BIREME / PAHO / WHO

Latin American and Caribbean Center on Health Sciences Information

# CISIS Utilities - Reference Manual

Version 5.2

Sao Paulo - 2005-2007

CISIS Utilities - Reference Manual



**Card catalog**

BIREME / PAHO / WHO (Brazil)

CISIS Utilities - Reference Manual. / BIREME (org.). Sao Paulo : BIREME / PAHO / WHO, 2005-2007.

188 p.

1. User manual. 2. Information access. 3. Information systems. 4. Information management. 5. Public health. 6. Public Health services. I. BIREME II. Title

**Warning** - Any mention in this document to companies, institutions, persons or products are not an endorsement or recommendation given by BIREME / PAHO / WHO, thus it does not mean a preference to a similar one, cited or not.

BIREME / PAHO / WHO

Latin American and Caribbean Center on Health Sciences Information

Rua Botucatu 862 V Clementino

*This document was produced with the Documents Conformation Methodology (NorDoc) developed by BIREME.*

# Table of contents

# Abbreviations used

- **ANSI**. American National Standards Institute.
- **ASCII**. American Standard Code for Information Interchange.
- **BIREME**. Latin American and Caribbean Center on Health Sciences Information.
- **BVS**. Biblioteca Virtual em Saúde (*see* VHL).
- **CGI**. Common Gateway Interface.
- **FST**. Field Selection Table.
- **HTML**. HyperText Markup Language.
- **HTTP**. HyperText Transfer Protocol.
- **ISO**. International Organization for Standardization.
- **MFN**. Master file number.
- **PAHO**. Pan American Health Organization.

- **STW**. STop Word file.
- **UNESCO**. United Nations Educational, Scientific and Cultural Organization.
- **URL**. Universal Resource Locator.

# How to use this manual

This manual has been conceived as a reference to the use of CISIS utilities and its contents are divided into four chapters, being:

1. Presentation: CISIS description, interface, installation, execution and conventions used by the utilities;
2. MX utility: with a full description of all parameters and functions available;
3. Master file utilities: describes the syntax and usage of the available utilities that operate with master files;
4. Inverted file utilities: describes the syntax and the usage of the available utilities that operate with inverted files.

There are also four Appendices containing additional and transversal information to the chapters and about the internal structure of ISIS.

A *Glossary* and an Abbreviations list complete the documentation.

# Preface

## About BIREME

Year after year, BIREME has been following its mission of being a center dedicated to scientific and technical health information for the region of Latin America and the Caribbean. Founded in Brazil in 1967, under the name of Regional Medicine Library (which the acronym BIREME comes from), it has always met the growing demand for up-to-date scientific literature from the Brazilian health systems and the communities of healthcare researchers, professionals and students. Then, in 1982, its name changed to Latin-American and Caribbean Center on Health Sciences Information so as to better express its dedication to the strengthening and expansion of the flow of scientific and technical health information across the region, but kept the acronym.

Networking, based on decentralization, on the development of local capacities, on sharing information resources, on developing cooperative products and services, on designing common methodologies, has always been the foundation of BIREME's technical cooperation work. It has been like this that the center established itself as an international model that fosters professional education with managerial and technical information with the adoption of information and communication paradigms that best meet local needs.

The main foundations that gave origin and which support the existence of BIREME are following:

- ✓ access to scientific and technical health information is essential for the development of health;
- ✓ the need to develop the capacity of Latin American and Caribbean countries to operate their sources of scientific-technical health information in a cooperative and efficient manner;
- ✓ the need to foster the use and to respond to the demands for scientific-technical health information from governments, health systems, educational and research institutions.

BIREME, as a specialized center of the Pan-American Health Organization (PAHO)/ World Health Organization (WHO), coordinates and conducts technical cooperation activities on the management of scientific information and knowledge with the aim of strengthening and expanding the flow of scientific health information in Brazil and in other Latin American and Caribbean countries as a key condition for the development of health, including its planning, management, promotion, research, education, and care.

The agreement that supports BIREME is renewed every five years by the members of the National Advisory Committee of the institution (PAHO, Brazilian Ministry of Health, Brazilian Ministry of Education and Culture, Secretary of Health of the State of São Paulo, and Federal University of São Paulo – Unifesp). The latter provides the physical infrastructure necessary for the establishment of the institution.

In 2004 the institution took on the responsibility of becoming a knowledge-based institution.

# The Virtual Health Library (VHL)

With the rise and consolidation of the internet as the prevailing means of access to information and communication, BIREME's technical cooperation model evolved,

as of 1998, to build and develop the Virtual Health Library (VHL) as a common space for the convergence of the cooperative work of producers, intermediaries, and users of information. The VHL promotes the development of a network of sources of scientific and technical information with universal access on the internet. For the first time there has been a real possibility of equal access to health information.

To BIREME, the Virtual Health Library is a model for the management of information and knowledge, which includes the cooperation and convergence between institutions, systems, networks, and initiatives of producers, intermediaries, and users in the operation of networks of local, national, regional and international information sources favoring open and universal access.

Today, every country in Latin America and the Caribbean (Region) participates either directly or indirectly in the cooperative products and services offered by the VHL, which includes over 1,000 institutions in more than 30 countries.

The VHL is simulated in a virtual space of the internet formed by a collection or network of health information sources in the Region. Users of different levels and locations can interact and navigate in the space of one or many information sources, regardless of where they are. Information sources are generated, updated, stored and operated on the internet by producers, integrators, and intermediaries, in a decentralized manner, following common methodologies for their integration in the VHL.

The VHL organizes information in a structure that integrates and interconnects reference databases, specialist directories, events and institutions, a catalogue of the information resources available on the internet, collections of full texts with a highlight for the SciELO (*Scientific Electronic Library Online*) collection of scientific journals, selective information dissemination services, information sources to support education and decision-making, news, discussion lists, and support to virtual communities. The space of the VHL is, therefore, a dynamic and decentralized network of information sources based on which it is possible to retrieve and extract information and knowledge to support health decision-making processes.

The Virtual Health Library can be visualized as a distributed base of scientific and technical health knowledge that is saved, organized and stored in electronic format in the countries of the Region, universally accessible on the internet and compatible with international databases.

# Presentation

## CISIS - Interface

*CDS/ISIS for Mini-microcomputers* (also known as *MicroISIS* in Latin America) is a software program developed by UNESCO for databases mostly consisting of text. CDS/ISIS handles fields (data elements) of variable length. A field may be absent in one or more records, it may contain only one element of data, or two or more subfields of variable length. A field may also be repeatable, that is, a given record may contain more than one occurrence of a field.

The CISIS Interface is a library of functions, written in the C programming language, designed to allow the development of CDS/ISIS database applications (without calling the CDS/ISIS software). CISIS applications are fully compatible with CDS/ISIS, including multi-user applications.

There are diverse implementations of the original CDS/ISIS structure: for that reason nowadays it is more appropriate to refer to these variants as the “Isis family”. It is important to emphasize that the data created under any of these “family” variants are compatible and can be interchanged.

Applications developed with the CISIS Interface can manipulate various databases at the same time; the master file and the inverted file are processed

independently. It is not necessary to have the database definition in order to run the CISIS applications.

The CISIS Interface and the CISIS Utility Programs were designed and implemented at BIREME, the Centre of Health Science Information for Latin America and the Caribbean, and currently they are available for the following platforms:

- IBM-compatible PC 32 bits
- UNIX using Intel processors (LINUX, SCO, etc.)
- UNIX using other processors (HP-UX, Sun, IBM-AIX, CDC/S4320, etc.)
- VAX under VMS
- HP3000/950 under MPE/XL

# CISIS – Utility Programs

The CISIS Utilities are a group of programs developed in the C programming language that call functions offered by the CISIS Interface in order to carry out different functions on the Isis family of databases, such as finding and displaying records, maintenance of databases, etc. They can also carry out special functions that allow you to arrange master file records, generate tables from a master record, change the field tags etc.

This group of utility programs is offered under four versions: 10/30 and 16/60, LIND, FFI. The main differences are in the length of the inverted file keys and the maximum supported record size measured in bytes, as is shown in the following table.

| | **10/30** | **16/60** | **LIND** | **FFI** |
|---|---|---|---|---|
| **Inverted file key length** | 30 | 60 | 60 | 60 |
| **Maximum record size** | 32.767 | 32.767 | 32.767 | 1.048.576 |

*Note: version 10/30 is the only one compatible with CDS/ISIS from UNESCO*

For more details on the structure of the master and inverted files see the Appendix: *Structure of ISIS database records.*

The particular characteristics of these programs can be verified in the version declaration that you can obtain with *what*

For example

```
mx what
CISIS Interface v5.2a/PC32/M/32767/10/30/I - Utility MX
CISIS Interface v5.2a/.iy0/Z/4GB/GIZ/DEC/ISI/UTL/INVX/B7/FAT/CIP/CGI/MX
Copyright (c)BIREME/PAHO 2006. [http://www.bireme.br/products/cisis]
```

| **Acronymn** | **Description** |
|---|---|
| V5.2a | Version number |
| PC32 | Computer used (in this case Windows PC) |
| L | Lind version if it is present |
| M | Multi-user support |
| 32767 | Maximum size of the record in bytes, value by default |
| 10/30 | Inverted file keys |
| I | Permission to update the I/F |
| Utility MX | Name of the program |
| .iy0 | Single physical file for I/F (made for mkiy0) |
| Z | Compressed I/F (made for myz – discontinued) |
| 4GB | Maximum size of the master file |
| GIZ | Gizmo |
| DEC | Decod |
| ISI | Iso-2709 import |
| UTL | Ciutl module |
| INVX | Multiple I/F searching |
| B7 | Version of the internal search mechanism |
| FAT | Fatal() |
| CIP | Cipar() |
| GCI | Supports operation in a CGI environment |
| MX | Cisis_mx() |

## MX Utility

`MX` The MX Program is a general purpose utility for working with MicroISIS databases. It can carry out most of the CISIS Interface functions, including the import/export of ISO-2709 records, searches, processing global changes of patterns, joining records from a master file by record number or key from the inverted file, incorporating fields with data generated through a field selection table (FST), and functions relating to the editing of fileds.

## Utilities for the master file

`MXF0` Analyzes all the records of a given master file, producing information on fields present and the characters used in them.

`MXTB` The MXTB program counts the content of the fields, for example, number of times that each author appears, each descriptor, or the combination of an author and title of the publication, etc.
The result of running MXTB is a master file that contains a record for each different phrase found (category). These records have fields for storing the category and its frequency.

`MXCP` Copies records from an input master file to an output master file,

## Utilities for the master file

allowing you to input data to be modified by global editing and/or procedures that supress spaces at the beginning and end, blank spaces, non-printable characters and final punctuation characters.
It also converts fields that contain a specific delimeter into repeatable fields and it can discard input fields, according to the values of their tags.
Another characteristic of MXCP is the recovery (*undelete*) of records deleted from a master file.

`MSRT` — Orders the records in a master file in an ascending form, according to keys that are generated by applying a format to the records.

`RETAG` — This program has two functions :
Change the tags of the fields in a given master file, according to a renumbering table.
Unlock a master file.

`CTLMFN` — Displays and updates the master file control record.
It can be used when a master file is reinitialized by accident.

`MKXRF` — A program for restoring the master file, that reads the *.mst* file and creates the corresponding *.xrf* file.
It can be used to restore all the active records in a master file, which has been logically reinitialized.

`I2ID` — Reads a master file and generates an ASCII file that can be edited and modified.
It works together with the *ID2I* utility that carries out the opposite function: reads an ASCII file and converts the data into a master file.

`ID2I` — Reads an ASCII file generated by *I2ID* (or with the same structure as a file generated by this) and converts those data into master file records.

`CRUNCHMF` — Converts the master file from one operating system to another, for example from Windows to Linux.

## Utilities for inverted files

`IFKEYS` — Displays the terms from the inverted file and the number of *postings* of each of them.
Optionally the terms can be selected by the tags they were extracted from.

`IFLOAD` — Loads an inverted file from the link file, according to the processing options. It accepts other formats as well as the standard CDS/ISIS link file.

`MYS` — Sorts the link file in order to create the inverted file.

| | |
|---|---|
| IFMERGE | Combines various inverted files from different master files into a single inverted file, with a procedure to recover the records from the source master files. |
| MKIY0 | Combines the six files that make up the inverted file into a single file. |
| CRUNCHIF | Converts the inverted file from one operating system to another, for example from Windows to Linux. |

# Installation of the CISIS utilities

The installation of the CISIS utilities consists of creating a directory, usually \CISIS\SYS\, and copying all the utilities into this.

