CardAK

Card Interchange Files Manager and Helpers

(Card Army Knife)

DRAFT

Version: 0.8.x (and older)

Table of contents

Introduction

This tool was created to help handling **MasterCard** interchange files

It was created as a personal helper, as these files are not easy to visualize and handle because of their format.

In its current state, most of the implemented functions can handle **MasterCard IPM** files, with some limited functionality over **MPE** files

How to use this manual

Due to the complexity of the tool, and the use of concepts that are not necessarily frequently used, it has been decided to separate this document into three large areas.

The first one, containing practical usage examples and how to use the tool to solve them.

The second one, is a detailed explanation of each of the supported commands, which have been developed to be used both by an operator or as part of automated procedures.

The third and last one, describes and shows how to use the **TUI** (Text User Interface), where an operator can do most of the actions on the files, requiring the usage of keyboard combinations.

It is suggested to read the Use cases section to better understand how to solve typical or similar problems. To better understand each command, please refer to the Commands section

Features and functionalities

The main features are:

- Multi platform (It works in Linux and in Windows)
- It does not require instalation, it is just an executable binary that needs to be put in the **PATH**
- It is a CLI application, so it is executed from a terminal. This allows it to be used in remote servers using (for example) a ssh connection
- It can be used both as an interactive application, or in automated tasks

Among its functionality, we find:

- Identify the file types by its contents and not by its name
- Convert files between formats (**ASCII**/**EBCDIC**, with/without blocks 1014, etc.)
- Validate the files for format errors and some data errors
- Possibility to automatically fix some errors
- It supports logic files inside phisical files (transmissions), allowing to separate then into physical files (and vice-versa)
- It lets you export data into readable formats, like plain text files, **CSV** files (that can be open with spreadsheets), o like Hex values to be imported in another file
- Import data from a previously exported data
- Detect duplicate records inside a file
- Split big files into smaller ones for easier handling
- Search for data inside files by using flexible conditions
- Distribution of registers among several files according to some user defined criteria
- Deletion of records from a file
- Open the files interactively, where the user can perform several actions like:
  - Visualize the contents of the file
  - Delete records or fields
  - Edit records

- Perform searches and filters
- Export selected records
- Import records previously exported from another file
- Add fields and records

Besides, other functionality is being developed

Use cases

Here we will go through some situations we can face when dealing with these files, and we will see how the tool can help us deal with them. They are practical examples that allows us to understand the possibilities of the tool and how to use it, which in some cases require usinig more than one command.

To learn more about the different commands, we can use the section Commands in this same document.

Identify and obtain file information

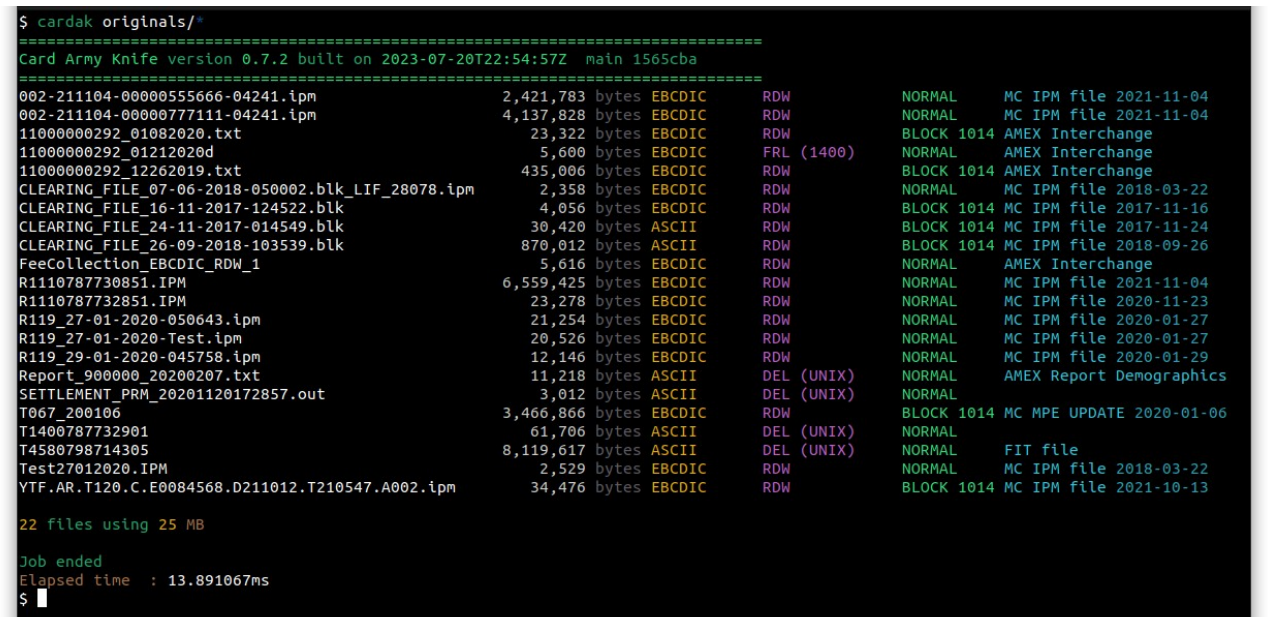
Many times we find ourselves with a list of files of different types, and just with their names is hard to determine their contents. Just listing the contents of the directory we may not be able to determine the kind of information inside, and it is almos impossible to know the file format.

As an example, let's see the contents of a directory with some files to better explain this situation, with an example:

```
$ ls -l originals/*
-rw-rw-r-- 1 eduardo eduardo 2421783 feb  7 15:13 originals/002-211104-00000555666-04241.ipm
-rw-rw-r-- 1 eduardo eduardo 4137828 feb  7 15:13 originals/002-211104-00000777111-04241.ipm
-rw-rw-r-- 1 eduardo eduardo   23322 abr 19  2022 originals/11000000292_01082020.txt
-rw-rw-r-- 1 eduardo eduardo    5600 abr 19  2022 originals/11000000292_01212020d
-rw-rw-r-- 1 eduardo eduardo  435006 abr 19  2022 originals/11000000292_12262019.txt
-rw-rw-r-- 1 eduardo eduardo    2358 abr 19  2022 originals/CLEARING_FILE_07-06-2018-050002.blk_LIF_28078.ipm
-rw-r--r-- 1 eduardo eduardo    4056 set 26  2018 originals/CLEARING_FILE_16-11-2017-124522.blk
-rw-r--r-- 1 eduardo eduardo   30420 set 26  2018 originals/CLEARING_FILE_24-11-2017-014549.blk
-rw-r--r-- 1 eduardo eduardo  870012 set 26  2018 originals/CLEARING_FILE_26-09-2018-103539.blk
-rw-rw-r-- 1 eduardo eduardo    5616 abr 19  2022 originals/FeeCollection_EBCDIC_RDW_1
-rw-r--r-- 1 eduardo eduardo 6559425 nov  4  2021 originals/R1110787730851.IPM
-rw-rw-r-- 1 eduardo eduardo   23278 abr 19  2022 originals/R1110787732851.IPM
-rw-rw-r-- 1 eduardo eduardo   21254 abr 19  2022 originals/R119_27-01-2020-050643.ipm
-rw-rw-r-- 1 eduardo eduardo   20526 abr 19  2022 originals/R119_27-01-2020-Test.ipm
-rw-rw-r-- 1 eduardo eduardo   12146 abr 19  2022 originals/R119_29-01-2020-045758.ipm
-rw-rw-r-- 1 eduardo eduardo   11218 abr 19  2022 originals/Report_900000_20200207.txt
-rw-rw-r-- 1 eduardo eduardo    3012 abr 19  2022 originals/SETTLEMENT_PRM_20201120172857.out
-rw-rw-r-- 1 eduardo eduardo 3466866 abr 19  2022 originals/T067_200106
-rw-rw-r-- 1 eduardo eduardo   61706 abr 19  2022 originals/T1400787732901
-rw-r--r-- 1 eduardo eduardo 8119617 may 27  2020 originals/T4580798714305
-rw-rw-r-- 1 eduardo eduardo    2529 abr 19  2022 originals/Test27012020.IPM
-rw-r--r-- 1 eduardo eduardo   34476 nov  4  2021 originals/YTF.AR.T120.C.E0084568.D211012.T210547.A002.ipm
$
```

We can see a list of files. Some of the names give us a clue of what they are, but extensions are sometimes meaningless and not all files correspond to a particular name pattern. Besides, we can't know the used encoding (either **ASCII** or **EBCDIC**), if they are files with fixed length, variable length or delimited records, or if they are marked in blocks (**MasterCard** usually sends its files in blocks).

Let's see now how to use the tool to determine all this. When we just execute the tool without any command, it assumes we are using the command IDENTIFY



We can see the following information: first, the file name and then its size in bytes, the encoding (**ASCII** or **EBCDIC**), the record type (**RDW** – Variable length records, **FRL** – Fixed length records and their size, **DEL** – Delimited and the Unix or Windows delimiters), and if the files is in blocks or not.

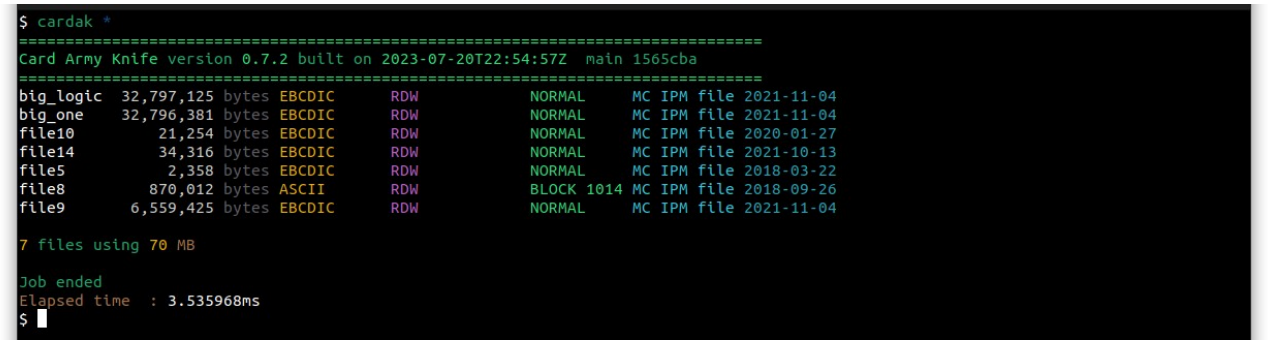
Optionally, it shows the type of the file. We can see that most of them are **MasterCard IPM** files, but there are some **MPE files**, and some **VISA** and **AMEX** files**.**

For **MasterCard IPM** and **MPE** files, we can see the date present in the Header of the files.

Even though the tool is mostly focused on working with **MasterCard IPM** files, some operations work with other file types, like converting the file format as we will see later..

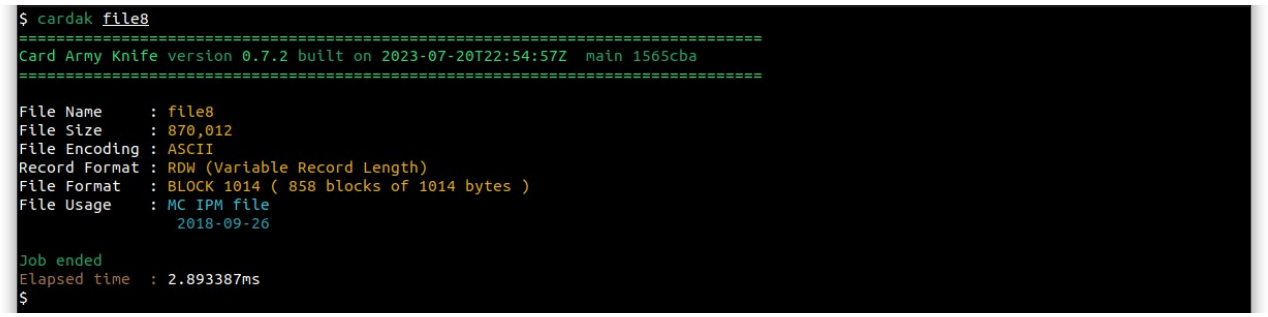
Next, and to use some examples, we will work with the following files:

Let's see what are those files:



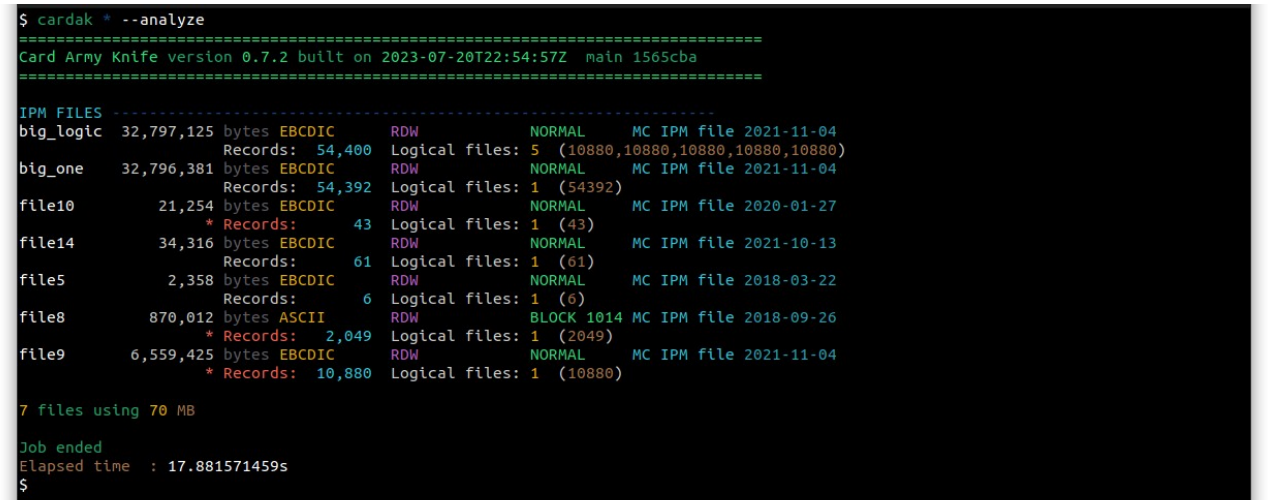
They are all **MasterCard IPM** files, and we have some in **ASCII** some in **EBCDIC**, and some in blocks. All the **IPM** files contain variable length records (**RDW**)

This is a compact view, where the information for each file is printed on one line. We can have a different view if we just supply a single file (for example *file8)*, supplying the file name as parameter like this:



This information is obtained by sampling the first 10Kb of the file (or the whole file if its size is less than 10 Kilobytes), so this is performed pretty fast.

If we want to have more information, we can specify the flag --analyze (-a), that will force the tool to read the whole file (it will be much slower) but it will display more information. For example:



In this case, for each file, and extra line is displayed showing the number of records inside the file, the number of logical files and number of records for each logical file. In this example, we can see that the first file contains 5 logica files, with each one containing 10880 recors (NOTE: the 5 logical files contain the same number of records because the file was created by concatenating the same file 5 times in order to have a big file for the examples)

We an also see that some lines contain the word “Records” in red and with an asterisk in from. This means that the file contains some errors detected when reading the file.

In order to get more information, we will indicate only one file as a parameter, which will display more detailed information. We will check for this example the file *file10*

```
$ cardak file10 -a
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-20T22:54:57Z  main 1565cba
================================================================================

IPM FILES -------------------------------------------------------------------

File Name     : file10
File Size     : 21,254
File Encoding : EBCDIC
Record Format : RDW (Variable Record Length)
File Format   : NORMAL
File Usage    : MC IPM file
                2020-01-27

FILE HAS ERRORS:
                LF: 1 PRN:     43 DE71 in Trailer is not greater than the last one, current: [00000043], previous: [00000209]
                LF: 1 PRN:     43 PDS 0301 (checksum) in Trailer is not correct, value: [0000000000000000], expected: [00000000043
79378]

Total records : 43
Logical files : 1

Records by MTI      :      33 - 1240 Presentment
                            8 - 1740 Fee Collection
                            2 - 1644 Administrative

Records by Tran type:      33 - First Presentment
                            8 - Fee Collection (Customer-generated)
                            1 - File Trailer
                            1 - File Header

Job ended
Elapsed time  : 20.892598ms
$
```

We can see that the file contains errors in the logical file number 1 (LF: 1), and in particular in the physical record number 43 (PRN: 43) which is the file trailer. We can see that the field **DE71** (Line number indicator inside the file) does not comply being greater than the value of the previous record. We can also see that the field PDS0301, containing the amount checksum, is incorrect.

We will also get statistical information like the count of records by **MTI**, or the count of records by each transaction type.

If we add the flag --verbose (-v), we will see the count of records by each present **MCC**

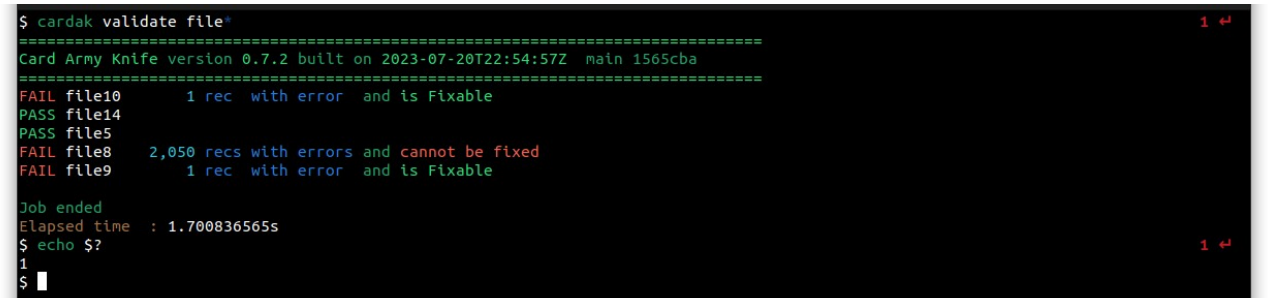
```
$ cardak file10 -a -v
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-20T22:54:57Z  main 1565cba
================================================================================

IPM FILES -------------------------------------------------------------------

File Name     : file10
File Size     : 21,254
File Encoding : EBCDIC
Record Format : RDW (Variable Record Length)
File Format   : NORMAL
File Usage    : MC IPM file
                2020-01-27

FILE HAS ERRORS:
                LF: 1 PRN:     43 DE71 in Trailer is not greater than the last one, current: [00000043], previous: [00000209]
                LF: 1 PRN:     43 PDS 0301 (checksum) in Trailer is not correct, value: [0000000000000000], expected: [00000000043
79378]

Total records : 43
Logical files : 1

Records by MTI      :       2 - 1644 Administrative
                           33 - 1240 Presentment
                            8 - 1740 Fee Collection

Records by Tran type:       1 - File Header
                           33 - First Presentment
                            8 - Fee Collection (Customer-generated)
                            1 - File Trailer

Records by MCC      :      11 - 5251 HARDWARE STORES
                            6 - 5541 SERVICE STATIONS WITH OR WITHOUT ANCILLARY SERVICE
                            9 - 5732 ELECTRONIC SALES
                            7 - 5812 EATING PLACES, RESTAURANTS

Job ended
Elapsed time  : 31.550791ms
$
```

Validate the files for errors

This is useful to be used in an automated process, where it is better to perform corrective actions before sending or processing files with detectable errors. They can be validated just after being generated and before being transferred to avoid rejections and possible fines by the brand.

The validation can be performed on a file by file basis, or over a list of files. The tool returns an error code to the operating system if any file presents errors, or zero if all the files appear to be fine. This allows to be performed automatically inside tasks and perform different actions as needed.

Let's see how it works. First, we will validate all files with name file*



We see two files that pass the validations and three that fail. We can see the number of records containing errors, and a message indicating if the errors found can be corrected automatically or not. The value returned to the operating system in this case in 1, so we know some files contain errors.

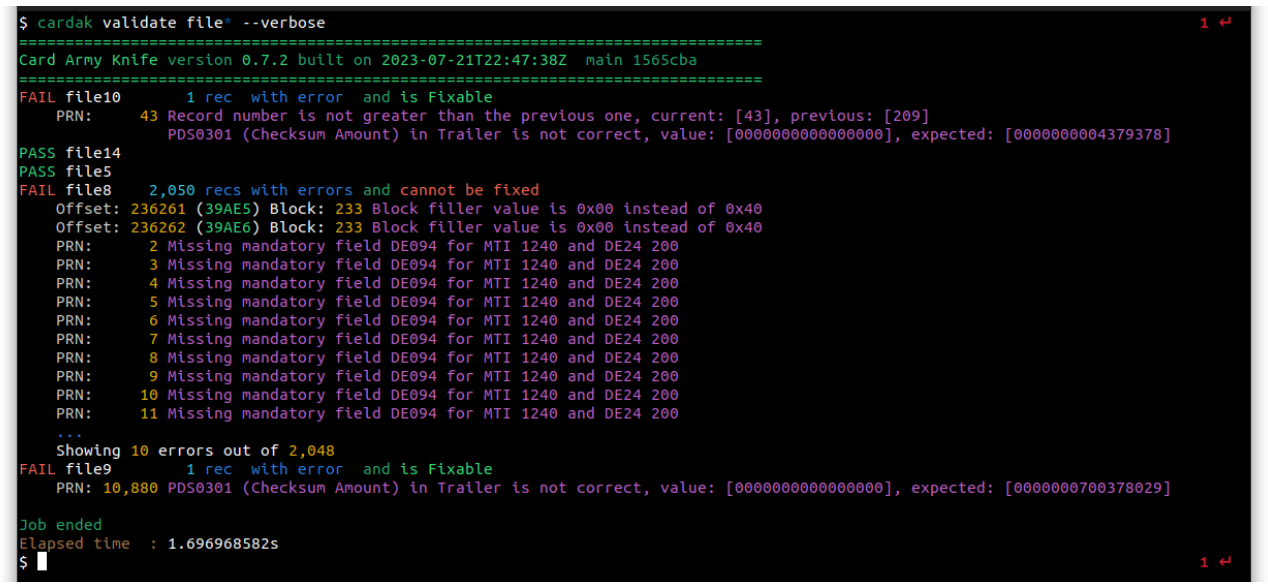
We can add the flag --silent (-z) to simplify the output in order to be used in automated tasks.

The flag --silent omits all output to the console, except for the results. We can see the output is one line per file, containing 4 sections separated by the character ":". The first section indicates the validity of the file, the second is the file name, the third is the number of records with errors, and the last one indicates if the errors can be fixed automatically or not.

An alternative useful for automations, is to specify if we just want a list with files that pass or fail the validation. For example::

This way, we can just get the list of files that fail or pass, and perform some action with them.

Returning to the validation, we can have more detailed information about the errors by adding the flag --verbose (-v) like this:



We can see that the file file10 contains the error in the trailer record (that can be corrected automatically), and file9 also contains an error in the trailer record because the checksum does not contain the expected value. This error can be fixed automatically by putting the correct value in this field.

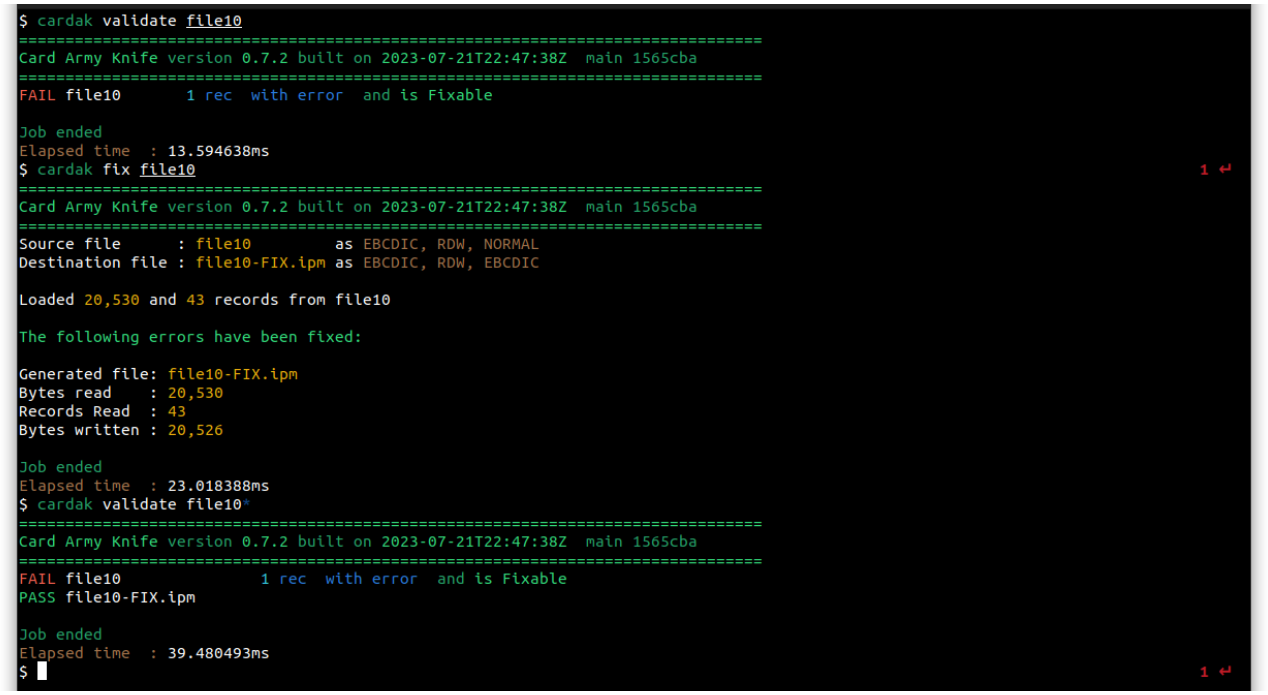
But, we can see that *file8* contains several errors of different types. The first two errors correspond to the file format, indicating that in positions 236261 y 236262 the bytes contain the value **0x00** when the correct values would be **0x40**. These two errors can be automatically corrected, but there are other errors that cannot. We just see the first 10 errors because the tool found 2048 which are too much to be displayed. We see the errors are that for the combination of **MTI** (1240) and Function Code **DE24** (200), the field **DE094** (Transaction Originator Institution ID Code) is mandatory, but not present. This kind of errors cannot be corrected automatically as we cannot insert a field without a previous analysis.

Automatically correct errors

Some errors can be corrected automatically. For example, record numbers inside a file need to be in ascending order, and each logical file header must start with record number 1. The checksum of the trailer can also be calculated and fixed, and the fill characters of blocks can be fixed (the standard stipulates that the value must be **0x40** but sometimes other value comes, like **0x00**). All these errors can be fixed just by calculating the correct values.

Other error types, like missing mandatory fields, or malformation of the file (like incorrect lengths or unauthorized characters on a field) probably cannot be fixed automatically. In those cases, the tool has some options to fix them manually (Please, refer to the section TUI - Text User Interface)

We previously saw that file10 contains two fixable errors. To do this, we will use the command **FIX**



The fixed file keeps the original name, but adds the string “-FIX.ipm” at the end. This extension is added because the command **FIX** only works on **IPM** files.

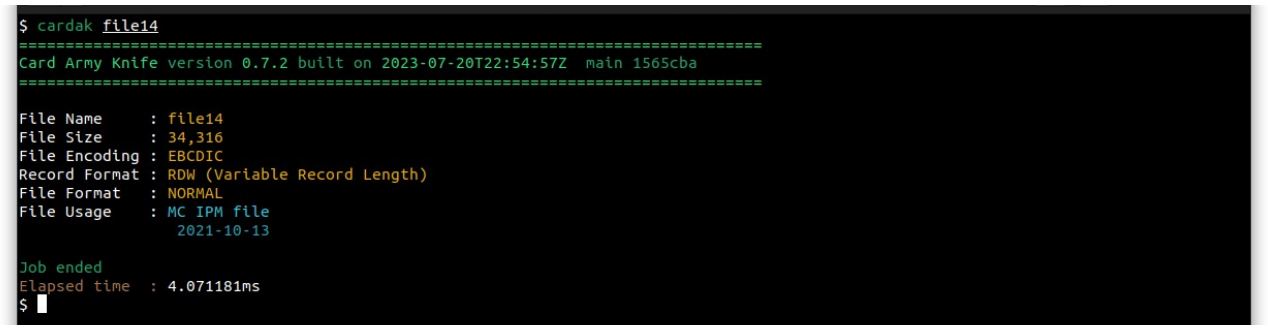
We can perform this command on one file at a time, or on a list of files, but in this case, only **MasterCard IPM** files with fixable errors will be fixed.

Convert file formats

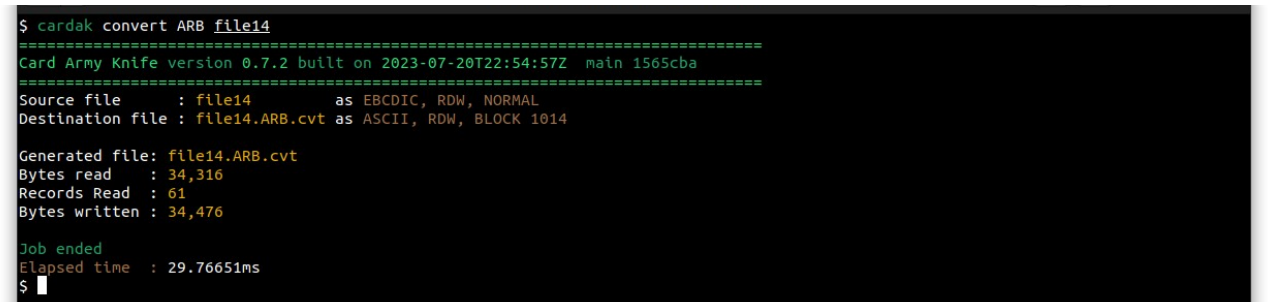
Some times, especially during certification and developing stages, the format of the generated or received files is not the expected one. In some implementations, it is not easy or convenient to change the configuration and reprocess the file.

In these cases, the tool allows us to easily convert the file from one format to another.

For example, let's see the format of file14:



It is a file encoded in **EBCDIC** and it is not in blocks 1014. Suppose we need it encoded in **ASCII** and in blocks 1014. We can easily perform this conversion like this:



We use the command **CONVERT**, specifying the desired format (by supplying three letters for the encoding, record format and file format) – For more details, please see the detailed usage of the command CONVERT in the corresponding section.

The generated, converted file, keeps the name of the original file and adds to the end the three letters used for the conversion, plus the extension .cvt

This command works for different file types, not just **MasterCard IPM,** like **MPE** and some other file types.

This command can also be used with one file at a time, or we can batch convert several files if the final conversion type is the same.

