EXTRACT Rexx variables:

scale.0	3
scale.1	The current setting of the SCALE option, **ON** or **OFF**.
scale.2	The current setting of the SCALE option, **c1**.
scale.3	The current setting of the SCALE option, **c2**.

SESSION - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ SESSion --------------------------------------------------><

>>--- EXTract --- /SESSion/ -------------------------------------------------><
```

Description:

Obtain general information about the current Structured Data (SDE) Edit/Browse window view.

QUERY Response:

Displays SESSION followed by SDE session information.

EXTRACT Rexx variables:

`session.0`	2
`session.1`	The SDE window view type. **Browse** BROWSE. Also indicated by "Browse" in the title bar. **Reuse** EDIT REUSE. Also indicated by "Edit" in the title bar. **Update** EDIT UPDATE. Also indicated by "Edit(UP)" in the title bar. **Auxiliary** EDIT AUX. Also indicated by "Edit(AUX)" in the title bar.
`session.2`	The SDE record management technique. **InMemory** For EDIT REUSE only (i.e. full edit capability.), In-memory loads all records into storage and rewrites the file's records when saved. **InPlace** For EDIT UPDATE and BROWSE only, In-place keeps only displayed and modified records in storage. Record move, insert, delete and length change are not allowed. **InPlaceInMemory** For EDIT UPDATE only, InPlaceInMemory is the same as In-place except that as much of the file as possible is kept in storage. **Auxiliary** For EDIT AUX only, Auxiliary copies all records of the file to an auxiliary file in order to support full edit capability for files too large to be loaded entirely into storage (InMemory). **KSDS** For EDIT of VSAM KSDS data sets, KSDS uses the random access and update features of the VSAM KSDS organisation to provide full edit capabilities without having the whole file in storage.

SHADOW - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- SHADow --+-- ON ---+-------------------------------------->
   |           |            |         |
   +- SET -----+            +-- OFF --+

                          +-------------------- ALL ---------------------+
                          |                                              |
 >------------------------+------------+-+---------------+-+-------------+----><
                          |            | |               | |             |
                          +- EXcluded -+ +- NOTselected -+ +- SUPressed -+

>>--- Query ------ SHADow ---------------------------------------------------><

>>--- EXTract --- /SHADow/ --------------------------------------------------><
```

Description:

This option controls the display of shadow lines used to indicate the presence of records not displayed due to one of three reasons.

EXCLUDED identifies a record group that has been excluded by the user using selective line editing techniques e.g. using the WHERE or EXCLUDE commands.

NOTSELECTED identifies a record group that has not been assigned a record type. Reasons for this are:

1. The length of the data record does not fit that required to match any of the RTO definitions.
2. Fields within the data record do not satisfy the criteria associated with any of the available RTO definitions. (RTO criteria are defined as part of the RTO via a USE WHEN clause.)
3. Fields within the data record contain invalid data when an RTO definition is applied.

SUPPRESSED identifies a record group of a record type that has not been selected for view. (See the VIEW command.)

The **SHADOW** option operates at the **view** level.

SET Value:

`ON`
: Shadow display is set ON for the types that follow.

`OFF`
: Shadow display is set OFF for the types that follow.

`ALL`
: The setting affects all types of shadow lines.
SET SHADOW OFF ALL is equivalent to HIDE.
SET SHADOW ON ALL is equivalent to RESET HIDE.

`EXcluded`
: The setting affects shadow lines for EXCLUDED records.

`NOTselected`
: The setting affects shadow lines for NOTSELECTED records.

`SUPressed`
: The setting affects shadow lines for SUPPRESSED records.

QUERY Response:

The current setting of the SHADOW options, **SHADOW EXCLUDED**, **SHADOW NOTSELECTED** and **SHADOW SUPPRESSED**, each followed by **ON** or **OFF**.

EXTRACT Rexx variables:

shadow.0	6
shadow.1	The literal **EXCLUDED**
shadow.2	The **ON** or **OFF** setting for EXCLUDED shadow lines.
shadow.3	The literal **NOTSELECTED**
shadow.4	The **ON** or **OFF** setting for NOTSELECTED shadow lines.
shadow.5	The literal **SUPPRESSED**
shadow.6	The **ON** or **OFF** setting for SUPPRESSED shadow lines.

SIZE - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ SIZE ----------------------------------------------------><


>>--- EXTract --- /SIZE/ ---------------------------------------------------><
```

Description:

Obtain the number of segments and records in the current file or DB2 table. For DB2 table rows and non-segmented records, the number of segments and records are the same.

Size also returns ">" (greater than) to indicate that the number of records reported does not reflect the whole file size, or "=" (equals) to indicate that the number of records reported does reflect the whole file size.

">" may be displayed for BROWSE or UPDATE-in-place EDIT, where sequential processing of the file has been suspended until more records are required for display. "=" is displayed sequential processing has occurred until end-of-file was reached.

QUERY Response:

Displays "SIZE" followed first by the number of segments then the number of records in the file or DB2 table, then either ">" or "=".

EXTRACT Rexx variables:

size.0	3
size.1	The number of record segments in the file.
size.2	The number of records/rows in the file or DB2 table.
size.3	Indicator of whether or not the size value reflects the file size. (">" or "=")

SMFMAPLIB SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- SMFMAPLIB ---- libname -----------------------------------><
   |           |
   +- SET -----+


>>--- Query ------ SMFMAPLIB ------------------------------------------------><


>>--- EXTract --- /SMFMAPLIB/ -----------------------------------------------><
```

Description:

SMFMAPLIB identifies the library from which System Management Facility (SMF) record type layouts are retrieved by FileKit when generating its SMF structure (SDO). This SDO is used to browse and generate reports on SMF record data.