For convenience it is possible to add the \CISIS\SYS directory to the operating system PATH, so that it is possible to run the utilities from the location you are in, without having to reference the \CISIS\SYS directory.

**Examples**

The examples are mostly based on the database CDS, and presume that it is located in the directory:

```
\CISIS\DATABASES\
```

They are carried out on the database and often will modify it, therefore it is advisable to make a backup copy.

# Execution of the utilities

The CISIS program is executed by a command, from the operating system *prompt*, or from *batch* files from MS-DOS or *scripts* (shell scripts) in UNIX.

Any program that uses CISIS can be executed by entering its name and one or more parameters, if the directory \cisis\sys (the directory where the CISIS utilities are found) is included in the system PATH list. If you do not provide parameters in the command call, each utility program displays a brief description of its use. For example, typing only the name MXCP at the DOS prompt displays:

```
CISIS Interface v5.2a/PC32/M/32767/10/30/I - Utility MXCP
Copyright (c)BIREME/PAHO 2006. [http://www.bireme.br/products/cisis]
```

```
mxcp {in=<file>|<dbin>} [create=]<dbout> [<option> [...]]

options: {from|to|loop|count|tell|offset}=<n>
         gizmo=<dbgiz>[,tag_list>]
         undelete
         clean [mintag=1] [maxtag=9999]
         period=.[,<tag_list>]
         repeat=%[,<tag_list>]
         log=<filename>

 Ex: mxcp in create=out clean period=.,3 repeat=;,7

      in = 3 «   Field 3 occ 1. »
           3 «Field 3 occ 2 . »
           7 « Field 7/1;Field 7/2 ;Field 7/3.»

     out = 3 «Field 3 occ 1»
           3 «Field 3 occ 2»
           7 «Field 7/1»
           7 «Field 7/2»
           7 «Field 7/3.»
```

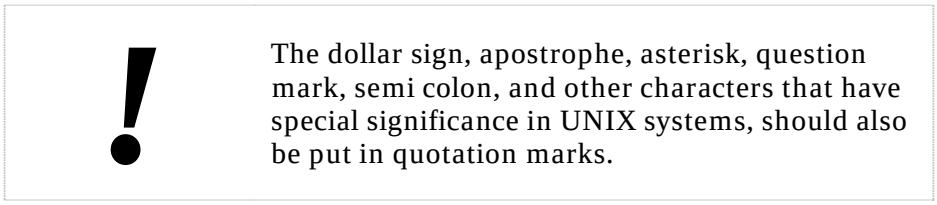
The parameters are displayed as a list separated by blank spaces and, for that reason, each individual parameter should be put in quotation marks when it contains blank spaces or any special system character (such as angle brackets, pipe, etc.).

The following example executes the MX program with three parameters (database name, search expression and a display format specification):

```
mx \cisis\databases\cds "plants*water" "pft=mfn/,'Ti: 'v24/,(|Au: |v70/)"
```

In order to use the colon as part of the parameter, it should be preceded by the pipe:

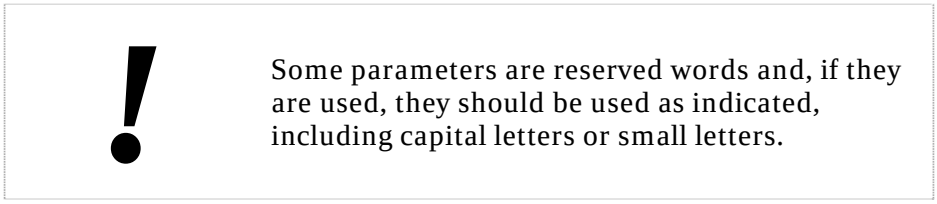
```
mx \cisis\databases\cds "plants*water" "pft=mfn/, \" Ti: \"v24/,(|Au: |v70/)"
```



# Syntax conventions

The following conventions are used to describe the syntax of the CISIS Utility Programs:

| | |
|---|---|
| <parameter> | mandatory parameter |
| [<parameter>] | optional parameter |
| {<option 1>\|<option 2>} | can choose between < option 1> or < option 2> |
| <option> [...] | <option> can repeat |



Some parameters are reserved words and, if they are used, they should be used as indicated, including capital letters or small letters.

For example, MXCP has the general syntax:

```
mxcp <dbin> [create=]<dbout> [<option> [...]]

options:

{from|to|loop|count|tell|offset}=<n>
gizmo=<dbgiz>[,<tag_list>]
undelete
clean [mintag=1] [maxtag=9999]
period=.[,<tag_list>]
repeat=%[,<tag_list>]
log=<filename>
```

showing that two parameters are mandatory: (a) name of the input database and (b) name of the output database.

Then, the command:

```
mxcp \cisis\databases\cds newcds
```

copies the master file *cds* located in the directory \cisis\bases to the master file *newcds* located in the same directory. The master file *newcds* must already exist, otherwise an error will be produced.

If *newcds* does not exist you can create it using the optional parameter *create* as can be seen in the following example:

```
mxcp \cisis\databases\cds create=newcds
```

The process options can be indicated using the optional parameters. To indicate, for example, the range of records to process use *from* and *to*

```
mxcp \cisis\databases\cds create=newcds from=10 to=20
```

# MX Utility

## Presentation

### Introduction

MX is a general use program for CDS/ISIS databases that carries out most of the CISIS Interface functions. Similarly to the other CISIS utility programs, MX is executed from the operating system command line, indicating the operations to carry out with parameters.

MX is used, for example, to search for and show a set of database records, according to a search expression and a display format, as in the following line:

```
mx \cisis\databases\cds "plants * water" "pft=mfn,x1,v24/"
```

Also, MX allows free-text searches even if an inverted file does not exist.

MX can also read ISO-2709 files or ASCII files, using delimiters as field separators. In these cases the input records are converted to master file records through which they are read.

The following procedures can be applied to the input records:

5. Global change of patterns.
6. Joining of records, by record number or inverted file key.

7. Add fields with data generated by a field selection table.
8. Import and export of records, specified through a format language.

Records processed by MX can be sent to a master file, an ISO-2709 file or to a standard output (which can be directed to a file or printer). Lines produced by a format can be sent to the operating system.

The execution of MX can generate a call to the operating system so that a certain program is run.

The result of applying a Field Selection Table (FST) to a master file can be sent to a link file or combined into an inverted file.

The output file can be the same as the input.

MX also works in multi-user environments.

## General description

In order to implement MX it is necessary to specify where the data are located that it will work on. You can provide a master file, an ISO-2709 file or a text file. This is the only mandatory parameter for the program.

The line

```
mx \cisis\databases\cds
```

generates an on-screen list of the *cds* database, that is found in the\cisis\databases directory. The records are displayed without formatting.

Other processing parameters can be specified, for example:

```
mx \cisis\databases\cds from=10 to=20
```

presents the records 10 to 20 of the database *cds* on the screen. The database can be found in the directory \cisis\databases and the records are displayed without formatting.

The command line

```
mx \cisis\databases\cds from=10 to=20 "pft=mfn,x1,v24(0,7)/"
```

displays records 10 to 20 from the *cds* database on screen, applying the format specified in the parameter pft=mfn,x1,v24(0,7)/. The database is found in the \cisis\databases directory.

It is important to bear in mind that the order in which the optional parameters are entered does not affect the execution of MX. The execution of these parameters is made in the order in which they appear in the syntax declaration.

So the previous line could be:

```
mx \cisis\databases\cds   to=20   "pft=mfn,x1,v24(0,7)/"   from=10
mx \cisis\databases\cds   "pft=mfn,x1,v24(0,7)/"   from=10   to=20
```

The following declarations are equivalent:

```
mx \cisis\databases\cds  pft=@file1  proc=@miproc.prc
mx \cisis\databases\cds proc=@miproc.prc  pft=@file1

mx \cisis\databases\cds gizmo=gizfile1  proc=@miproc.prc
mx \cisis\bases\cds   proc=@miproc.prc  gizmo=gizfile1
```

the gizmo parameter can be applied before the proc parameter, because it is identified by the syntax.

If the first parameter is a database and its corresponding inverted file is available, the set of records to be processed can be obtained through a search.

The following example returns *cds* database records, that are found in the \cisis\databases directory, containing the words plants and water.

```
mx \cisis\databases\cds "plants * water"
```

MX can read input data from a ISO-2709 file or a delimited text file.

The following line displays the first five records of an ISO-2709 file called *cds.iso*, that is found in the \cisis\databases directory.

```
mx iso=\cisis\databases\cds.iso to=5
```

In the next example MX uses an ASCII file called *name* as its input source, whose content is:

```
Author 1|title 1|^aParis^bUnesco^c1965
        |title 2|^aParis^bUnesco^c1965
Author 3|title 3|^aParis^bUnesco^c1965
```

This can be executed with the following call to MX:

```
mx seq=name "pft=mfn,c11,v1,c21,v2,c31,v3/" now
```

That generates the output:

```
000001     Author 1  Title 1   ^aParis^bUnesco^c1965
000002               Title 2   ^aParis^bUnesco^c1965
000003     Author 3  Title 3   ^aParis^bUnesco^c1965
```

The processed records can be stored in a master file. The following lines create a master file *sample*:

```
mx\cisis\databases\cds "plants * water" create=sample -all now
mx iso=\cisis\databases\cds.iso to=5 create=sample -all now
mx seq=name create=sample -all now
```

These records, can also be exported to anISO-2709 file, for example *sample.iso:*

```
mx \cisis\databases\cds "plants * water" iso=sample.iso -all now
mx iso=\cisis\databases\cds.iso to=5 iso=sample.iso -all now
mx seq=name iso=sample.iso -all now
```

When MX carries out one or more processes that modify records (whether read from a database, ISO-2709 file or a text file), these modifications are carried out in memory and **do not modify the database**, unless explicitly indicated.

The modified records can be viewed on the screen or written to an output file.

The main modification processes are:

- procedures for changing global patterns (gizmo).
- joining records (or part of them) from another database (join).
- carrying out field operations (proc).
- applying a field selection table from CDS/ISIS and aggregating the results in a file in memory (fst).

The following example shows records from the master file *cds*, and indicates the number of records to display using the keyboard.

- MS-DOS:

```
mx seq=con "join=cds='mfn='v1" "proc='D1/1D32001'"
```

- UNIX:

```
mx seq=/dev/ttyp0 "join=cds='mfn='v1" "proc='D1/1D32001'"
```

MX can carry out the record modifications on the same master file that it uses as input:

```
mx cds "proc='D24'" copy=cds -all now
```

The example deletes the field tag 24 from all the cds database records, making the changes on the same database.

MX can take parameters from a text file, allowing it to exceed the limitations of the operating system that are described here:

1. A call to MX has more characters than is permitted in a operating system command line (128 characters in MS-DOS 512 characters in UNIX).
2. The command line that is added by keyboard contains special operating system characters.