Search for data inside IPM files

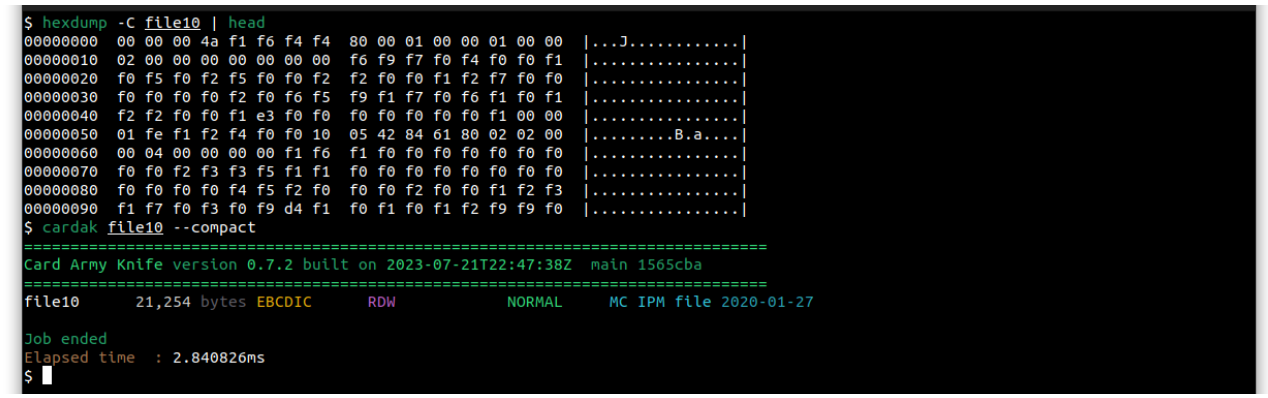
On Linux (and Unix in general), we ca use the grep command to search inside files. This works great when working with plain text files, but **IPM** files are not the case.

As an example, suppose we know that in some of these files we have some purchases in a store named “Carpinteria”, but we don’t know which files contain these transactions.

First, we will try with the system’s grep utility:

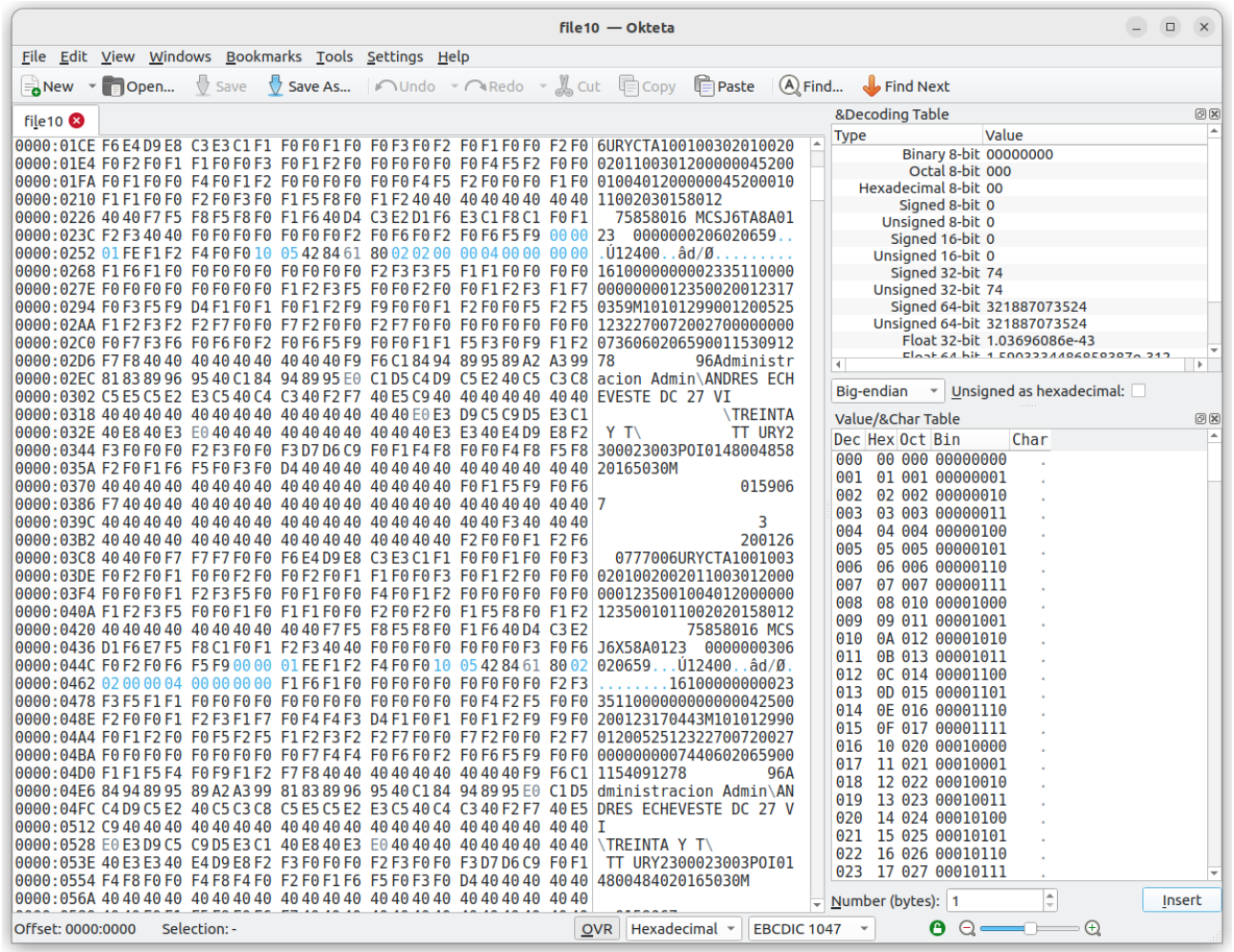
As we can see, we get no results, and we also get a return code of 1, saying that binary search is not allowed.

We can, of course, open the file with an hexadecimal editor and perform the search. Let’s try with the hexdump utility on the command line:



It doesn’t seem to contain legible data, so what is the problem? Well, we see that the encoding is **EBCDIC**, so we can clearly see the byte values as hexadecimal values, they don’t contain a simple **ASCII** representation. Of course, we could first convert the file into **ASCII** and then use hexdump, but having manyu files this is not a very practical solution.

So, let’s try using another tool, in this case a graphical tool (there are several hexadecimal editors for Windows and Linux):



Now, after changing the codification to **EBCDIC**, we start to see values that can be easier to identify. But this option requires to open the files one by one to perform the search, and when trying to search for something more specific, not just a string of characters, the tools is not much better.

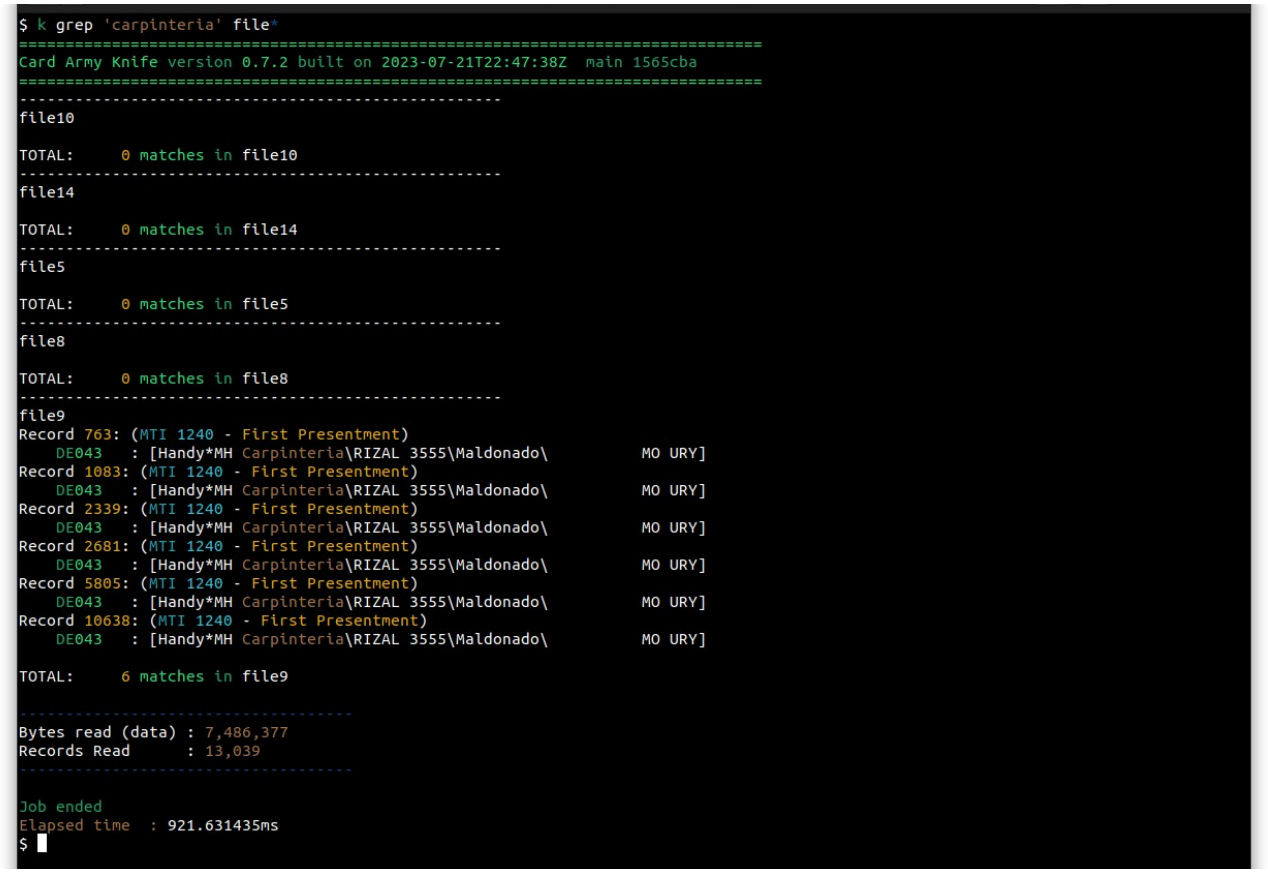
So, let's see how to use the tool to perform searches insode **IPM** files.

As we mentioned earlier, we wanto to search for files containing transactions performed in a store containing the word "carpinteria" in its name. We know the merchant name comes in field **DE043**, but let's start performing a global search (like in all fields)

For that, we call the tool with the command **GREP**, specifying first the search criteria and the the file(s) where to perform the search.

About the search criteria, this is very flexible, and it allows us to perform global searches (anywhere in the record), on specific fields (including subfields) or using logical operators like **OR** and **AND** (for more information, please refer to the corresponding section for the command GREP)

For the moment, just search for the text "carpinteria". As a note, the search is not case-sensitive



Well, we see that in the file file9 there are 6 records containing the string in field **DE043**. We also see the record numbers where the value is present, and we also see that those 6 records correspond to first presentments.

From now on, we can start narrowing the search, but now focusing on just the file file9 which is the one that contains our possible transaction.

Our next step could be, as an example, that besides displaying the record number that contain the searched value, we would like to see the contents of fields **DE004** (transaction amount) and **DE012** (Date and Time). To do that, we can add the flag -F where we can indicate the fields we want to show from matching records, even if those fields don't contain the searched value.

To indicate the desired fields, we just write the list of fields separated by commas, and we can indicate if we are referring to a Data Element (**DE),** a Private Data Sub-element (**PDS**), or if we want to specify a Sub-field of any field. For example, the Date and Time come in field **DE012**, it contains two sub-fields that are the Date in sub-field 1, and the time in sub-field 2. So, instead of specifying that we want the whole field **DE12**, we can just indicate the sub-fields **DE12 SF 1** and **DE12 SF 2** separately.

For more information about how to specify the fields, sub-fields and how to use predefined lists, please refer to the section FILTER

Let's see the output of this command:

```
$ k grep 'carpinteria' file9 -F 'D4,D12s1,D12s2'
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
------------------------------------------------
file9
Record 763: (MTI 1240 - First Presentment)
    DE004   : [000000059017]
    DE012S01: [211029]
    DE012S02: [171126]
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 1083: (MTI 1240 - First Presentment)
    DE004   : [000000140656]
    DE012S01: [211029]
    DE012S02: [175500]
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 2339: (MTI 1240 - First Presentment)
    DE004   : [000000197000]
    DE012S01: [211030]
    DE012S02: [171923]
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 2681: (MTI 1240 - First Presentment)
    DE004   : [000000233115]
    DE012S01: [211030]
    DE012S02: [110446]
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 5805: (MTI 1240 - First Presentment)
    DE004   : [000000296066]
    DE012S01: [211031]
    DE012S02: [113844]
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 10638: (MTI 1240 - First Presentment)
    DE004   : [000000204591]
    DE012S01: [211103]
    DE012S02: [105208]
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]

TOTAL:     6 matches in file9

-----------------------------------
Bytes read (data) : 6,559,425
Records Read      : 10,880
-----------------------------------

Job ended
Elapsed time  : 812.682041ms
$
```

Now we can also see the amounts and times of the present transaction for this merchant.

Now, imagine we have a lot of records that match the searched criteria, but if we also know some other data from the transaction we could narrow down the search. For example, if we know that the transaction we are looking for has the amount is 1970.00, we could specify the search like this:

```
$ k grep 'carpinteria,D4:197000' file9
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
------------------------------------------------
file9
Record 2339: (MTI 1240 - First Presentment)
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
    DE004   : [000000197000]

TOTAL:     1 matches in file9

-----------------------------------
Bytes read (data) : 6,559,425
Records Read      : 10,880
-----------------------------------

Job ended
Elapsed time  : 789.731388ms
$
```

Now we are getting just one one result, record 2339, containing the transaction we are looking for.

At this point we may want to see the full record for the transaction. We could supply a list of all the fields as we have seen before, but there are better ways.

For this, we could use the command PRINT. This command allows us to display in a friendly way, the contents of a file or specified records. We could redirect the output to a text file and open it in a text editor.

So, let's see the contents of the record previously found with the **GREP** command. We saw that the matching record was number 2339, so we could just specify to PRINT only that record.

```
$ cardak print file9 -R 2339
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
file9
------------------------------------------------------------------------------------------------------------------------
 2339: MTI 1240 - First Presentment
        DE002   DE003   DE004   DE012   DE022   DE024   DE026   DE031   DE033   DE038   DE040   DE042   DE043   DE049   DE055   DE
063   DE071   DE094
       PDS0023 PDS0148 PDS0158 PDS0159 PDS0165 PDS0208 PDS0777 PDS1001 PDS1002 PDS1003 PDS1004 PDS1011

-----------------------------------
Bytes read (data) : 6,559,425
Records Read      : 10,880
-----------------------------------

Job ended
Elapsed time  : 69.054472ms
$
```

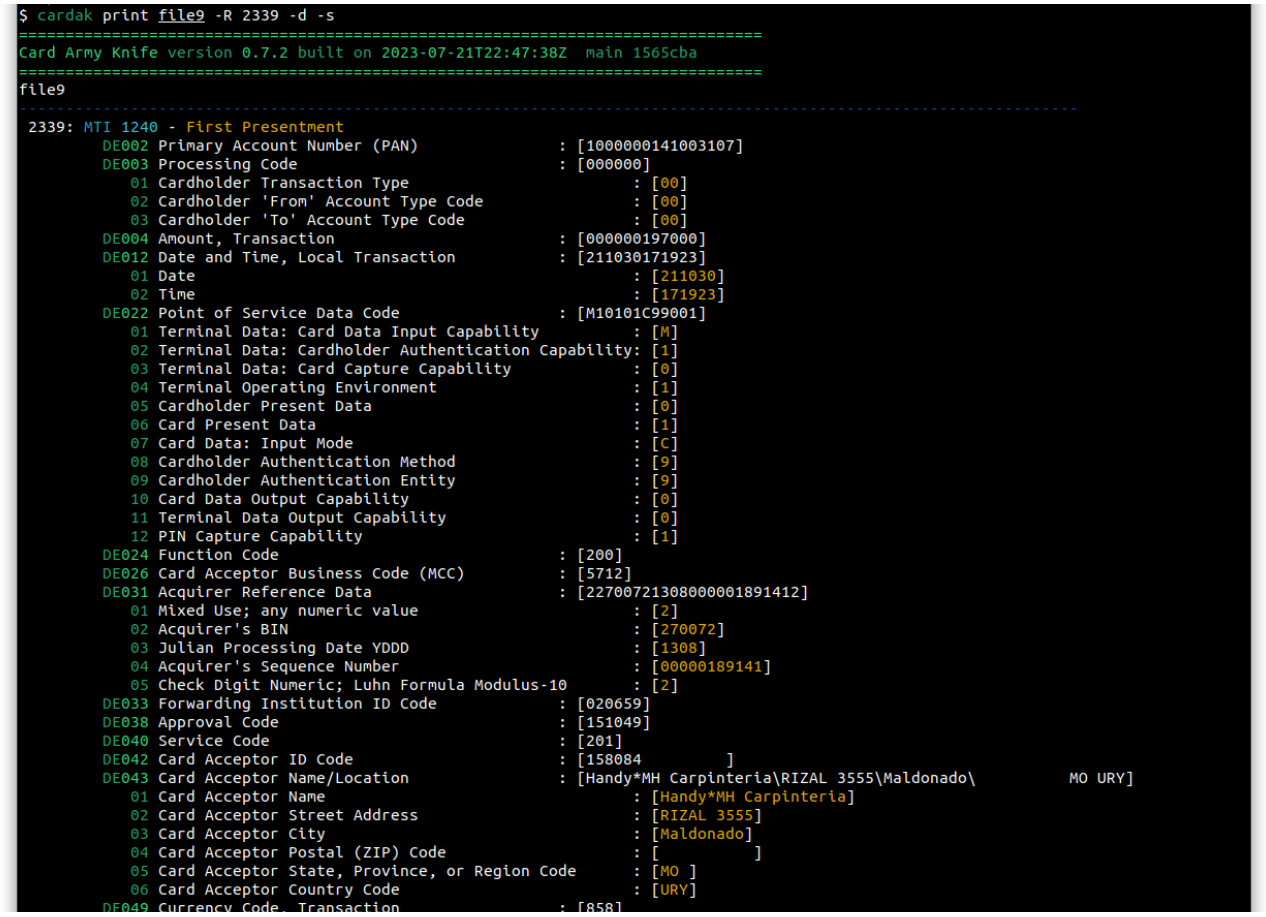
By default, the **PRINT** command shows the list of present fields on each record (first the DE and then the PDS).

If we want to see the values of each field, we just add the flag --detailed (-d)

```
$ cardak print file9 -R 2339 -d
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
file9
------------------------------------------------------------------------------------------------------------------------
 2339: MTI 1240 - First Presentment
        DE002 Primary Account Number (PAN)              : [1000000141003107]
        DE003 Processing Code                           : [000000]
        DE004 Amount, Transaction                       : [000000197000]
        DE012 Date and Time, Local Transaction          : [211030171923]
        DE022 Point of Service Data Code                : [M10101C99001]
        DE024 Function Code                             : [200]
        DE026 Card Acceptor Business Code (MCC)         : [5712]
        DE031 Acquirer Reference Data                   : [22700721308000001891412]
        DE033 Forwarding Institution ID Code            : [020659]
        DE038 Approval Code                             : [151049]
        DE040 Service Code                              : [201]
        DE042 Card Acceptor ID Code                     : [158084         ]
        DE043 Card Acceptor Name/Location               : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
        DE049 Currency Code, Transaction                : [858]
        DE055 Integrated Circuit Card (ICC) System Related Data: [◆◆\0Hn       ◆◆◆Rm◆&◆◆◆◆◆◆◆◆◆ ◆◆◆◆◆◆◆Eh^◆◆◆◆b◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆◆d◆
◆^◆◆◆]
        DE063 Transaction Life Cycle ID                 : [ MCS5O4W2E1030  ]
        DE071 Message Number                            : [00002339]
        DE094 Transaction Originator Institution ID Code : [020659]
       PDS0023 Terminal Type                                        : [POI]
       PDS0148 Currency Exponents                                   : [8582]
       PDS0158 Business Activity                                    : [          06]
       PDS0159 Settlement Data                                      : [                                        3          N0
000000022020200]
       PDS0165 Settlement Indicator                                 : [M                              ]
       PDS0208 Additional Merchant Data                             : [00000268588HY000923       ]
       PDS0777 Promotion Code                                       : [URYCTA]
       PDS1001 Installment Plan Type                                : [23]
       PDS1002 Installment Number                                   : [01]
       PDS1003 Installment Amount                                   : [000000032833]
       PDS1004 Total Transaction Amount                             : [000000197000]
       PDS1011 Total Number of Installments                         : [06]

-----------------------------------
Bytes read (data) : 6,559,425
Records Read      : 10,880
-----------------------------------

Job ended
Elapsed time  : 57.387614ms
$
```

In this way, we see the list of present fields and their corresponding value.

We can also add the flag --subfields (-s) to see the contents of each sub-field of the fields that contain sub-fields.



If we want to redirect the output to a file to be opened with a text editor, it is convenient to add the flags flags --silent (-z) or --mono so the output will not contain the codes used to display colours.

How to see the file contents in an external program

Maybe we want to see or manipulate the data that come in an **IPM** file with some external program, like a text editor or an Excel spreadsheet. This tool allows to print or export all or some records to a file that can be used by those external programs.

First, let's see how we look at the contents of some records of an **IPM** file in a text editor.

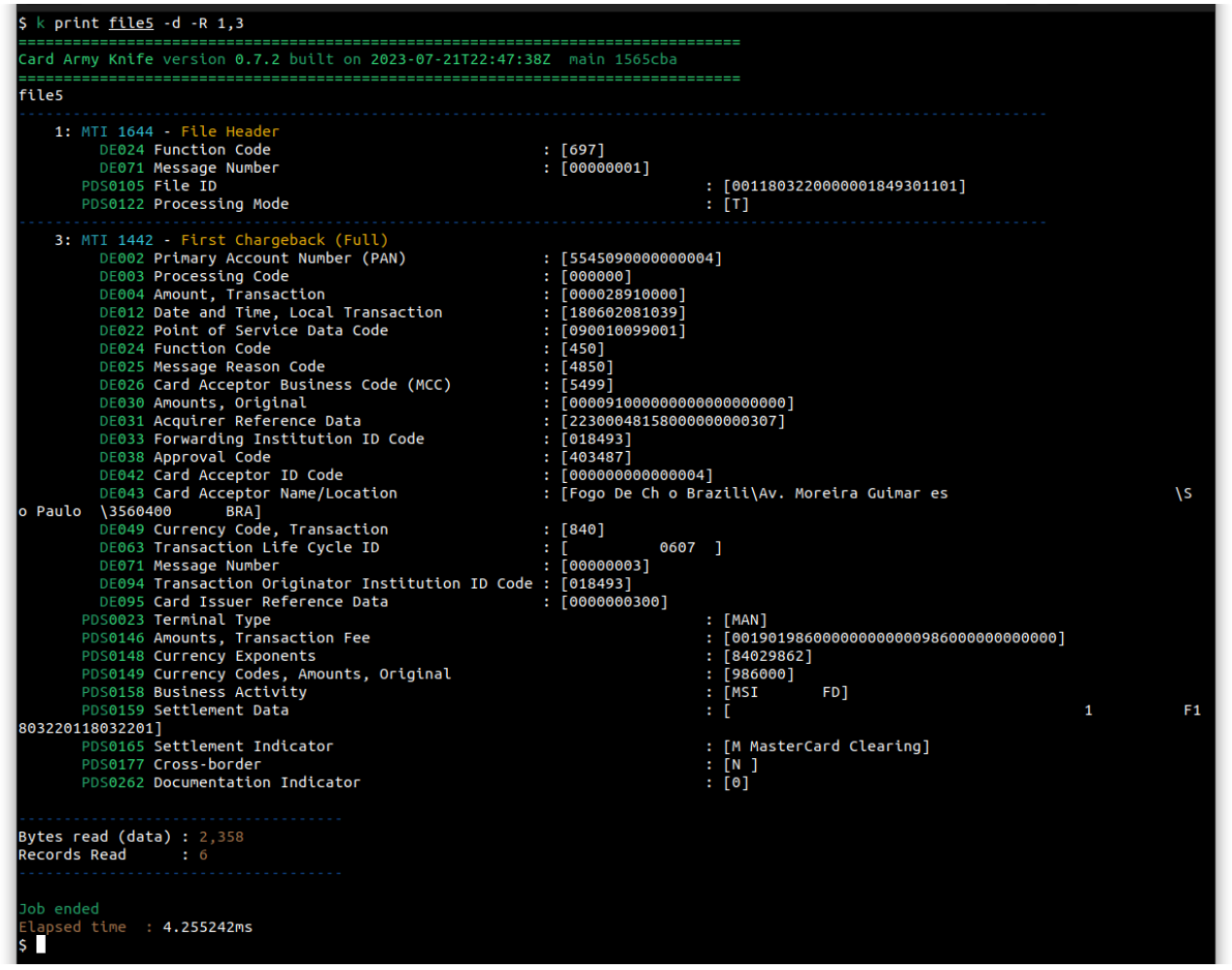
To do this, we will use the **PRINT** command. With this command, we can display all the records from an **IPM**, or just the ones that we want.

By default the **PRINT** command shows the record number, the record type and the list of fields present on each record.

If we also want to display the values of the fields, we need to add the flag --detailed (-d).

We can also select the records to show by using the flag -R, and display just the wanted fileds by using the flag -F

To see how to use these flags in more detail, please refer to the section called FLAGS and FILTERS



We can use the flag --silent and redirect the output to a file, and then open it with a text editor and see the contents, perform searches, print in paper, etc.

Another option is to use the command **EXPORT**. This command, by default, generates a **CSV** file which can be open with Excel or similar spreadsheet. This file contains the first line with the name of the fields, and on each consecutive line, the corresponding values for each record.

You can use the same filters to select the records and fields to export to the file as with other commands

There is a flag, --console that if present, instead of generating the file, displays the output on STDOUT, so we can redirect it to any file.

Manage logical files

We may have a file that contains more than one logical file inside (**MasterCard** calls these logical files “transmissions”), and we may want to generate physical files for each one of those logical files (or vice-versa).

For that, we can use the commands **SPLIT** and **JOIN**

The command **SPLIT** takes the contents of a file, and creates a physical file for each logical one contained in the source file.

The command **JOIN** does the opposite. Takes a list of physical files and creates a new file, generating inside a logical file for each file in the list.

```
$ cardak join file5 file10 file14
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
 +           6 records to logical file 1 from file5
 +          43 records to logical file 2 from file10
 +          61 records to logical file 3 from file14

Generated 20230721233425_J.ipm -> 110 records using 57,200 bytes

Job ended
Elapsed time  : 43.55596ms
$ cardak 20230721233425_J.ipm -a --compact
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================

IPM FILES ---------------------------------------------------------------------
20230721233425_J.ipm      57,200 bytes EBCDIC      RDW            NORMAL     MC IPM file 2018-03-22
                               * Records:     110  Logical files: 3  (6,43,61)

Job ended
Elapsed time  : 29.71457ms
$ cardak split 20230721233425_J.ipm
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
Source file      : 20230721233425_J.ipm         as EBCDIC, RDW, NORMAL

Loaded 57,200 and 110 records from 20230721233425_J.ipm
Analyzing file...
? Please, choose an action: Fix
Fix - Fix the file and proceed

The following errors have been fixed:

      6 records (     2,358 bytes) written to 00118032200000001849301101-230721233456-1.ipm
     43 records (    20,526 bytes) written to 00220012700000002065917061-230721233456-2.ipm
     61 records (    34,316 bytes) written to 00121101300000001444402201-230721233456-3.ipm

Job ended
Elapsed time  : 13.152925545s
$
```

We will see these commands in action. First, we join three files (file5, file10 y file14) generating a new one with name 20230721233425_J.ipm (it is the generation date and time, plus the string “_J.ipm”

We see that effectively, this new file contains three logical files with 6, 43 y 61 records each.

The, we split that file using the command **SPLIT**. When trying to separate them, it detects some errors and we are given the chance to cancel, ignore or correct those errors. We choose to Fix them, and three new files are generating, which names are the value of field **PDS0105** from the header, followed by a dash and date/time of creation, plus a sequential number (the number of logical file) and the extension “.ipm”

Delete records from a file

Maybe, after generating a file, we need to remove some records before sending the file, just because we don’t want to send a particular transaction or because for some rejection in a previous transfer.

Sometimes we can just regenerate the file, but not all systems have an easy or quick wayt to do that.

We first need to identify the record(s) we want to remove from the file. We can do this by performing a search as seen previously in the Search for data inside IPM files section.

What we need is to get the physical record numbers of the records to be removed.