The SMFMAPLIB option operates at the Global level and sets User INI file variable **SMF.SMFMAP**. Following product installation, the value assigned to the SMF.SMFMAP variable in the System INI file will be "*hlq*.SZZSDIST.SMFMAP", where *hlq* is the product install library high level qualifier.

SET Value:

libname
The data set name of the PDS/PDSE library containing the SMF record type layout members.

QUERY Response:

The name of the PDS/PDSE library assigned to the SMFMAPLIB option.

EXTRACT Rexx variables:

smfmaplib.0	1
smfmaplib.1	The current PDS/ODSE library name assigned to the SMFMAPLIB option.

STRUCTURE - SET/QUERY/EXTRACT Option

Syntax:

```
                         +--- Edit ----------------------------------------+
                         |                                                 |
                         |                              +- FOR -+          |
                         |                              |       |          |
>>-+-------+- STRUCTure -+-+------------- struct_name +-+-------+- fileid -+-><
   |       |             | |                          |                    |
   +- SET -+             | +- ASM ------+ copybook ---+                    |
                         | +- ASSembler +             |                    |
                         | +- HLASM ----+             |                    |
                         | |            |             |                    |
                         | +- COBol ----+             |                    |
                         | |            |             |                    |
                         | +- PL1 ------+             |                    |
                         | +- PLI ------+             |                    |
                         |                            |                    |
                         +--- UNMapped ---------------+                    |
                         +--- OFF --------------------+                    |
                         |                                                 |
                         +--- Reload --------------------------------------+
>>--- Query ------ STRUCTure ------------------------------------------------><

>>--- EXTract -- / STRUCTure / ---------------------------------------------><
```

Description:

SET STRUCTURE is used to do one of the following:

- Add or remove an association definition to/from the structure to dataset associations table.
- Edit the structure to dataset associations table.
- Reload the in-storage copy of the associations table from the table library member.

Note that only one structure may be associated with a particular data file mask. If an association definition is added for a data file mask that already exists, then the existing association definition will be replaced.

SET Value:

`EDIT`
: Open a data edit view of the structure to data set associations table. This is equivalent to entering primary command EDIT from the Manage Copybook Associations panel (=0.4.6).

`struct_name`
: The FileKit SDO structure (dataset) name that will be associated with the specified data file mask (*fileid*).

`ASM | ASSEMBLER | HLASM | COBOL | PL1 | PLI` *`copybook`*
: Identifies the structure data file (*copybook*) as being a non-SDO structure of the specified source language type. This structure will be associated with the specified data file mask (*fileid*).

ASM, ASSEMBLER and HLASM indicates High Level Assembler source, COBOL indicates a COBOL copybook source, PL1 and PLI indicate a PL1 source. The source language specification governs the compiler/assembler module that will be used by FileKit to generate a temporary SDO structure.

`UNMAPPED | OFF`
: UNMAPPED or OFF are synonymous and remove the association definition from the associations table for the specified data file mask (*fileid*).

`fileid`
: Identifies the data file mask for which an association definition will be added or removed from the associations table.

The data file mask may be the DSN of a specific dataset or library member. Alternatively, standard pattern matching wildcards may be specified in the data file mask as follow.

`*`	A single asterisk indicates that either a qualifier or one or more characters within a qualifier can occupy that position. An asterisk can precede or follow a set of characters.
`**`	A double asterisk indicates that zero or more qualifiers can occupy that position. A double asterisk cannot precede or follow any characters; it must be preceded or followed by either a dot or a blank.
`%`	A single percent sign indicates that exactly one character can occupy that position. (Up to 8 percent signs can be specified in each qualifier.)

An optional library member name mask may also be specified in "( )" (parentheses) following a library DSN mask. This supports wildcards as follow.

`*`	A single asterisk represents an entire member name or zero or more characters within a member name mask.
`%`	A single percent sign represents exactly one character within a member name mask. Up to 8 percent signs can be specified in each member name mask.

`RELOAD`
: Reloads the in-storage copy of the associations table from the the ZZSDSUSE FileKit table library member.

This option is only required if the ZZSDSUSE member is updated by any method other that STRUCTURE EDIT, and the updated associations table is to be activated without restarting FileKit.

QUERY Response:

For each association definition in the table, QUERY returns a message line containing "STRUCTURE" followed by either the source language and *copybook* name or the SDO *structure* name, then "FOR" followed by the data file mask *fileid*.

EXTRACT Rexx variables:

`structure.0`	The number of association definitions.

structure.n	The SDO *structure* name or source language and *copybook* name followed by "FOR" and the data file mask *fileid*.

TITLE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- TItle ---------- short_title ----------------------------><
   |           |
   +- SET -----+


>>--- Query ------ TItle ---------------------------------------------------><


>>--- EXTract --- /TItle/ -------------------------------------------------><
```

Description:

This option displays and assigns the title that may be defined as part of an SDE structure (SDO) file. See CREATE STRUCTURE.

SET TITLE values take effect at the File level.

SET Value:

short_title
Specifies a structure title text string which may be no longer than 64 characters. Enclosing quotation marks (") or apostrophes (') may be specified and are stripped when saved in the structure.

QUERY Response:

The string "TITLE", followed by the complete structure title text.

EXTRACT Rexx variables:

title.0	1
title.1	The complete structure title text.

TYPE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- TYpe --+-- ON -------+-----------------------------------><
   |           |          |             |
   +- SET -----+          +-- OFF ------+
                          |             |
                          +-- Default --+
                          |             |
                          +-- FORMat ---+
                          |             |
                          +-- FMT ------+
                          |             |
                          +-- OFFSet ---+
                          |             |
                          +-- PICture --+

>>--- Query ------ TYpe ---------------------------------------------------><


>>--- EXTract --- /TYpe/ -------------------------------------------------><
```

Description:

This option controls display of the field data type, location and length display for SDE EDIT and BROWSE window views in either mapped table view (VFMT) or mapped single record view (MAP). The format and content is controlled by the TYPE option specified.