The following example shows how to use a parameters file:

```
mx in=somefile
```

Where the file *somefile* contains:

```
\cisis\databases\cds
proc='D1'
copy=\cisis\databases\cds
-all
now
```

# Syntax

MX version 5.2a, syntax table:

```
CISIS Interface v5.2a/PC32/L/M/32767/16/60/I - Utility MX
Copyright (c)BIREME/PAHO 2006. [http://www.bireme.br/products/cisis]

mx [cipar=<file>] [{mfrl|load}=<n>] [cgi={mx|<v2000_fmt>}] [in=<file>]
   {[db=]<db>|
    seq[/1m]=<file>|
    iso[={marc|<n>}]=<isofile> [isotag1=<tag>]|
    dict=<if>[,<keytag>[,<posttag>[/<postsperrec>]]] [k{1|2}=<key>]}

options:
   {from|to|loop|count|tell|btell}=<n>
   text[/show]=<text>
   [bool=]{<bool_expr>|@<file>}  [invx=<tag101_mf>] [tmpx=<tmp_mf>]

   gizmo=<gizmo_mf>[,<taglist>]  [gizp[/h]=<out_mfx>] [decod=<mf>]

   join=<mf>[:<offset>][,<taglist>]=<mfn=_fmt>
   join=<db>[:<offset>][,<taglist>]=<upkey_fmt> [jmax=<n>]
   jchk=<if>[+<stwfile>]=<upkey_fmt>

   proc=[<proc_fmt>|@<file>]

         D{<tag>[/<occ>]|*}
         A<tag><delim><data><delim>
         H<tag> <length> <data>
         <TAG[ <stripmarklen>[ <minlen>]]><data></TAG>

         S[<tag>]
         R<mf>,<mfn>
         G<gizmo_mf>[,<taglist>]
         Gsplit[/clean]=<tag>[={<char>|words|letters|numbers|trigrams}]
         Gsplit=<tag>=6words[/if=<if>]
```

```
        Gload[/<tag>][/nonl][=<file>]
        Gmark[/<tag>]{/<elem>|/keys|/decs|/<mf>,<otag>[,<ctag>]}=<if>
        Gmarx[/<tag>]/<elem>[@<att>="x"] =<tag>[:&[<att>]|/c[=224]|/i]
        Gdump[/<tag>][/nonl][/xml][=<file>]
        =<mfn>
        X[append=]<mf>

   convert=ansi [uctab={<file>|ansi}]         [actab={<file>|ansi}]
   fst[/h]={<fst>|@[<file>]}                  [stw=@[<file>]]

   [mono|mast|full] {create|copy|append|merge|updatf}=<out_mf>
   [out]iso[={marc|<n>}]=<out_isofile>        [outisotag1=<tag>]
   fullinv[/dict][/keep][/ansi]=<out_if>      [maxmfn=<n>|master=<mf>]
   ln{1|2}=<out_file>                         [+fix[/m]]
   fix=<out_file> tbin=<tag>
   tab[/lines:100000/width:100/tab:<tag>]=<tab_fmt>
   {prolog|pft|epilog}={<diplay_fmt>|@<file>} [lw={<n>|0}]

   {+|-}{control|leader|xref|dir|fields|all} mfrl now
```

MX takes the parameters in the order shown in the table. In first place should be, the initialisation (setup) parameters, followed by the source of the input data, and finally the optional processing parameters. There are some exceptions that are pointed out in the manual, for example *btell*= should go before *bool*=.

## Parameters. General description

If you enter the name of the MX program without parameters, a menu of all possible options and a brief description of their use is displayed, as is shown in the previous section.

## Initialization parameters (setup)

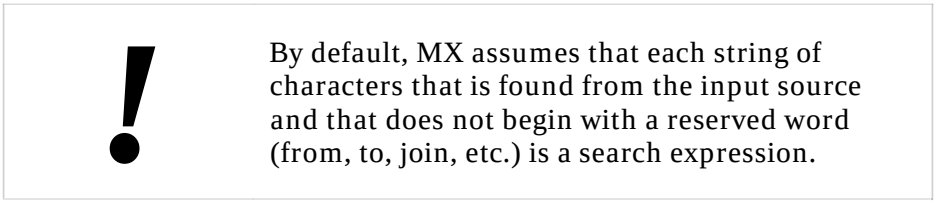
When one or more of the optional initialization parameters (files, mfrl, fmtl, load) are present, they should be placed before any other parameter.

## Parameters that indicate the database source

A mandatory parameter that indicates the database source (database name, ISO-2709 file or text file), should be the first parameter, except for the initialization parameters, in which case it should be entered immediately after.

## Parameters for processing data

Optional parameters that carry out tasks on the input data. In the command line these follow the parameter that indicates the source.




Processing parameters can be classified as:

### Parameters for searching records

With these parameters you define a subgroup of source data on which to perform a task. The method of defining this subgroup can be by:

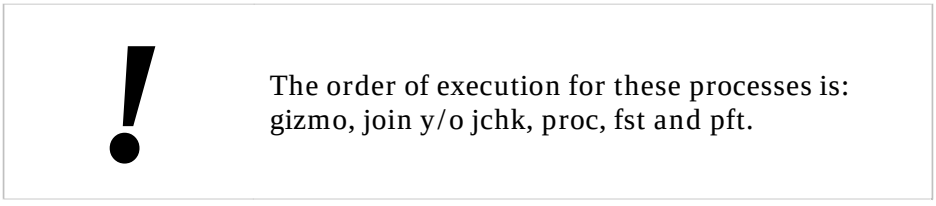
- A search (*bool*)
- A free-text search expression (*text*)
- A range of records (whose limits are indicated with *from*, *to*)
- Number of records (*count*)
- Repeating for each record until a condition is met (*loop*)

### Parameters that carry out processes

These are parameters which call internal processes that carry out tasks in memory on a set of records.

These tasks could be:

- Carry out global changes (gizmo)
- Join records (join)
- Compare master files with inverted files (jchk)
- Carry out modifications in the record fields (proc)
- Apply the field selection table (fst) to records
- Apply formats to the records (pft)



### Parameters for data output

These are the parameters that, for example, indicate:

- The output database (create, copy, append, etc.)
- The name of an output file ISO-2709 (iso)
- The name of a link file (ln1, ln2)
- Calls to the operation system (sys)

### General parameters

These are parameters that carry out general tasks, for example:

- Deactivate the *prompt* (interaction with the user) between the processed records (*now*)
- Change the mx prompt text (*p1, p2*)
- Modify display options (+*fields*, +*all*, -*all*, etc.).
- Redirect the standard output (>, >>)
- Continue the execution of CISIS (*trace=rec*, *trace=all*, etc.)

# Parameters that indicate the input source

## Input database

**<[db=]<db>>**

Specifies the master file to be read. Processes are carried out on records in this master file.

```
C:\isis\data> mx cds
```

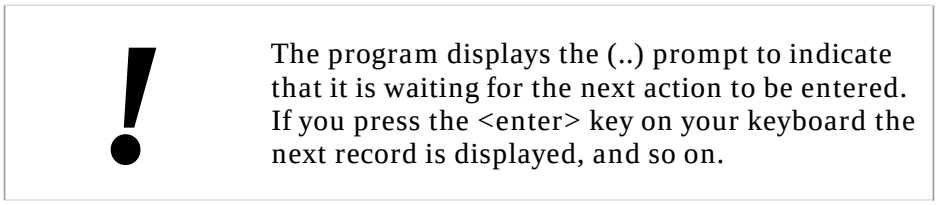
or

```
X:\otherdirectory> mx C:\isis\data\cds
```

Generates unformatted output, showing all the fields for each record in turn.

```
mfn= 1
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
50 «Incl. bibl.»
69 «Paper on: <plant physiology><plant transpiration> <measurement and
instruments>»
24 «Techniques for the measurement of transpiration of individual plants»
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
```

```
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
..
```



The program displays the (..) prompt to indicate that it is waiting for the next action to be entered. If you press the <enter> key on your keyboard the next record is displayed, and so on.

```
mfn= 2
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
50 «Incl. bibl.»
69 «Paper on: <plant evapotranspiration>» 24 «<The> Controlled climate in the
plant chamber and its influence upon assimilation and transpiration»
26 «^c1965»
30 «^ap. 225-232^billus.»
70 «Bosian, G.»
..
```

At the *prompt(..)* it is possible to take three actions:

a) Continue displaying records by pressing <enter>
b) Enter a lowercase x and press <enter> to exit the program.
c) Any other data entered will be interpreted as a search expression. On pressing <enter> MX will carry out the search and display the records returned.

## ISO-2709 input file

**iso[={marc|<n>}]=<isofile> [isotag1=<tag>]**

In the previous section the input source was considered as a database in the CDS/ISIS format. The MX program can also read files in the ISO-2709 format and apply the same processes as on master files (excepting those processes that require the use of the inverted file or dictionary, for example, a search).

Each record in ISO-2709 format that is entered is transformed internally into a record in ISIS format, on which it will work.

The field separators and the record separators of the ISO-2709 records **are not** taken into account by MX.

**Examples:**

- Browse an ISO-2709 file:

```
mx iso=\isis\sys\cds.iso
```

Browses an ISO-2709 file called *cds.iso* that is found in the directory *\isis\sys*.

- Read an ISO file and create a master file with its content. The *prompt* is cancelled by the *now* (no wait) parameter.

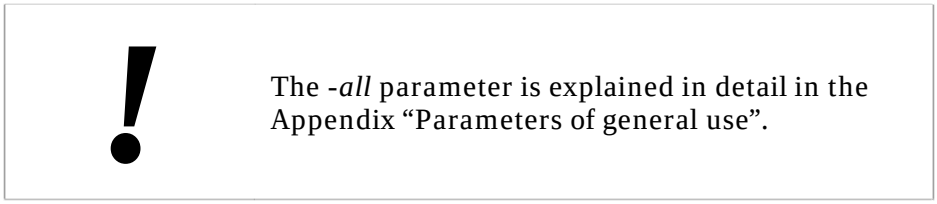
```
mx iso=cds.iso create=newcds now
```

The *now* parameter is explained in detail in the Appendix “Parameters of general use”.

- Read an ISO-2709 file that is found in the current directory and create a master file called *newcds* with the data from the ISO file.

  This example carries out the same process as the previous one but without displaying the information on the screen. The parameter *-all* deactivates the display of information on the screen.

```
mx iso=cds.iso create=newcds now -all
```



The *-all* parameter is explained in detail in the Appendix “Parameters of general use”.

- Read an ISO file and create a master file called *newcds* with the input records, that begin with mfn 1001. Similar to the previous example it does not display information on the screen (*-all*) nor does it stop between each record waiting for user intervention (*now*).

```
mx iso=cds.iso create=newcds from=1001 -all now
```

## Fixed line length

MX can read ISO files with a fixed line length. This option is used to interchange PC ISO files with computers and software that requires it (such as the HP MINISIS). In order to read files with fixed line length, indicate the length of the line with the *ISO* parameter:

```
mx iso=80=cds.iso create=newcds now -all
```

The example reads the ISO file expecting the line length to be 80 characters.

If the line length is variable then the parameter is 0 (zero).

```
mx iso=0=cds.iso create=newcds2 now -all
```

## Note on MARC files

For files in the MARC format, enter the line end as the character ‘\0Bx’ and not ‘\0Cx’ (<CR>). For these files enter the parameter as follows:

mx iso=marc=input.iso create=output.iso now –all

### Leader data in a MARC record

MARC records have data in the leader that is not automatically converted from CDS/ISIS. If these data are necessary they should be converted to conventional fields, entering a value in the isotag1=<n> parameter as a place where it will load the bytes from the leader. For example, if isotag1=3000 is entered, the byte position 5 in the leader will be loaded in field 3005.

```
mx iso=marc=input.iso  isotag1=3000  create=newcds now -all
```

## Input ASCII text file

**seq[/1m]=<file>**

**<file>={filename|con|null}**

MX can take as an input source a plain text ASCII file. The line length can be up to 1MB, which is indicated with /1m.

Each line in this input file is converted to a master file record. The data in each line can be separated with delimiters, which means each part of the line is added to the records as consecutive fields. The number of fields is one more than the number of delimiters.

The default MX delimiter is the pipe (|). Each line of the input ASCII file can be up to 32743 characters in length for one field, 32740 for two fields, 32735 for three fields, etc.

Suppose there is a text file with a single line, called *input.in*, whose content is:

```
agua|tierra|abono
```

Using the line:

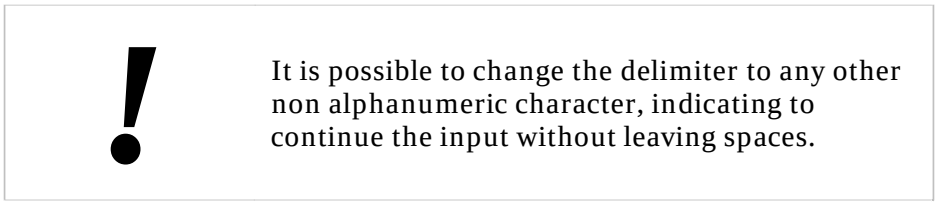
```
mx seq=input.in create=output now
```

it is converted to:

```
mfn= 1
1 «agua»
2 «tierra»
3 «abono»
```

Two consecutive delimiters produce a jump in field number:

```
agua||tierra||abono
mfn= 1
1 «agua»
3 «tierra»
5 «abono»
```




In the following example the semi colon (**;**) is used as a delimiter.