As an example, we will need the same criteria in the previous example by using the **GREP** command. We will eliminate from file9 the record corresponding to the merchant “Carpinteria” and with an amount of 1970,00

```
$ cardak grep 'carpinteria,D4:197000' file9 --matches --silent
file9
2339
$
```

Here we will use the reduced version of the search, as we already know that the transaction is located in that file and we have also found the record, so we just need to know the record number (we will use the --matches and –silent flags), getting the record number 2339

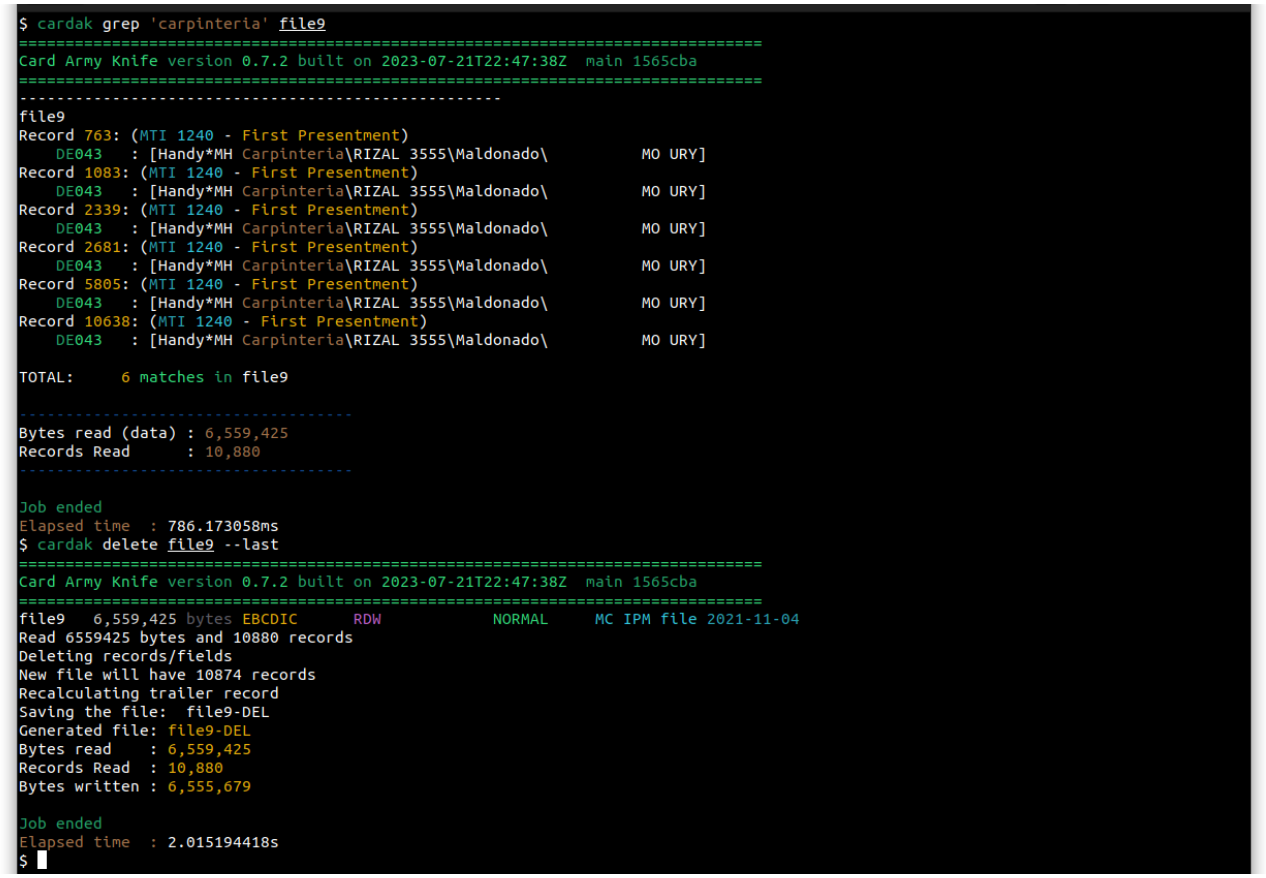
```
$ cardak delete file9 -R 2339
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
file9   6,559,425 bytes EBCDIC      RDW            NORMAL     MC IPM file 2021-11-04
Read 6559425 bytes and 10880 records
Deleting records/fields
New file will have 10879 records
Recalculating trailer record
Saving the file:  file9-DEL
Generated file: file9-DEL
Bytes read    : 6,559,425
Records Read  : 10,880
Bytes written : 6,558,793

Job ended
Elapsed time  : 1.788785402s
$ cardak file9* -a -v
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================

IPM FILES ------------------------------------------------------------------
file9       6,559,425 bytes EBCDIC      RDW            NORMAL     MC IPM file 2021-11-04
                    * Records:  10,880  Logical files: 1  (10880)
    LF: 1 PRN: 10,880 PDS 0301 (checksum) in Trailer is not correct, value: [0000000000000000], expected: [0000000700378029]
file9-DEL   6,558,793 bytes EBCDIC      RDW            NORMAL     MC IPM file 2021-11-04
                    Records:  10,879  Logical files: 1  (10879)

2 files using 12 MB

Job ended
Elapsed time  : 3.068579703s
$
```

We can observe that one record has been eliminated from the file, which originally contained 10880 records, and now it contains 10879. Se also see that the original file had a fixable error in the trailer, but the new file, as trailer values had to be recalculated, is now correct..

We also see that the original file is not really modified, but a new one is generated, with the same name as the original (by default) and adding “-DEL” (We can specify the name of the new file by using the flag --out)

But what if we want to eliminate, for example, all the records corresponding to the merchant “Carpinteria”? We could delete records one by one, or take note of the record numbers in order to supply a list of records to delete, but that can be a little unpractical if we need to delete several, non consecutive records.

In that case, we can use the flag --last that uses the records numbers returned by the last execution of the **GREP** command realized over that file. Let’s see how to use it:



We search for the value “carpinteria”, and in return it says there are 6 records that match the criteria.

The next command to use is the **DELETE** command but adding the flag --last, so that it uses the result of the last **GREP** command, deleting those 6 records. We see that effectively, a new file is generated with 10874 records (the original 10880 minus the 6 deleted ones).

Delete fields from some records

We may also need to eliminate certain fields from some given records. For example, if we have included by mistake some field that should not be present.

Just like the previous case, we just need to use the flag -F with the list of the fields we want to remove. These fields will be removed from the requested records, or from all the file if we don’t specify any records.

Care must be taken as there is no check on the removed fields, so we can be removing mandatory fields.

Transfer records from one file to another

Let’s say we want to transfer some records from one file to another one. This can be done in two steps, first, removing the records from the first file, and second, inserting those records into the other file.

The first step is achieved with the command **EXPORT**, and the second one with the command **IMPORT**

For the command **EXPORT** we have two options. Either export to a **CSV** or an **HEX** file. Both formats can be used for the import, but the **HEX** format is considered a better way as it is less likely to be modified.

First, we need to select the records to export. We can use the same method as the previous example, by searching for records containing the string “Carpinteria”:

```
$ cardak grep 'carpinteria' file9                                                                    1 ↵
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
------------------------------------------------
file9
Record 763: (MTI 1240 - First Presentment)
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 1083: (MTI 1240 - First Presentment)
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 2339: (MTI 1240 - First Presentment)
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 2681: (MTI 1240 - First Presentment)
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 5805: (MTI 1240 - First Presentment)
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]
Record 10638: (MTI 1240 - First Presentment)
    DE043   : [Handy*MH Carpinteria\RIZAL 3555\Maldonado\          MO URY]

TOTAL:     6 matches in file9

-----------------------------------
Bytes read (data) : 6,559,425
Records Read      : 10,880
-----------------------------------

Job ended
Elapsed time  : 801.227122ms
$
```

Next, we can export those records into a **HEX** file:

```
$ cardak export file9 --last --hex
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
6      lines written to file9-EXP.ckh

Job ended
Elapsed time  : 56.223089ms
$
```

Then, we will import those records into the file named file14, that we previously saw has 61 records:

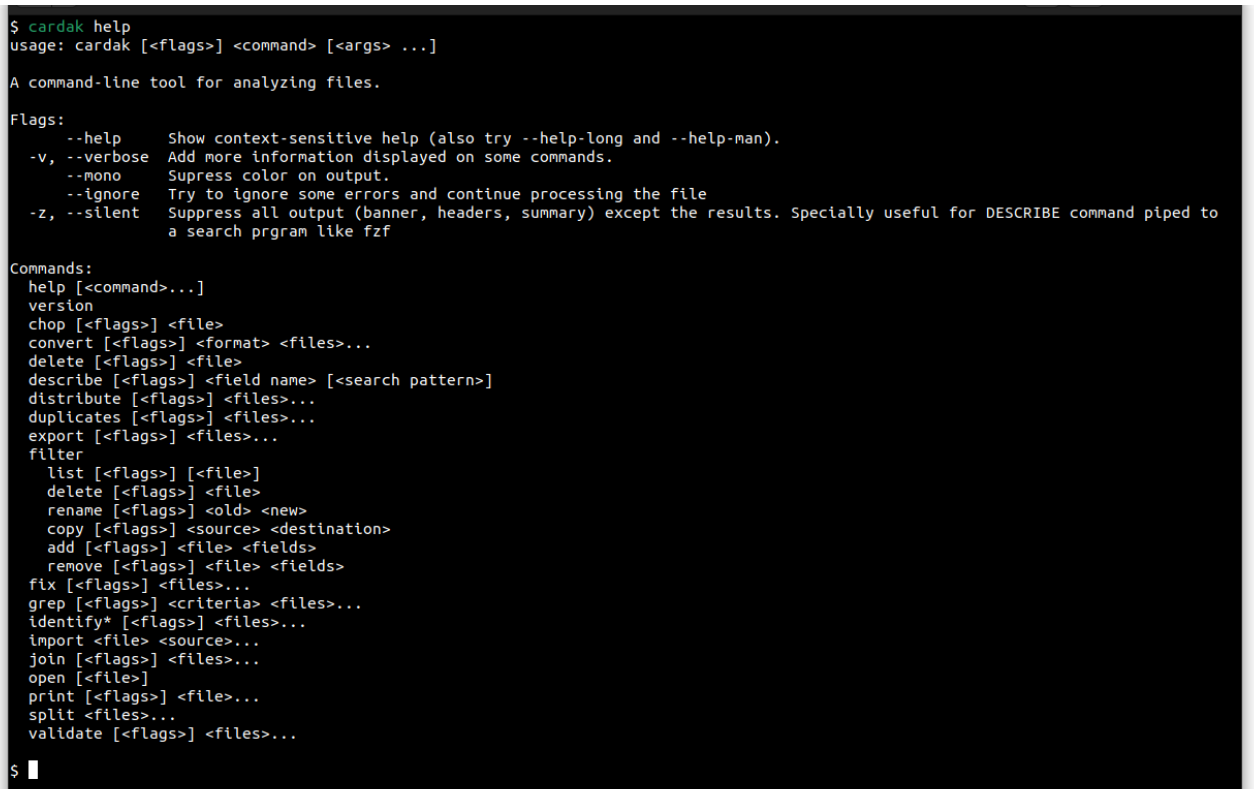
```
$ cardak file14 -a --compact
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================

IPM FILES ------------------------------------------------------------------
file14      34,316 bytes EBCDIC      RDW           NORMAL     MC IPM file 2021-10-13
                 Records:      61  Logical files: 1  (61)

Job ended
Elapsed time  : 41.590677ms
$ cardak import file14 file9-EXP.ckh
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================
file14      34,316 bytes EBCDIC      RDW           NORMAL     MC IPM file 2021-10-13
Loaded 61 records from file14

Import from file9-EXP.ckh
             6 records added

Generated file: file14-IMP.ipm
Records Added : 6
Total records : 67
Bytes written : 38,062

Job ended
Elapsed time  : 17.870269ms
$ cardak file14* -a
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-21T22:47:38Z  main 1565cba
================================================================================

IPM FILES ------------------------------------------------------------------
file14              34,316 bytes EBCDIC      RDW           NORMAL     MC IPM file 2021-10-13
                           Records:      61  Logical files: 1  (61)
file14-IMP.ipm      38,062 bytes EBCDIC      RDW           NORMAL     MC IPM file 2021-10-13
                           Records:      67  Logical files: 1  (67)

2 files using 71 kB

Job ended
Elapsed time  : 60.233875ms
$
```

We see a new file is created (file14-IMP.ipm) containing 67 records, the original 61 plus the exported 6 records.

Commands

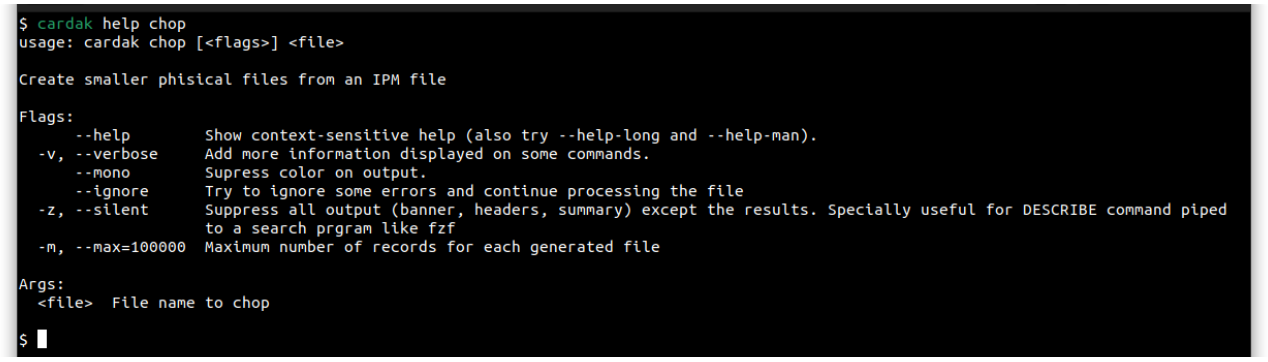
Here is a detailed description of usage and parameters that can be used for each command

To list all the available commands, we can use the **HELP** command that shows the following:



To get more information on a particular command, we can add the command name, by using the syntax “cardak help <command>”

CHOP



This command separates big files into smaller ones. It is useful to easily handle very big, in particular with interactive commands like OPEN, where memory consumption increases with the number of records.

For example, a file containing 50.000 records needs a little more than 1 Gb RAM, while a file containing 100.000 records need around 2.2 Gb RAM, so opening bigger files can be a little problematic. With this command, we can split big files into smaller ones.

Most of the commands work by processing records as they are read from disk, so the memory consumption is limited and independent of the number of records, there are a few that need to be loaded at once in memory (like the **OPEN** or **EXPORT** to **CSV** commands).

This commands reads the contents of a file, performs a fast, heuristic estimate of the number of records (to have the exact number of variable length records it would need to read the whole file). Then it reads the records, and every time it finds a trailer record (end of a logical file), or when we get the maximum number of records for the output file, it proceeds to generate the file and start another one. Each of the new generated files has as name, the original file name plus a sequential number..

We can include the flag --max (-m) to indicate the maximum number of records we want for each file. It this value is not specified, it defaults to 100.000 records (which will consume approximately 2Gb of RAM) which is considered acceptable.

CONVERT

```
$ cardak help convert
usage: cardak convert [<flags>] <format> <files>...

Convert files between formats.

Files have three characteristics:

  Encoding         : Which characters each byte value represents (ASCII or EBCDIC are supported)
  Record definition: Records can be of fixed length, delimited, or determined by a length
  File packing     : Then can have no packing (NORMAL) or can be packed in blocks (typically 1014)

This tool allow to convert files changing those formats

Flags:
      --help                 Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose              Add more information displayed on some commands.
      --mono                 Supress color on output.
      --ignore               Try to ignore some errors and continue processing the file
  -z, --silent               Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE
                             command piped to a search prgram like fzf
  -T, --file-type=FILE-TYPE  Filter by file type when supplying several files. File types are represented by a single letter as:
                             I-IPM files, M-MPE files

Args:
  <format>  New file format The format consists of three consecutive characters that indicate the Encoding, the record format,
            and the file format. Valid values are:

              Encoding      : [E] - EBCDIC, [A] - ASCII
              Record format : [R] - RDW, [F] - Fixed 1400, [1] - Fixed 1400, [6] - Fixed 600, [D] - Delimited
              File format   : [B] - Block 1014, [N] - Nomrmal
  <files>   List of files. This can be a single file or you can use wildcards

$
```

This command is used to convert between file formats.

Files, and in particular those used by **MasterCard**, have three main properties which are the encoding, the record format and the file format.

The supported encodings are **ASCII** and **EBCDIC**

The record format can be of variable, fixed or delimited formats.

IPM and **MPE** files handled by **MasterCard** contain variable length records. Files handled by other brands are usually of fixed length (typically 600 or 1400 characters), and text files are usually delimited (on Windows by the bytes **0x0A** y **0x0D**) and in Unix and similar, by the character **0x0D**

Besides, **MasterCard** usually formats the contents of their files by using blocks of 1014 bytes, so the file content is divided in 1014 bytes blocks (so its size is a multiple of 1014), and on each block, the data is held in the first 1012 bytes, adding two bytes at the end with the value **0x40**. In turn, the last block is padded with bytes with value **0x40** untl it has the size of 1014

Usually, systems that generate or process **MasterCard** files are configured to handle just one combination of these formats, so it is important to coordinate with the brand the format to be used. On early stages of development, texting and certification, we can find files that are not in the expected format. Changing the configuration for each case can be very difficult or almost impossible.

This tool allows to convert the format of a file between different combinations of these three components, regardless of the input format.

The format is specified by using three consecutive letters, where the first one corresponds to the encoding, the sencond one to the record format, and the last one to the file format.

For the encoding, we can use the letter A for **ASCII**, and the letter E for **EBCDIC** (the two options for the interchange files)

For the record format, we can use the letter R for variable length records (or **RDW**), the letters F or 1 for fixed length records of 1400 characters, 6 for fixed length records of 600 characters, or D for delimited files (the delimiters to be used depends if we are using the tool under Windows or Linux)

Lastly, for the file format, we can use the letter B if we want the file to be formatted with blocks of 1014, or the letter N for a normal format (without blocks).

The first parameter must be the three letters that indicate the format, and then the file(s) to convert.

The converted files is named using the original file name, a dot, the three letters indicating the destination format, and the string “.cvt”

This command can be used with any file type, but **IPM** are treated in a special way, as they may contain binary data and those should not be converted, otherwise we would be changing the binary values. So, the tool, in those cases, performs the conversion on a record by record basis, considering the field types.

This command is optimized and processes the stream of records, so it can handle huge files while using a fixed amount of memory (of course the speed will depend on the number of records present in the file)

In case we supply a list of files by using a wildcard, we can add the flag -T to indicate the file type to process and ignore the rest. In order to indicate that we want to to convert **IPM** files, we use the letter I, and for **MPE** files, the letter M**.**

CREATE

```
$ cardak help create
usage: cardak create [<flags>] <file>

Create a new IPM file

Flags:
      --help                 Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose              Add more information displayed on some commands.
      --mono                 Supress color on output.
      --ignore               Try to ignore some errors and continue processing the file
  -W, --width                Ignore small terminal width check and force execution
  -z, --silent               Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE
                             command piped to a search utility like fzf
      --import=IMPORT        File containing the records to be imported
      --pds105=PDS105        String representing the value of PDS 0105
      --file-type=FILE-TYPE  File Type to be used in the header
      --file-date=FILE-DATE  File date, defaults to today
      --processor-id=PROCESSOR-ID
                             The processor ID
      --sequence-number=SEQUENCE-NUMBER
                             The sequence number
      --processing-mode=PROCESSING-MODE
                             The processing mode, T or P
      --encoding=ENCODING    Encoding for new file, default ASCII
      --block                If present, the file will be in block 1014

Args:
  <file>  IPM file name to create

$
```

This command lets us create a new **IPM** file, and optionally load it with records previously exported with the **EXPORT** command.

To create the new file, we need to supply the File ID value and the File Processing Mode, which are fields that are part of the file header. We can specify the complete value of the **PDS 0105** field (the 25 characters of the File ID), or the different sub-fields of the mentioned PDS.

If we decide to indicate the sub-fields, we need at least indicate the SF 01(File Type) and the SF 03 (Processor ID). If we do not supply the SF 02 (File Reference Date), the current date will be used, and for SF 04 (File Sequence Number) we will take the value 1000

If no Processing Mode is supplied, by default it will use the value “T” (Test)

Other parameters that we can indicate are the encoding (**ASCII** by default) and the file format of Block 1014 or normal (normal by default)

If we don’t indicate the usage of an export, the generated file will be empty, just containing the File Header and File Trailer.

In case we use a file with previously exported records, it can be a file containing complete **CSV** records (no missing fields), or an export in **HEX** format (.ckh extension)

DELETE

```
$ cardak help delete
usage: cardak delete [<flags>] <file>

Delete records and/or fields from the IPM file

Flags:
      --help              Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose           Add more information displayed on some commands.
      --mono              Supress color on output.
      --ignore            Try to ignore some errors and continue processing the file
  -W, --width             Ignore small terminal width check and force execution
  -z, --silent            Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE
                          command piped to a search utility like fzf
  -R, --records=RECORDS   List of record numbers to be considered for deletion. Values are separated by comma (,) and ranges are
                          indicated by the starting and ending record separated by a hyphen (-)
  -F, --fields=FIELDS     List of IPM fields to be deleted (can use a filter name)
  -l, --last              Use the record numbers returned on the last GREP command
  -O, --omitdel           Do not create a file with deleted records
  -x, --export            Export the deleted records to a .ckx file (in Hex format)

Args:
  <file>  IPM file name to delete the records from

$
```

This command is used to delete records or fields from an **IPM** file.

The original file is not modified at all, and by default two new files are generated, one containing the deleted records, and another one with the remaining records. The file containing the deleted records is named after the original file name plus the string "**_DISCARD**", and the other file is named after the original file name plus the string "**_KEEP**".

If, what is deleted are just fields and not complete records, the resulting file will be named after the original file name, plus the string "**_DISCARDEDFIELDS**".

In all cases, the extension ".ipm" is added

The required parameter is the file name containing the records or fields to be deleted, and to indicate the records to be deleted or containing the fields to be removed, we use the flag -R (please refer to the section Flags and Filters for more information on the usage of this flag.

As an example, we will delete the records numbers 5 and the records 10 to 15 from the file file10:

```
$ cardak delete file10 -R 5,10-15
================================================================================================================================
Card Army Knife version 0.9.9413 built on 2023-08-30T18:23:12Z  delete 4c92352
================================================================================================================================
file10      21,254 bytes EBCDIC      RDW            NORMAL     MC IPM file 2020-01-27
Loaded 20,530 bytes and 43 records from file10

Generated files:
        9 records using        3,442 bytes to file10_DISCARD.ipm
       36 records using       17,286 bytes to file10_KEEP.ipm

Job ended
Elapsed time  : 18.618694ms
$
```

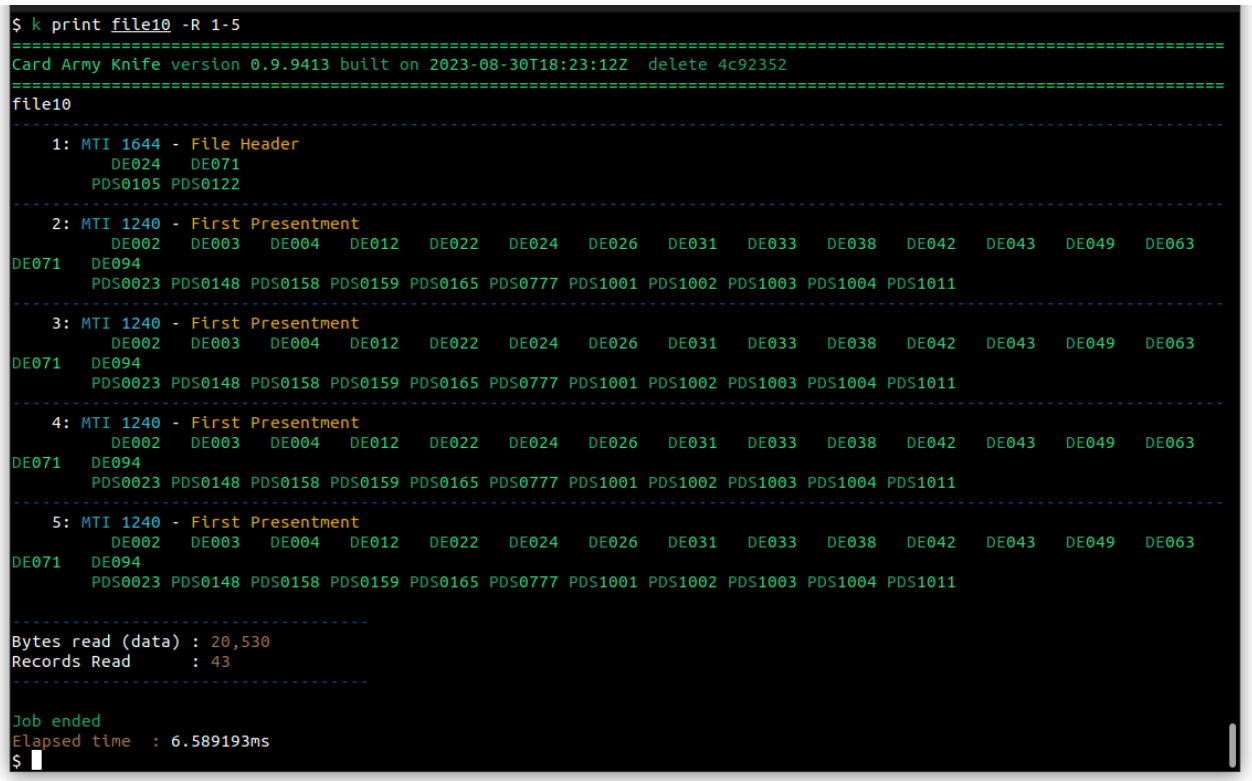
From the original 43 records, a new file is created with 9 records (the selected 7 records plus the new header and trailer), and another file with 36 records(the remaining 34 plus header and trailer)

In order for the files to be able to be processed correctly by other systems, the value of field PDS0105 SF04 (File Sequence Number) of the file header in the **_KEEP** file will contain the the value of the original file plus one, and the **_DISCARD** will be two values above.

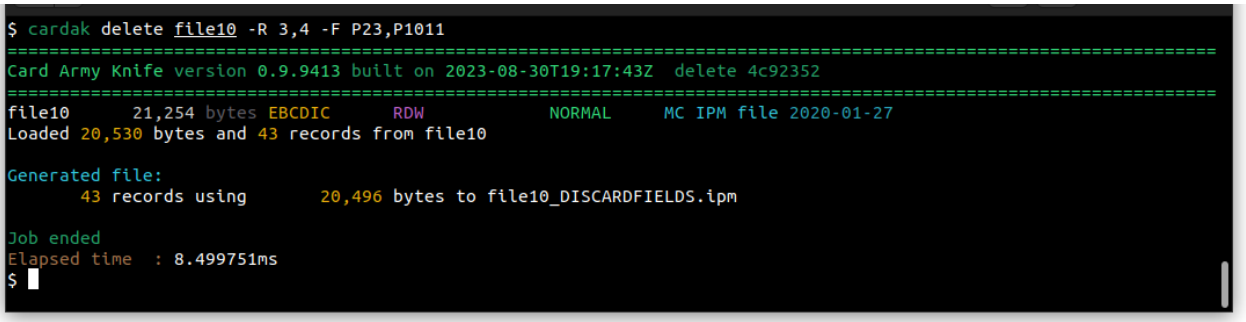
If we don't want to generate the **_DISCARD** file, we can use the flag --omitdel (-O), and using the flag --export (-x), we will generate a file with extension .ckx containing the deleted records in **HEX** format (that can be later be imported into a file)

If we just want to delete some fields but not complete records, we can use the flag -F to indicate the list of fields to delete. If no records are specified, the fields will be deleted from all records, otherwise, the fields will be deleted just on the indicated records (by using the flag -R).

For example, let's look at the first 5 records:



We now delete the fields PDS0023 and PDS1011 from records 3 and 4 as follows:



If we now print again the first 5 records, we will see that records 3 and 4 no longer contain the deleted fields.

It is important to note that no checks are performed in the fields to be deleted, so it is possible to eliminate fields that are mandatory, resulting in a file with errors.

```
$ cardak print file10_DISCARDFIELDS.ipm -R 1-5
================================================================================================================================
Card Army Knife version 0.9.9413 built on 2023-08-30T19:17:43Z  delete 4c92352
================================================================================================================================
file10_DISCARDFIELDS.ipm
--------------------------------------------------------------------------------------------------------------------------------
    1: MTI 1644 - File Header
          DE024   DE071
        PDS0105 PDS0122
--------------------------------------------------------------------------------------------------------------------------------
    2: MTI 1240 - First Presentment
          DE002   DE003   DE004   DE012   DE022   DE024   DE026   DE031   DE033   DE038   DE042   DE043   DE049   DE
063   DE071   DE094
        PDS0023 PDS0148 PDS0158 PDS0159 PDS0165 PDS0777 PDS1001 PDS1002 PDS1003 PDS1004 PDS1011
--------------------------------------------------------------------------------------------------------------------------------
    3: MTI 1240 - First Presentment
          DE002   DE003   DE004   DE012   DE022   DE024   DE026   DE031   DE033   DE038   DE042   DE043   DE049   DE
063   DE071   DE094
        PDS0148 PDS0158 PDS0159 PDS0165 PDS0777 PDS1001 PDS1002 PDS1003 PDS1004
--------------------------------------------------------------------------------------------------------------------------------
    4: MTI 1240 - First Presentment
          DE002   DE003   DE004   DE012   DE022   DE024   DE026   DE031   DE033   DE038   DE042   DE043   DE049   DE
063   DE071   DE094
        PDS0148 PDS0158 PDS0159 PDS0165 PDS0777 PDS1001 PDS1002 PDS1003 PDS1004
--------------------------------------------------------------------------------------------------------------------------------
    5: MTI 1240 - First Presentment
          DE002   DE003   DE004   DE012   DE022   DE024   DE026   DE031   DE033   DE038   DE042   DE043   DE049   DE
063   DE071   DE094
        PDS0023 PDS0148 PDS0158 PDS0159 PDS0165 PDS0777 PDS1001 PDS1002 PDS1003 PDS1004 PDS1011

-----------------------------------
Bytes read (data) : 20,496
Records Read      : 43
-----------------------------------

Job ended
Elapsed time  : 7.968503ms
$
```

We can always perform a validation on the generated file:

```
$ cardak validate file10_DISCARDFIELDS.ipm -v
================================================================================================================================
Card Army Knife version 0.9.9413 built on 2023-08-30T19:17:43Z  delete 4c92352
================================================================================================================================
PASS file10_DISCARDFIELDS.ipm

Job ended
Elapsed time  : 10.40563ms
$
```

We can also generate a **HEX** file containing the deleted records:

```
$ cardak delete file10 -R 2-10 -x
================================================================================================================================
Card Army Knife version 0.9.9413 built on 2023-08-30T21:43:04Z  delete 4c92352
================================================================================================================================
file10      21,254 bytes EBCDIC      RDW           NORMAL     MC IPM file 2020-01-27
Loaded 20,530 bytes and 43 records from file10

Generated files:
       34 records using       16,423 bytes to file10_KEEP.ipm
       11 records using        4,305 bytes to file10_DISCARD.ipm
        9 records as HEX lines to file10_DISCARD.ckx

Job ended
Elapsed time  : 8.846529ms
$
```

DESCRIBE

```
$ cardak help describe
usage: cardak describe [<flags>] <field name> [<search pattern>]

Describe IPM fields and functions.

It accepts the following:

  MT[I]              - Show MTI values
  TRAN[SACTIONTYPES] - Show valid transaction types
  FUNC[TIONCODES]    - Show valid function codes
  D[E]n              - Data Element n (can use wildcards)
  P[DS]n             - PDS element n (can use wildcards)

Flags:
      --help       Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose    Add more information displayed on some commands.
      --mono       Supress color on output.
      --ignore     Try to ignore some errors and continue processing the file
  -z, --silent     Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE command piped
                   to a search prgram like fzf
  -x, --extended   Display extended information
  -s, --subfields  Show subfields if they exist

Args:
  <field name>        Field name to describe (can use wildcards)
  [<search pattern>]  Pattern used as filter for results

$
```

This command does not work on files, it is a quick helper to find information about fields and values that can be used instead of reading the manuals.

It can be helpful when we need to use a filter on other commands, but we don't remember which field to use.

This command is very flexible on how to search, allowing the usage of wildcards or by searching by part of the description of the field.

We can look for fields like Data Elements (**DE**), Private Data Sub-elements (**PDS**), **MTI**, Function Codes and Transaction Codes.

When searching for fields, we can use the same pattern used for almost every other part of the tool (form more details plese refer to section Flags and Filters). We can also use wildcards if we want more than one field in the result, and we can optionally add a text to filter the results if the description contains that text. The information shown is similar to the one on the brand manuals

For the Function and Transaction codes, we are presented with the values and descriptions of the different valid combinations.

By default, only a short description is displayed, but we can use the flag --subfields (-s) to show also the sub-fields (if they exist), or the flag --extended (-x) to show extended information (if possible).