See also the *SHOW* command which may be used as a shortcut combining control of the *TYPE* and *OFFSET* set options.

SET TYPE takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect. If multiple SDE edit views are open for the same file then this option may be individually controlled for each view.

SET Value:

`OFF`
: Removes TYPE from display for the current view.

`ON | Default`
: For non-DB2 display is of the format ***data-type position*:*length*** where *position* and *length* identify the field's position and length within the unformatted record.
e.g. **'AN 111:30'** indicates an Alpha-Numeric character field of length 30 bytes starting at position (decimal) 111 of the unformatted record..

  For DB2 display is of the format ***data-type***, ***data-type*(*length*)** or ***data-type*(*precision*,*scale*)** representing the field's DB2 built-in data type and, where applicable, its length or its precision and scale.

  See *"SDE Data Types"* for descriptions of each of the supported data types and their representation in default TYPE display.

`FORMat | FMT`
: For non-DB2 only, display is of the format ***length*/*format*** where *length* identifies the field's length within the unformatted record, and *format* identifies the field's data-type in a descriptive form.
e.g. **'30/CHAR'** indicates an Alpha-Numeric character field of length 30 bytes.

`OFFSet`
: For non-DB2 only, display is of the format ***nnnnn*** representing the field's location within the unformatted record. The format of the displayed location may be decimal (absolute position or offset) of hexadecimal (offset) as controlled by the set option *OFFSET.*

`PICture`
: For non-DB2 only, displays the field's picture string.
e.g. **'X(30)'** indicates an Alpha-Numeric character field of length 30 bytes.

QUERY Response:

The current setting of the TYPE option, **ON/OFF/FORMAT/OFFSET/PICTURE**.

EXTRACT Rexx variables:

type.0	1
type.1	The current setting of the TYPE option, **ON/OFF/FORMAT/OFFSET/PICTURE**.

UNDOING - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- UNDOING --+- ON --+--+-----------------------------+------><
                             |       |  |                             |
                             +- OFF -+  +- n_levels --+------------+--+
                                                      |            |
                                                      +- n_kbytes -+


>>--- Query ------ UNDOING --------------------------------------------------><


>>--- EXTract --- /UNDOING/ -------------------------------------------------><
```

Description:

UNDOING defines whether the UNDO (and REDO) facility is enabled for SDE, the number of change levels that SDE will attempt to maintain and the maximum amount of storage SDE can allocate in order to store this information.

The third number following "Alt=" on the status line displays the current number of stored change levels.

The change level count is incremented by any command for which the UNDO operation is supported. i.e. any command that changes the data in the display area or the flag bits of a line. Flag bits include a line's excluded and suppressed indicators.

Multiple changes made to a file as a result of a macro execution are considered to be one change level only.

SDE is informed of any changes to the 3270 terminal when an Attention ID (AID) is generated (e.g. on hitting the Enter key or any of the PF Keys). It is only then that changes to the file are committed to the copy of the file data in SDE storage and the change level is updated. Therefore, where changes have been made to text on multiple lines, SDE has no indication as to the order in which the lines were changed and so assigns a change level to each updated line in ascending order of line number.

The UNDOING option operates at the File level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON`
`OFF`
: Set UNDOING ON or OFF.
Default is ON.

`n_levels`
: Number of change levels maintained by SDE for the file. If this value is exceeded, SDE drops the oldest undoable change level.
Default is 100.

`n_kbytes`
: Maximum amount of storage (KB) that may be obtained by SDE for storing undo information for the file.
Default is 64K.

QUERY Response:

The current setting of the UNDOING option, **ON** or **OFF** followed by number of change levels and maximum storage allocation in KBytes.

EXTRACT Rexx variables:

`undoing.0`	3
`undoing.1`	The **ON** or **OFF** setting for UNDO.
`undoing.2`	The number of change levels.
`undoing.3`	The maximum storage allocation in KBytes.

UNNAMED - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- UNNamed --+-- ON ---+-----------------------------------><
   |           |             |         |
   +- SET -----+             +-- OFF --+

>>--- Query ------ UNNamed -------------------------------------------------><

>>--- EXTract --- /UNNamed/ ------------------------------------------------><
```

Description:

This option controls whether unnamed fields appear in the display. Note that COBOL **FILLER** fields are treated as unnamed.

With UNNAMED=ON in effect unnamed fields will appear as a normal column of data in table type display, except that the field-name column heading will appear blank (or FILLER).

The UNNAMED option operates at the View level, affects all record types and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON`
`OFF`
: Set display of UNNAMED fields ON or OFF.

QUERY Response:

The current setting of the UNNAMED option, **ON** or **OFF**.

EXTRACT Rexx variables:

unnamed.0	1
unnamed.1	The current setting of the UNNAMED option, **ON** or **OFF**.