The file input.in contains the line:

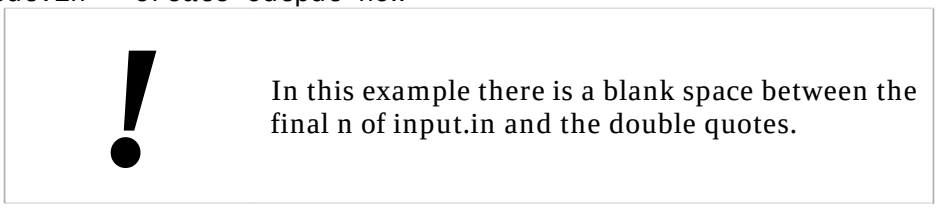
```
agua;abono;tierra
```

To read the file input.in with a semi colon (;) as a field separator, type:

```
mx "seq=input.in;" create=output now
```

It is possible to use a blank space as a delimiter:

```
mx "seq=input.in " create=output now
```




The input file can be generated directly by entering the data from the keyboard, that is, using the standard input device *con* (console) as the data entry source. Therefore, it is possible to create CDS/ISIS database records by entering the data directly from the keyboard.

Examine the following sequence:

| **PROCESS WITH THE PARAMETER *con*** | | |
|---|---|---|
| **Step** | **Explanation** | **Lines to type** |
| 1 | From the operating system prompt give the instruction | `C:\path> mx seq=con create=out`<br>`(MS-DOS)`<br>`unixuser:~$ mx seq=/dev/tty1 create=out`<br>`(input from console 1 in UNIX)` |
| 2 | Enter a line that finishes by pressing <enter> | `agua\|tierra\|vegetales\|abono<enter>` |
| 3 | This automatically creates the record mfn=1 in the master file *out* and displays the record fields | `mfn= 1`<br>`1 «agua»`<br>`2 «tierra»`<br>`3 «vegetales»`<br>`4 «abono»` |
| 4 | The MX prompt remains waiting | `..<enter>` |

| PROCESS WITH THE PARAMETER *con* | | |
|---|---|---|
| **Step** | **Explanation** | **Lines to type** |
| | for new instructions. Continue creating records by pressing <enter> at the MX prompt | |
| 5 | Add a new line that finishes by pressing <enter> | bovinos\|ovinos\|equinos<enter> |
| 6 | This creates the record mfn=2 and displays the record fields | mfn= 2<br>1 «bovinos»<br>2 «ovinos»<br>3 «equinos» |
| 7 | Finish the creation of records by entering a lowercase x at the MX prompt | ..x<enter> |
| 8 | The execution of the command finishes | C:\path><br>(MS-DOS)<br>unixuser:~$<br>(UNIX) |

The process can be made quicker by avoiding the use of the MX *prompt* by entering the parameter *now*. In this case to end the entry process press the keys <ctrl>+<Z> or <F6> (in MS-DOS) and <ctrl>+<D> (in UNIX).

The following table describes the process with the parameter *now*:

| PROCESS WITH THE PARAMETER *now* | | |
|---|---|---|
| **Steps** | **Explanation** | **Lines to type** |
| 1 | From the operating system prompt give the instruction | C:\path>mx seq=with create=out now<br>(MS-DOS)<br>unixuser:~$ mx seq=/dev/tty1 create=out now<br>(input from console 1 in UNIX) |
| 2 | Enter a line that finishes by pressing <enter> | agua\|tierra\|vegetales\|abono<enter> |
| 3 | This creates the record mfn=1 and displays the record fields | mfn= 1<br>1 «agua»<br>2 «tierra»<br>3 «vegetales»<br>4 «abono» |
| 4 | Enter a new line that finishes by pressing <enter> | bovinos\|ovinos\|equinos<enter> |
| 5 | This creates the record mfn=2 and displays the record fields | mfn= 2<br>1 «bovinos»<br>2 «ovinos»<br>3 «equinos» |
| 6 | On finishing the record creation, from the keyboard enter <ctrl>+<Z> (or the key F6) inDOS, or the keys <ctrl>+<D> in UNIX | <Ctrl>+<Z> o <F6><br>(MS-DOS)<br><ctrl>+<D><br>(UNIX) |

| PROCESS WITH THE PARAMETER *now* | | |
|---|---|---|
| **Steps** | **Explanation** | **Lines to type** |
| 7 | The execution of the command finishes | C:\path><br>(MS-DOS)<br>unixuser:~$<br>(UNIX) |

## Dummy database

**con | null**

MX allows the input source to be a dummy database called *null*. This database *null* consists of an unlimited number of records without data fields (empty records). It is possible to add data directly from the keyboard, using the input parameter *con* (console). Each <enter> creates a record. In order to finish the process, use <ctrl>+Z.

The use of this parameter is not directly related to the operations that are carried out on a database. It is used in processes that involve counting, generating lists of numbers, etc.

**Examples:**

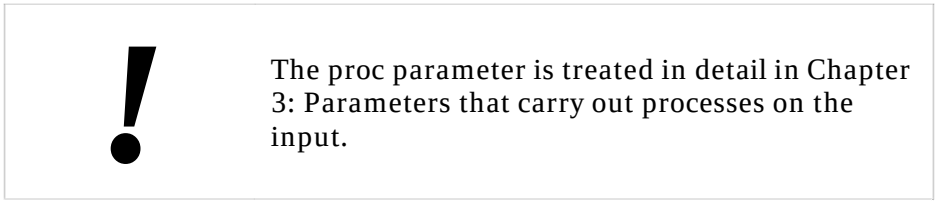
- Display a list of consecutive numbers from 1 to 50:

```
mx null to=50 pft=mfn(3)/ -all now
```

- Create a master file OUT with records from 100 to 200 containing the literal DBN in field 999:

```
mx null from=100 to=200 proc='A999#DBN#' now -all create=OUT
```

The parameter *proc* carries out the adding and deleting of fields in a record. In the example, proc adds in the field 999 (*A999*) the content DBN (*#DBN#*).



The proc parameter is treated in detail in Chapter 3: Parameters that carry out processes on the input.

## Parameters file

**in=<file>**

The parameters for the MX program can be read from an external ASCII file, where each parameter is added as a separate line. In this way it is possible to program command lines for MX that are longer than permitted by the operating system (normally limited to 128 characters for MS-DOS and to 512 characters for UNIX).

Supposing there is a file called input.in, whose content is:

```
iso=input.iso
create=output
now
from=10
to=20
-all
```

The command line:

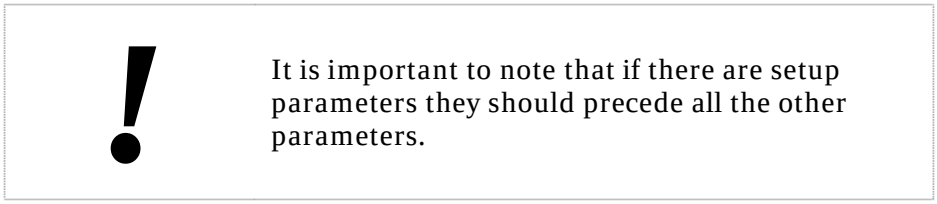
```
mx iso=input.iso create=output now from=10 to=20 -all
```

could be written as:

```
mx in=input.in
```

The parameter *in* can be used at any part of the command line after the first position, it may be the only parameter or it may be a part of a line of MX parameters.

If it is found in the first position (or it is the only parameter) the first line in this file should be the name of the input file.




It is possible to use more that one *in* file in a command line, and to use the *in* parameter in an *in* file up to 8 recursive levels.

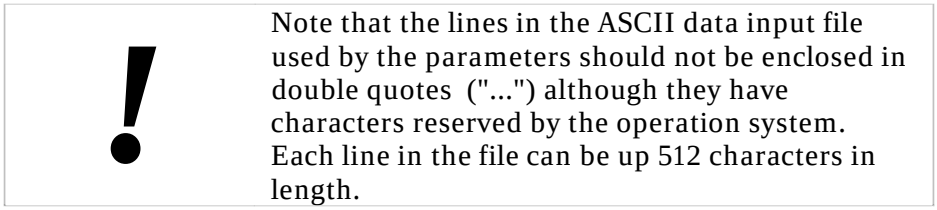
The following example shows a call to MX using an *in* parameters file between other parameters:

The file is called print:

```
\cisis\databases\cds
pft=v70+|; |, " / "v26". "/# mhl,(v69/)
now
-all
tell=10
```

The command line would be:

```
mx in=print from=10 to=50 > file.txt
```



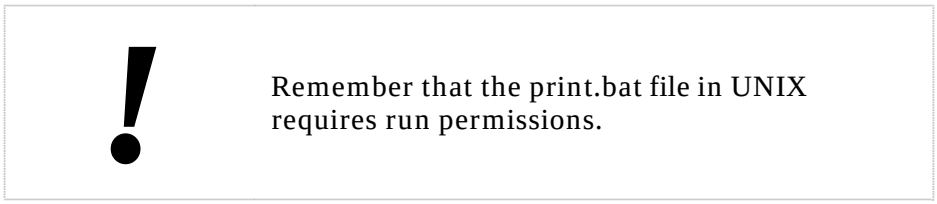

The use of the parameter *in* allows you to prepare files for batch tasks (*bat* in MS-DOS and *scripts* in UNIX). For the file print.bat whose content is as follows:

MS-DOS version

```
REM You should enter the values from= to=
REM For example: print 10 20
mx in=print from=%1 to=%2 > file.txt
```

UNIX version

```
# You should enter the values from= to=
# For example: print 10 20
mx in=print from=$1 to=$2 > file.txt
```




The operation could be entered as:

- MS-DOS

```
C:\cisis\sys> print 10 40
```

- UNIX

```
unixuser:~/cisis/sys$ print 10 40
```

That is translated as:

```
mx in=print from=10 to=40 > text.txt
```

which will then become:

```
mx \cisis\databases\cds pft=v70+|; |, " / "v26". "/# mhl,(v69/) now -all
tell=10 from=10 to=40 > text.txt
```

## Inverted file as input

**dict=<if>[,<keytag>[,<posttag>[/<postsperrec>]]]**
**[k{1|2}=<key>]**

MX reads the keys in an inverted file as input data producing records with field labels <ktag>, between the keys k1 and k2, with the following parameters.

| | |
|---|---|
| **if** | Name of the inverted file (I/F) |
| **keytag** | Tag that will have the lead key (*key tag*), by default 1<br>This field has the following sub fields<br>^l is the type of key (1=short, 2=long)<br>^s is the size of the key (number of characters = *keylength*)<br>^t total number of pointers (*totalpostings*)<br>^k sequential number of reading (*keyorder*) |
| **posttag** | Tag that will have the pointer (*posting tag*)<br>If this parameter is indicated, MX will combine the keys in the inverted file I/F and its corresponding pointers. The field ptag will have the following sub fields<br>^m MFN of the pointer shown<br>^t the field tag where the key was found<br>^o the occurrence of the label where the key was found<br>^c the position, counted in words, of the key in this field<br>^p the sequential number of the shown pointer<br>^k the sequential number of reading of the key |
| **postsperrec** | Maximum number of pointers read per record, by default all (*all*) |
| **k1=<key>** | Initial key to show |
| **k2=<key>** | Final key to show |

**Example 1** Read the keys from the CDS inverted file between the terms AFRICA and AFRICAN LANGUAGES

```
mx dict=cds  k1=africa  "k2=african languages"  now
mfn= 1
1 «AFRICA^l1^s6^t5^k1»
mfn= 2
1 «AFRICAN LANGUAGES^l2^s17^t1^k2»
```

**Example 2** Read three keys and postings from the CDS inverted file from the term Africa

```
mx dict=cds,1,2 k1=africa count=3 now
mfn=     1
  1  «AFRICA^l1^s6^t5^k1»
  2  «^m93^t69^o1^c9^p1^k1»
  2  «^m111^t24^o1^c3^p2^k1»
  2  «^m111^t69^o1^c3^p3^k1»
  2  «^m115^t69^o1^c7^p4^k1»
  2  «^m121^t69^o1^c4^p5^k1»
mfn=     2
  1  «AFRICAN LANGUAGES^l2^s17^t1^k2»
  2  «^m75^t69^o1^c10^p1^k2»
mfn=     3
  1  «AGE^l1^s3^t1^k3»
  2  «^m136^t24^o1^c7^p1^k3»
```

**Example 3** Read the keys from the CDS inverted file showing a maximum of three postings each time from the term AFRICA, show three output records, in this case three keys

```
mx dict=cds,1,2/3   k1=africa   count=3 now
```

```
mfn=     1
  1  «AFRICA^l1^s6^t5^k1»
  2  «^m93^t69^o1^c9^p1^k1»
  2  «^m111^t24^o1^c3^p2^k1»
mfn=     2
  1  «AFRICA^l1^s6^t5^k1»
  2  «^m115^t69^o1^c7^p4^k1»
  2  «^m121^t69^o1^c4^p5^k1»
mfn=     3
  1  «AFRICAN LANGUAGES^l2^s17^t1^k2»
  2  «^m75^t69^o1^c10^p1^k2»
```

**Example 4** Read the keys from the CDS inverted file showing the postings from the term Africa one by one

```
mx dict=cds,1,2/1   k1=africa   count=5 now
mfn=     1
  1  «AFRICA^l1^s6^t5^k1»
  2  «^m93^t69^o1^c9^p1^k1»
mfn=     2
  1  «AFRICA^l1^s6^t5^k1»
  2  «^m111^t24^o1^c3^p2^k1»
mfn=     3
  1  «AFRICA^l1^s6^t5^k1»
  2  «^m111^t69^o1^c3^p3^k1»
mfn=     4
  1  «AFRICA^l1^s6^t5^k1»
  2  «^m115^t69^o1^c7^p4^k1»
mfn=     5
  1  «AFRICA^l1^s6^t5^k1»
  2  «^m121^t69^o1^c4^p5^k1»
```

# Parameters that process the input

## Parameters that apply formatting to the input

This parameter provides the formatting specifications for the record display. The logically deleted records are not displayed with use of the parameter *pft*=.