As an example, we can take the field **DE43**. If we are not sure what is it about, we can just ask for a description:

```
$ cardak describe D43
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================
DE ------------------------------------------------------------------------------
    DE043: Card Acceptor Name/Location
           01 Card Acceptor Name
           02 Card Acceptor Street Address
           03 Card Acceptor City
           04 Card Acceptor Postal (ZIP) Code
           05 Card Acceptor State, Province, or Region Code
           06 Card Acceptor Country Code

              DE 43 (Card Acceptor Name/Location) contains the card acceptor's name and
              location as known to the cardholder


Job ended
Elapsed time  : 2.666349ms
$
```

Or, let's say we want to search for an amount but we don't remember the field where this value is used. We can search by description on all **DE** fields:

```
$ cardak describe 'D*' amount
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================
DE --------------------------------------------------------------------------
    DE004: Amount, Transaction
    DE005: Amount, Reconciliation
    DE006: Amount, Cardholder Billing
    DE030: Amounts, Original
    DE054: Amounts, Additional
    DE111: Amount, Currency Conversion Assessment

Job ended
Elapsed time  : 4.701286ms
$
```

In the previous example we used a wildcard, asking for all Data Elements by using ‘D*’. It is necessary to surround this with quotes, otherwise the shell will try to expand that as a file (in Linux environments, it is called globbing)

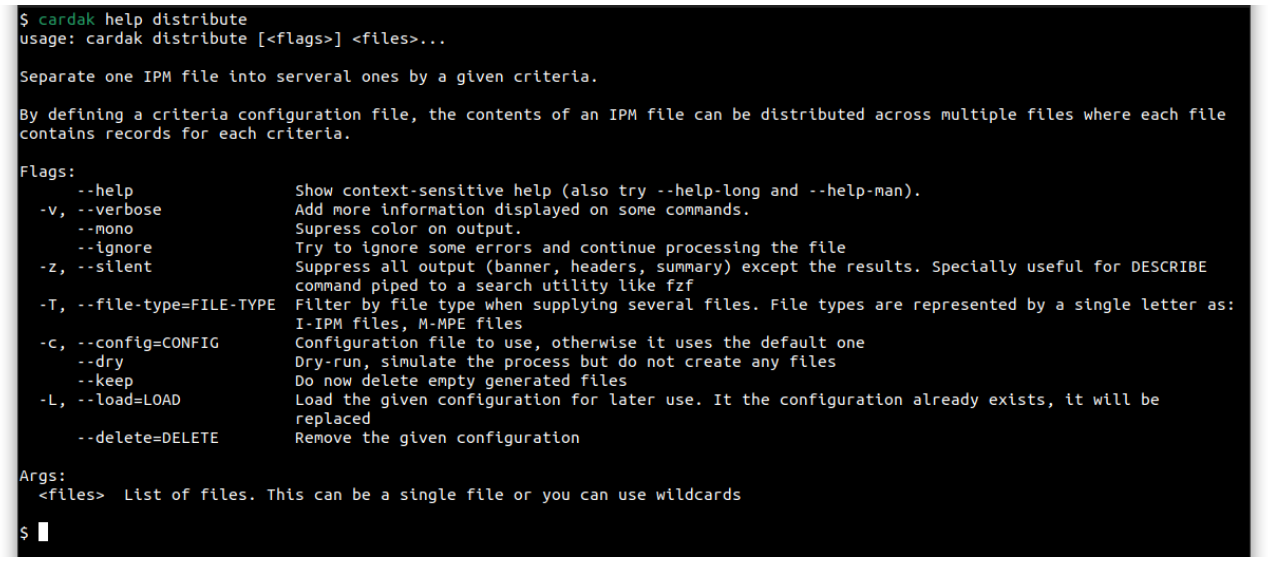
For example, let’s see the Function codes:

```
$ cardak describe func
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================

Function Codes ----------------------------------------------------------------
MTI 1240 - Presentment
           DE24(200) : First Presentment
           DE24(205) : Second Presentmnent (Full)
           DE24(282) : Second Presentment (Partial)
MTI 1442 - Chargeback
           DE24(450) : First Chargeback (Full)
           DE24(451) : Arbitration Chargeback (Full)
           DE24(453) : First Chargeback (Partial)
           DE24(454) : Arbitration Chargeback (Partial)
MTI 1644 - Administrative
           DE24(603) : Retrieval Request
           DE24(605) : Retrieval Request Acknowledgement
           DE24(640) : Currency Update
           DE24(680) : File Currency Summary
           DE24(685) : Financial Position Detail
           DE24(688) : Settlement Position Detail
           DE24(691) : Message Exception
           DE24(693) : Text Message
           DE24(695) : File Trailer
           DE24(696) : Financial Detail Addendum
           DE24(697) : File Header
           DE24(699) : File Reject
MTI 1740 - Fee Collection
           DE24(700) : Fee Collection (Customer-generated)
           DE24(780) : Fee Collection Return
           DE24(781) : Fee Collection Resubmission
           DE24(782) : Fee Collection Arbitration Return
           DE24(783) : Fee Collection (Clearing System-generated)
           DE24(790) : Fee Collection (Funds Transfer)
           DE24(791) : Fee Collection (Funds Transfer Backout)

Job ended
Elapsed time  : 9.542244ms
$
```

Here, we see for each **MTI**, the values that can be used on field **DE24** and their meaning.

This saves us from searching in the manuals the possible values for each type of transaction, and it is useful when searching in files if we don’t remember the values we need.

DISTRIBUTE



This command can clasify and distribute records from an **IPM** file and put them in different files depending on defined rules. A typical use is when we receive a file from the brand that contains transactions from different entities, and we need to send each processor only their transactions. Or, maybe, we just want to have a file containing just presentments and another one containing only Fee Collections, or maybe classify them by currency, or any criteria we can imagine.

To execute this command, we need to have extra configuration files where we define the criteria to use for clasification.

We can see the existing configuration files by executing “cardak distribute list”

These configuration files can be located in two different locations. They can be present in the local folder (the one we are when issuing the command), or they can be located in the “HOME” fonder. We can automatically copy a configuration file to the “HOME” location using the flag --load without needing to know the real location, as we will see later.

The configuration files are in **JSON** format, and the template is like the following:

```
{
    "<ID>": {
      "FileName": "File1",
      “Directory”: “/usr/dir/”,
      "Description": ""
      "ProcessorID": "123456789",
      "FileSequence": 1,
      "BinRanges": [
        {
          "BinMin": 0,
          "BinMax": 9999
        }
      ],
      "MtiList": {
        "1240": true
        "1442": true
        "1644": true
        "1740": true
      },
      "FieldValues": "DE49:858"
    }
}
```

Let’s explain its contents in detail:

In the first level of the **JSON**, we have entries where each of them corresponds to one file we want to generate. Each of these entries need a name (that is the “<ID>” we see that is to be replaced with any descriptive name as we will see later).

For each entry, we can define the generic name of the file(“FileName”), an optional description (“Description”), and a value for “ProcessorID” and “FileSequence” that will be used to create the file header. This is needed because all records that meet the conditions will be written to a new file, and in the header, to populate field **PDS0105**, we need some data. The **SF01** (File Type) is copied from the original file, and so is field **SF02** (File Reference Date).

The generated files will have as a name, the original file name (without the extension), an underscore and the string indicated in the entry “FileName”, and with the extension “.ipm”

If we include a directory, the file will be generated in that location. If the directory does not exist, it will be created, but will fail if, for example, the user does not have enough permissions to create it..

Fields **SF03** (Processor ID) and **SF04** (File Sequence Number) are the indicated.

Following those entries, we specify the conditions that a record must meet to be included in this file. The conditions can be some BIN ranges, **MTI**, or a value present in any field. Those conditions are optional, but we must have at least one of them.

The BIN ranges consist in a list of pairs represinting the Minimum and Maximum values of the range. If the value of the **BIN** in field **DE002** is contained in any of these ranges, it is considered a match. Anyway, the record must match ALL the conditions to be included in the file. The **BIN** length is not fixed, it is calculated depending on the length needed to be compared in the range (if, for example, we define max to be 9999, only the four first digits of the **PAN** will be considered).

We can also indicate which values of **MTI** we want to consider.

Lastly, we can apply the rule “**FieldValues**”, where we put a list of field name and value pairs, using the same rules used for searches using the **GREP** command (that is, a list of pairs of field:value separated by commas or semicolons)

To see the existent configuration, we can use the “list” option like this:

```
$ cardak distribute list
================================================================================
Card Army Knife version 0.8.3 built on 2023-08-19T14:08:51Z  main ad92f3f
================================================================================

Config files in HOME directory

bybin
default

Job ended
Elapsed time  : 2.232883ms
$
```

We see we have two defined configurations. The one named “default” is always present and is the one to be used if we don’t specify anyone else.

To see the contents of the configurations, we can use the flag -v or specify the name configuration name we want to see.

```
$ cardak distribute bybin
================================================================================
Card Army Knife version 0.8.3 built on 2023-08-19T14:08:51Z  main ad92f3f
================================================================================
bybin
--------------------------------------------------------------------------------
Using configuration file /home/eduardo/.cardak/bybin_dist.json

BANK_1
  Registros que van al Banco 1
  Processor ID : 123456789
  File sequence: 100
  Bin ranges   :         0 to 100000012
                 100000014 to 999999999

BANK_2
  Registros que van al Banco 2
  Processor ID : 112233445
  File sequence: 200
  Bin ranges   : 100000012 to 100000013

BANK_3
  Registros que van al Banco 3
  Processor ID : 987654321
  File sequence: 300
  Bin ranges   : 100000013 to 100000014

--------------------------------------------------------------------------------

Job ended
Elapsed time  : 1.929097ms
$
```

Let's see an practical and simple example. We want to classify the transactions of a file into two different files, so that transactions in USD go to one file, and transaction in UYP go to another file.

To do this, we will create a file named "money.json" with the following content:

```
$ cat money_dist.json| jq
{
  "1-PESOS-858": {
    "FileName": "pesos",
    "Description": "Registros en Pesos Uruguayos",
    "ProcessorID": "123456789",
    "FieldValues": "D49:858"
  },
  "2-DOLARES-840": {
    "FileName": "dolares",
    "Description": "Registros en Dolares",
    "ProcessorID": "987654321",
    "FieldValues": "D49:840"
  }
}
$
```

Here I indicate I will generate two files, one with name "pesos" and another one with name "dolares".

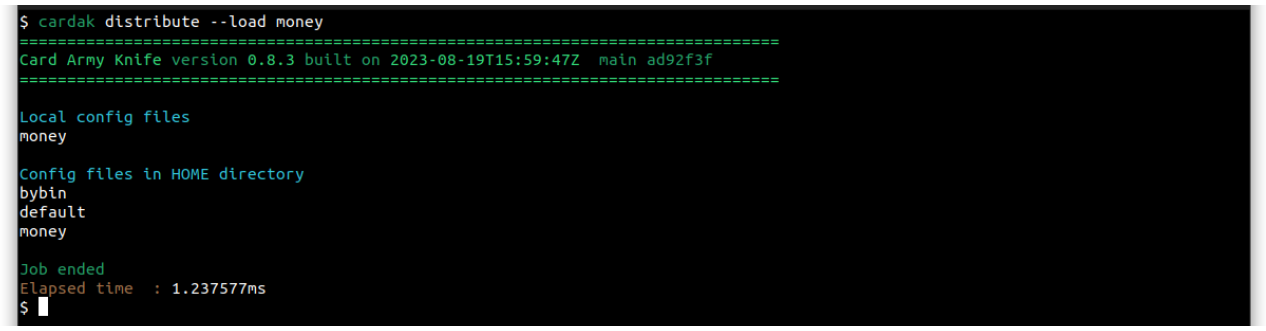
Each new file will have the indicated ProcessorID, but I am not specifying any FileSequence, so we will take the same as the original file

For the conditions, we will not use the bin ranges or **MTI**, but I will want the file with UYP to match **DE049** with value **858**, and for USD with value **840**

So, let's see the available configurations:

```
$ cardak distribute list
================================================================================
Card Army Knife version 0.8.3 built on 2023-08-19T14:08:51Z  main ad92f3f
================================================================================

Local config files

money
Config files in HOME directory

bybin
default

Job ended
Elapsed time  : 2.616722ms
$
```

It is not practical to leave the configuration in the local folder, so we will load it into the applications HOME directory by using the flag --load

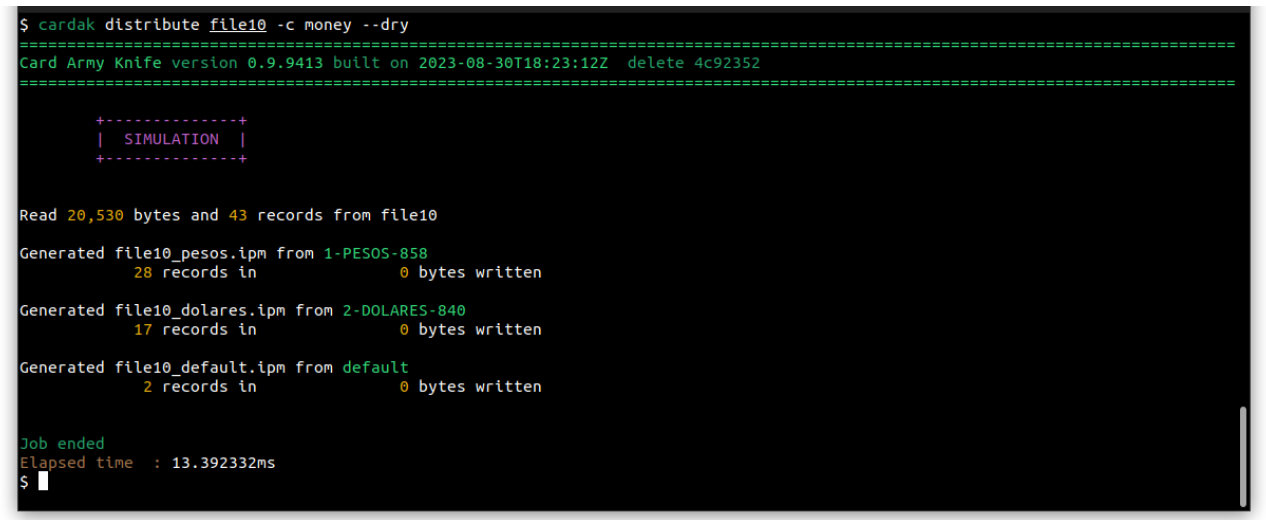


Now we can just delete the local file “money_dist.json”, but the configuration will be available from its copy in the HOME directory. If the same configuration name is present in the HOME and the local folders, the local one will be used, so we can easily override it at any time.

To use this command we must supply the name of the original file, and optionally the name of the configuration to apply.

If we don’t put any configuration name, the tool will try to find one named “default”. This is handy if we have one configuration that will be used most of the time.

In the example, we will take the contents of the file file10. But first, we will simulate the process by using the flag --dry (this step is optional, we just use it for demonstration and tests)



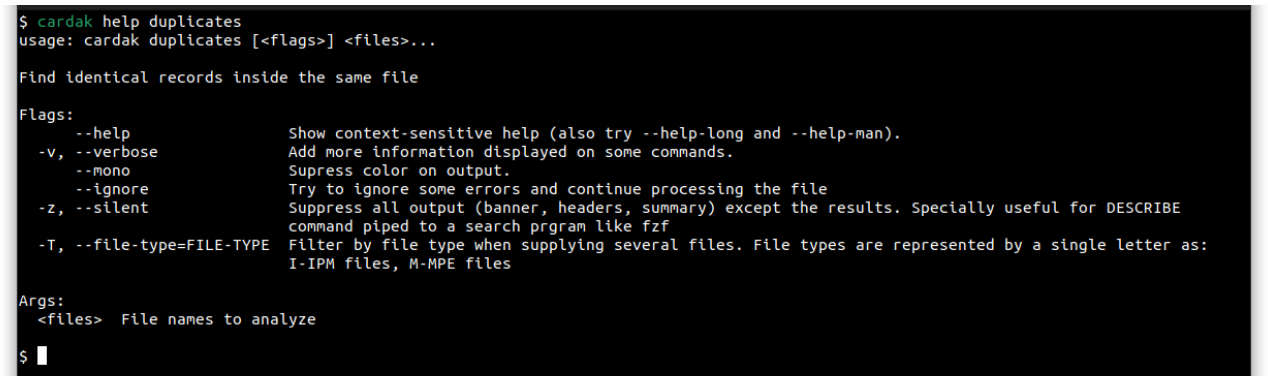
The tool will read all 43 records from file10,and it will create a file with 28 records in UYP, and another one with 17 records in USD.

We see a third file named also “default”. This file will receive all records that don’t match any criteria. In this case, as all records match, this “default” file would be empty (just with a header and trailer), and when executing the command without the --dry option, it will not be created, unless we force it with the flag --keep

If we want to physically generate de files, we just omit the flag --dry

```
$ ls -l
total 24
-rw-rw-r-- 1 eduardo eduardo 21254 ago 30 15:30 file10
$ cardak distribute file10 -c money
================================================================================================================================
Card Army Knife version 0.9.9413 built on 2023-08-30T18:23:12Z  delete 4c92352
================================================================================================================================

Read 20,530 bytes and 43 records from file10

Generated file10_pesos.ipm from 1-PESOS-858
           28 records in          12,802 bytes written

Generated file10_dolares.ipm from 2-DOLARES-840
           17 records in           7,926 bytes written


Job ended
Elapsed time  : 8.010495ms
$ ls -l
total 48
-rw-rw-r-- 1 eduardo eduardo 21254 ago 30 15:30 file10
-rw-rw-r-- 1 eduardo eduardo  7926 ago 30 15:31 file10_dolares.ipm
-rw-rw-r-- 1 eduardo eduardo 12802 ago 30 15:31 file10_pesos.ipm
$
```

DUPLICATES



This command tries to find duplicate records in **IPM** files.

Two records are considered duplicated when they contain the same information in all fields except **DE71** (the record number, as it probably has a different value on each record)

EXPORT

```
$ cardak help export
usage: cardak export [<flags>] <files>...

Exports the contents of the file as CSV (comma separated values) or HEX records

It can write the values to a file or display them on the console

Flags:
      --help                 Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose              Add more information displayed on some commands.
      --mono                 Supress color on output.
      --ignore               Try to ignore some errors and continue processing the file
  -z, --silent               Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE
                             command piped to a search prgram like fzf
  -T, --file-type=FILE-TYPE  Filter by file type when supplying several files. File types are represented by a single letter as:
                             I-IPM files, M-MPE files
  -R, --records=RECORDS      List of record numbers to be exported. Values are separated by comma (,) and ranges are indicated by
                             the starting and ending record separated by a hyphen (-)
  -F, --fields=FIELDS        List of IPM fields to be exported (can use a filter name)
      --console              Do now wirite the file and display output on console
  -C, --code=CODE            Filter by Function Code description
  -x, --hex                  Export a .ckh file with records in HEX format
  -l, --last                 Use the record numbers returned on the last GREP command

Args:
  <files>  File names to export

$
```

This command is used to extract full or partial records from **IPM** files, and store them in **CSV** or **HEX** files, that can be used with external programs or to be imported into other **IPM** files.

By default, if we don't specify any other option, a **CSV** file is created with the same number as the original, and adding "-EXP.csv" to the name.

This file can also be opened directly with a spreadsheet like Excel, where we can examine the information using the tools provided by these spreadsheets.

Binary fields are exported as their HEX representation.

If we just want to export some records, we can use the flag -R, and if we just want to include some fields, we can use the flag -F (please refer to the section Flags and Filters for more information on how to use them)

We can use the flag --hex (x) to export the records in a **HEX** format file instead of a **CSV**. In this case, full records are exported and the field filter is ignored.

We have the option ot display the exported records on screen without generating any file. For this we justuse the flag --console

We can also use the flag --last to automatically export the records returned by the las **GREP** command we may have run on the file.

FILTER

```
$ cardak help filter
usage: cardak filter <command> [<args> ...]

Manages filter files

Flags:
      --help     Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose  Add more information displayed on some commands.
      --mono     Supress color on output.
      --ignore   Try to ignore some errors and continue processing the file
  -z, --silent   Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE command piped to
                 a search prgram like fzf

Commands:
  filter
    list [<flags>] [<file>]
    delete [<flags>] <file>
    rename [<flags>] <old> <new>
    copy [<flags>] <source> <destination>
    add [<flags>] <file> <fields>
    remove [<flags>] <file> <fields>

$
```

This command is not used on files, but it allows us to maintain what we call filters. Filters are just helper configurations that helps us by defining a list of fields, so instead of manually entering the list of fields when filtering, we can just use the name that contains the list of fields. This command uses sub-commands to operate.

To better understand the concept, we will see an example. Imagine that we frequently want to use fields DE2, DE3, DE4, DE12, DE24, DE25 and DE26, because we always want to show the values of those fields when using other commands.

We could, for example, add this when using the **PRINT** command:

-R D4,D12,D24,D25,D26

This is unpractical, so we can just define a new filter with that list of fields. We give that filter a name (which is just a string of characters that do not contain spaces).

Here is an example. We start using the sub=command LIST to see the defined filters:

```
$ cardak filter list
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================

IPM field filter configurations:

WARNING: No configurations have been found

Job ended
Elapsed time  : 2.315716ms
$
```

As we see, we don’t have any defined filters. Let’s change that by defining a filter named “myfilter” containing the previous list of fields.

```
$ cardak filter add myfilter D4,D12,D24,D25,D26
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================
WARNING: Configuration with name "myfilter" does not exist, a new one will be created

WARNING: Are you sure you want to add 5 fields to "myfilter"? (y/n):
y

Original number of fields : 0
Fields added              : 5
Fields already present    : 0
New number of fields      : 5

New contents of myfilter:

  DE004    Amount, Transaction
  DE012    Date and Time, Local Transaction
  DE024    Function Code
  DE025    Message Reason Code
  DE026    Card Acceptor Business Code (MCC)

Job ended
Elapsed time  : 2.751821294s
$
```

The **FILTER ADD** command will add the supplied field list if the filter already exists, or it will create a new one. We will be prompted and if we confirm, the new filter will be created or updated. It will also show the list of fields contained in the filter, in this case “myfilter”

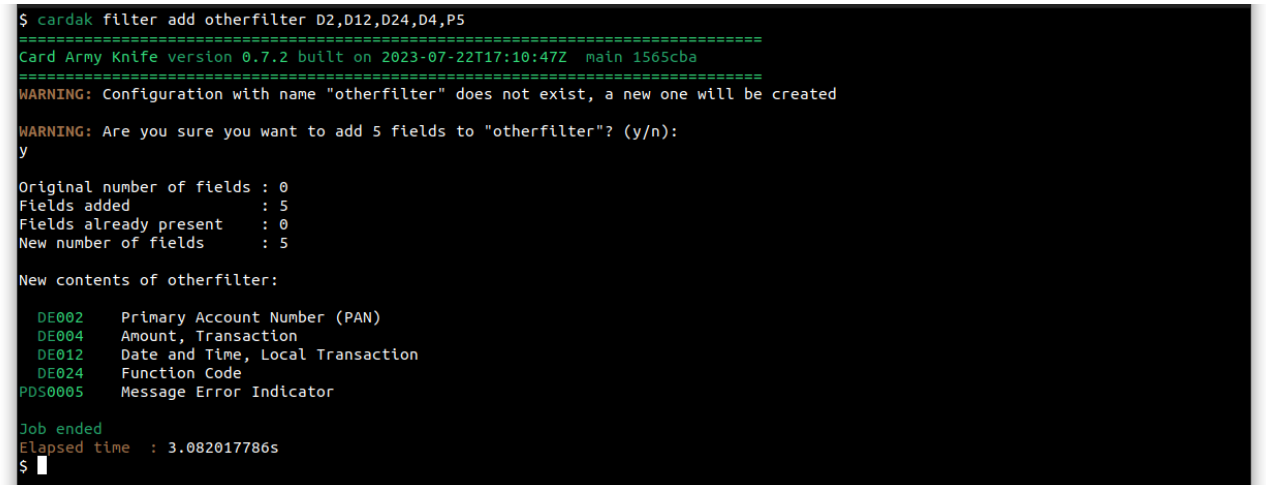
From now on, every time we need to specify the filter so use, we can either use:

-R D4,D12,D24,D25,D26

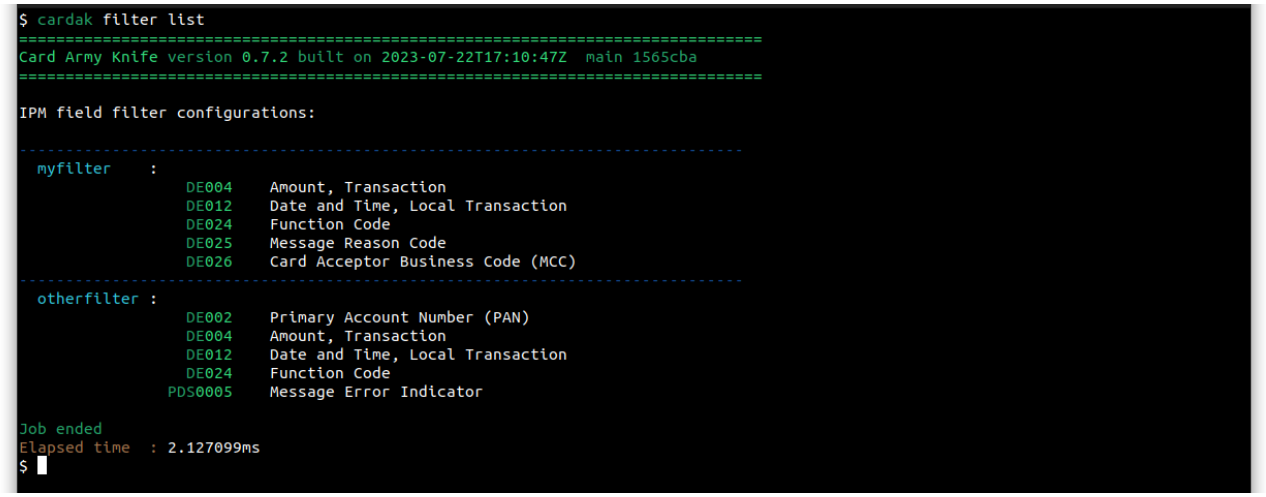
or just the corresponding filter name:

-R myfilter

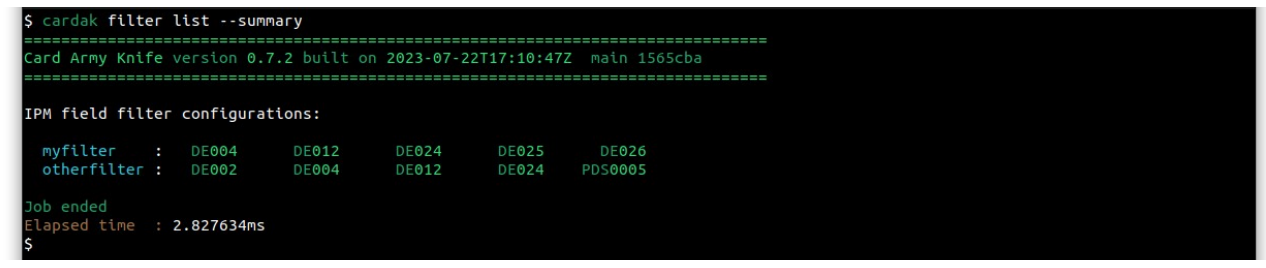
Let’s create another filter with name “otherfilter” but with a different list of fields



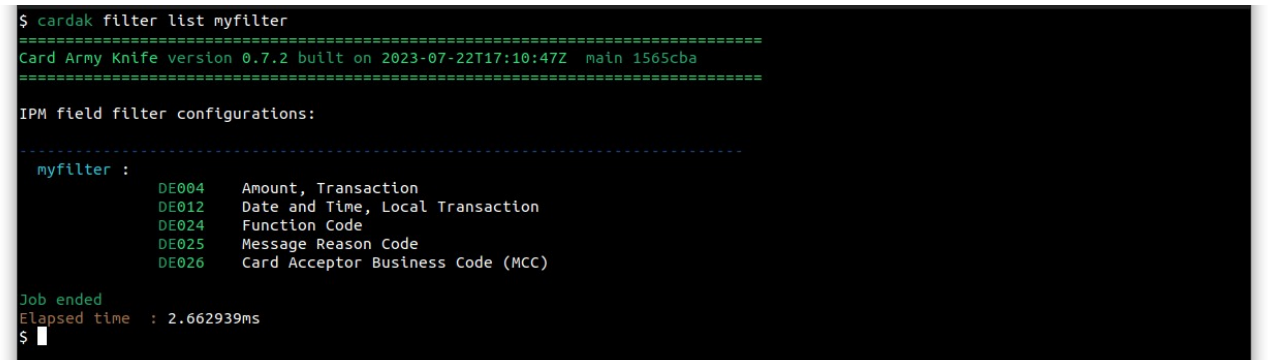
To see the defined filters, we use the **FILTER LIST** command



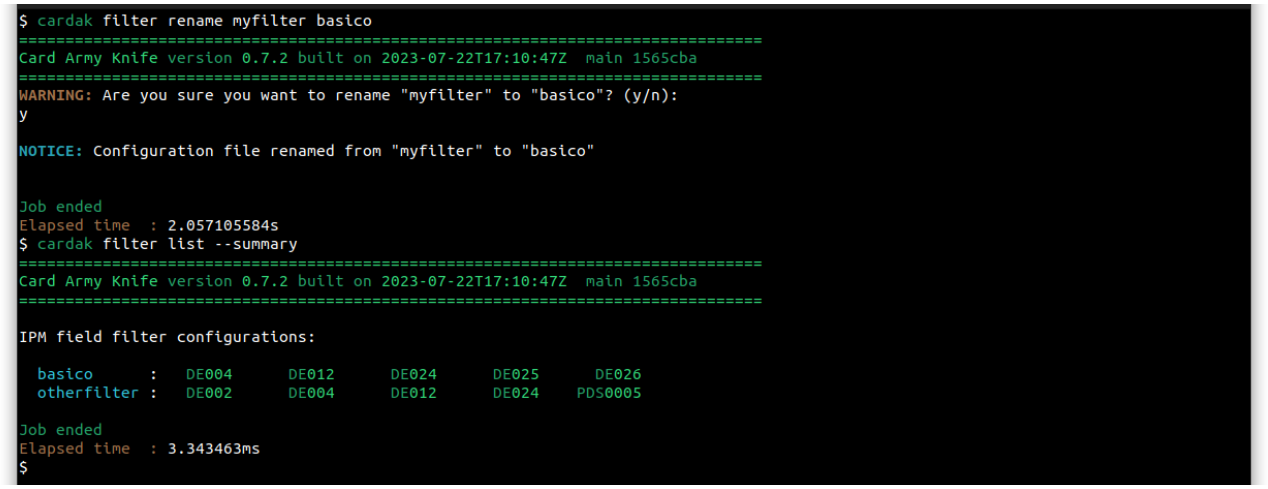
So, we have two defined filters. We can define as many as we want, and if we have many of them, we can list them in a more compact view by using the flag --summary



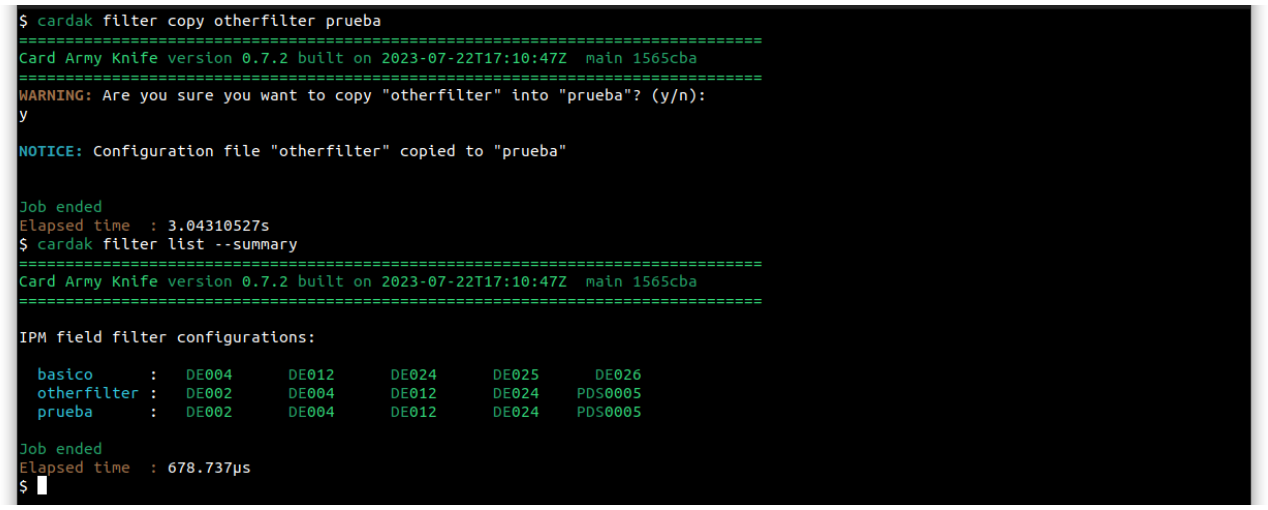
If we want to see one filter in detail, just use its name like this:



We now realize the name of “myfilter” is not helpful as it gives us no clue about it. We would prefer it to be called “basico”. We can rename it like this:



Another option is to create a new filter based on another one. We will take the filter “otherfilter” and create a copy named “prueba”



After having a copy, let’s add some fields to this filter named “prueba”. As a test, we will use some fields already present in the filter.

```
$ cardak filter add prueba D12S2,0005,0149,D22s5,D2,D12S1,D24,P23,p148,d4
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================
WARNING: Are you sure you want to add 10 fields to "prueba"? (y/n):
y

Original number of fields : 5
Fields added              : 6
Fields already present    : 4
New number of fields      : 11

New contents of prueba:

  DE002    Primary Account Number (PAN)
  DE004    Amount, Transaction
  DE012    Date and Time, Local Transaction
  DE012S01 Date
  DE012S02 Time
  DE022S05 Cardholder Present Data
  DE024    Function Code
PDS0005    Message Error Indicator
PDS0023    Terminal Type
PDS0148    Currency Exponents
PDS0149    Currency Codes, Amounts, Original

Job ended
Elapsed time  : 2.602334706s
$
```

We can see that the existing fields are ignored, so we can just add some sub-fields without the need to include their parent field (like adding **DE022S05** without including the parent **DE022**).