USEOFFSET - SET/QUERY/EXTRACT Option

Syntax:

```
                                                +- + (plus) -+
                                                |            |
>>-+-----------+-- USEOffset +---------------+--+------------+-- n_bytes --->
   |           |             |               |  |            |
   +- SET -----+             +- record_type -+  +- - (minus) +
                             |               |
                             +----- ALL -----+
                             |               |
                             +-----  *  -----+

 >------------------+------------------------------------+-------------------><
                    |                                    |
                    +- IN -+-------------+- struct_name -+
                           |             |
                           +- STRUCTure -+
```

```
>>--- Query ------ USEOffset +---------------+------------------------------>
                             |               |
                             +- record_type -+
                             |               |
                             +----- ALL -----+
                             |               |
                             +-----  *  -----+

 >------------------+------------------------------------+-------------------><
                    |                                    |
                    +- IN -+-------------+- struct_name -+
                           |             |
                           +- STRUCTure -+
```

```
>>--- EXTract---- /USEOffset +---------------+------------------------------>
                             |               |
                             +- record_type -+
                             |               |
                             +----- ALL -----+
                             |               |
                             +-----  *  -----+

 >------------------+------------------------------------+----- / ----------><
                    |                                    |
                    +- IN -+-------------+- struct_name -+
                           |             |
                           +- STRUCTure -+
```

Description:

This option determines the offset into a record at which data mapping by the associated record type object (RTO) is to begin. This offset may be a negative value, effectively beginning the record map before the start of the record data itself and so bypassing fields defined at the start of the RTO.

The offset value is stored as part of the RTO definition and becomes permanent when the structure is saved (e.g. using the SAVESTRUCTURE command.)

Therefore, when SDE attempts to match an RTO within the specified structure with a particular data set record, any offset value saved within the RTO is added to that RTO's determined maximum and minimum record length values. Only if the record's length falls between these adjusted minimum and maximum values (and any USE WHEN conditions have been satisfied), will the RTO become associated with the data record.

Following execution of USEOFFSET, all records that are currently in storage are re-assessed and assigned an appropriate RTO accordingly.

A negative offset cannot be applied if it would result in removing from the display a field mapping which defines the length of another field, defines the size of an array or is included as part of a USE WHEN expression.

Where an RTO containing a negative offset value is associated with a record, data at the beginning of the record may only be partially mapped by a field definition within the RTO. In this case, the partially mapped data is omitted so that record data is displayed starting at the first complete field mapping.
e.g. RTO begins with a 4-byte field map but has an offset value of -2, therefore data is displayed using the 2nd RTO field map starting at offset 2 into the record.

Where a positive offset is applied so that fields defined within the RTO correspond to positions beyond the length of the record data, then a length error occurs and **=ERRV>** is displayed in the record's prefix area.

SET USEOFFSET takes effect at the Global level.

SET, QUERY & EXTRACT Values:

`record_type`
: Set, Query or Extract the offset value for the specified record type (RTO).
Defaults to the Default Record Type.

`ALL`
`*`
: Set, Query or Extract the offset value for all record types (RTO) defined within the structure (SDO) *struct_name*.

`+ (plus)`
`- (minus)`
: Set a positive "+" (plus) or negative "-" (minus) offset value.
Defaults to a positive offset.

`n_bytes`
: Set an integer number of bytes at which record mapping is to be offset.

`IN struct_name`
: Set, Query or Extract the name of the structure (SDO) to which the specified RTO belongs.
Defaults to the SDO being used in the current SDE window.

QUERY Response:

Displays the current USEOFFSET value for the specified record type on a single message line. If parameter ALL or * is specified, the USEOFFSET value for each record type within the SDO is displayed on a separate message line.

EXTRACT Rexx variables:

useoffset.0	Number of record types (RTO) if parameters ALL or * is specified; otherwise 1.
useoffset.i	The record type followed by its current USEOFFSET value.

USERNAME - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ USERname -------------------------------------------------><

>>--- EXTract --- /USERname/ -----------------------------------------------><
```

Description:

Obtain the current user's RACF login id and the executing job name.

QUERY Response:

Displays "USERNAME RACF user" followed by the user's RACF login id, then "JOB name" followed by the currently executing job name.

EXTRACT Rexx variables:

username.0	2
username.1	The name of the user's RACF login id.
username.2	The name of the currently executing job.

USING - QUERY/EXTRACT Option

Syntax:

```
>>--- Query ------ USING ---------------------------------------------------><

>>--- EXTract --- /USING/ --------------------------------------------------><
```

Description:

QUERY/EXTRACT USING reports the structure name (SDO) and record type of the default record within the current SDE window view.

The **USING** option operates at the **view** level.

QUERY Response:

Message ZZSD079I stating the structure name and record type. e.g.

```
  ZZSD079I USING Structure CBL.FILEKIT.SDO(DIRAMEMP) Record type EMP
```

EXTRACT Rexx variables:

using.0	5
using.1	The structure name used in the current SDE window view.
using.2	The default record type.
using.3	PERM or TEMP. TEMP indicates the SDO was created internally by FileKit e.g. for a DB2 table or because a COBOL copybook was directly referenced.
using.4	The structure source. DIRECT, COBOL, ADATA COBOL, PL1, ADATA PL1, ASM, ADATA ASM, DB2, SEGMENT, RECORD.
using.5	EXPLICIT or AUTO. AUTO indicates that the mapping was automatically applied due to a dataset/structure association (See AUTOSTRUCTURE).

VALUE - EXTRACT Option

Syntax:

```
>>--- EXTract --- /VALUE/ --------------------------------------------------><
```

Description:

For use in macros, EXTRACT VALUE obtains the values of each field belonging to the default record type.

Values are obtained for each displayed field column in the order in which they appear in the display. Therefore, the SELECT command will influence the values returned by EXTRACT VALUE.

EXTRACT Rexx variables:

value.0	Number of currently selected fields belonging to the default record type.
value.i	The character representation of the *i*th field value in the display.

See Also:

EXTRACT Options: FIELD FOCUS FVALUE

VBASE - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- VBASE ------+-- ON ---+-----------------------------------><
   |           |               |         |
   +- SET -----+               +-- OFF --+


>>--- Query ------ VBASE -----------------------------------------------------><

>>--- EXTract --- /VBASE/ ----------------------------------------------------><
```

Description:

Applicable only to segmented record display, VBASE controls whether or not record data assigned to a secondary segment record type mapping is displayed or suppressed. If suppressed, only primary (base) segments are displayed.