MX supports all the standard CDS/ISIS formatting language instructions for DOS (except *format exits*) and adds some extensions developed for the CISIS Interface. Many of these new instructions are incorporated in Winisis, but MX does not accept the RTF graphical formatting instructions that Winisis uses.

The complete formatting manual is available on the VHL Model website, at the URL: http://bvsmodelo.bvsalud.org/.

## Specifying the display format from the command line

**pft=<display_fmt>**

The following example applies the format mfn/v24/v26 to the records retrieved from the master input file (cds):

```
mx cds pft=mfn/v24/v26
```

If the formatting instructions incorporate characters reserved by the operating system (such as: > | % etc.) or blank spaces, the parameter should be encased in double quotes:

```
mx cds "pft=mfn,/(v70+|; |)/v24/#"
```

## Specifying the display format through a file

**pft= @[<file>]**

MX allows you to specify a file (*pft=@[<file>]*) with the format to be used. This is a more practical method for specifying a display format, the call to MX is clearer and, also, you do not lose the format once the command has been implemented.

Also, this overcomes the operating system's limitations for the length of a command line (128 characters for MS-DOS and 512 characters for UNIX) for which an extensive format could not explicitly be written.

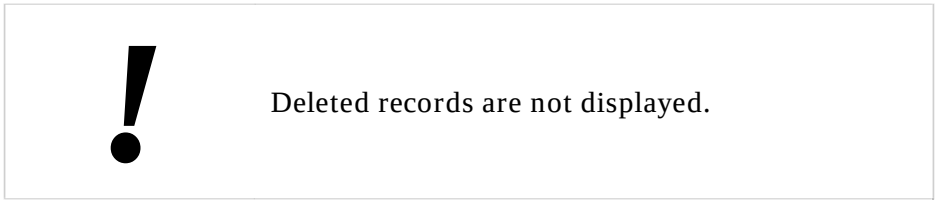
If the name of a file is not provided, MX will use the format with the same name as the database by default:

mx cds pft=@ is equivalent to mx cds pft=@cds.pft

On specifying a file, its name can have more than six characters in length, it can be located in a different directory from the database, and it may or may not have an extension. (If the file name has an extension this should be written, even if it is *.pft*).

**Examples:**

```
mx cds pft=@cdsnew.pft
mx cds pft=@\dbisis\other_dir\other.pft
mx cds pft=@long_name.pft
mx cds pft=@sin_ext
```




## Conditional formats

**prolog | epilog**

*prolog* is a specification that is executed in the first processed output record, and *epilog* in the last record.

**Example:**

```
mx cds prolog='First:' pft=mfn/ epilog='last'  from=10 to=20 now
```

## Line width

**lw={<n>|0}**

The output line has a predefined width of 78 characters. It is possible to change the line width with the parameter *lw=n*.

```
mx cds "pft=mfn,/(v70+|; |)/v24/#" lw=40 to=20 now
```

## Extract data from a CGI variable

**getenv('cgi=',<varfmt>)**

<varfmt> is a format specification that generates a name of a cgi variable.

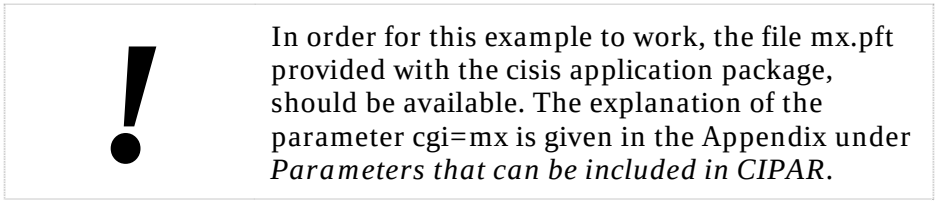
Multiple occurrences are separated with the character %

```
set "REQUEST_METHOD=GET"
set "QUERY_STRING=db~cds&btell~0&bool~plants*water"
mx cds cgi=mx "pft='Searching for \"',getenv('cgi=bool'),'\" in the database:
',getenv('cgi=db')/,'  MFN   Title'/,mfn,x2,mhl,v24.50,'...'/"
```

displays as output:

```
 Searching for "plants*water" in the database: cds
  MFN   Title
000004  Mc An Electric hygrometer apparatus for me...
..
Searching for "plants*water" in the database: cds
```

```
  MFN   Title
000011  Measurement of water stress in plants...
..
Searching for "plants*water" in the database: cds
  MFN   Title
000013  Experience with three vapour methods for measuring...
->x
```



In order for this example to work, the file mx.pft provided with the cisis application package, should be available. The explanation of the parameter cgi=mx is given in the Appendix under *Parameters that can be included in CIPAR*.

### Obtain a temporary empty file

**getenv('tmp=',[<pathfmt>])**

The format to obtain a name of an empty temporary file in the current working directory, or in one specified by a path, that is specified by the format <pathfmt>.

If 'ci_tmpdir' is specified then the path is obtained from the following: *ci_tempdir, temp, tmp*.

```
mx null pft=getenv('tmp=')
TMP1.$$$

mx null pft=getenv('tmp=','ci_tempdir')
C:\windows\TEMP\TMP1.$$$

set ci_tempdir=C:\work
mx null pft=getenv('tmp=','ci_tempdir')
C:\work\TMP1.$$$

echo ci_tempdir=C:\work2 >xcip
mx cipar=xcip null pft=getenv('tmp=','ci_tempdir')
C:\work2\TMP1.$$$
```

## Parameters that select a set of records to be processed

### Search expression specification in the command line

**bool=<bool_expr_spec>**

The parameter *bool* allows you to carry out searches in the database. MX provides all the Boolean operations of the CDS/ISIS search language.

The result of a search is a set of records that fulfil the search expression.

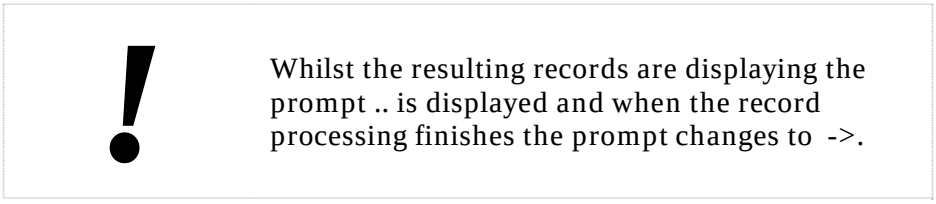
The term *bool* is optional, since MX considers the search expression to be everything that is not a parameter. The two examples produce the same results:

```
mx cds bool=water
mx cds water
```

The output of the search is:

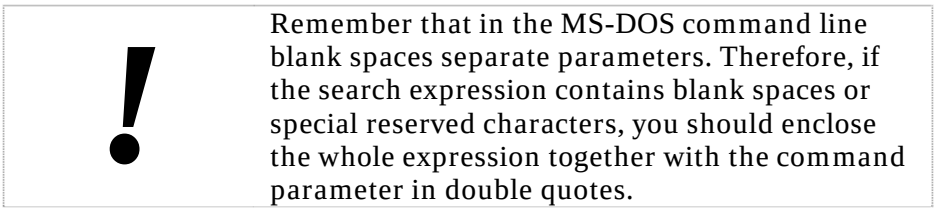
```
mx cds culture pft=mfn/
2     2     CULTURE
2     2     Set #000001
Hits=2
000050
..
000064
->
```

First the result of the search is presented and it is assigned a sequence number (*Set #000001*). Then the records are displayed according to the specified format.



Whilst the resulting records are displaying the prompt .. is displayed and when the record processing finishes the prompt changes to ->.

It is possible to continue entering instructions from the *prompt* interactively, but it is not possible with this parameter to reference the previous results in instructions such as #1 + #2, etc. For this it is necessary to use the parameter tmpx (which is described later in this chapter).



Remember that in the MS-DOS command line blank spaces separate parameters. Therefore, if the search expression contains blank spaces or special reserved characters, you should enclose the whole expression together with the command parameter in double quotes.

```
mx cds "bool=water + soil"
mx cds "water + soil"
```

The MX program accepts the Boolean operators +, *, ^ that CDS/ISIS uses as well as the literal expressions OR, AND, NOT.

```
mx cds "bool=water or soil"
mx cds "agricult$ and plants"
```

## Loading the search expression from a file

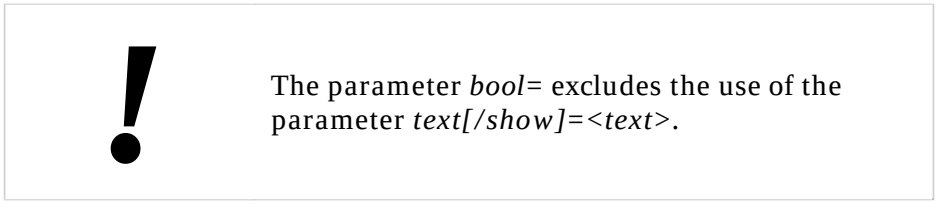
**bool=@<file>**

*bool=@<file>* reads the search expression from an external ASCII file. In this case the expression does not need to be enclosed in quotes.

Supposing that the file *consulta.001* contains the expression: agricult$ and plants, then the last example can take any one of the following forms:

```
mx cds bool=@consulta.001
mx cds @consulta.001
```

The *bool* parameter is optional, but its use may be necessary when the search expression begins with a word reserved by MX.



The parameter *bool=* excludes the use of the parameter *text[/show]=<text>*.

If you want to retrieve records that contain the word now:

| | |
|---|---|
| `mx cds now` | Will not do what is expected, since it interprets now as a parameter, not as a search expression. |
| `mx cds "now"` | Will not do what is expected, since it interprets now as a parameter, not as a search expression. |
| `mx cds bool=now` | Is correct. |
| `mx cds "bool=now"` | Is correct. |

## Utilization of intermediate search results

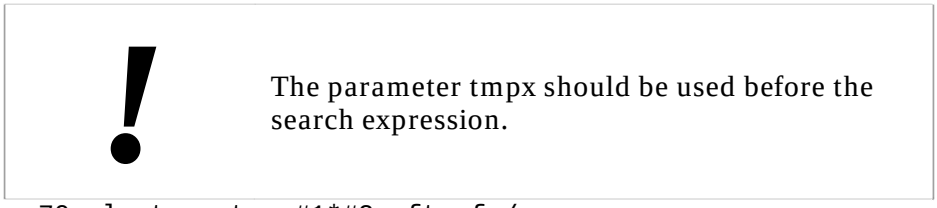
**tmpx=<tmpx_query_dbn>**

**obsolete parameter:**

**b70=<b70_query_dbn>**

The parameter *tmpx* stores the intermediate results of a search session, allowing you to reference the results in the form #1 + #2.

The name given to the right of the equal sign is an ISIS database that MX automatically creates and, and is not deleted at the end of execution.