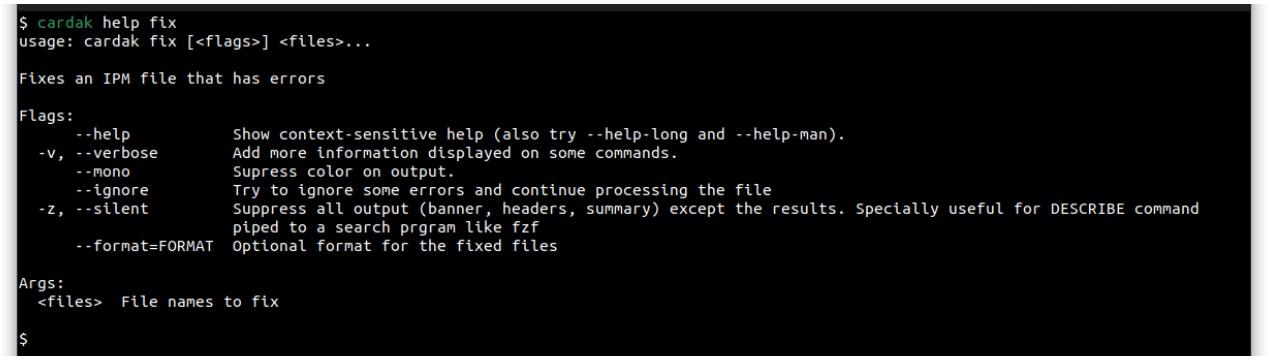
We can also remove fields from a filter. As an example, we will remove field **PDS0023**

```
$ cardak filter remove prueba P23
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================
WARNING: Are you sure you want to remove 1 fields from "prueba"? (y/n):
y

Original number of fields : 11
Fields removed            : 1
Fields ignored            : 0
New number of fields      : 10

New contents of prueba:

  DE002    Primary Account Number (PAN)
  DE004    Amount, Transaction
  DE012    Date and Time, Local Transaction
  DE012S01 Date
  DE012S02 Time
  DE022S05 Cardholder Present Data
  DE024    Function Code
PDS0005    Message Error Indicator
PDS0148    Currency Exponents
PDS0149    Currency Codes, Amounts, Original

Job ended
Elapsed time  : 2.078789417s
$
```

And finally, we can eliminate a filter that we don't need any more.

```
$ cardak filter delete prueba
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================
WARNING: Are you sure you want to delete the file "prueba"? (y/n):
y

NOTICE: Configuration file "prueba" has been deleted


Job ended
Elapsed time  : 1.624941732s
$ cardak filter list --summary
================================================================================
Card Army Knife version 0.7.2 built on 2023-07-22T17:10:47Z  main 1565cba
================================================================================

IPM field filter configurations:

  basico      :   DE004     DE012     DE024     DE025     DE026
  otherfilter :   DE002     DE004     DE012     DE024    PDS0005

Job ended
Elapsed time  : 2.790468ms
$
```

FIX



Some errors in files can be automatically fixed.

Not all errors are fixable, especially when mandatory fields are missing or header records are not present.

For other errors, like wrong record numbers or wrong checksums, an automatic fix can be applied. This command does that, creating a new file with the same name as the original but appending “-FIX.ipm” to it.

The new generated file has the same format as the original one by default, but we have the option to specify a different format for the fixed file by using the flag --format

GREP

```
$ cardak help grep
usage: cardak grep [<flags>] <criteria> <files>...

Find data in files.

It can search for values regardless of the file format, and it has the ability to understand IPM records to specify in what fields
to perform the search

Flags:
      --help                 Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose              Add more information displayed on some commands.
      --mono                 Supress color on output.
      --ignore               Try to ignore some errors and continue processing the file
  -z, --silent               Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE
                             command piped to a search prgram like fzf
  -T, --file-type=FILE-TYPE  Filter by file type when supplying several files. File types are represented by a single letter as:
                             I-IPM files, M-MPE files
  -R, --records=RECORDS      List of record numbers to be printed. Values are separated by comma (,) and ranges are indicated by
                             the starting and ending record separated by a hyphen (-)
  -F, --fields=FIELDS        List of IPM fields to list from a matching record, even if these fields don-t have a match (can use a
                             filter name)
      --summary              Only display file names and the matching count for each one.
      --matches              Only display file names and the list of record numbers that match.
  -C, --code=CODE            Filter by Function Code description

Args:
  <criteria>  Search criteria. This is a list of criteria, elements separated by a comma (,) are ANDed together, while elements
              separated by a semi-colon are ORed together. Each element consists of an optional field descriptor followed by a
              colon (:) and the value to search. Field descriptors consist of an optional letter (D-DE fields, P-PDS fields) and
              the corresponding field number. If the field descriptor consist of only numbers, then a three digit number is taken
              as a DE field, and a four digit number is taken as a PDS field. An example could be: 'DE43:Supermarket' to search for
              records having field DE43 that contain the string "Supermarket"
  <files>     List of files. This can be a single file or you can use wildcards

$
```

This command is used to perform searches in **IPM** files.

The first parameter is the search criteria, and the rest are file names where to perform the search.

This first parameter consists on a list of matching criteria, where each element of the list is either a value to search anywhare in the record, of a field name and the value to be searched in that field. The field name and the value to search are separated by a colon (:). This field name is used in the usual way as defined in the section Flags and Filters

This list of criteria consists of one or more criteria, and the elements of the list can be separated by a comma or a semi-colon. If we use the comma(,) the criteria is applied using the **AND** logic (all of them must match to consider the record matches), and if we use a semi-colon (;) they are applied using a logical **OR**

We can also limit the search in a certain records, by using the flag -R where we can specify individual records or ranges where to perform the search.

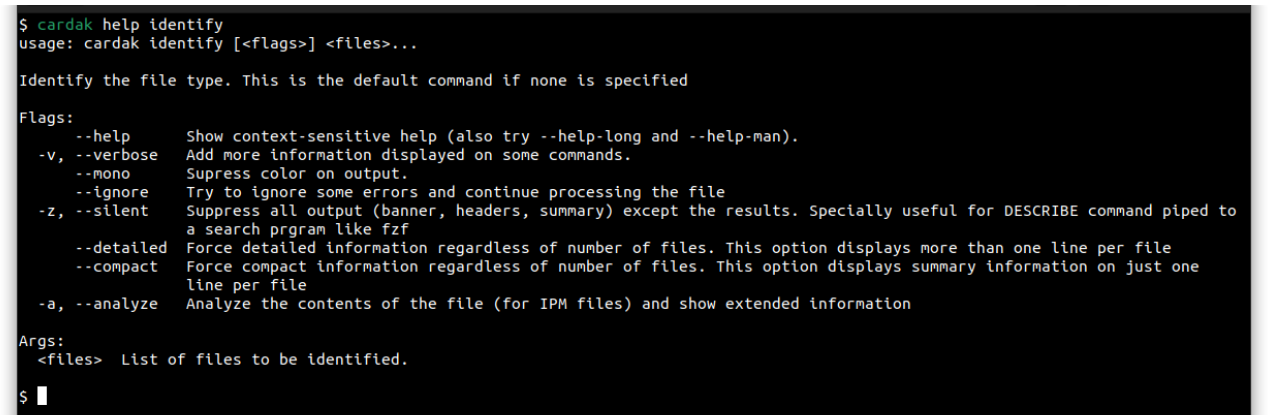
Using the flag -F we can define a list of fields to be displayed in the results and are independent on the fields used in the search criteria. This is handy to display information of the records that are nor part of the searched criteria.

If we use the flag --summary, we will only see the file names and number of coincidences inside each one.

Adding the flag --matches, we will see the name of the file and the list of records numbers that matched.

Another filter we can use to limit the search is by using the flag --code (-C), where we can write text as part of the description of the desired Function Code (Like, for example, “Second Presentment”, or “Partial”

IDENTIFY



This is the command by default when we don't supply any command

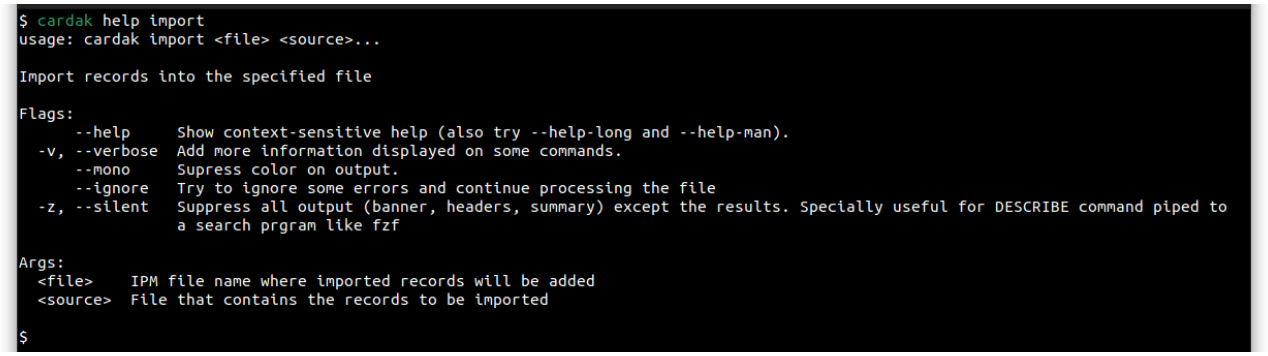
It receives a list of file names and displays information about them, like the size in bytes, the encoding, the record side and if the file is in Block 1014.

If we just supply one file name, it will display detail information, otherwise, a more compact view is displayed, showing one file per line.

We can always force the output by using the flags --detailed or --compact

If we add the flag --analyze (-a), the file will be read and analyzed to determine the exact number of records, perform a quick check in search for some errors, and also show some statistics about the contents, like the number of logical files present and the number of records of each logical file, the number of records by **MTI**, number of records by Transaction Type, and the number of records by **MCC** (if we add the flag --verbose)

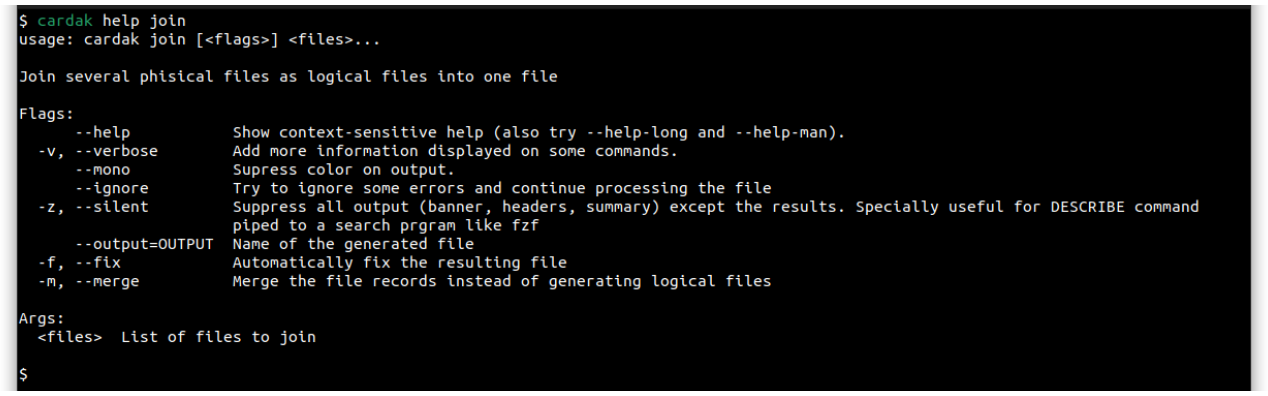
IMPORT



This command is the inverse of the **EXPORT** command

We can add previously exported records to an **IPM** file. We can import files in **HEX** or **CSV** formats that contain complete records.

JOIN



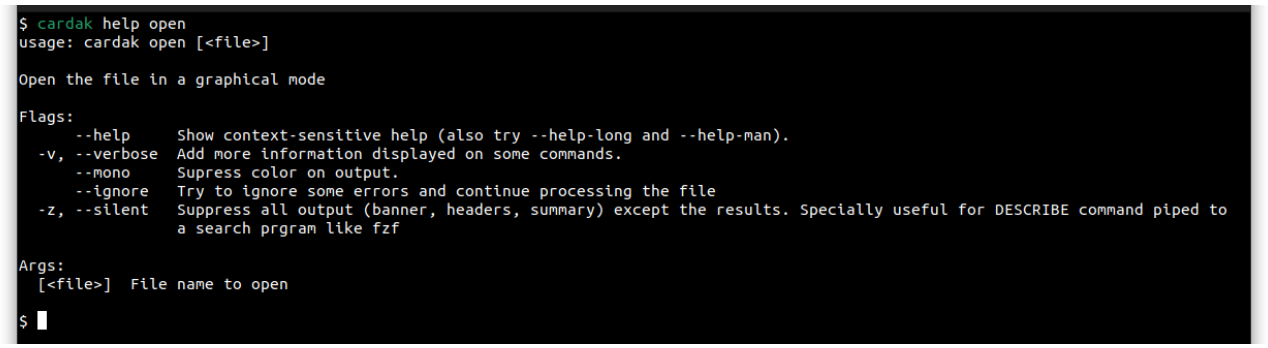
This command use used to join multiple **IPM** files in another one. We just put the list of file names and a new one is generated with the contents of all of them.

By default, the new file will contain as many logical files as physical files we are joining, so effectively, we till have the files separated into logical units.

We can change this behavior by using the flag --merge (-m), in which case all records from all the files will be merged into a single logical file. The header of this new file will be the same as the header of the first file to be joined.

We also can use the flag --fix (-f) to automatically try to fix errors that can be present in the files to be joined.

OPEN



This command can open and visualize the contents of an **IPM** file interactively. This is discussed in a separate section called TUI (Text User Interface)

PRINT

```
$ cardak help print
usage: cardak print [<flags>] <file>...

print the contents of the file record by record

Flags:
      --help                 Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose              Add more information displayed on some commands.
      --mono                 Supress color on output.
      --ignore               Try to ignore some errors and continue processing the file
  -z, --silent               Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE
                             command piped to a search prgram like fzf
  -T, --file-type=FILE-TYPE  Filter by file type when supplying several files. File types are represented by a single letter as:
                             I-IPM files, M-MPE files
  -R, --records=RECORDS      List of record numbers to be printed. Values are separated by comma (,) and ranges are indicated by
                             the starting and ending record separated by a hyphen (-)
  -F, --fields=FIELDS        List of IPM fields to be listed (can use a filter name)
  -d, --detailed             Print detailed information showing the contents of the fields
  -s, --subfields            Show subfields contents
  -C, --code=CODE            Filter by Function Code DESCRIPTION
  -l, --last                 Use the record numbers returned on the last GREP command

Args:
  <file>  File name to print

$
```

This command is used to display the contents of **IPM** files in a more friendly way.

It receives as a parameter the file name to visualize, and by default it will display all records of the file, showing the record number, the **MTI** and Function Code, and the list of **DE** and **PDS** fields present on each record.

This view is really not very useful, it is just an overview, so we usually use it with some flags like --detailed (-d) that also display the values of each fields.

As usually these files contain many records, the output can be really big, and even if we can redirect the output to a file to be handled by some external tool like a text editor, the file can be huge.

It can be more practical to just specify the records we need and ignoring the rest, by using the flag --records (-R).

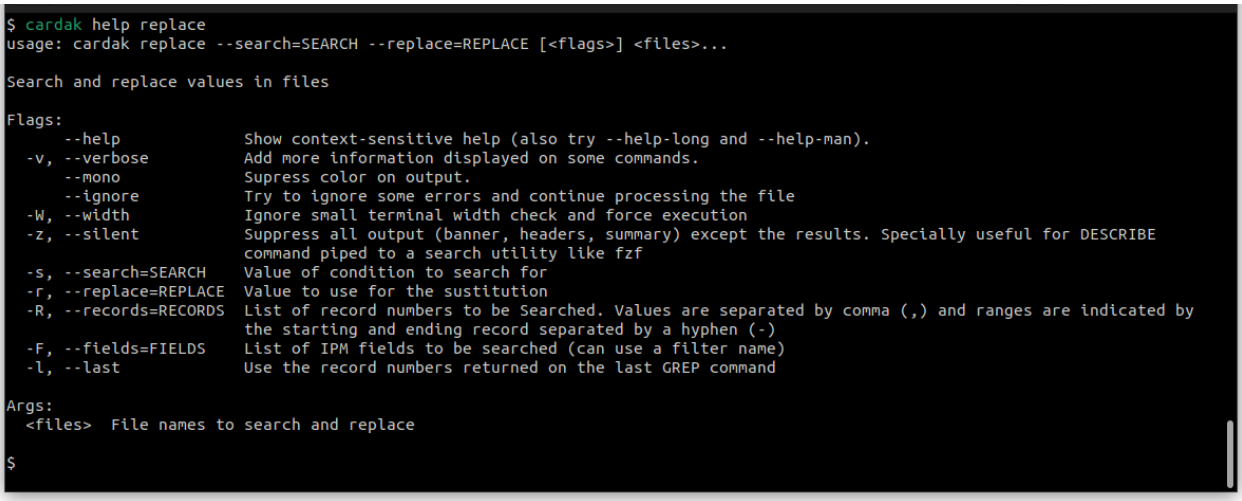
We can further shrink the output but filtering by fields, by using the flag --fields (-F)

As with other commands, we can also filter records by their Function Code using the flag --code (-C) and putting some text that must be part of the description of the Function Code.

We can also use the flag --subfields (-s) to show more details of fields that contain sub-fields.

We can also use the flag --last (-l) to just show records returned by the last **GREP** command performed on the same file.

REPLACE



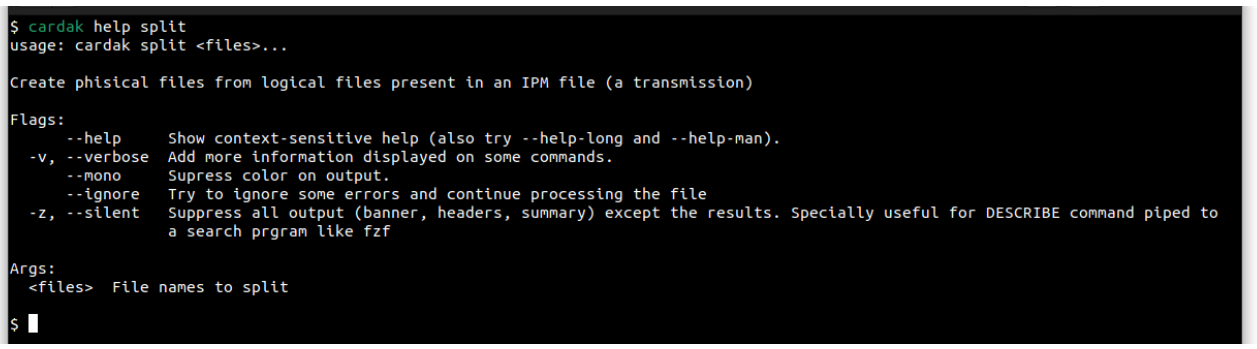
With this command we can replace values inside an **IPM** file. We search for a value using the flag --search (-s) and the value to use in the replacement with the flag --replace (-r)

The value to be searched can be indicated by a constant value, or by using regular expressions.

As usual, if we want to limit the search and replace to some records, we can use the flag --records (-R), and to search only in some fields, we can use the flag --fields (-F)

We can also use the result of the last **GREP** command by using the flag --last (-L) to only replace the values in those records.

SPLIT



This command is the inverse of the **JOIN** command, generating a physical file for each logical file present in an **IPM** file.

VALIDATE

```
$ cardak help validate
usage: cardak validate [<flags>] <files>...

Validate the IPM files

Flags:
      --help                 Show context-sensitive help (also try --help-long and --help-man).
  -v, --verbose              Add more information displayed on some commands.
      --mono                 Supress color on output.
      --ignore               Try to ignore some errors and continue processing the file
  -z, --silent               Suppress all output (banner, headers, summary) except the results. Specially useful for DESCRIBE
                             command piped to a search prgram like fzf
  -T, --file-type=FILE-TYPE  Filter by file type when supplying several files. File types are represented by a single letter as:
                             I-IPM files, M-MPE files
      --mcc                  Check that MCC values are valid (incomplete list, use with caution)
      --fail                 Just display the list of file names that have errors
      --pass                 Just display the list of file names that don't have errors

Args:
  <files>  List of files to validate.

$
```

With this command we can verify the integrity and validity of **IPM** files, and perform some basic validation. It is an advanced version of the **IDENTIFY** command when we use it with the flag --analyze, and it is implemented to be used mainly in automation chains.

The indicated files are read and analyzed, and results are shown for each of those files. The results can be either PASS or FAIL

In addition, a return code is returned to the operating system, being 0 as all files passed the validation, and 1 if any of the files contain errors.

It also shows if the errors found are automatically fixable (by using the **FIX** command) or not, and also the number of records with errors.

We can use the flag --silent (-z) for a compact output without colors, to be used by an external process.

We can also use the flag --pass to only have the list that have no errors, or the flag --fail to get the list of files with errors. This can be handy to automate processes by performing different action on files with or without errors, like moving them to different locations, display alerts, etc.

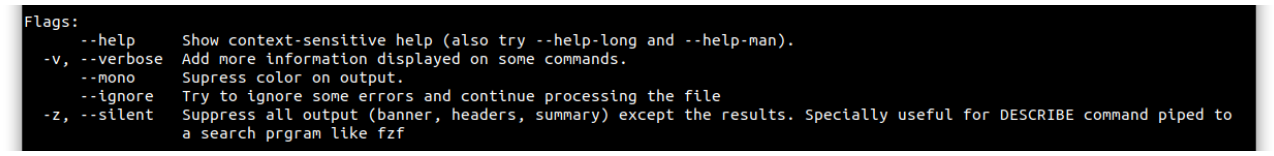
Flags and Filters

Many times it is convenient to limit the scope of some commands to some records or fields. Most of the commands can accept filters to limit those scopes. Here he will see how to use them.

The flags are used on the command line, and they indicated by a couple of colons followed by a string. Sometimes an alias exist consisting on just a colon and a single letter. Some flags can accept values to change its behavior, and some other times it is enough for the flag to be present or absent.

Some flags are global to the application (they work regardless of the command), and some other are specific to each command.

Here are the global flags:



-- help: This flag is similar to the **HELP** command, but can be used with any other command (sometimes we just add this flag to a command to get help on that particular command)

--verbose (-v): In some cases, it increases the information to be displayed. Usually the output is limited to avoid an excess of information on screen, and using this flag we get the full output.

--mono: By default, the output displays colors to improve the readability (by using terminal ANSI codes). This can be a problem when redirecting the output to a file, as those color codes can add noise to the data. In those cases, especially when redirecting the output to a file, it is convenient to use this flag.

--ignore: This flag is mostly deprecated, but in some cases, when reading a file, some errors can be found (like wrong block delimiters). Using this flag the tool will try to read the file ignoring those errors.

--silent (-z): Usually, the output of the commands include extra information, like the banner with version and build information, and the total execution time. But sometimes, and especially when using the tool as part of an automation process, this extra information can make the process more difficult. By using this flag, the color is eliminated, the banner is not displayed, the final execution time is not displayed, and progress bars when handling big files are also not displayed.

Every command can have specific flags, but others are commonly used on most commands (general purpose flags). Here are some of them:

-T

This flag requires an extra value, which is a single letter. These letters can be “I” or “M”

It is used to filter files when we pass a list of file names, like when using wildcards, but the command only accepts **IPM** or **MPE** files

If the flag is present, only files of the corresponding type will be processed, and the rest will be silently ignored.

-C

This flag is used to filter records by Function Code. It receives a string and only records having a Function Code which description contains that text are processed.

If, for example, we want to only consider records containing Chargeback information, we can add to the command the flag “-C chargeback”.

To see the full list of available Function Codes, we can use the DESCRIBE command by executing the following command: “cardak describe func”

-R

Filter by record numbers. By using this flag, we can specify the record numbers to be considerd. The record numbers are the physical record number in the file.

It receives a comma separated list of values, which can be a single number or a range (by putting a dash between the limits of the range, where both extremes of the range are included).

If, for example, we want to process records number 10, 12, 30 to 39 y 50, we can add this to the command line: “-R 10,12,30-39,50”

There is a particular behavior with the **GREP** command, as when we use it with a single file and records match the search condition, the list of record numbers of matched ones is saved, so they can be automatically be used by a subsequent command by using the flag --last, so we don’t need to manually specify the record numbers if we want to apply the command to the records found by a previous **GREP.**

The tool checks that when using this flag, the used file is the same as the last one used by the GREP command.

-F

Filter by fields. Sometimes, we want to limit the actions to some fields. Similar to the filter by record, this flag lets us indicate which fields to consider. But it has some other differences.

Before anything else, let’s see how to identify a field. On **IPM** files we have two different element types, the Data Elements (DE) and the Private Data Sub-elements (PDS). These elements can be subdivided into smaller components called sub-fields.

The way to specify any of these fields is like this: DE are followed by a number between 1 and 3 digits, that uniquely identify each one, while PDS are followed by a four digit number. Sub-Fields are indicated by appending the letters “SF” and a two digit number. For example, the field named “Card Acceptor Name” is DE043, and is subdivided into 6 parts, being them DE043SF01, DE043SF02, etc.

In order to simplify the naming of these files, follow these rules:

DE can be represented with the letters DE or D, followed by one to three digits, or simply by specifying the three digits.

PDS can be represented with the letters PDS, PD or P followed by one to four digits, or just using four digits.

Sub-Fields are represented by adding the letters SF o S, followed by one or two digits

Following these rules, field DE43 can be represented in any of these ways:

DE043, DE43, D43, D043, 043

If we want to reference a specific Sub-Field, to any of the previous options we can add SF or S and the sub-field number. For example, to specify Sub-Field 03, we can use:

DE043SF03, D43S3, 043S3, etc.

For the PDS it is similar, but if we don’t add the letters, we must use 4 digits to differentiate it from a regular DE. As an example, to reference PDS0021 we can use:

PDS0021, P21, PD21, P21, 0021, etc.

To reference a PDS Sub-Field, we apply the same rules as for the DE

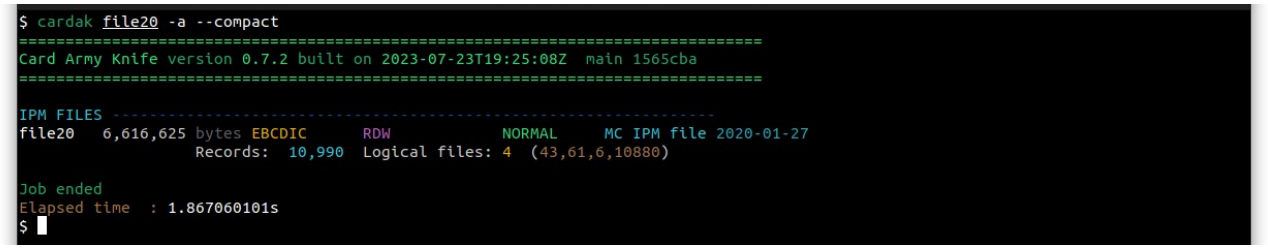
TUI (Text User Interface)

Most of the commands are not interactive, but we have to option to visualize and perform some operation on **IPM** files interactively. It is not a graphical application, but it is something similar but confined to a text terminal. This allow the tool to be used on a remote server by using, for example, SSH.