If secondary segments are suppressed (VBASE ON) and SHADOW ON (or RESET HIDE) is in effect, then groups of lines assigned the same secondary segment time are displayed as a single suppressed shadow line.

Compare with the VIEW primary command and prefix (line) command equivalents ("V", "V+" and "V-") which suppress or display segments on an individual segment record-type basis.

SET VBASE takes effect at the view level, so separate windows open on the same file may independently control this option.

SET Value:

`ON | OFF`

VBASE ON suppresses display of all secondary segment lines and displays all base (primary) segment lines only.

VBASE OFF resets the suppression of ALL secondary segments allowing the prevailing VIEW option to take full effect.

QUERY Response:

The current setting of the VBASE option, **ON** or **OFF**.

EXTRACT Rexx variables:

vbase.0	1
vbase.1	The current setting of the VBASE option, **ON** or **OFF**.

VIEW - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- View ---- view_parms -------------------------------------><
   |           |
   +- SET -----+

>>--- Query ------ View -----------------------------------------------------><

>>--- EXTract --- /View/ ----------------------------------------------------><
```

Description:

This option controls which records or record segments are in view (not suppressed) based on their assigned record types. The SET VIEW option is equivalent to the VIEW command.

SET VIEW values take effect at the View level.

See also VBASE SET/QUERY/EXTRACT option, which provides a short cut for suppressing ALL secondary segments.

SET Value:

view_parms

Specifies parameters as supported by the VIEW command.

QUERY Response:

The string "VIEW", followed by the record type names of each non-suppressed record in the file.

EXTRACT Rexx variables:

view.0	Number of non-suppressed record type names.
view.i	The name of the *i*th non-suppressed record type entry.

VSHOWEND - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+- VSHOWend -+-- ON ---+----------------------------------><
   |           |            |         |
   +- SET -----+            +-- OFF --+


>>--- Query ------ VSHOWend ---------------------------------------------><


>>--- EXTract --- /VSHOWend/ --------------------------------------------><
```

Description:

VSHOWEND only applies to DB2 edit and browse of variable length columns.

When VSHOWEND is ON, the end of the variable length column data is marked with the VENDCHAR output indicator character. When VSHOWEND is OFF, the end of the variable length column data is not marked.

SET VSHOWEND takes effect at the file level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

ON | OFF

Specifies whether the end of the variable length column data is marked with the VENDCHAR output indicator character (ON) or not (OFF).

QUERY Response:

The current setting of the VSHOWEND options, **ON** or **OFF**.

EXTRACT Rexx variables:

vshowend.0	1
vshowend.1	The current setting of the VSHOWEND option, **ON** or **OFF**.

VSTRIP - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+- VSTRIp -+-- ON ---+---------------------------------><
   |           |          |         |
   +- SET -----+          +-- OFF --+


>>--- Query ------ VSTRIp ---------------------------------------------><


>>--- EXTract --- /VSTRIp/ -------------------------------------------><
```

Description:

VSTRIP only applies to DB2 edit of variable length columns.

When VSTRIP is ON, any trailing blanks in the column value are stripped and the length of the column data adjusted accordingly.

When VSTRIP is OFF, there is no special treatment of trailing blanks.

SET VSTRIP takes effect at the file level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON | OFF`
Specifies whether trailing blanks are to be stripped from the values of variable length columns (ON) or not (OFF).

QUERY Response:

The current setting of the VSTRIP options, **ON** or **OFF**.

EXTRACT Rexx variables:

vstrip.0	1
vstrip.1	The current setting of the VSTRIP option, **ON** or **OFF**.

WINNAME - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option WINNAME, have the same effect in a Data Editor view as that supported by a Text Editor view. See SET WINNAME in Text Editor documentation.

WINPOS - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option WINPOS, have the same effect in a Data Editor view as that supported by a Text Editor view. See SET WINPOS in Text Editor documentation.

WINSIZE - SET/QUERY/EXTRACT Option

SDE SET, QUERY and EXTRACT for option WINSIZE, have the same effect in a Data Editor view as that supported by a Text Editor view. See SET WINSIZE in Text Editor documentation.

WRAP - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- WRap ----+-- ON ---+-------------------------------------><
   |           |            |         |
   +- SET -----+            +-- OFF --+

>>--- Query ------ WRap ----------------------------------------------------><

>>--- EXTract --- /WRap/ ---------------------------------------------------><
```

Description:

WRAP defines whether a LOCATE *where_clause* search wraps around the end of the range of displayable records to continue searching until either the condition is found or the original focus line is encountered. i.e. when WRAP is set ON, searching forwards through the data continues from the Top of Data after End of Data has been reached. Likewise, searching backwards through the data continues from the End of Data when Top of Data has been reached.

Where WRAP is OFF and the End of Data or Top of Data is reached, then the following message is returned:

```
ZZSD176W Top/End of file or record range reached.
```

SET WRAP takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON | OFF`
Specifies whether LOCATE searches are allowed to wrap (ON) or not (OFF).