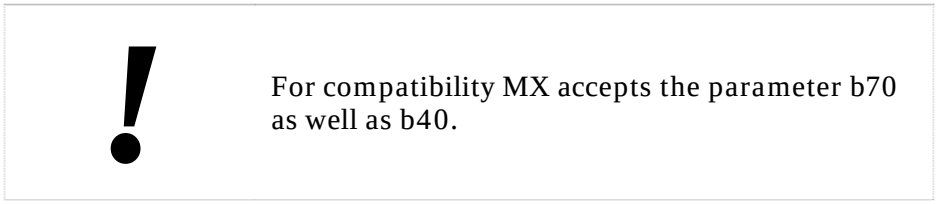


The parameter tmpx should be used before the search expression.

```
mx cds tmpx=x70 plants water #1*#2 pft=mfn/ now
      8  PLANTS
      8  Set #000000001
Hits=8
     14  WATER
     14  Set #000000002
```

```
Hits=14
      3  Operation *
      3  Set #000000003
Hits=3
000004
000011
000013
```

Note that if the parameter *tmpx* is indicated it is possible to carry out instructions interactively from the MX *prompt* making reference to the previous results.



For compatibility MX accepts the parameter b70 as well as b40.

In the following example the operator introduces the terms one by one by, as answered at the *prompt*, results are presented in black. The result of the process is stored in the X40 database.

```
mx cds tmpx=x70 plants pft=mfn/
      8  PLANTS
      8  Set #000000004
Hits=8
000001
..water
     14  WATER
     14  Set #000000005
Hits=14
000004
..#1 and #2
      3  Operation *
      3  Set #000000006
Hits=3
000004
..
000011
..
000013
->x
```

It is possible to carry out complete instructions reading the search expression from an external file. Supposing that the file profile.001 has the following search strategy:

```
Water
Plants
#1 and #2
Transpiration
#3 or #4
```

The following command line produces an output text file with the records retrieved according the profile given. Observe that each individual instruction should be added as independent lines in the file PROFILE.001.

```
mx CDS tmpx=x70 in=profile.001 pft=@cds.pft now > out_001.txt
```

## Elimination of search data

**btell=0**

It is possible to suppress the output on the screen that displays the intermediate processing steps and end of the search. In this case the parameter *btell=0* should go before the search expression.

**Example:**

| **Without the parameter *btell*=** | **With the parameter *btell=0*** |
|---|---|
| `mx cds "water * plants" pft=mfn/ now` | `mx cds btell=0 "water * plants" pft=mfn/ now` |
| `14 WATER`<br>`8 PLANTS`<br>`3 Operation *`<br>`3 Set #000000001`<br>`Hits=3`<br>`000004`<br>`000011`<br>`000013` | `000004`<br>`000011`<br>`000013` |

**btell=2**

lists the dictionary terms when $ truncation is performed.

## Search in various inverted files

**invx=M/F**

Invx contains records with the repeatable field v101, with the following format:

```
^pPrefix or asterisk^yFile[^uUseprefix][^mMessage]
```

Example:

A master file **cdsinvx** is created with the following fields

```
^p*^ycds^mText words
^pAU ^ycdsaut^mAuthor
^pTI ^ycdstit^mTitle words
```

```
       ^pKW ^ycdkw^mKeywords
```

The associated CDS inverted files are created

```
mx cds "fst=1 0 (v70/)" fullinv=cdsaut
mx cds "fst=1 4 v24" fullinv=cdstit
mx cds "fst=1 2 v69" fullinv=cdskw
```

Instructions

```
mx cds invx=cdsinvx "KW plants " pft=mfn,x1,v24,x1,v69
      2  PLANTS
      2  Set #000000001
Hits=2
000054 Vegetation as a geological agent in tropical deltas Paper on:
<vegetation><geology><deltas><tropical zones><sedimentation><plants><organic
matter><wetlands>..
000069 Some important animal pests and parasites of East Pakistan Paper on:
<pests><parasites><biology><ecology><plants><agriculture><public
health><food><Bangladesh>
->x

mx cds invx=cdsinvx "KW plants * east"
      2  PLANTS
      9  EAST
      1  Operation *
      1  Set #000000001
Hits=1
mfn=    69
 24  «Some important animal pests and parasites of East Pakistan»
 26  «^c1966»
 30  «^ap. 285-291^billus.»
 44  «Scientific problems of the humid tropical zone deltas and their
implicatio
ns: proceedings of the Dacca Symposium»
 50  «Incl. bibl.»
 69  «Paper on:
<pests><parasites><biology><ecology><plants><agriculture><public
 health><food><Bangladesh>»
 70  «Yosufzai, H.K.»
100  «1001»
->x
```

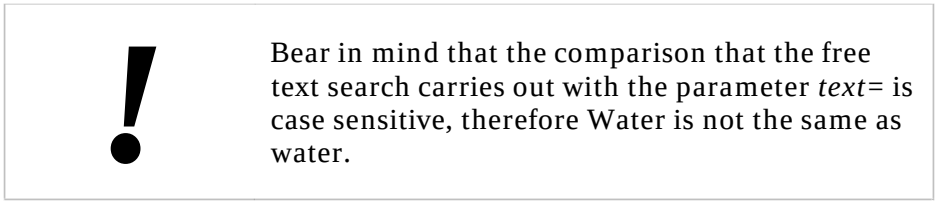
If the prefix is indicated between [ ], then it is applied to all the Boolean terms:

```
mx cds invx=cdsinvx "[KW] plants * east"
      2  PLANTS
         EAST
         Operation *
         Set #000000001
Hits=0
->x
```

## Free text searches

**text[/show]=<text>**

This function allows you to carry out searches in free text (not using the index). It searches the character string indicated in <*text*> in all the fields of the input database and selects the records that fulfil the search.




**Example:**

```
mx cds text=water iso=output.iso now -all
```

The example carries out a search and exports the records found to an ISO-2709 file.

## Parameter text/show

**text/show**

The parameter *text/show* only displays the record and field information which is produced by the search. It displays two lines, one that contains the record number, tag where the text searched for appears, and the search text; The second line displays the complete occurrence of the text searched for.

**Example:**

```
mx cds text/show=water now
```

Output on the screen:

```
mfn 56|tag 69|occ 1|water
69 «Paper on: <regime of waters><sediment transport><deltas>»
```

This indicates that record mfn=56 contains the character string water in the first occurrence of the field with the tag 69.

# Other ways of selecting the set of records to process

## Selecting by range

**from= <n> to=<n>**

```
mx cds from=24 to=50
```

This command line displays record 24 to record 50 inclusive on the screen. If *from* is not indicated the first record in the database will be assumed as the start; if *to* is not indicated it will process until the last record in the database, or until the program is exited, by pressing x.

> **!** The parameters should be written strictly in lower case, without spaces between the equals sign (=).

The following examples ARE NOT correct:

| | |
|---|---|
| `mx cds FROM=10` | The parameter should be in lowercase |
| `mx cds from= 10` | There is a blank space between = and 10 |
| `mx cds from 10` | Missing the = sign |

> **!** If the syntax is not correct there are two possible consequences: the program will not run and will display a cancellation message with the type of error, or it will interpret the badly written parameter as a search expression.

- The expression in example (a) will generate the result:

```
FROM=10
Set #000001
Hits=0
->
```

This means that it searched for the expression FROM=10 in the database.

- The expression in example (b) will generate the result:

```
FROM
Set #000001
Hits=0
Expression syntax error 5: '='
->
```

MX interprets an error in the search expression owing to the = sign

- The expression in example (c) will generate the result:

```
FROM
Set #00001
Hits=0
10
Set #000002
Hits=0
->
```

This implies that it searched for the words FROM and 10 and didn't find any records.

With this command line only the MFN numbers of records 24to 50 will be displayed.

```
mx cds from=24 to=50 pft=mfn/
```

## Selecting every *n* records

**loop=<n>**

The parameter *loop=n* processes a record and skips *n* records, *n+1* is processed, it skips another *n* and so on successively.

**Example:**

```
mx cds loop=10
```

Returns records 1, 11, 21, etc.

## Selecting *n* records

**count=<n>**

The parameter *count=n* selects exactly *n* records from a given start. If the start record is not indicated if will start from the first record.

| Example | Output |
|---|---|
| `mx cds from=24 to=50 loop=5 pft=mfn/ now` | `000024`<br>`000029`<br>`000034`<br>`000039`<br>`000044`<br>`000049` |
| `mx cds from=24 to=50 count=3 loop=5 pft=mfn/ now` | `000024`<br>`000029`<br>`000034` |
| `mx cds from=24 to=50 count=9 loop=5 pft=mfn/ now` | `000024`<br>`000029`<br>`000034` |

| Example | Output |
|---|---|
| | 000039<br>000049 |

When parameters like count, to, etc. are found in the same line the process finishes when it has completed the first one of them.

## Parameters that modify records

**proc=[<proc_fmt>][@<proc_fmt_file>]**

**<proc_fmt> and <proc_fmt_file> sintax**

```
D{<tag>[/<occ>]|*}
A<tag><delim><data><delim>
H<tag> <length> <data>
<TAG[ <stripmarklen>[ <minlen>]]><data></TAG>
S[<tag>]
R<mf>,<mfn>
G<gizmo_mf>[,<taglist>]
Gsplit[/clean]=<tag>[={<char>|words|letters|numbers|trigrams}]
Gload[/<tag>][/nonl][=<file>]
Gmark[/<tag>]{/<elem>|/keys|/decs|/<mf>,<otag>[,<ctag>]}=<if>
Gmarx[/<tag>]/<elem>[@<att>="x"] =<tag>[:&[<att>]|/c[=224]|/i]
Gdump[/<tag>][/nonl][/xml][=<file>]
=<mfn>
X[append=]<mf>
```

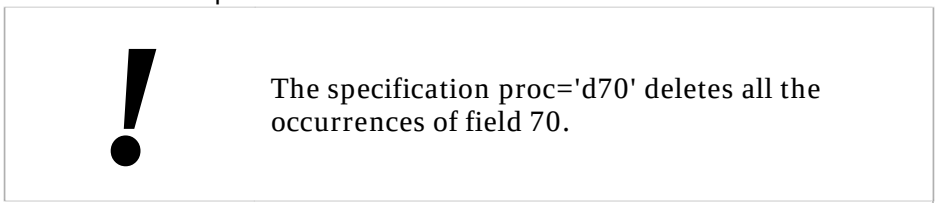
The parameter *proc* allows you to specify, through a format, modifications to carry out on the fields of a source record. It is possible to delete, add and replace the content of the fields. The operations to carry out are defined as format instructions, that can be provided directly in the command line or from an external file.

It is also important to emphasize that the modifications on the records are not carried out on the input database (the database that provides the data) but on the output database. If the specification for the output database is not found, the

changes can be viewed on the screen but they will be lost once the process finishes. In order to carry out changes to the same database as is used for input, this should be specified as the output database.

**Examples:**

```
mx cds from=1 to=10 now proc='d70'
```




The changes will be seen on the screen but will not actually be carried out because the output database is not specified. In order for the changes to be reflected in the same master file as is specified as the input, you must indicate this using the parameter *copy*:

```
mx cds from=1 to=10 proc='d70' copy=cds now
```

Another option is to specify a file instead of writing the format in the command line:

```
mx cds from=1 to=10 now proc=@change
```

This carries out the modifications in a file that contains the following format:

```
'd70'
```

Maximum number of accepted parameters

| | |
|---|---|
| in= | 1024 |
| files in= (recursive) | 16 |
| proc= | 1024 |
| join= | 128 |
| gizmo= | 16 |
| lines in the file stw | 799 |

It is possible to specify up to 1024 proc= parameters, including fst, read, write, I/F update *proc* in an MX command line. Each successive *proc* will act on the record in its current situation, so if a *proc* (or any previous procedure in the process) modifies the original input records, the next *proc* will act on the existing current data.

All the standard CDS/ISIS format instructions are allowed, including conditional type *IF*, but calls to programs in IsisPascal are not accepted. It also accepts the variables e0...e9, s0 ..s9 and the command while. It is possible to include proc inside a proc (recursive) without limit.

Also, the CDS/ISIS format language accepts the instruction proc( ). For details, see the CDS/ISIS format language manual.