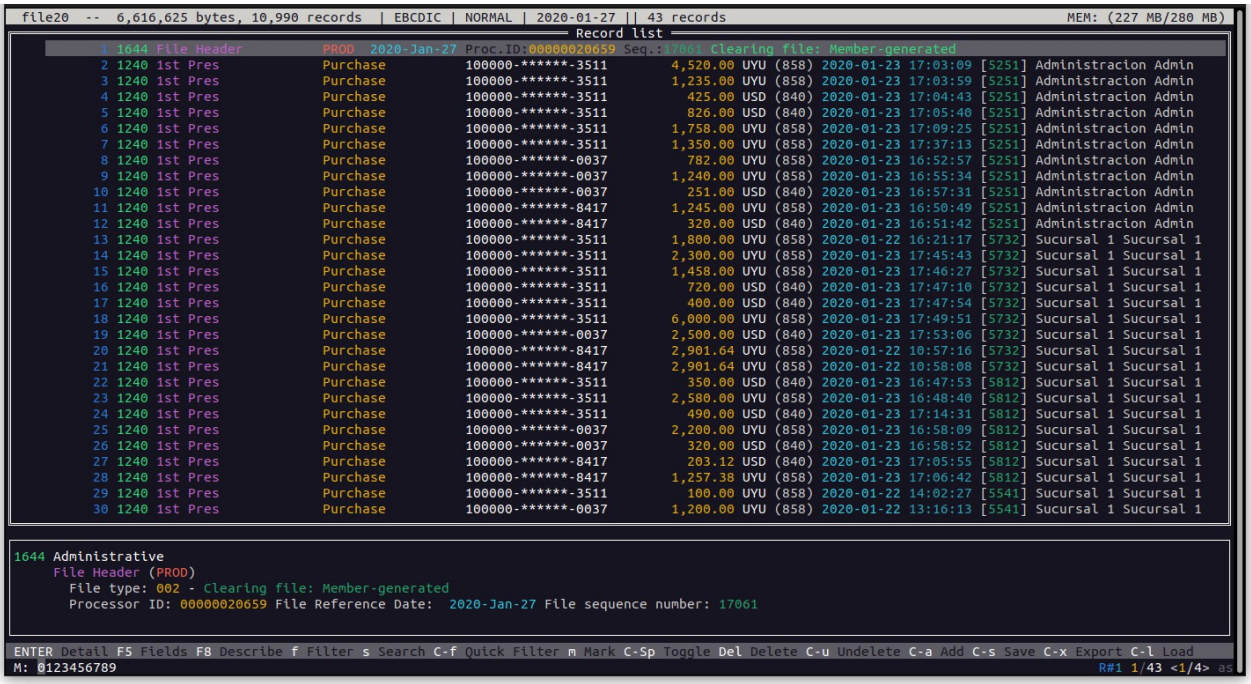
This working mode is called Text User Interface.

To get into this mode, we use the **OPEN** command, which reads the full content of the file and presents it in a format that occupies the full terminal, allowing the user to navigate and perform actions by using some keyboard keys and combinations. In some terminal, a mouse can also be used.

In this example, we will use the file file20 that contains 4 logical files, as seen here:



Now let's execute the command "cardak open file20"



The main screen is displayed with the contents of the file (file20 in this case). The screen is divided in 5 vertical sections as described:

In the upper part, we have a bar containing informacion of the file. We see the size in bytes, the number of total records, the encoding and format (block or normal). We can also see the date included in the first header, and an area where we can see the number of active records (more about this in a following section of this manual). On the right side of the bar, we see information about memory usage (reserved by the application and total usage on the system).

Below this bar we have the main area, where we can see the main information of the records. We see the record number inside the file, the **MTI** value and the record type. The information displayed depends on the

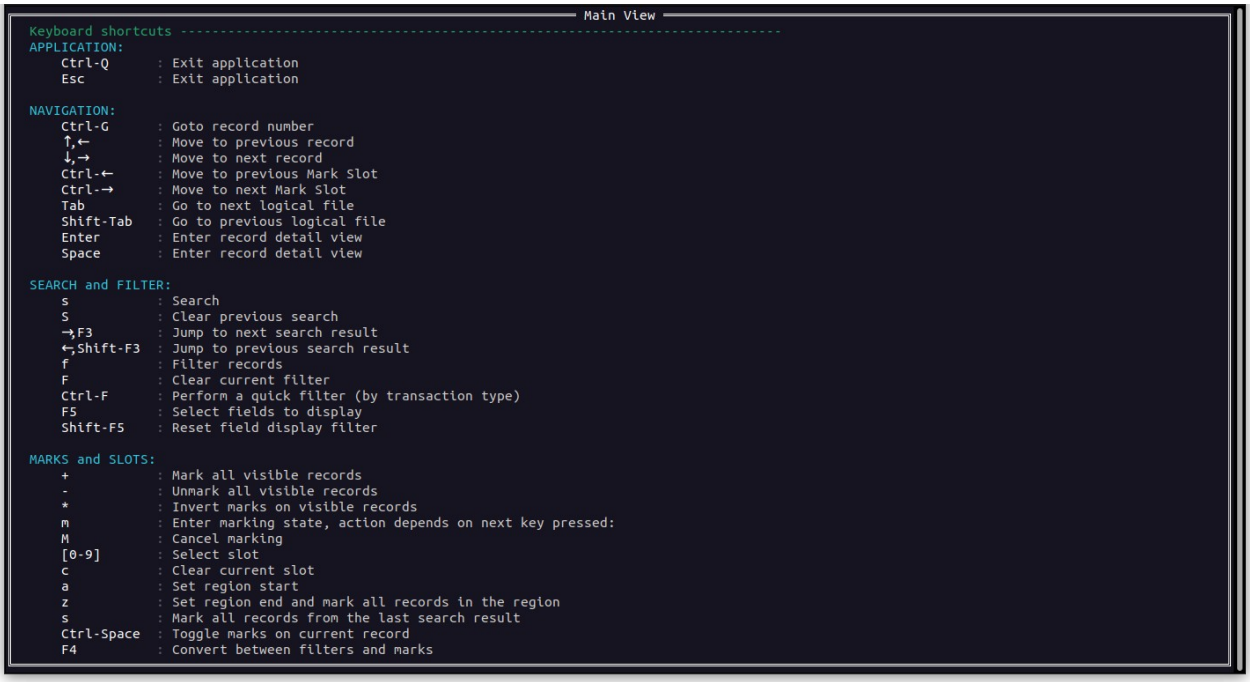
type of the transaction contained in the record. For example, for regular purchases, we see the masked **PAN,** the amount, the currency and its code, Date and Time of the transaction, the **MCC** and the merchant name.

Below this main area, we see a smaller area with more details about the record we are on.

Belos that detail area we have another bar with gray background, where we see a help summary of the available keys and their actions.

At the bottom, we have the status bar. On the left we have the “slots” indicator (as discussed later), a centra message area, and on the right more information about the record we are on, like the record number and the total number of records, the logical file number (here we are on the first of 4 logical files), and an “Auto Save” indicator (discussed later in this manual)

Pressing the **F1** key we get help about the screen we are on. For example, in this main scren, pressing the key **F1** we get this:



We will see now the actions we can perform.

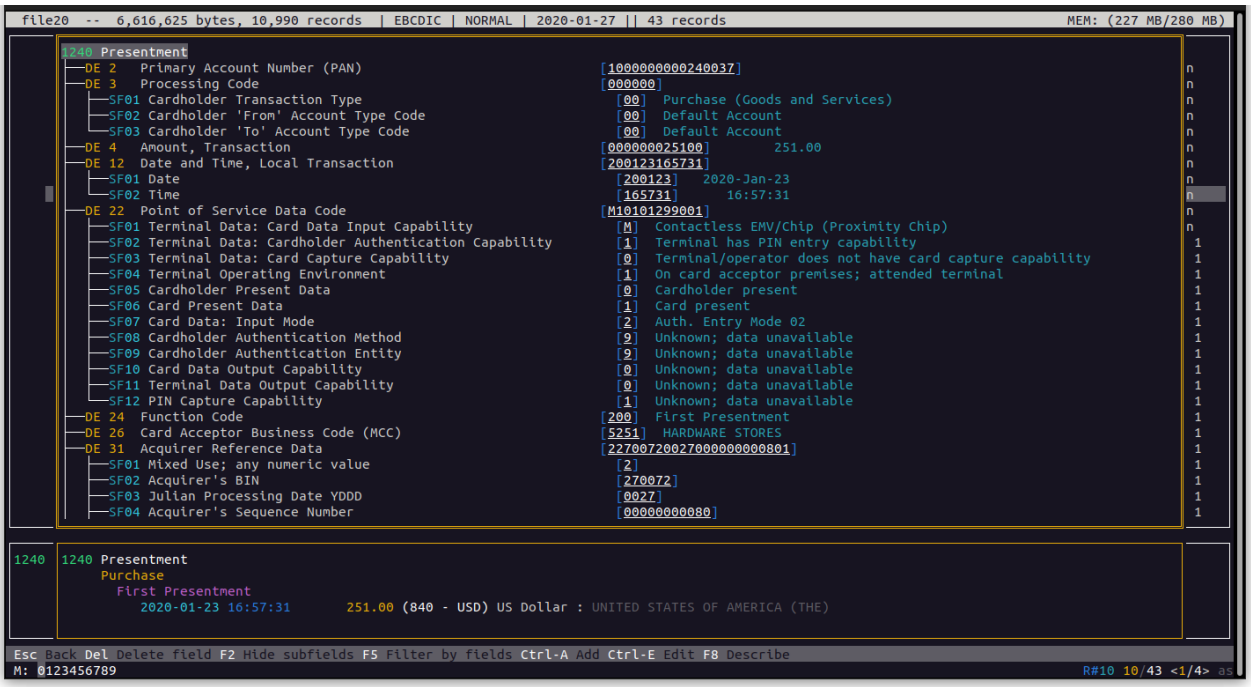
Navigation

In order to navigate between the records, we can use the usual cursor keys, plus **PgUp**, **PgDn**, **Home** and **End.**

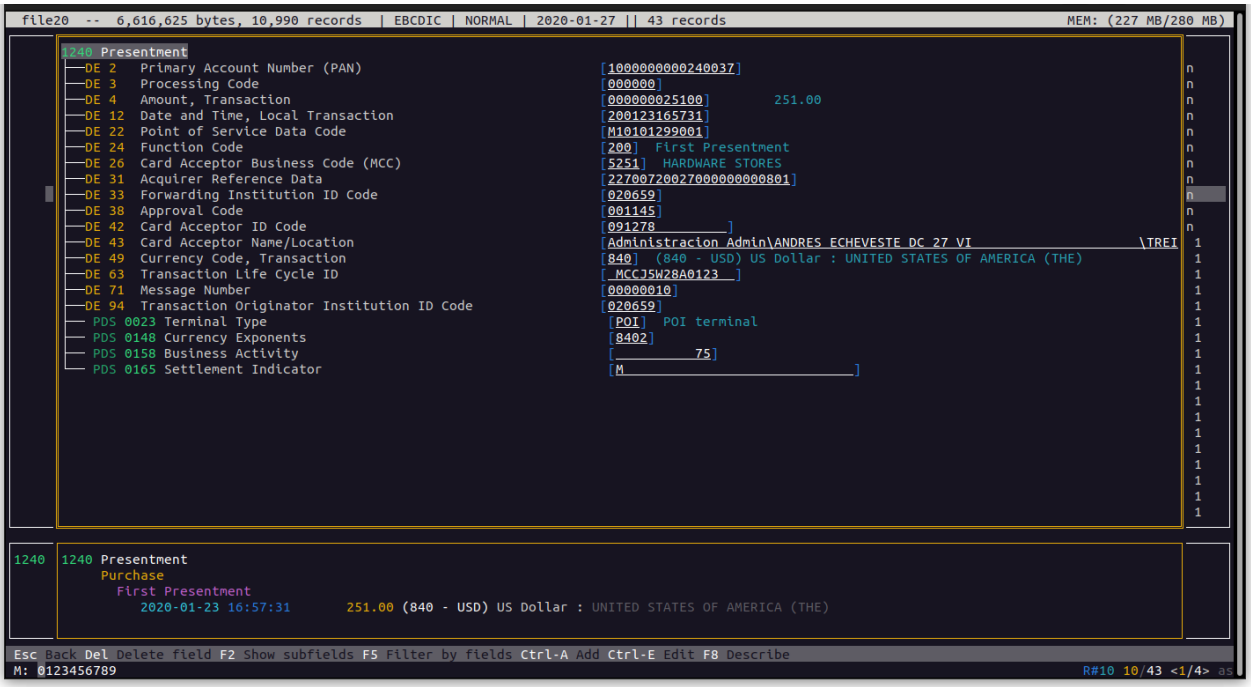
We can jump to a specific record number by pressing **Ctrl-G.** We will be asked to enter a record number, and by pressing **ENTER** we will jump to the desired record as seen here:

By using the **Tab** and **BackTab** (**Shift-Tab**) keys, we will cycle between logical files if present. The logical file we are on will be reflected on both bars.

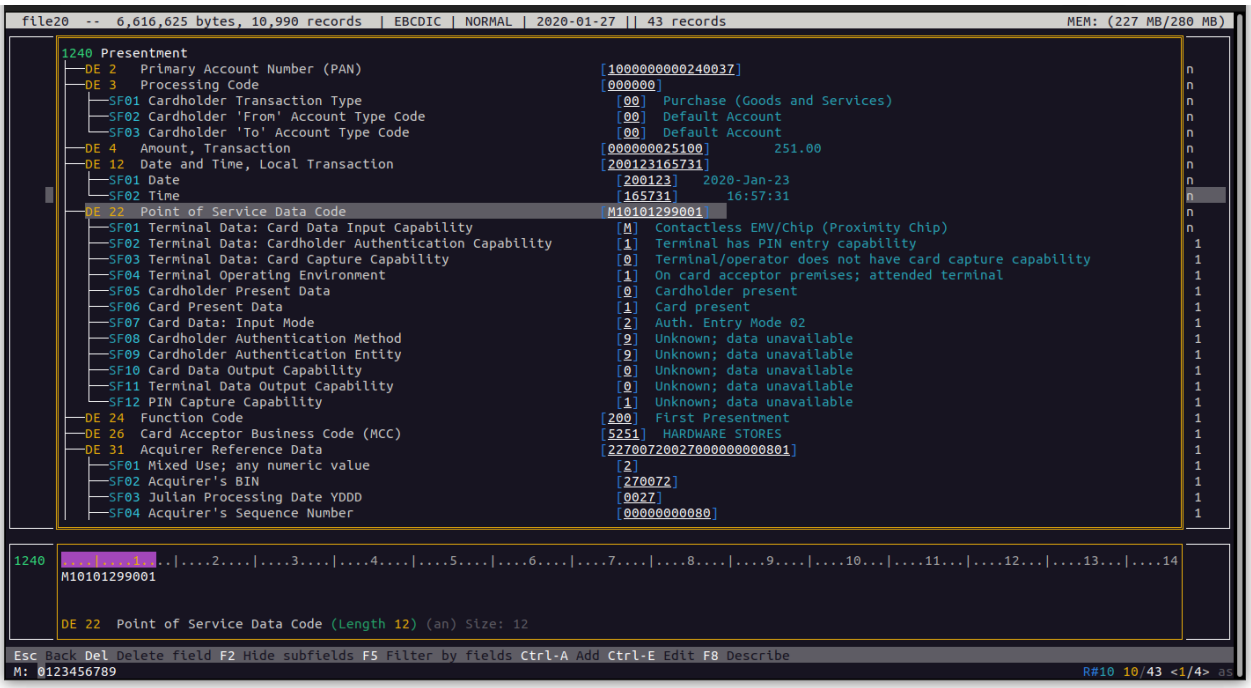
If we want to see the full details of a record, we just press **SPACE** or **ENTER**, and a new pane will be open with the full contents of the record.



This view shows the present fields along with the description and values. By default this view also shows the sub-fields and in some cases the meaning of the value is displayed. If we want s more compact view by hiding the sub-fields, we can toggle the views by pressing **F2**



In this detailed view, as we move through the different fields, in the pane below we will see information for that field. For example, if we move to field **DE022**, we can see the description and definition of the field, along with the value with a ruler to help visualize the value.



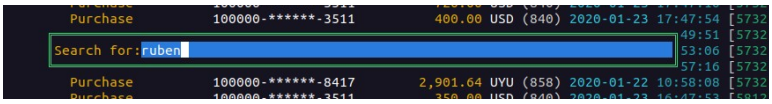
To return to the main view, we just press the **Esc** key.

Search and filters

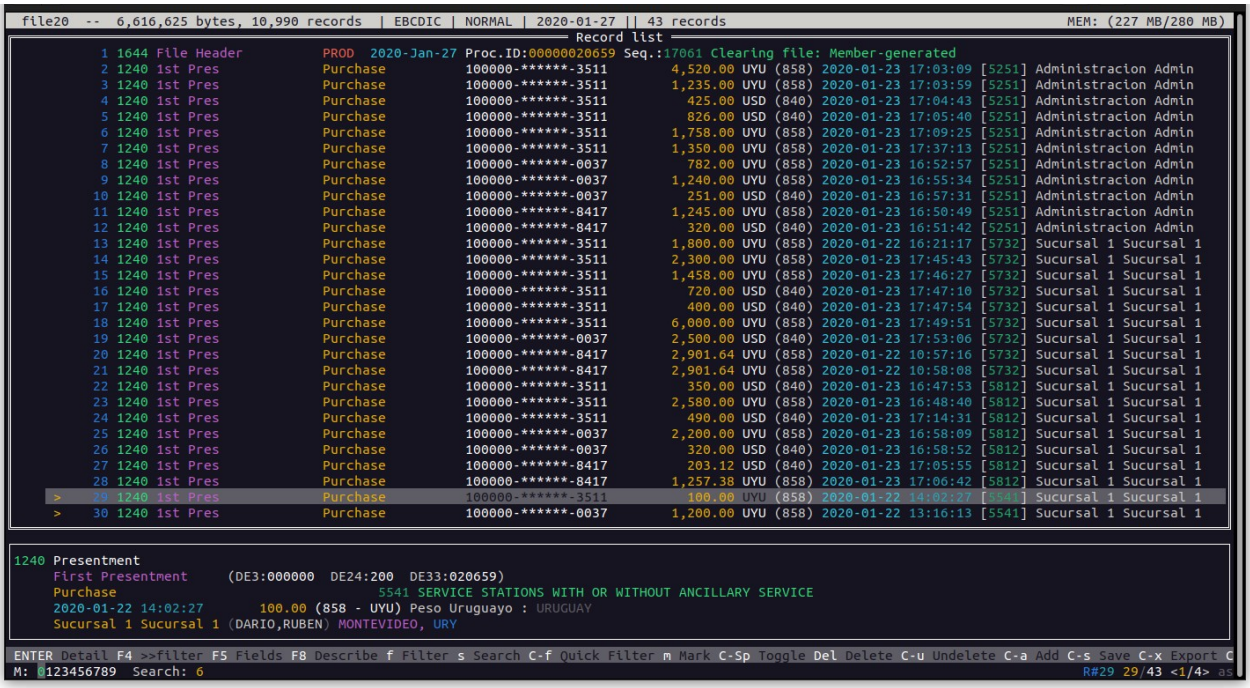
Being able to see the records is great, but when we are dealing with files containing thousands of records, it can be difficult to navigate to get the data we are looking for. For this we can perform searches and filter records. Both options are similar, being the main difference that when performing a search, we jump to the first record matching the search criteria, while filtering hides all records except the ones that match the criteria. Anyway, we can easily switch between the two modes as we will see later.

To see how this work, we will start performing a global search. We want to search for the word “RUBEN”, but we are not sure in which field it is present.

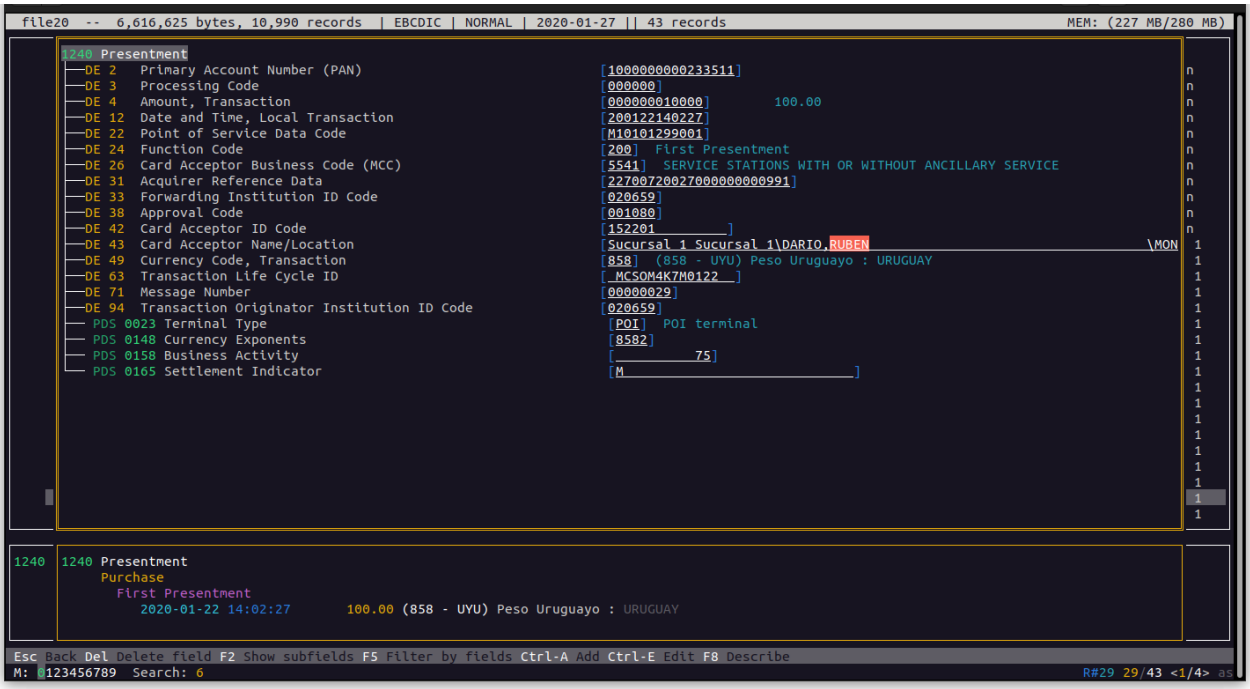
To perform the search, we just press the “s” key and we will get a dialog where we can write the search criteria.



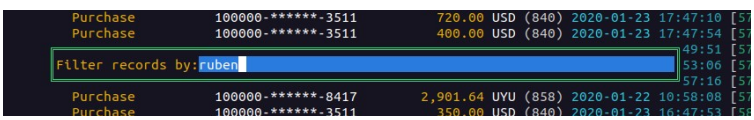
By pressing **ENTER**, the search is performed and the matching records are marked, jumping to the first result. Besides the usual navigation keys, we can use right and left cursors to jumpt to the next and previous search result.



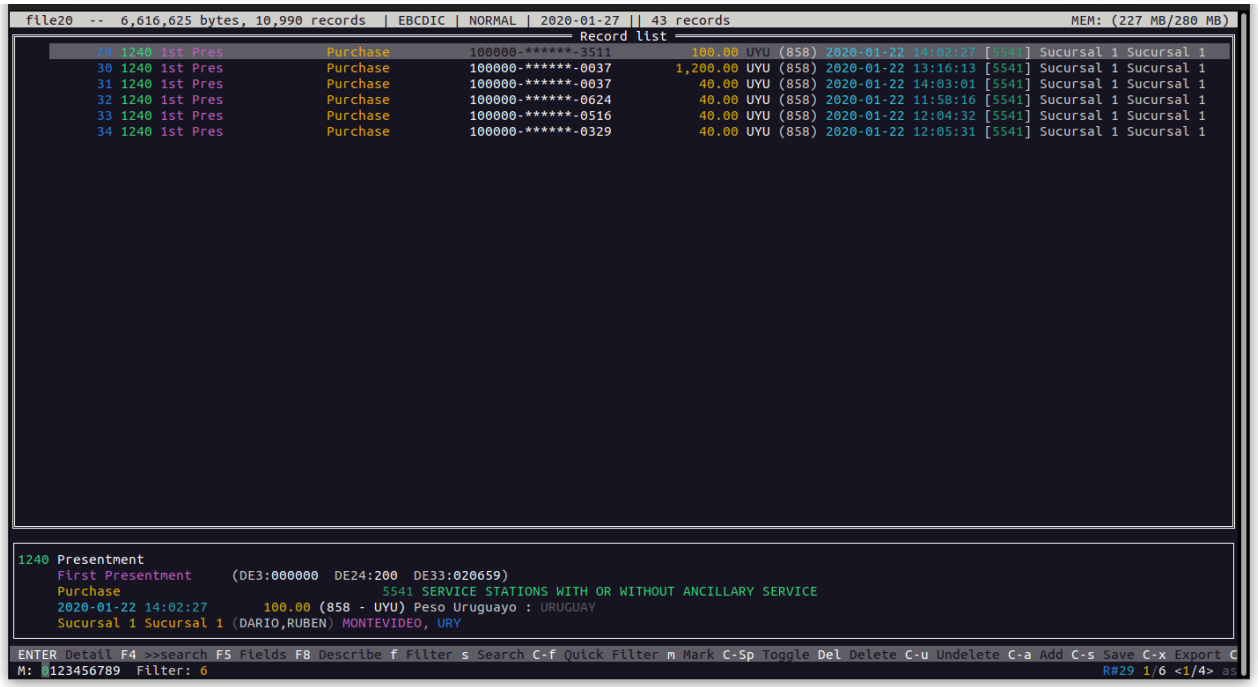
Entering the detailed view (by pressing **ENTER**) we can see the matching value highlighted



Now, lets perform the same search but using a filter instead. To do this, instead of pressint the “s” key, we press the “f” key:



Now, we will only see the matching records

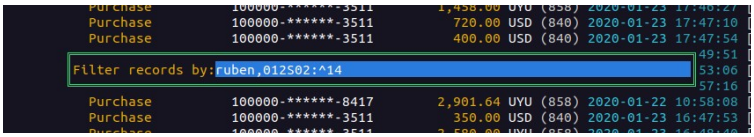


From the filter view we can change to the search view and vice versa, just pressing the **F4** key

But the search engine allows more options when performing the search. We can use the same methods used in the GREP command described in another part of the document.

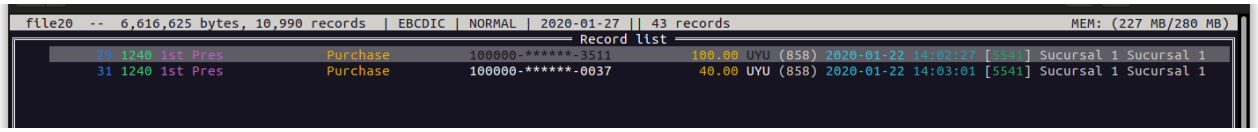
For example, we can limit the search in some fields. For that, we must put the field name and the value to search for, separated by a colon (:)

So, let's search again transactions that contain the word "RUBEN", but only those made between the time of 14:00 and 15:00. We know the time is present in **DE12S02**, and as we can use regular expressions, we can perform the search like this:



We search for the string "ruben" in all the fields, and a value in field **DE012SF02** that starts with "14". Both conditions are separated by a comma, so they both must apply in order for the record to match.

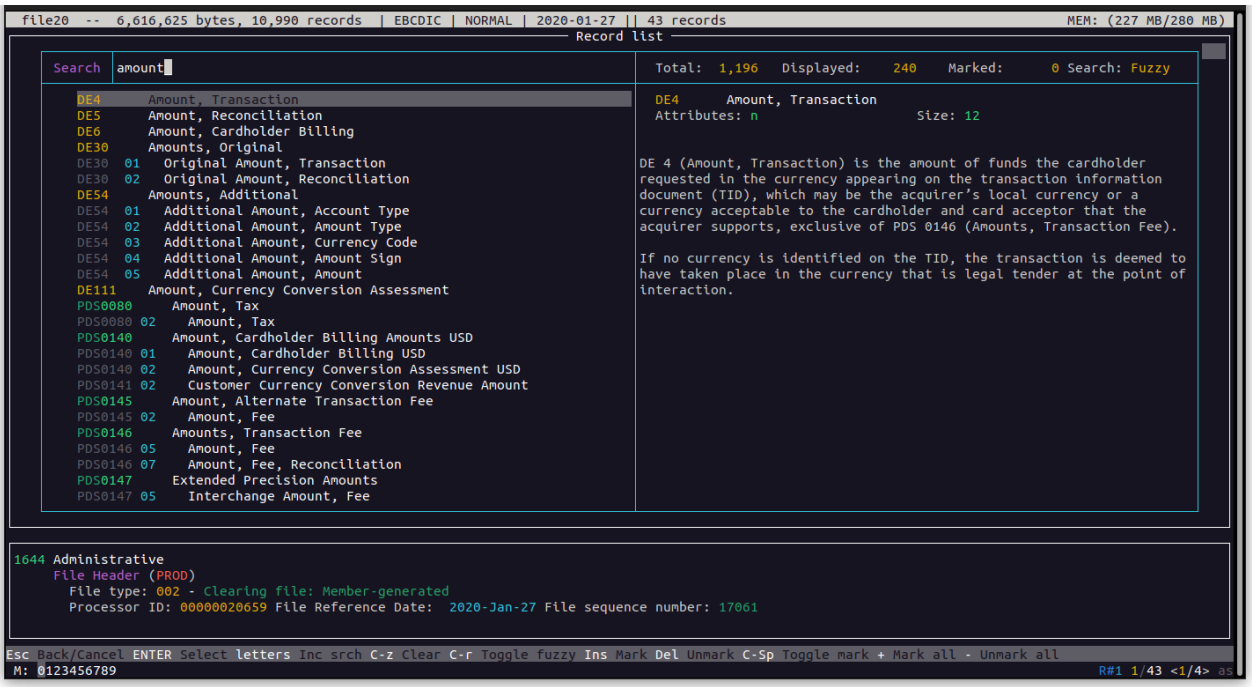
Now we only have two records:



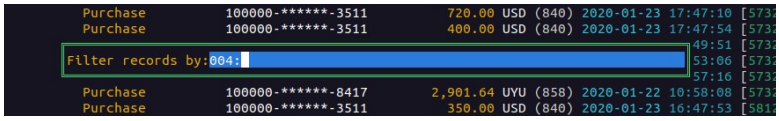
Another example. Suppose we want to search for an amount of 40

We can start the search by pressing the "f" key, but we don't remember the field that holds the transaction amount. In this case, when entering the search condition, we can press the **F8** key to get help. A new pane is shown with all the possible fields. Typing "amount" we will get all the fields that contain the word "amount" in their description.