QUERY Response:

The current setting of the WRAP options, **ON** or **OFF**.

EXTRACT Rexx variables:

wrap.0	1
wrap.1	The current setting of the WRAP options, **ON** or **OFF**.

ZEROS - SET/QUERY/EXTRACT Option

Syntax:

```
>>-+-----------+-- Zeros -------+-- ON ---+----------------------------------><
   |           |                |         |
   +- SET -----+                +-- OFF --+

>>--- Query ------ Zeros ----------------------------------------------------><

>>--- EXTract --- /Zeros/ ---------------------------------------------------><
```

Description:

This option causes values in all numeric fields to be prefixed by zeros up to the width of the field display.

SET ZEROS takes effect at the View level and its setting is saved if SAVEOPTIONS ON is in effect.

SET Value:

`ON|OFF`
Controls whether ZEROS prefixing is in effect (ON) or not (OFF).

QUERY Response:

The current setting of the ZEROS option (ON or OFF).

EXTRACT Rexx variables:

`zeros.0`	1
`zeros.1`	The current setting of ZEROS, **ON** or **OFF**.

Prefix Area (Line) Commands

The following commands can be entered in the prefix area of an SDE browse or edit window:

.name Set a line pointer (line name).

([n] (([n] For unformatted record display only, SHIFT a line or block of lines *n* columns to the left. Characters shifted past the current BOUNDS setting are truncated.

)[n]))[n] For unformatted record display only, SHIFT a line or block of lines *n* columns to the right. Characters shifted past the current BOUNDS setting are truncated.

A Make this line the target for a move or copy (move or copy lines After this line).

B Make this line the target for a move or copy (move or copy lines Before this line).

C[n] CC Mark a line or a block of lines for copying. Lines may be copied or cut to the clipboard (using the CLIPBOARD COPY/CUT command) or copied to another position within the same edited data using prefix commands, A or B.

Only visible (not excluded) data records and shadow lines representing EXCLUDED record groups are eligible to be copied. i.e. records flagged as NOTSELECTED, SUPPRESSED or, if SHADOW OFF EXCLUDED is in effect, records flagged as EXCLUDED, are **not** eligible to be copied.

Note that, although not eligible for copy, shadow lines representing NOTSELECTED or SUPPRESSED records groups are still included within a C[n] count of lines to be copied. If a shadow line is not displayed (SHADOW OFF), it is not included within a C[n] line count.

D[n] DD Delete a line or a block of lines.

Alternatively, for RRDS/VRDS Update-in-place editing, selects lines to become Empty Slots.

Only visible (not excluded) data records and shadow lines representing EXCLUDED record groups are eligible for delete. i.e. records flagged as NOTSELECTED, SUPPRESSED or, if SHADOW OFF EXCLUDED is in effect, records flagged as EXCLUDED, are **not** eligible to be deleted.

Note that, although not eligible for delete, shadow lines representing NOTSELECTED or SUPPRESSED records groups are still included within a D[n] count of lines to be deleted. If a shadow line is not displayed (SHADOW OFF), it is not included within a D[n] line count.

E Applicable to edit of DB2 tables, E is used to display an SQL error panel describing the SQLCODE returned from the last attempt to save the row.

Refer to the ERROR primary command, which has the same effect as this prefix command, for a description of the SQL error panel.

F[n] Show the first n records of an excluded record group.

FMT Display the record or record segment in single record, formatted view.
Prefix commands FMT and MAP are synonyms.

HEX or HEXD Display the record or record segment in HEXDUMP format.

I[n] Insert a new line or a block of n new lines of the default record type.

If display format is TABLE, then numeric/bit fields are initialised to zero and character fields to blank, otherwise the whole record is initialised to blank.

For RRDS/VRDS Update-in-place editing, I[n] selects a line or block of lines to be searched for empty slots. All empty slots found within this range of lines will be activated.

Shadow lines representing NOTSELECTED or SUPPRESSED records groups are still included within an I[n] count of lines to searched. If a shadow line is not displayed (SHADOW OFF), it is not included within an I[n] line count.

ID[n] IDD Remap (IDENTIFY) a line or a block of lines.

IDENTIFY will action FileKit SDE record type assignment processing to re-assign a new record type to the records/segments in the selected lines.

L[n] Show the last n records of an excluded record group.

M[n] MM Mark a line or a block of lines for move. Lines may be moved to the clipboard (effectively performing a CLIPBOARD CUT operation) or moved to another position within the same edited data using prefix commands, A or B.

Only visible (not excluded) data records and shadow lines representing EXCLUDED record groups are eligible to be moved. i.e. records flagged as NOTSELECTED, SUPPRESSED or, if SHADOW OFF EXCLUDED is in effect, records flagged as EXCLUDED, are **not** eligible to be moved.

Note that, although not eligible for move, shadow lines representing NOTSELECTED or SUPPRESSED records groups are still included within a M[n] count of lines to be moved. If a shadow line is not displayed (SHADOW OFF), it is not included within a M[n] line count.

MAP

Display the record or record segment in single record, formatted view.
Prefix commands FMT and MAP are synonyms.

**R[n] RR[n]
"[n] ""[n]**

Replicate (duplicate) a line or a block of lines n times.

Only visible (not excluded) data records and shadow lines representing EXCLUDED record groups are eligible for replication. i.e. records flagged as NOTSELECTED, SUPPRESSED or, if SHADOW OFF EXCLUDED is in effect, records flagged as EXCLUDED, are **not** eligible to be replicated.

RE

Applicable to edit and browse of DB2 tables, RE is used to list, edit or browse tables related to the current table.