The following table describes the most frequently used commands that *proc* can use, the others are individually explained in detail. (MX accepts all the commands of the CISIS Interface function *fldupdat( )* ):

| **Commands of the function fldupdat ( )** | | |
|---|---|---|
| **Command** | **Explaination** | **Example** |
| D. | Logically delete the record. | proc='d.' |
| D* | Delete all fields in the record | proc='d*' |
| Dtt | Delete all occurrences of the field tt. | proc='d26' |
| Dtt/occ | Delete the occurence occ from the field tt. | proc='d26/3' |
| Att#str# | Add the character string str as a new occurrence in the field tt. | proc='A999#cds#' |
| Htt n str_n | Add the character string str_n of n bytes in length as a new occurrence in the field tt. | proc='H99 8 CDS/ISIS' |
| =<mfn> | Change the record number (mfn) for n . | proc='=10' |
| S<tag> | Order the record input directory by tag. | proc='s'<br>proc='s70' |

**!**

In the same proc parameter all the delete commands should precede the add commands.
In the same proc parameter you should not use two or more Dtt/occ commands for the same field tt.
In the same proc parameter you should not use the command =n or the command S together with other commands.

The commands listed in the previous table, can be written in uppercase or lowercase.

Basically the idea is to carry out a format that, as a result, generates a character string of the type:

```
d36a999#1999#a70#Magalhaes, A.C.#
```

by which field 36 is deleted, 1999 is added to field 999 and Magalhaes, A.C. is added to field 70.

This *string* (of characters) can come from a format like the following:

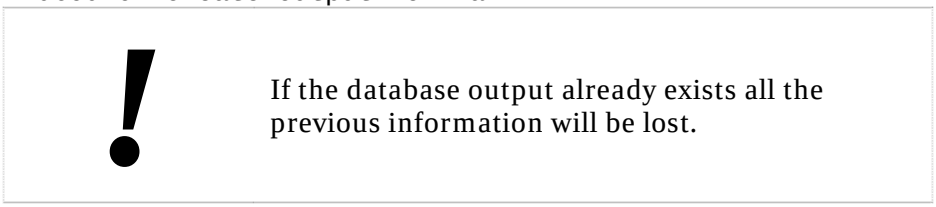
```
if p(v36) and a(v999) then 'd36a999#'v47'#'fi,"a70#"v36"#"
```

The *string* is formed by the presence of field 36 whose content is Magalhaes, A.C., and the absence of field 999.

**Examples:**

Create a database with all the records from the input database (CDS) deleting from those fields 69 and 70:

```
mx cds proc='d69d70' create=output now -all
```




Add information to field 999 in a group of records:

```
mx cds proc='A999#1998#' from=100 to=120 copy=cds now -all
```

In this example the changes are carried out on the same database that is used as the input source, therefore the changes are reflected in the same database.

To import an ISO-2709 file with MFN stored in field 999 of the ISO records:

```
mx iso=datos.iso proc='='v999 create=master -all now
```

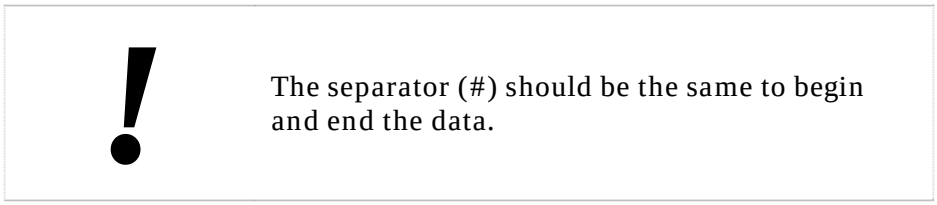
To export records to an ISO-2709 file adding the MFN of the input record to field 999 in the ISO records:

```
mx master proc='D999A999/'mfn'/' -all now iso=dato.iso
```

## Function ‘A' (add a field) att#str#

The ‘A' function is composed of:

| Command | Description |
|---|---|
| *a* | Indicates that a field will be added. |
| *tt* | Field tag where that data will be added. |
| *#* | Separator between *tt* and the data.<br>The separator can be any character but not numeric and that does not occur in the character string to add. |
| *str* | Character string to add.<br>The string can be added as a static character string or can be extracted from a data field in the same record or another (including a record from another database) |
| *#* | Separator that indicates the end on the string of data. |




The example:

```
mx cds proc='A999#1998#' from=100 to=120 copy=cds now -all
```

is equivalent to:

```
mx cds "proc='a999{1998{'" from=100 to=120 copy=cds now -all
```

**Examples**:

- Delete field 70, add only the first occurrence of this field as a new occurrence of field 100.

The following examples show two ways to carry out the same task:

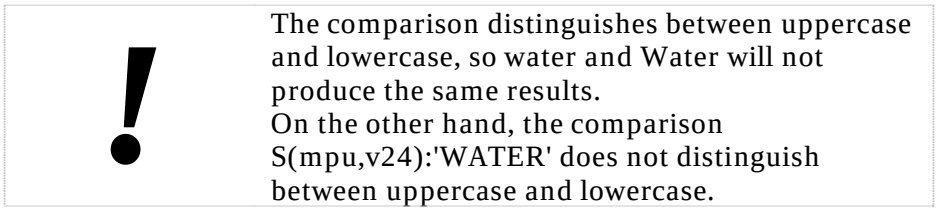
```
mx cds "proc='d70a100#',v70[1],'#' " from=10 to=20 now -all
mx cds "proc='d70',|a100@|v70[1]|@| " from=10 to=20 now -all
```

- Delete all the record fields where field 24 contains the word water, and export the record to a database called output. The format that contains the instructions for *proc* is taken from an external file (*test*).

```
mx cds proc=@test now –all create=output
```

The file *test* contains the format line:

```
If v24:'water' then ‘D*' fi
```




It is also possible to operate on the same database that is used as the input source.

```
mx CDS proc='d70' copy=CDS now to=1
```

- Order the record fields by tag number:

```
mx CDS proc='s' copy=CDS now –all
```

Input record:

```
mfn= 1
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
50 «Incl. bibl.»
69 «Paper on: <plant physiology><plant transpiration> <measurement and
instruments>»
24 «Techniques for the measurement of transpiration of individual plants»
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
```

```
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
```

Output record:

```
mfn= 1
24 «Techniques for the measurement of transpiration of individual plants»
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
50 «Incl. bibl.»
69 «Paper on: <plant physiology><plant transpiration><measurement and
instruments>»
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
```

Notice that the output of the process in this example is the same as the input database.

## Function R<mf>,<mfn>

Add the fields of the active record to the output master file

| | |
|---|---|
| <mf> | name of the input master file |
| <mfn> | record number |

```
mx null proc='a10/1/a20/1/' create=xxx
10 «1»
20 «1»

mx null proc='a20/2/' append=xxx
20 «2»

mx out  proc='Rxxx,1' proc='Rxxx,2' copy=out count=1 now
mfn= 1
10 «1»
20 «1»
20 «2»
```

writes the fields from records mfn=1 and mfn=2 from the master file *xxx* into the master file *out* in the active record (mfn=1)

## Function <TAG[ <stripmarklen>[ <minlen>]]><data></TAG>

Adds <*data*> as a new occurrence in the field <*tag*>. The markers in the style "<*marks*>" are removed from the text up to a maximum of <*stripmarklen*> characters.

If the text that is left after removing the markers is shorter than <*minlen*> characters, the editing is not carried out.

| | |
|---|---|
| <tag> | Label of the field |
| <stripmarklen> | Maximum length of the markers (infinite by default) |
| <minlen> | Minimum length of the resulting field (zero by default) |

- Add an occurrence with the term *text* in field 10

```
mx null "proc='<10>text</10>'"
mfn= 1
10 «text»
```

- Include an occurrence with the term *x<mark>y</mark>z* in field 10 removing the marks *<mark></mark>* but not the content

```
mx null "proc='<10>x<mark>y</mark>z</10>'"
mfn= 1
10 «xyz»
```

- Include and occurrence with the term *x<mark>y</mark>z* in field 10 removing the markers *<mark>* based on the size of the markers, in this case six characters.

```
mx null "proc='<10 6>x<mark>y</mark>z</10>'"
mfn= 1
10 «xy</marc>z»
```

- Include an occurrence with the term *x<mark>y</mark>z* in field 10 removing the markers *<mark></mark>* but not its content with a minimum expected of three characters.

```
mx null "proc='<10 99 3>x<mark>y</mark>z</10>'"
mfn= 1
10 «xyz»
```

- Do not include an the occurrence with the term *x<mark>y</mark>z* in field 10, after removing the markers *<mark></mark>* because the content does not meet the minimum expected of four characters.

```
mx null "proc='<10 99 4>x<mark>y</mark>z</10>'"
mfn= 1
```

## Function X[{create|copy|append|merge}=]<mf>

Writes the current record data in a file <out_dbn>: it is created if it does not already exist. Alternatively, it is possible to assign different output files for the same record in the command line.

- Assign the file *dbn1* as output, then write the same record mfn=3 of the master file *dbn2*.

```
mx null proc='a10/1/' proc='Xdbn1' proc='a20/2/' proc='=3' proc='Xdbn2'
mx dbn1
mfn= 1
10 «1»
mx dbn2
mfn= 3
10 «1»
20 «2»
```

- First create a file *xfile.pft*, then assign output file names as the first occurrence of field 70 in records 1,2,3. Notice the format *xfile.pft* that adds the signs -+ to the name. Record mfn=1 of CDS is written to a master file "*Magalhaes*" and records mfn=2, mfn=3 are recorded in a master file "*Bosian-+*".

```
echo replace(replace(s(v70[1].8),',','-'),' ','+') >xfile.pft
mx cds proc='Xappend='@xfile.pft,  pft=mfn,x1,v70[1]/  count=3 now
000001 Magalhaes, A.C.
000002 Bosian, G.
000003 Bosian, G.
mx Magalhae    pft=mfn,x1,v70[1]/
000001 Magalhaes, A.C.
mx Bosian-+   pft=mfn,x1,v70[1]/
000002 Bosian, G.
000003 Bosian, G.
```

## Function G<gizmo_mf>[,<taglist>]

Applies a gizmo to the record, which can apply to a list of specific fields.

```
echo transpiration~xxx> xxx.lst
mx seq=xxx.lst~  create=xxx now
mx cds proc='Gxxx,2' from=2  count=1 now
mfn=     2
24  «<The> Controlled climate in the plant chamber and its influence upon
assimilation and xxx»
30  «^ap. 225-232^billus.»
26  «^aParis^c1965»
70  «Bosian, G.»
```

## Function Gsplit[/clean]=<tag>[={<char>|words|letters|numbers|trigram}]

- An example with <char>:

```
echo Perez, J.; Garcia, María; Machado, A. > xxx.lst
mx seq=xxx.lst proc='Gsplit/clean=1=;'
mfn=     1
  1  «Perez, J.»
  1  «Garcia, María»
  1  «Machado, A.»
..
```

- Field 44 of CDS is separated by word

```
mx cds proc='Gsplit=44=words'
mfn=     1
44  «Methodology»
44  «of»
44  «plant»
44  «eco»
44  «physiology»
44  «proceedings»
44  «of»
44  «the»
44  «Montpellier»
44  «Symposium»
..x
```

## Function Gsplit=<tag>=6words[/if=<if>]

Extract the words from a text in the following sequence:

- word 1, words 1+2, words 1+2+3 ... words 1+2+...6,
- when the series finishes, it transfers to the second word
- then the sequence begins again.

**Example:**

Methodology of plant eco-physiology: proceedings of the Montpellier Symposium

```
Methodology
Methodology of
Methodology of plant
Methodology of plant eco
Methodology of plant echo physiology
Methodology of plant echo physiology proceedings
of
of plant
of plant echo
of plant echo physiology
…
```

### Option /if=<if>

Performs a lookup in an inverted file and only extracts the validated expressions.

## Function Gload[/<tag>][/nonl][=<file>]

Load the file <file> in binary format in the field <tag>

If /nonl is indicated it eliminates exiting at the end of the line<CR>

## Function Gdump[/<tag>][/nonl][/xml][=<file>]

Writes the content of field <tag> in a file.

Option /nonl: eliminates exiting at the end of the line <CR>.

Option /xml: writes it in an xml structure.

# Global change of patterns

**gizmo=<gizmo_mf>[,<taglist>] [gizp[/h]=<out_mfx>]**
**[decod=<mf>]**

The parameter *gizmo* allows you to carry out global changes to the field content in a CDS/ISIS database, converts one character string (or a single character) to another, and carries out modifications, encoding/decoding, data compression, etc.