We see the transaction amount comes in field **DE4**, so we move to that line and press **ENTER**.

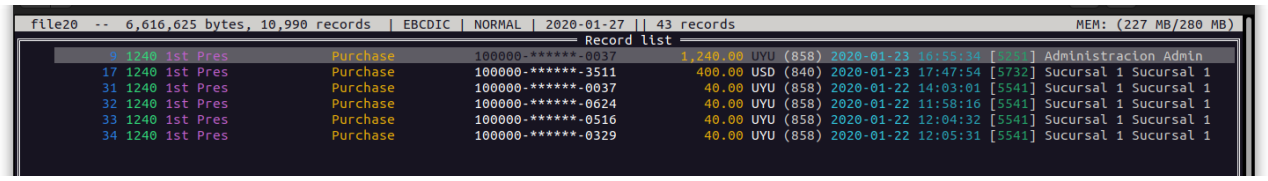


Doing this, makes the field identifier to be automatically inserted into the filter bos, and also the colon is added. We now just have to enter the value to be searched.



We type 40 and press **ENTER**

We have several records, not only the ones we want. That is because all records having the value “40” somewhere are shown, but that is not what we were looking for.



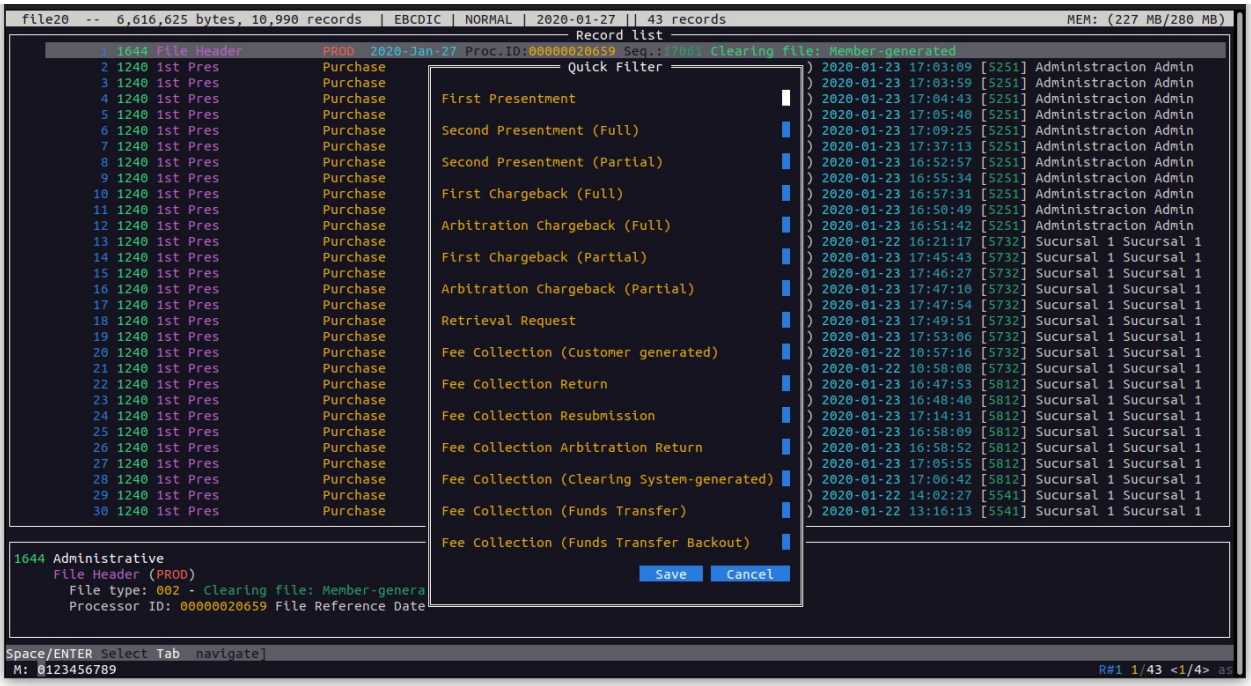
But we know that the amount is recorded with two decimals for this currency, and that the field is filled with 0 on the left to fill the length. So we can use another regular expression to force that condition. Something like this:

And now, only records with the exact value of 40.00 are shown

If we still have several records, we can refine the search adding more conditions, like the time, merchant, **PAN**, etc.

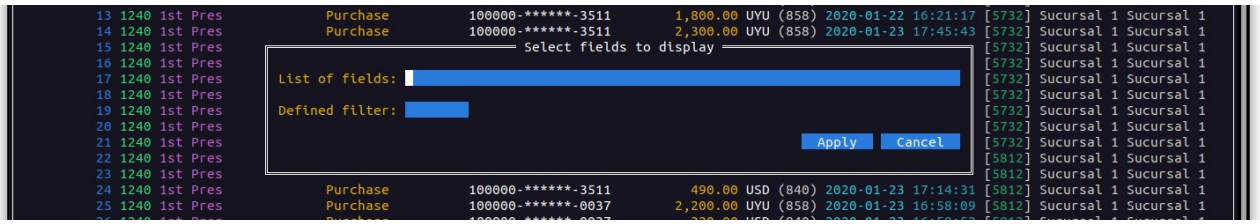
But we also have another quick filter option, that is performed by pressing **Ctrl-F**

Different transaction types are displayed, and we can selecte the ones that we want, without the need to enter the corresponding fields and values.

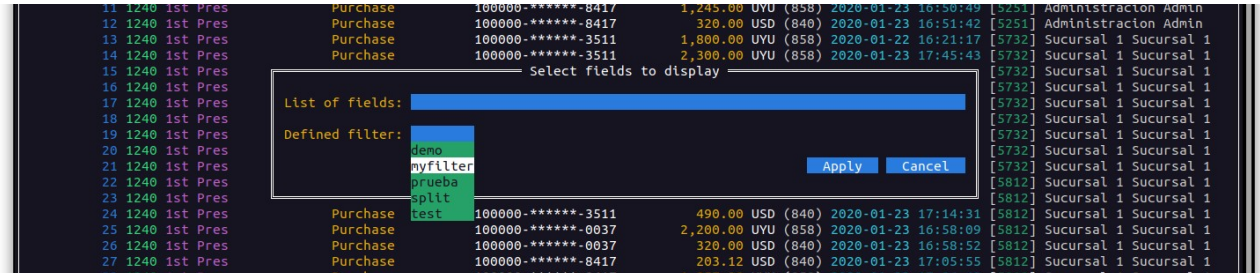


View filter

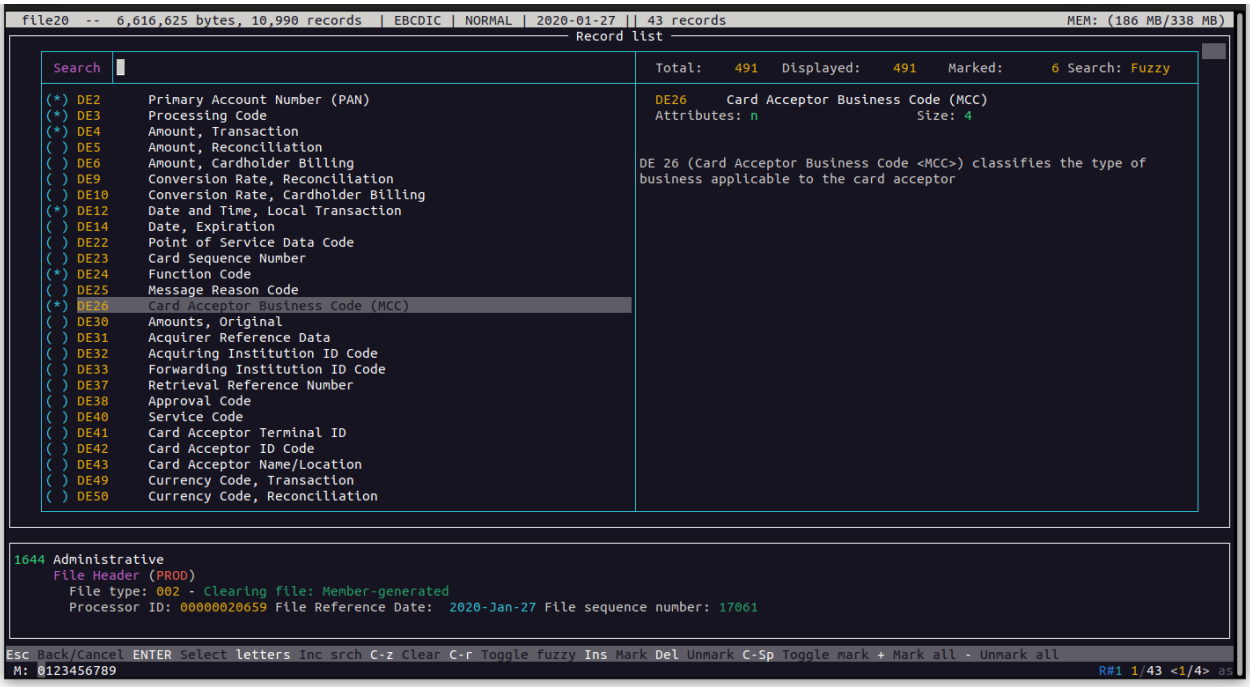
Another filter view is to select which fields to see in the detail view. Maybe we just want to see a couple of Data Elements and having all the fields makes it more difficult to focus on what we want. So, pressing the **F5** key we are asked to enter the list of desired fields and, optionally, we can use the existing filters as we previously saw in the section of the FILTER command.



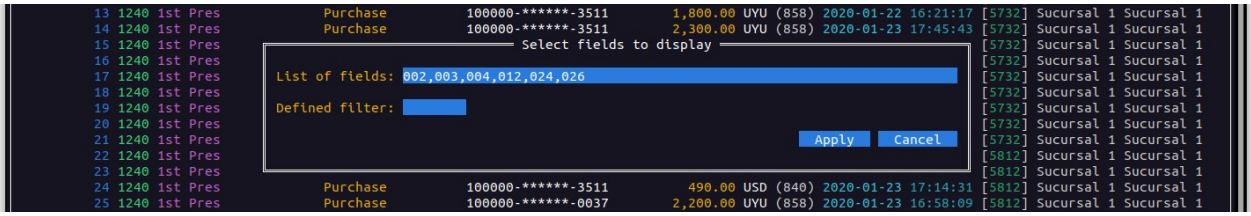
If we move the the “Defined filter” area and press the down cursor, we can select the defined filters



As with other places where we can enter a list of fields, we can press the **F8** key and get the list of possible fields. In this case, we can select multiple fields by pressing **Ctrl-Space** on each one, and when all the ones we want are selected, we just press **ENTER.**



After having the field list, we press **ENTER**

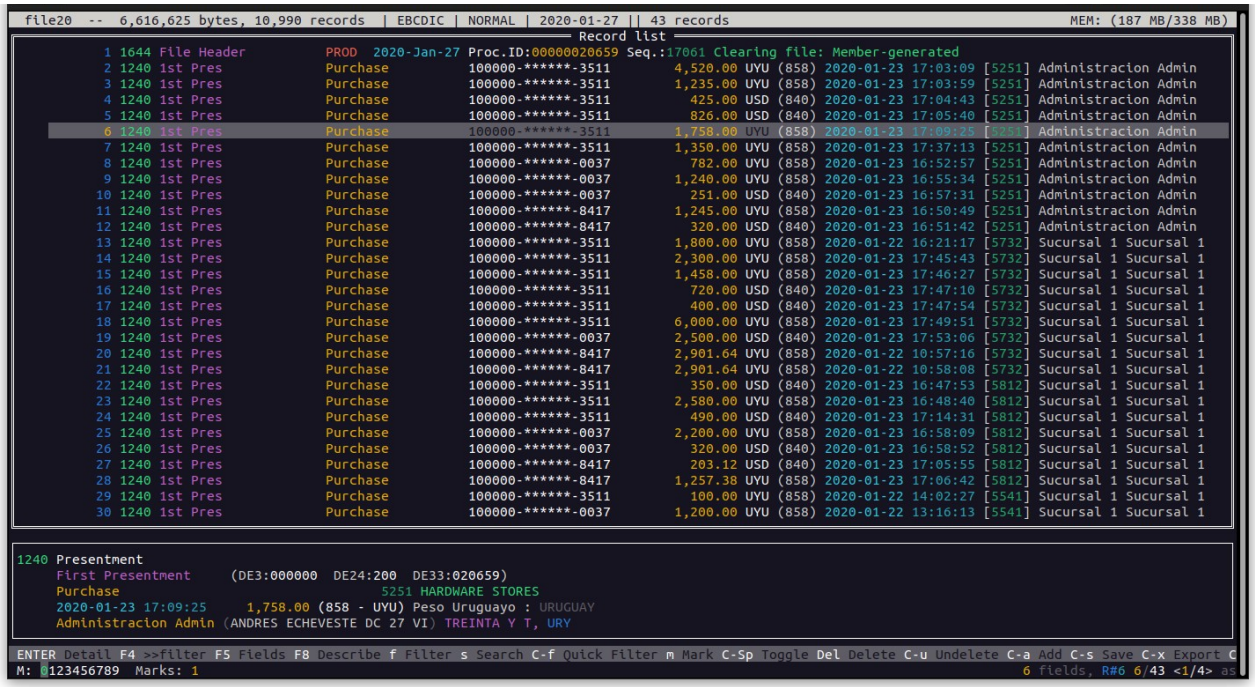


By pressing the **Apply** button, the filter becomes active, so when we enter into the detail view, we will see only the selected fields.

To reset this filter and see all fields again, we press **Shift-F5**

Marking records

We can mark individual records to perform some operation on them instead of the whole file. When we perform a search (using “s”), the matching records are automatically marked. We can also manually mark individual records by going to the desired one and pressing **Ctrl-Space**. In this example, we just go over record number 6 and press the keys. We see that the record number changes color, indicating that it is marked.



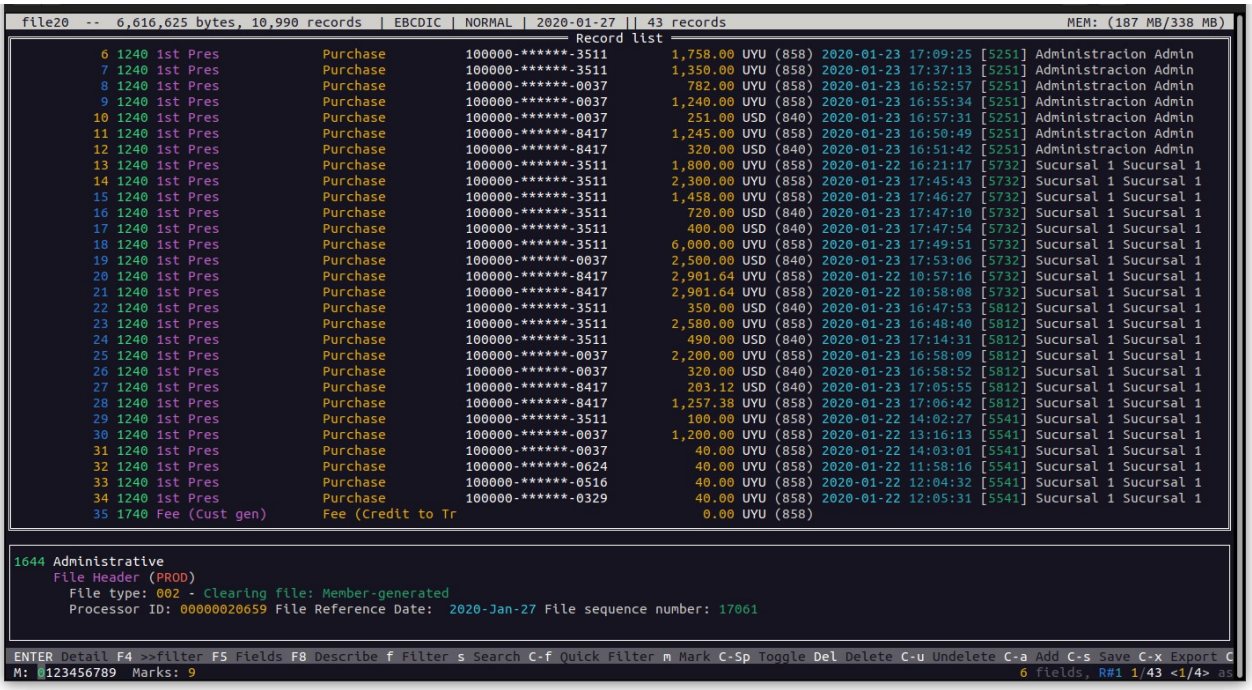
We can also see that in the status bar at the bottom, over the left, there is an indicator with the text “Marks” and the number of marked records, in this case 1

We can also mark a range of records, If we wanted to mark records 10 to 14, we go to record number 10, press the “m” followed by the “a” key (think of mark A). Then, we move to the end of the range, in this case record 14, and we press the “m” and “z” keys (mark Z). So records 10 through 14 are now marked. Thhe previously marked record number 6 is still marked, so using this method just adds records to the already marked ones. If we want to remove just one mark, we go to the corresponding record and press again the combination **Ctrl-Space** (this just toggles the mark status of a record). If we press the “-” key, then all marks are removed.

One option is to filter some records, and then pressing the “+” key, all visible records are marked. If we then press **Shift-F** to clear the filter, the marks will still be there.

Let’s add the records we found with the amount of 40.00

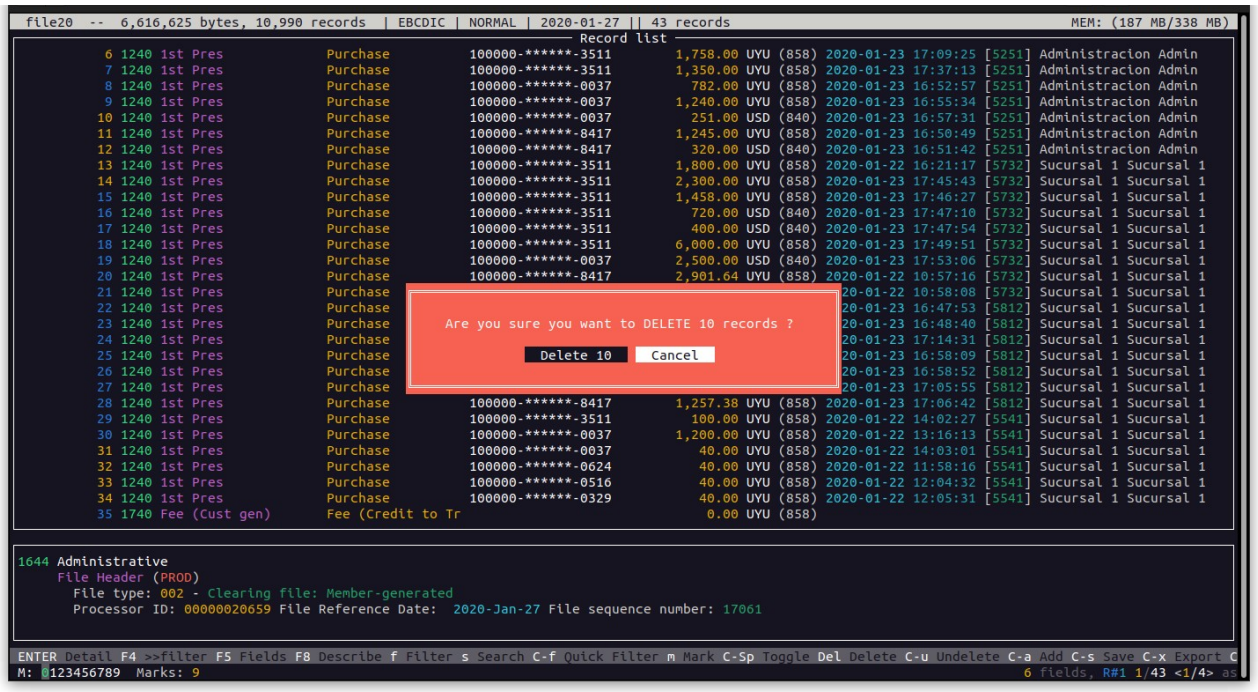
First, we apply the filter and then we press the “+” key, and after that we press **Shift-F5**



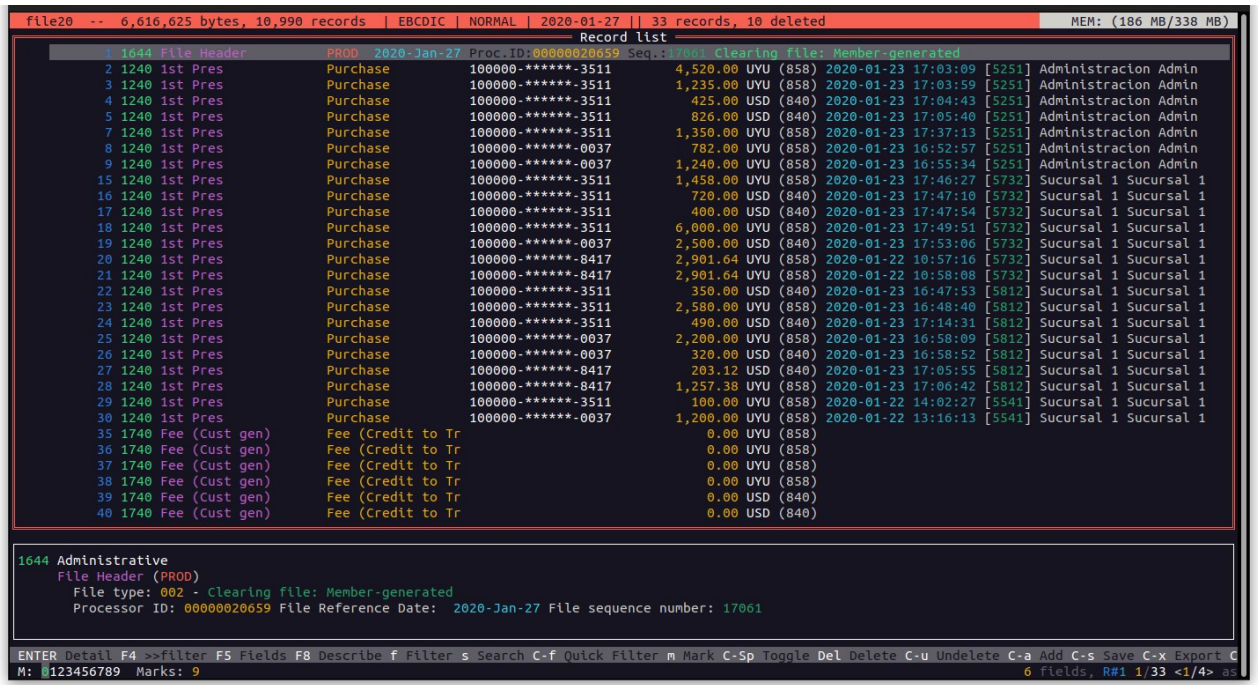
Delete records

Pressing **Del** or **Ctrl-D** we can delete records. If no records are marked, we will delete just the record we are on, otherwise, all marked records will be deleted.

Before records are deleted, we are asked to confirm the operation. The dialog box also shows the number of records to be deleted.



By pressing the confirmation button, those records will be deleted, and the upper bar and pane border will chage color to red to remind us that we have deleted records and changed the file. The upper bar will also show the number of deleted recods.



We need to note that all actions are non destructive, so the original file will not be modified in any way. So, in order to save the changes, we need to save the records in a new file as we will see later on this document.

Recovery of deleted records

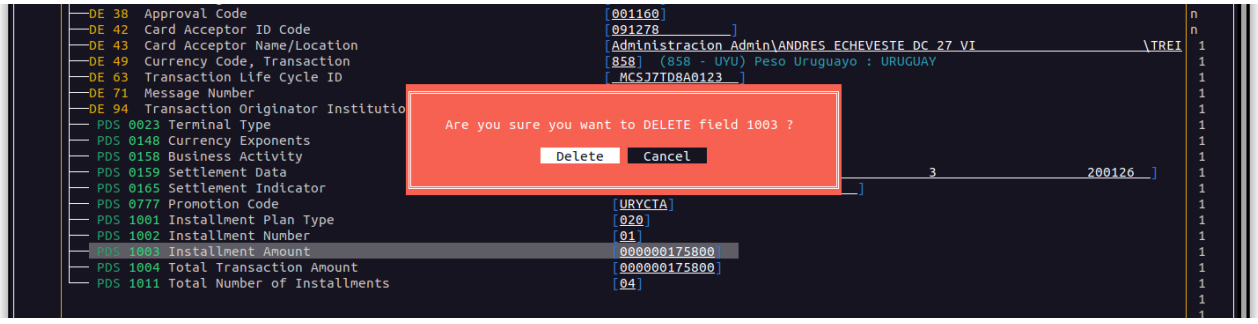
After deleting records, we may find that we deleted some that we wanted to keep. In this case, and before quitting the application, we can just press **Ctrl-U**

Doing this will show a new pane with all the deleted records. We can move to any of them and pressing **ENTER** will mark it again as undeleted.

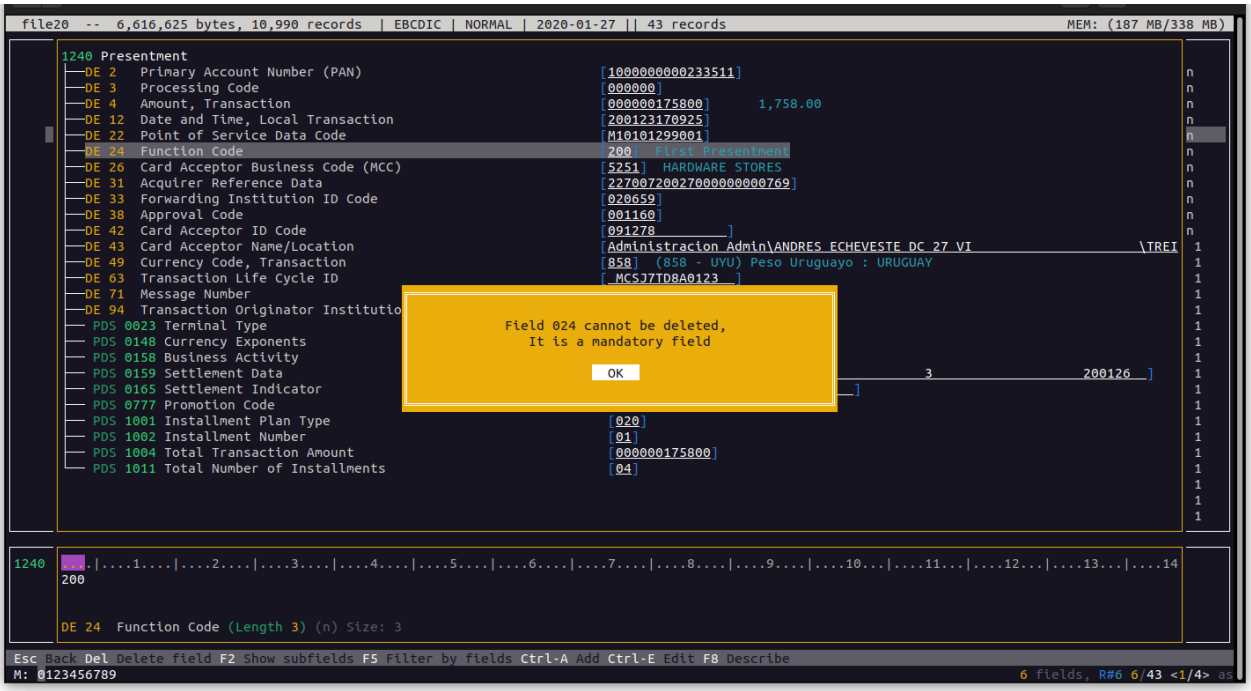
Modifying records

While in the detailed view (pressing **ENTER** or **SPACE** over any record in the main view), we can modify the records in several ways.

We can, for example, delete any field. For ecample, let's delete field **PDS1003**. To do so, we move over the field and press **Del**

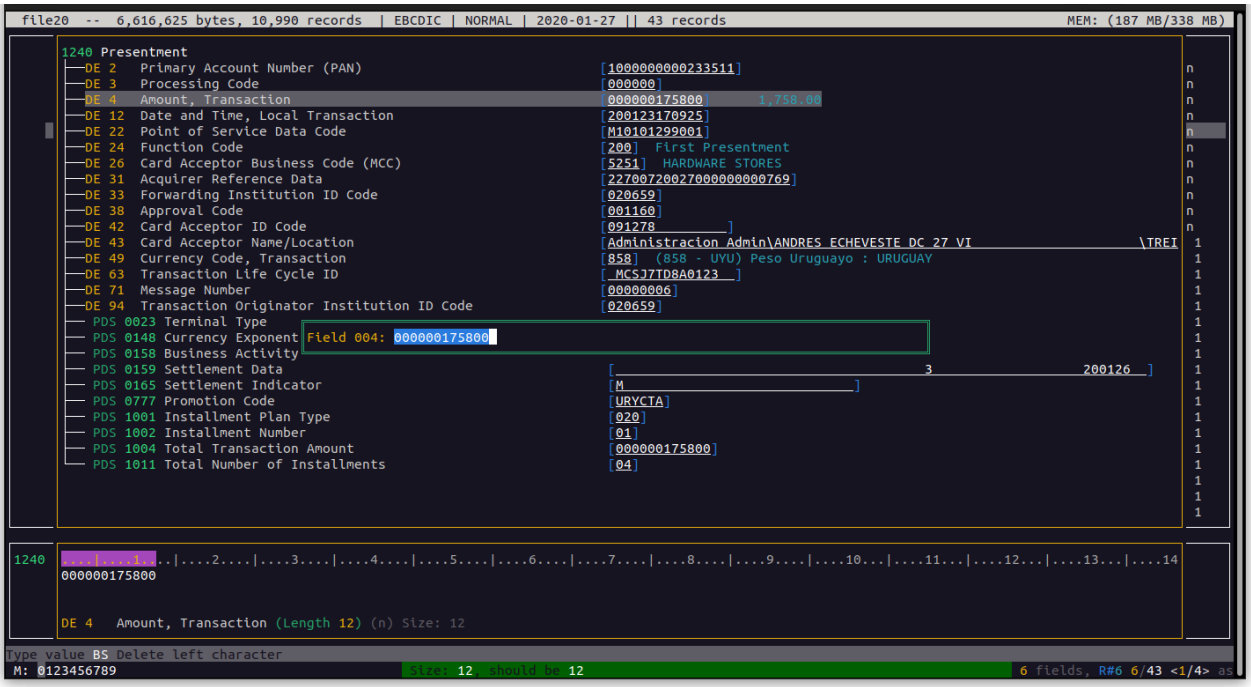


If we confirm the action by pressing the Delete button, the field will be eliminated from the record. Please note that we will not be able to delete mandatory fields. For example, we will not be able to delete field **DE024**.