This command is equivalent to issuing the REDIT primary command with no parameters, with the row on which the prefix command is entered being the focus row for the command.

SEL

Open the SDE SELECT Columns Panel to apply column selection, column sequencing and/or column width definitions to the record-type definition mapping the focus line.

STP

Applicable to edit of Segmented Records only.
Force a secondary segment to be a primary (base) segment so splitting the record into two. No action is taken if the segment is already a primary segment type.
An IDENTIFY operation is automatically performed on the focus record to remap all segments with appropriate record types.

STS

Applicable to edit of Segmented Records only.
Force a primary segment to be a secondary segment so joining the record with the record before. No action is taken if the segment is already a secondary segment type.
An IDENTIFY operation is automatically performed on the focus record to remap all segments with appropriate record types.

V

Display only records that are of the same record type as this line.
V may be specified on a visible data record or on EXCLUDED, SUPPRESSED or NOTSELECTED shadow lines.
The DRECTYPE value is updated to be this record type and this line becomes the first line of the display.

V+

Add to the display records that are of the same record type as this line.
V+ may be specified on a visible data record or on EXCLUDED, SUPPRESSED or NOTSELECTED shadow lines.
Note, however, that specification on a visible data record or EXCLUDED record group shadow line will not add new record types to the display.
The DRECTYPE value is updated to be this record type and this line becomes the first line of the display.

V-

Remove from the display records that are of the same record type as this line.
V- may be specified on a visible data record or on EXCLUDED, SUPPRESSED or NOTSELECTED shadow lines.
Note, however, that specification on a SUPPRESSED or NOTSELECTED record group shadow line will not remove record types from the display.
The DRECTYPE value is updated to be this record type and this line becomes the first line of the display.

X[n] XX

Mark a line or a block of lines for exclusion from the display.

Only visible (not excluded) data records are eligible to be excluded. i.e. records flagged as NOTSELECTED, SUPPRESSED or EXCLUDED are **not** eligible to be excluded.

Note that, although not eligible for exclusion, shadow lines representing NOTSELECTED, SUPPRESSED or EXCLUDED records groups are still included within an X[n] count of lines to be excluded. If a shadow line is not displayed (SHADOW OFF), it is not included within an X[n] line count.

Z

Switch to a zoomed (single record view) display of the record occupying this line.

**BLUE (BL)
GREEN (G)
PINK (P)
RED
TURQ (T)
WHITE (W)
YELLOW (Y)**

Set the colour for the focus record type.

Generates and executes an RCOLOUR command that is unconditional and will replace any existing row coloring options for the focus record type.

Function Keys

3270 Program Function Keys (PFKeys) may be assigned to SDE commands.

The default keylist for SDE browse/edit views is **DATAEDIT**.

The SDE program default function keys are:

F1	HELP	The standard HELP key.
F2	SPLIT	The ISPF SPLIT command.
F3	END	Quit the file (you will be prompted to save any changes).
F4	WINDOW	Navigate open FileKit windows.
F5	RFIND	Locate search string defined by last FIND or CHANGE command.
F6	RCHANGE	Repeat the change requested by the last CHANGE command.
F7	UP	Scroll the window display upwards by an amount determined by the scroll field or specified on the command line.
F8	DOWN	Scroll the window display downwards by an amount determined by the scroll field or specified on the command line.
F9	SWAP	The ISPF SWAP command.
F10	LEFT	Scroll the window display to the left by an amount determined by the scroll field or specified on the command line.
F11	RIGHT	Scroll the window display to the right by an amount determined by the scroll field or specified on the command line.
F12	CRETRIEV	Retrieve the last command to the command line.
F13	INSERT	Insert a new record following the focus line. (Same as "I" line-command)
F14	DELETE	Delete the focus line. (Same as "D" line-command)
F15	DUPLICATE	Duplicate the focus line (Same as "R" line-command)
F16	MACRO SDEUTIL	Open the SDE Edit/Browse Options Menu.
F17	MACRO SDEZOOMW	Open a new view window of the same file window ZOOMed-mode (MAP/FMT).
F21	SWAP LIST	The ISPF SWAP LIST comamnd.
F22	UNDO	Press repeatedly to UNDO your changes one at a time.
F23	REDO	Press repeatedly to REDO your changes one at a time.

Note that the contents of the command line is concatenated to the definition of the function key and the result executed as a single command.

Glossary

The following is a glossary of terms used in this document.

CLI (**Command Line Interface**)
A Command Line Interface is a text based method by which users can execute functions supported by the application.

Text Editor
A function rich Text Editor that runs as an MDI application in FileKit. The Text Editor supports its own command interface and has been developed based on specifications for both the ISPF PDF Editor and Mansfield Software's KEDIT for Windows.

Current Column
The first field column visible within the current display area, belonging to records of the default record type. The current column references the same field within all data records that are of the default record type.

Current Line
The first record group (data record or shadow line) displayed within the current display area view.

Current Record Group
The record group occupying the current line.

Current Data Editor View
The Data Editor View which received focus last.
The current Data Editor view persists even if the current focus window is not a Data Editor view. The concept of a current Data Editor view is important when executing Text/Data Editor REXX macros or when executing SDE commands from a Text Editor view via the SDATA command.

Default Record
The focus line if it contains a visible data record, otherwise the first visible data record following the focus line that is of the default record type.

Default Record Type
The default record type is defined as being the record type of the focus line if the focus line is a visible or EXCLUDED record group shadow line, otherwise it is the record type defined by DRECTYPE setting.
See section *Default Record Type*.

EXCLUDED
The status of a record group in a display of formatted or unformatted records which has been excluded by the execution of an exclude operation. (e.g. EXCLUDE or WHERE.) By default, record groups that are EXCLUDED are represented by an "Excluded" shadow line.

Focus Column
The field column on which the cursor is positioned within the focus line.
If the focus line is **not** a visible or EXCLUDED record group, or if the cursor is positioned outside the display area (e.g. the command line) or within the prefix area, the focus column is defined as being the current column. The focus column references the same field within all data records that are of the same record type as the focus line.

Focus Field
The individual field within the focus line referenced by the focus column.

Focus Line
The line within the display area on which the cursor is positioned.
If the cursor is positioned on a header line, the focus line is the first line that immediately follows the header line. If the cursor is positioned outside the display area (e.g. the command line), then the focus line is defined as being the current line.

Focus Record Group
The record group occupying the focus line.

Header Line
A line within the Data Editor view display area that contains column header information for the record group that follows. A header line constitutes one line within a group of header lines which scroll with the record data display.

INI file
File containing configuration options for FileKit. The System INI file is processed on startup of FileKit and contains options that apply to all users. The User INI file contains options specific to each user that may, where appropriate, override options set in the System INI file.

List Window
A FileKit window containing rows of associated information. List windows support point-and-shoot column sorting; select, sort and filter CLI commands; and prefix area commands.

MDI
Multiple Document Interface is a Microsoft specification for PC applications that enable the user to work with multiple documents at the same time. Each document is displayed in a separate child window within the client area of the application's main (frame) window. Typical MDI applications on PCs include word-processing and spread sheet applications.

MDI Client Area window
The MDI client area window is the display area within an MDI application's frame window. The MDI client area serves as the backround for MDI child windows.

MDI Child/Document Window
An MDI child or document window is opened in an application's client area window each time a document is opened. Each child window has a sizing border, title bar, window menu, minimise, maximise, restore and close buttons. A child window is clipped so that it is confined to the client window and cannot appear outside it.
When a child window is maximized, its client area completely fills the MDI client area window. In addition, the system automatically hides the child window's title bar, and adds the child window's window menu icon and Restore button to the MDI application's menu bar.

MDI Frame Window
An MDI frame window may be considered the main window of an MDI application. It is the parent window of the MDI client area window in which MDI child windows are opened. It has a sizing border, title bar, window menu, minimise, maximise restore and close buttons.

NOTSELECTED
The status of a record group in a display of formatted records which has no assigned record type. By default, record groups that are NOTSELECTED are represented by a "Not Selected" shadow line. Since the introduction of the default record type, "Unmapped", it is not possible for record groups to be NOTSELECTED. (See "Record Type Assignment")

Record Group
A record group is one or more data records that is represented by a single line in the Data Edtor view display.
A visible data record may be considered to be a record group of one record whereas a shadow line may be a record group of one or more consecutive records of the same record type.

Record Type
Term referring to the name assigned to a record type object (RTO) on execution of the CREATE STRUCTURE command.
This name is the highest level field name identifier for a record mapping in a COBOL or PL1 copybook or in a FileKit SDE data definition clause.

RTO (**Record Type Object**)
A single record type definition within an SDO.
RTO definitions are generated via the CREATE STRUCTURE command which uses record structures defined with FileKit's own SDE structure definition syntax or defined from within an existing COBOL or PL/1 copybook.
An RTO (and hence the SDO) may subsequently be altered (e.g. via the USE command) and saved to an SDF.

SD (**Structured Data**)
Structured Data refers to data within file records that have a pre-defined structure. See structured records.
SD is also the minimum abbreviation for SDATA, the Text Editor primary command used to prefix any of FileKit's SDE primary commands when executed from within a Text Editor view.

SDF (**Structure Definition File**)
A disk file (sequential data set or PDS/PDSE member) containing a saved Structure Definition Object (SDO).
If not already in storage, an SDF gets loaded (creating an SDO) during execution of an EDIT or BROWSE command in order to apply a structure to records processed by FileKit's SDE.
If an SDO is altered during a Data Editor session and is not flagged as being temporary, then the user will be prompted to save the SDO to an SDF. An SDO may also be saved to an SDF automatically during a CREATE STRUCTURE command.

SDE (**Structured Data Environment**)
The Structured Data Environment runs under FileKit and includes a number of primary commands and options which support processing data which is mapped into distinct fields by external structure definitions (e.g. COBOL, PL1, Assembler copy books).
FileKit utilities that use SDE features and so support structured data include the Data Editor, File Copy, File Search & Update, File Compare, Report, etc. Although applicable to a number of utilities, SDE primary commands and options are documented in the FileKit Data Editor manual.

Data Editor View
A Data Editor MDI document window that may contain a display of structured data. If the same file is displayed in multiple windows, then the user has multiple Data Editor views and/or Text Editor views of the file.

SDO (**Structure Definition Object**)
An in-storage structure definition consisting of one or more record type objects (RTO) that map records in a structured file.
An SDO is created by a CREATE STRUCTURE command or an EDIT or BROWSE command if the SDF is not already loaded in storage.

FileKit
The Interactive environment developed by CBL and supplied as part of SELCOPY and CBLVCAT licensable software products.

Structured Data Set
A data set containing structured records.

Structured Records
Records that consist of one or more data fields, each with a defined field start position, length and data type.

SUPPRESSED
The status of a record group in a display of formatted records which has been suppressed by the execution of a VIEW operation either on the initial BROWSE or EDIT command, or specifically by the VIEW primary or "V" line command. By default, record groups that are SUPPRESSED are represented by a "Suppressed" shadow line.