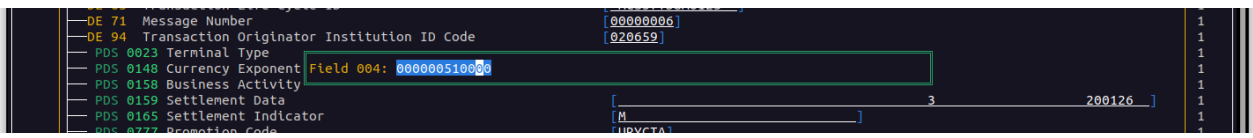


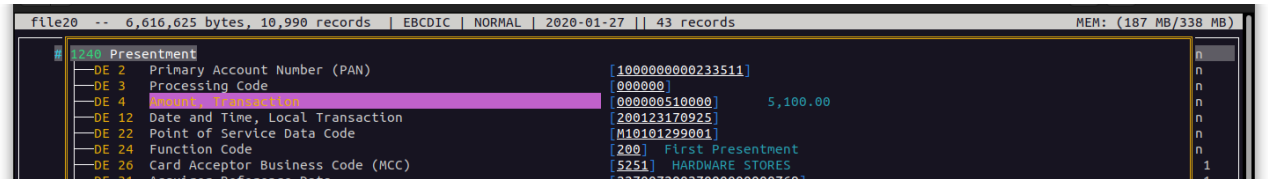
Another thing we can do is modify the value of a field. Let's change the amount of 1758,00 to a value of 5100,00

We first move to the field **DE4** and press **Ctrl-E**



We modify the value and press **ENTER** to confirm.

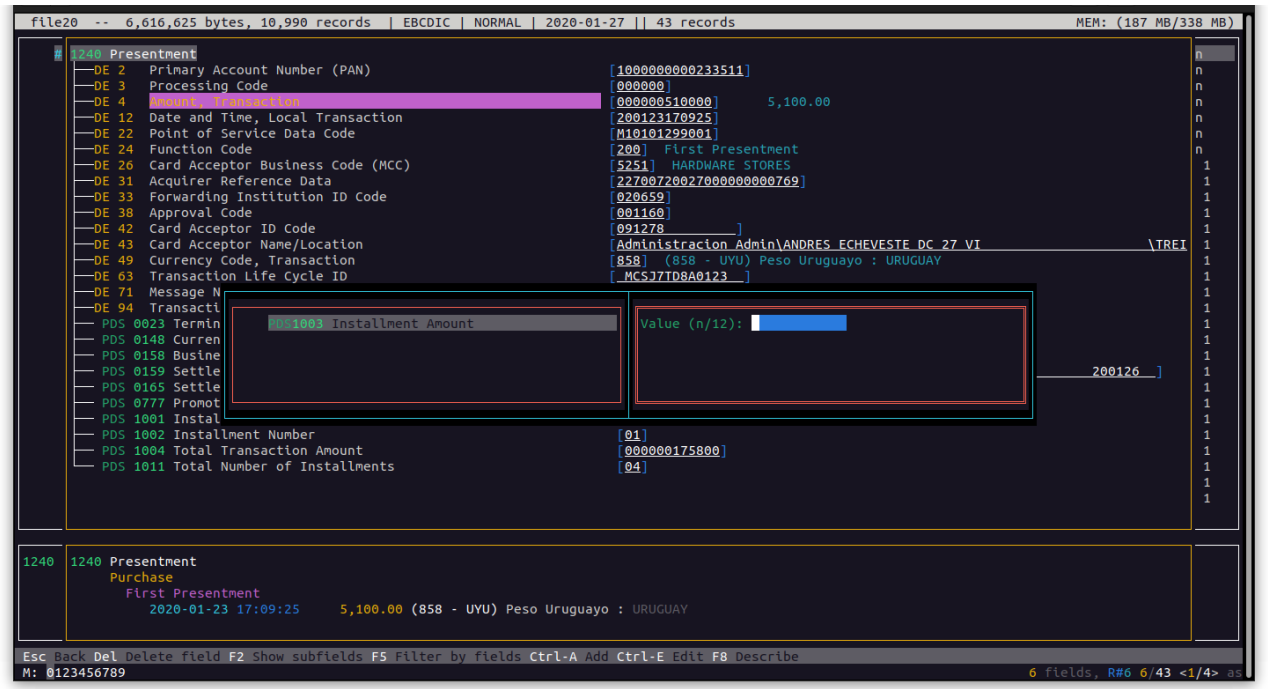




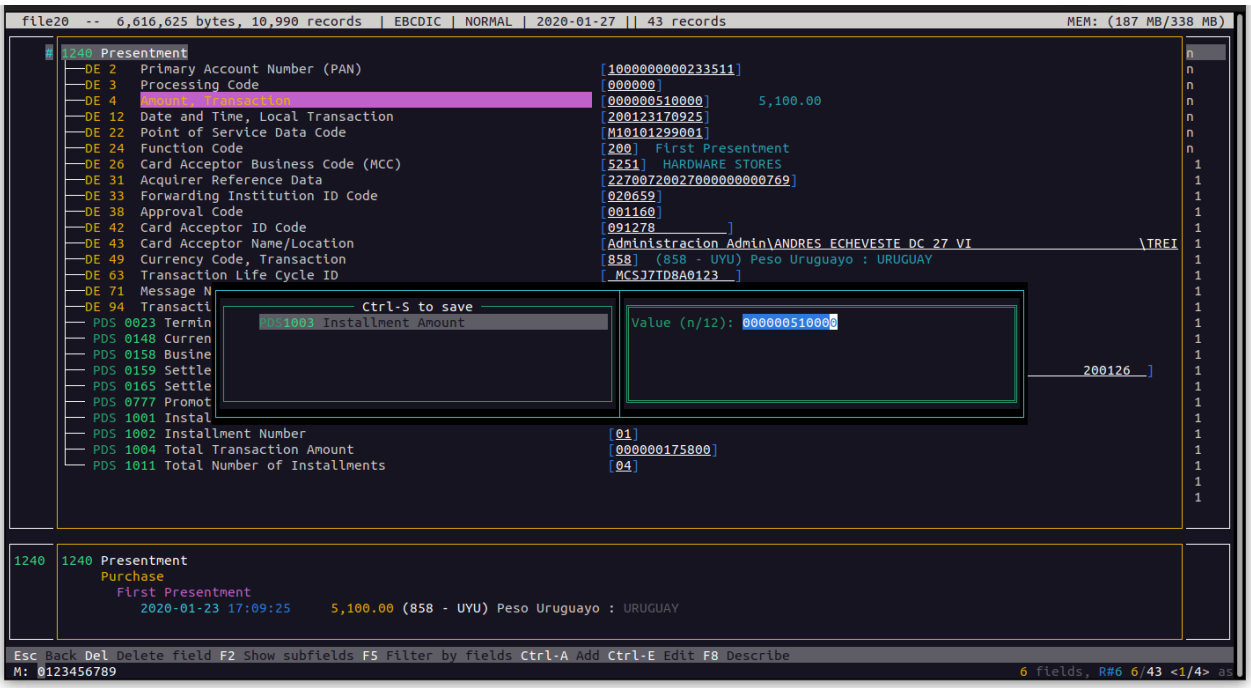
After changing the value, we see the new value and the description changes color to visually see that the field has been modified.

We can also add a field to a record. If we press **Ctrl-A** we will be presented with a list of possible fields to add (the existing ones are excluded). We select, for example, field 1003 (the one we previously deleted) and press **ENTER**

Then, we get a dialog of fields to add and on the right a place where to enter the values. Se can see the field data type and length as a reminder (in this case, 12 characters).

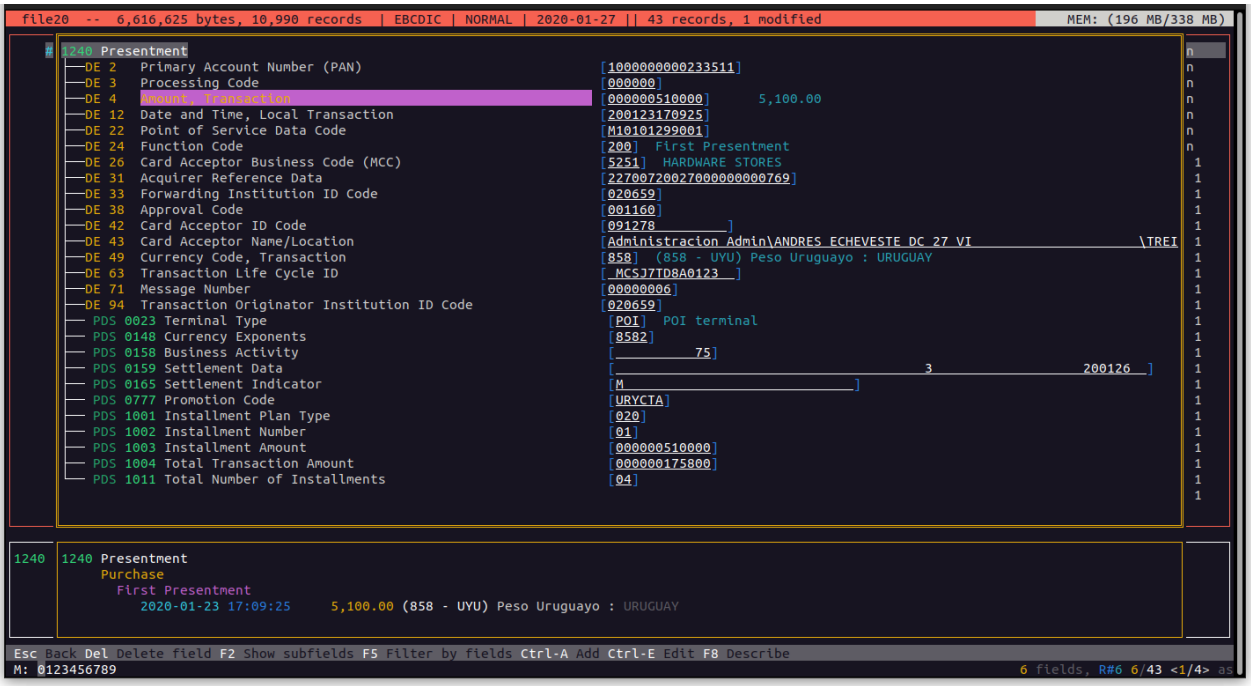


We then enter the new value, and when the value we enter is valid, the margin (initially red) turns int green, indicating that we have a valid value.



After all fields have been completed, we can press **Ctrl-S** to save the changes and insert the fields/

Again, we see the upper bar turns red to indicate unsaved changes in the file, and the border of the detailed pane turns into yellow to note changes in the record.

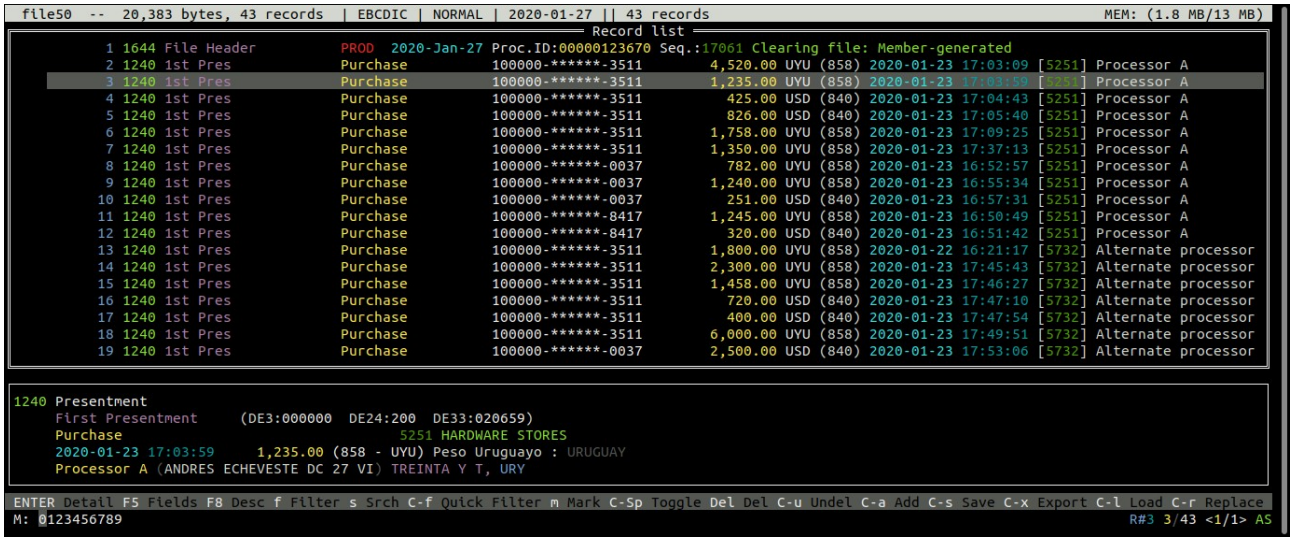


Returning to the main view (by pressing **Esc**) we see that record number 6 is marked with the character # to indicate a modified record.

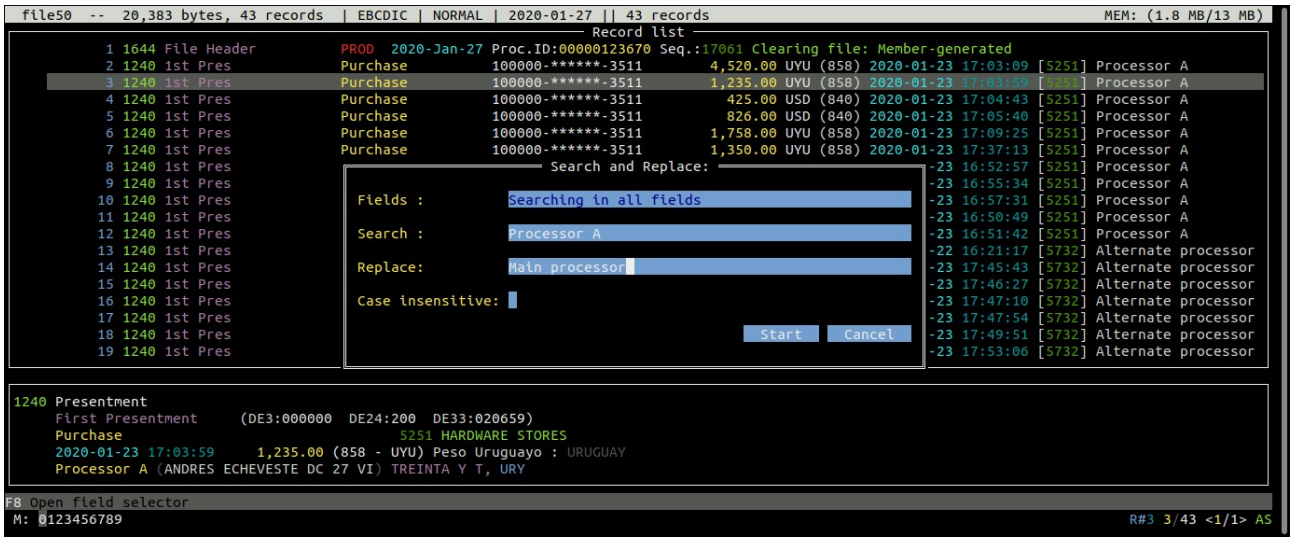
Replace values

By pressing **Ctrl-R** we are presented with a dialog box where we can enter a value to search for and a value to replace the original. The search entry can specify fields (like in a normal search) and it also can use regular expressions.

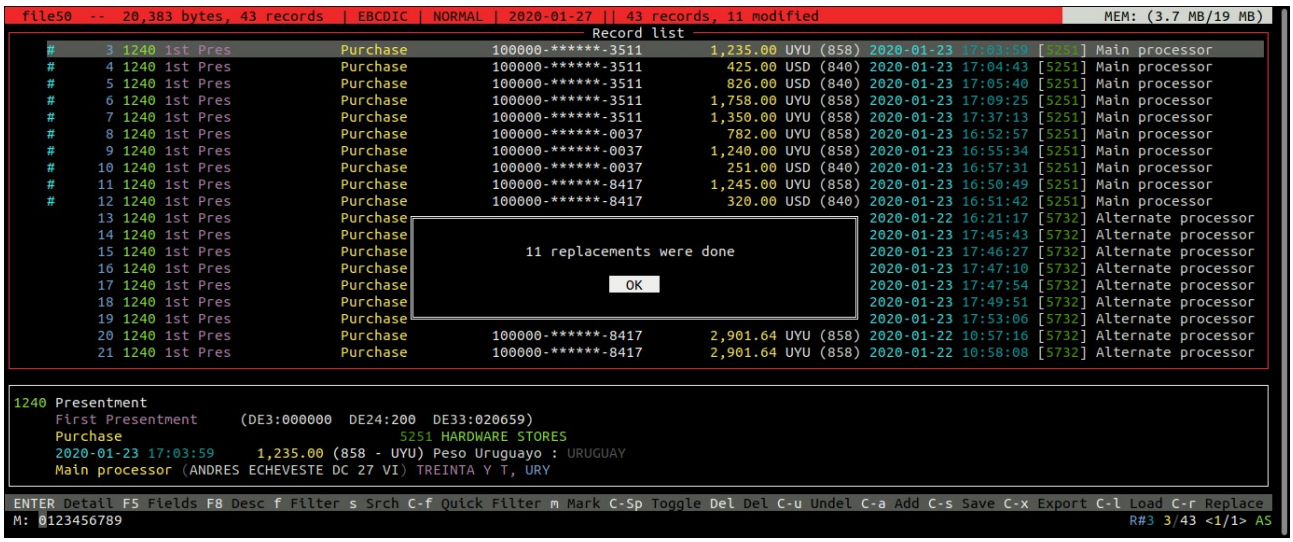
Let's take this file:



We want to replace the merchant name from "Processor A" to "Main processor". We press **Ctrl-r**



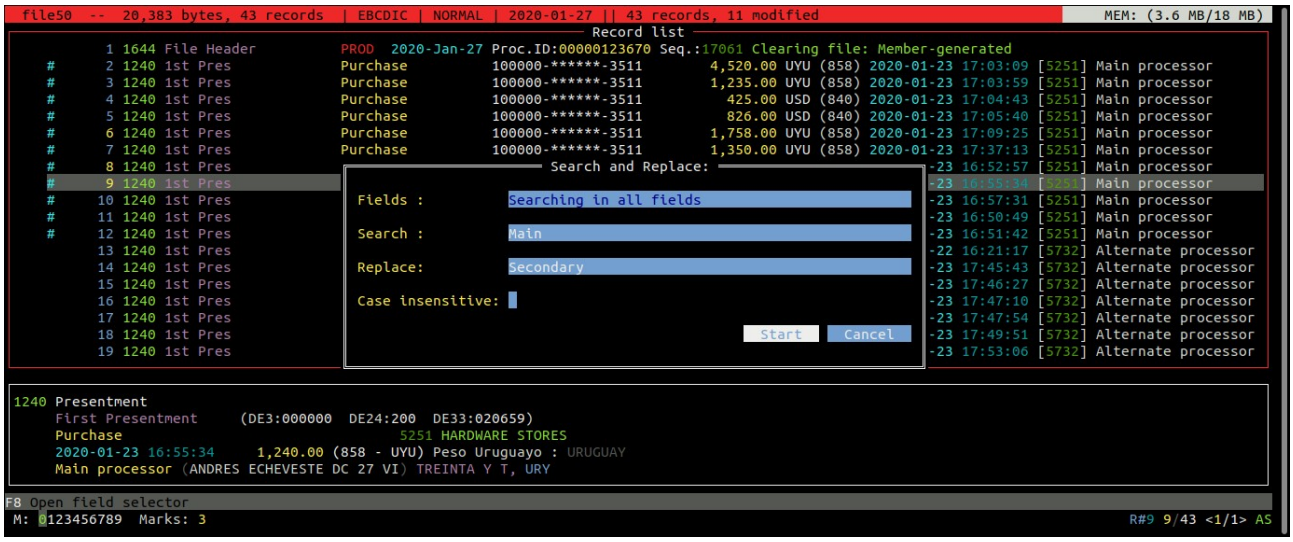
We fill the options and press the "Start" button, and the result is:



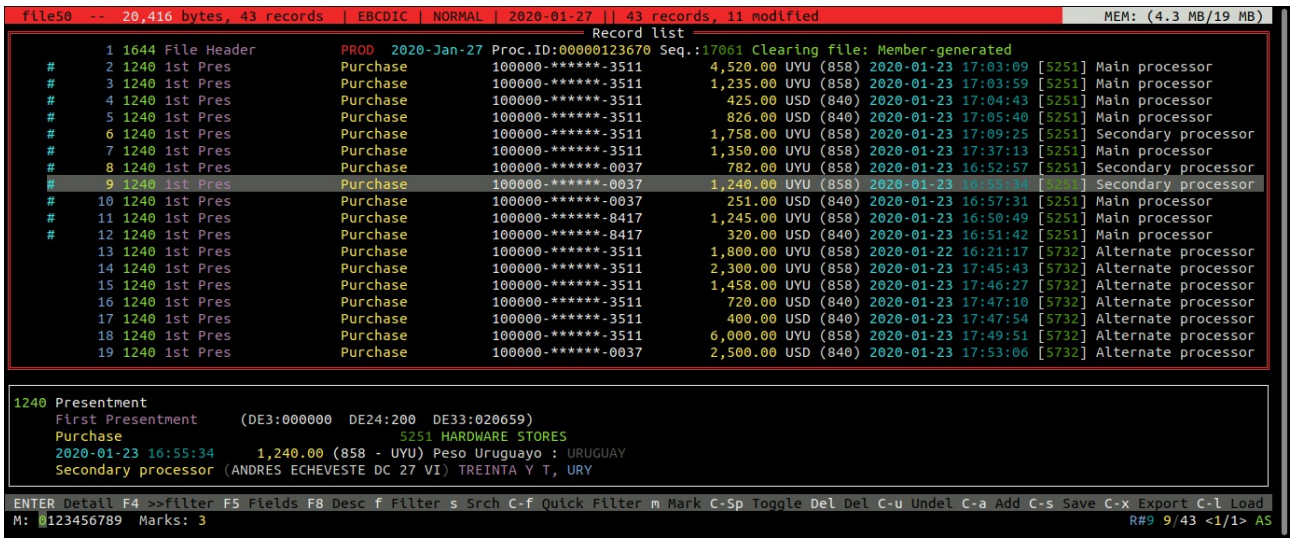
We are informed that 11 replacemens took place, and the upper bar turns into red to indicate modification in the file as usual. We can also see all the modified records with the character “#” in cyan.

We need to remember that in order to keep the changes, we need to save the file.

If we want to limit the replacement to just some records, we must mark them previously. In this case we mark records 6, 8 y 9 (we see the record numbers in yellow) and we replace “Main processor” for “Secondary processor” only in those records:



And here is the result:



Auto Save

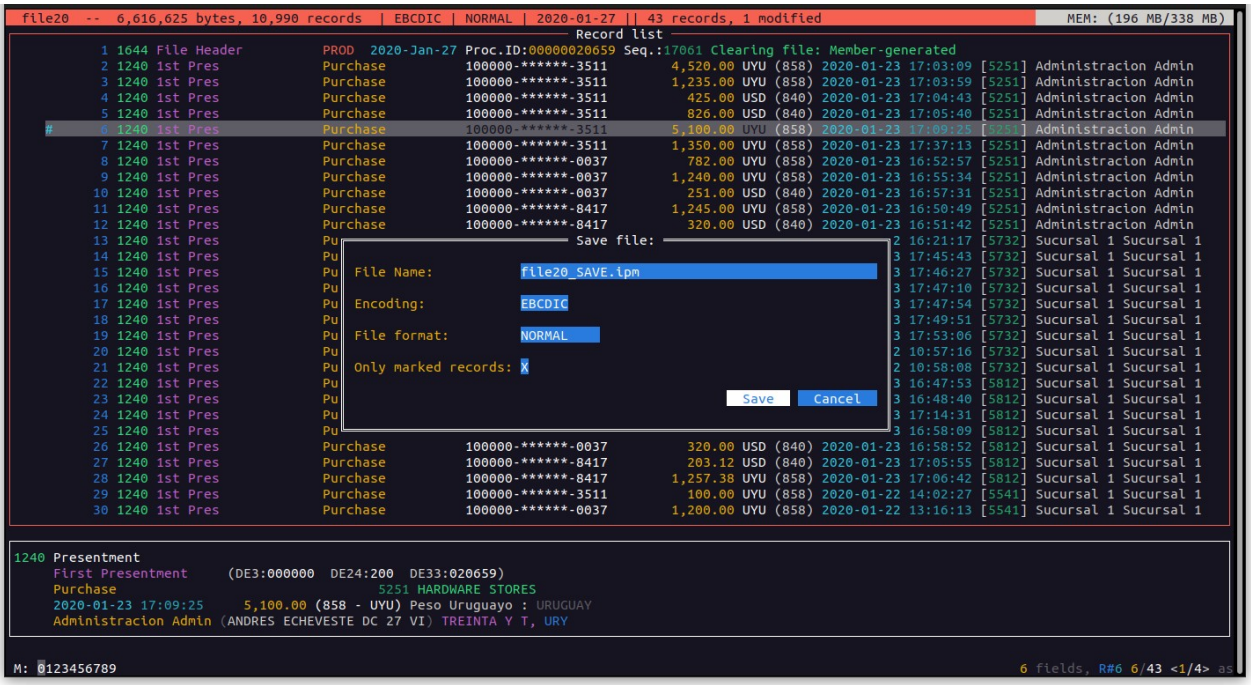
There is an auto save function which is automatically activated when the file contains less than 2000 recods, but can be activated any time by pressing **F6**. The reason it is not always activated automatically, is that when we have several records, saving to a new file can take some time, so as not to slow down normal operations, saving to a file is done manually. For really small files the time to save is bearly noticeable, so it is automatically activated.

When Auto Save is active, each modification is automatically saved in a file with the same name as the original, but adding “_AUTOSAVE.ipm” to the name

Manual Save

After modifying the file, we can opt to discard the changes or save them in a new file..

To save the file, we press **Ctrl-S** were we are presented with a dialog where we can select the name of the new file, the encoding and the file format. We can also choose to save just the marked records o all records (except the deleted ones).



By default, the new name is the same as the origiinal file plus “_SAVE.ipm”. The encoding and file format are initially the same as the original file, but we can choose to change them, so we will be converting the file between formats.

So this is an example of the utility of marking records. Another one si this:

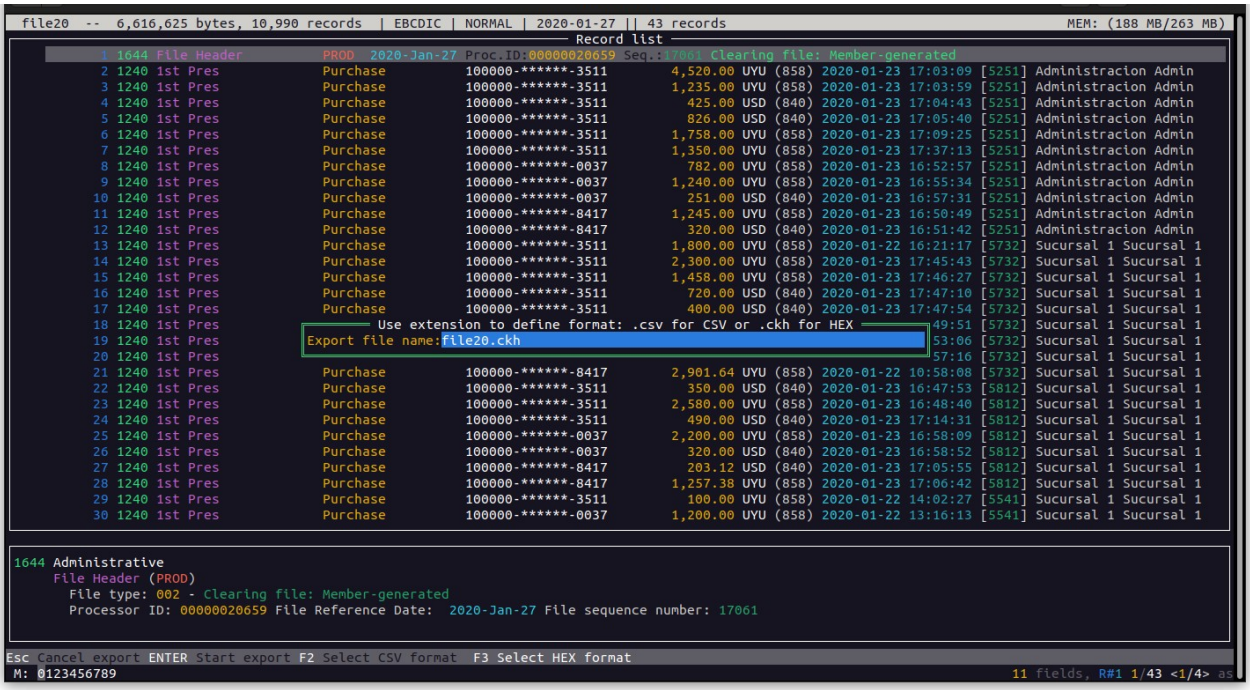
Export records

Pressing **Ctrl-X** we can export marked records or all records (if we don’t have any marked records).

The formats for the export files are the same as the ones used in the EXPORT command, that is, **CSV** or **HEX**. We can decide which format to use by changing the extension of the exported file name. If we choose “.csv” then the exported file will be a **CSV** file, and if the extension is “.ckh”, then the exported file will be a **HEX** file.

As a helper to easily change the extension, we can press the **F3** to toggle between the two extensions.

Selecting **CSV**, we need to remember that if a field filter is active, the exported file will only include the fields defined in the filter.



Import records

As with the IMPORT command, we can import records from a **CSV** or **HEX** file.

Files in **CSV** format must contain complete records, so they have to be exported including all fields, as partial records cannot be imported.

To import records, we press **Ctrl-L** and we are presented with a list of fields that contain exported records and can be imported, if any.

Pressing **ENTER** all records from the file will be appended (just before the trailer). We need to save the new file to save the changes.

Mark slots

In the bottom bar, on the left, we can see something like this:

This is an indicator of the “Mark slots” and an indicator of its contents. We have 10 different placeholders where we can save up to 10 list of marked records. Every time we mark some records, the list is saved to the active placeholder, which is number 0 by default, but we can change it anytime.

This allows us to hold up to 10 different selections, for example each one with the results of a different search, and when changing the slot we retrieve the list of marked records.

For example, we can perform 4 different searches and keep the results in separate areas. The, just by changing the active slot, the saved marks are restored accordingly.

To change the active slot, we just press the “m” key followed by a number from “0” to “9”

THIS PAGE INTENTIONALLY LEFT BLANK