These changes can be carried out on all the database records or on a set of records (selected by means of a search, a range, etc). Also, the changes can be to the whole record or to certain fields only.

To carry out changes, for example, the signs < > to / /, or the character string plants to PLANTS, etc., it is necessary to have a *gizmo* master file. This master file has at least two fields: field 1 contains the data to be changed, and field 2 the new value. Each data pair will be a record of the master file *gizmo*.

Each input record is submitted to changes set out in the *gizmo* master file. When MX starts up the data in the *gizmo* file are loaded as a table in memory and are sorted alphabetically by the value of field 1 and by its length. (In this way the longer character strings are converted before the shorter ones ).

It is possible to specify up to 16 gizmos in an MX command line. Each successive *gizmo* will act on the record in its current state, so if a *gizmo* modifies the original input record, the next *gizmo* will act on the resulting data.

**Examples:**

- In the following example a database called *test* is created using known MX parameters and entering the data directly from the keyboard:

| **MS-DOS** | **UNIX** |
|---|---|
| `mx seq=con create=test -all now`<br>`<\|/`<br>`>\|/`<br>`plants\|PLANTS`<br>`<ctrl>+<Z> (o <F6>)` | `unixuser:~$ mx seq=/dev/tty1 create=test -all now`<br>`<\|/`<br>`>\|/`<br>`plants\|PLANTS`<br>`<ctrl>+<D>` |

It produces the following records:

```
mfn= 1
1 «<»
2 «/»
mfn= 2
1 «>»
2 «/»
mfn= 3
1 «plants»
2 «PLANTS»
```

The content of the title and descriptor fields of MFN=1 is:

```
mx cds to=1 "pft=mfn/v24/v69"
000001
Techniques for the measurement of transpiration of individual plants
Paper on: <plant physiology><plant transpiration><measurement and instruments>
```

Applying:

```
mx cds gizmo=test to=1 "pft=mfn/v24/v69"
```

gives the result:

```
000001
Techniques for the measurement of transpiration of individual PLANTAS
Paper on: /plant physiology//plant transpiration//measurement and instruments/
```

The modification does not affect the cds database that provides the input data, because it is carried out on the output (in this case, the screen).

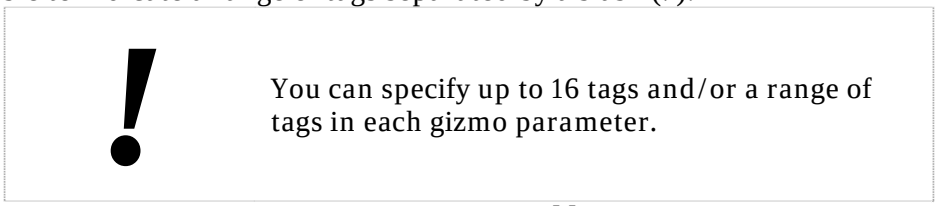
To actually modify the record, the result of the process should be output to the same master file, as in the following example:

```
mx cds gizmo=test to=1 copy=cds –all now
```

If, on the other hand, you want to generate a new database it is necessary to create it:

```
mx cds gizmo=test to=1 create=output –all now
```

It is possible to restrict the modification to a specific record field, indicating after the gizmo parameter the tag or tags on which the change will take place. It is possible to indicate a range of tags separated by a slash (/).



You can specify up to 16 tags and/or a range of tags in each gizmo parameter.

```
mx cds gizmo=test,69,24 to=1 create=output -all now
mx cds gizmo=test,35/56 to=1 create=output -all now
```

- Another application of the *gizmo* can be the encoding and resulting compression of data where frequently occurring terms can be substituted by codes or special values:

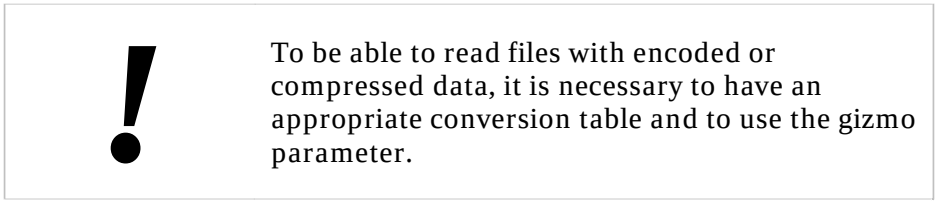
```
mx seq=con create=tabla1 now -all
^aParis^bUnesco|*1*
transpiration|*2*
<ctrl>Z
```

Before conversion:

```
mx cds to=1 pft=mfn/v69/v24/v26
000001
Paper on: <plant physiology><plant transpiration><measurement and instruments>
Techniques for the measurement of transpiration of individual plants
^aParis^bUnesco^c1965
```

After conversion:

```
mx cds gizmo=table1 to=1 pft=mfn/v69/v24/v26
000001
Paper on: <plant physiology><plant *2*><measurement and instruments>
Techniques for the measurement of *2* of individual plants
*1*^c1965
```




In the following example **table2** is created to carry out the inverse conversion to **table1**, using the *proc* parameter :

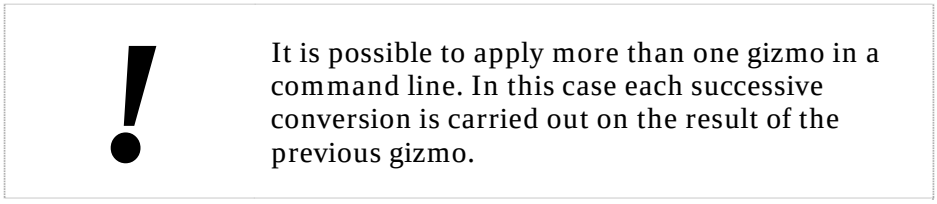
```
mx table1 create=table2 now -all "proc='d*a1#',v2,'#a2#',v1,'#'"
```

## Description

The instruction *proc* starts by deleting all the fields, then adds as field 1 the content of field 2 and finally adds to field 2 the content of field 1.

Using the databases *table1* and *table2* the following example can be carried out:

```
mx CDS to=1 create=OUT gizmo=table1 now –all
mx OUT gizmo=table2
```




- It applies two *gizmos* with *table1* and *table2* to the CDS database, that produce two reciprocal conversions (therefore the original data is not changed).

```
mx CDS gizmo=tabla1 gizmo=tabla2
```

The *gizmo* parameter has many applications. For example, you could have tables in various languages with equivalences between the numeric code of a descriptor and its version in that language. The database will have the numeral value as data and the conversion to the required language that will be carried out during the processing.

Another interesting example is to have a table with accented characters and symbols as field 1 and their corresponding codes in HTML (such as é for é) in field 2, so you could generate HTML documents as output.

Another application could be the decoding of abbreviations, institutional abbreviations, abbreviations of languages, of countries, etc., that in the standard CDS/ISIS is usually carried out with the function *ref+lookup*.

## Conversion tables with ASCII or hexadecimal codes

The *gizmo* conversion tables are master files consisting of two fields, whose contents are the string to change (field 1) and the string that it will be changed to (field 2).

Also, it is possible to specify these character strings through the numeric character code, allowing the gizmo table to include characters that cannot be typed or viewed or easily handled in a text file.

This type of table can have up to four fields, these are: fields 1 and 2 (string to change and string that it will be changed to) and fields 11 and/or 21 where you indicate what type of information fields 1 and/or 2 contain, respectively. The value of these fields can be: **hex** (if the content of field 1 or 2 is encoded in hexadecimal) or **asc** (if the content of field 1 or 2 is encoded in three decimal digits). It is possible to add comments in each record using the non-reserved fields.

**Example:**

The following gizmo tables are equivalent:

| TAG | Content |
|---|---|
| 1 | OF THE |
| 2 | Ç |

| TAG | Content |
|---|---|
| 1 | OF THE |
| 2 | 128 |
| 21 | Asc |

| TAG | Content |
|---|---|
| 1 | 079070032084072069 |
| 2 | Ç |
| 11 | asc |

| TAG | Content |
|---|---|
| 1 | 4F4620544845 |
| 2 | Ç |
| 11 | hex |

1. You have a master file where you want to change the character - (hexadecimal 2D, for the character ~ (hexadecimal 7E).

First you need to generate a table to carry out the change, that will be a master file that contains:

| TAG | Content |
|---|---|
| 1 | 2D |
| 2 | 7E |
| 11 | Hex |
| 21 | Hex |

For this you must create a text file called *change.seq*, whose content is:

```
2D|7E|hex|hex
```

Then apply the following command line:

```
mx seq=change.seq create=change -all now
```

This generates a table with four fields needed to carry out the change. Although fields 3 and 4 should have the tags 11 and 21 respectively, to change the *tag* numbers you could use the RETAG or ID2I utilities, but in this example it is done with MX using the *proc* command:

```
mx change proc='d3d4a11#hex#a21#hex#' copy=change
```

To change the tag numbers you could use the RETAG or ID2I utilities, that are covered in chapters 12 and 15 respectively.

Now it is ready to carry out the change:

```
mx cds gizmo=change -all now
```

## Statistics on conversion by gizmo

**gizp/h=<out_mfx>**

The parameter *gizp=<out_mfx>* generates a master file for each *gizmo* used in the command line, with information that records the process carried out on the records. Each database will have as a prefix ‘dbnx' plus a sequential number, for example: *dbnx0*, *dbnx1*, *dbnx2*, etc. The parameter /h indicates that the data are stored in hexadecimal.

**Example:**

```
mx cds gizmo=code,80 gizmo=precode,87 now –all create=OUT gizp=dbnx
```

will create two output master files: *dbnx0* corresponding to the process carried out with *gizmo=code*, and *dbnx1* corresponding to the process carried out with *gizmo=precode*.

Each of these master files will have a record for each gizmo input that has been used in the MX implementation. The fields created are:

| TAG | Description |
|---|---|
| 1 | Field 1 of the gizmo file |
| 2 | Field 2 of the gizmo file |
| 10 | Number of times that this conversion was used in the process |
| 31 | Length of field 1 in characters |
| 32 | Length of field 2 in characters |

## Option [decod=<mf>]

Name of a master file that encodes/decodes the data. The mf file contains the separator character of the elements and the fields where it is applied.

There are two encoding files, *decodp* and *zdecsp*:

```
mx decodp
mfn=     1
  1  "ZDECSP"
  2  ";"
  3  "87"
  3  "88"
  3  "71"
  3  "76"
..

mx zdecsp
mfn=     2
  3  "Temefos"
mfn=     3
  3  "Matadouros"
mfn=     4
  3  "Abreviaturas"
mfn=     5
  3  "Abdome"

mx lilacs "pft=mfn/(|87=|v87/)(|88=|v88/)(|71=|v71/)(|76=|v76)#"

432938
87=^d731;^d9525^s22004;^d2465^s22004

432937
87=^d3977;^d1732^s22045;^d30912^s22045;^d7840;^d6893
```

```
432936
87=^d1942^s22004;^d2465^s22004;^d731
76=841;21044;21030
```

# Join databases - JOIN

**join=<mf>[:<offset>][,<taglist>]=<mfn=_fmt>**

**join=<db>[:<offset>][,<taglist>]=<upkey_fmt> [jmax=<n>]**

The parameter *join* allows one or more records from the same or other databases to be joined to a record that is being processed. The record being processed can come from a master ISIS file, an ISO-2709, or an ASCII file. All the fields of the referenced records (or fields selected from them) are added to the output record and, also, control fields with process information are created (numbers from 32001).

The parameter *join* can take the following arguments:

| **Arguments of the parameter *join*** | |
|---|---|
| **Argument** | **Description** |
| <mf> / <db> | An alternative database that is searched for records to join with the record that is being processed. The alternative database can be the same as the input data. |
| [:offset] | Displacement for joined tags |
| [<taglist>] | A list of fields that will be extracted from records in the alternative database to add to the input record. If the list is not specified, then all the fields in the named records are added. The data fields of the named record can be renumbered. |
| <_fmt><br>@<file> | Fomat that reads the input record to extract keys with which records in the alternative database are searched. It can be an explicit format or a reference to an external file. |
| <upkey_fmt> | <_fmt>\|@<file><br>It is necessary to generate the key(s) according to I/F, typically in *uppercase* |
| [jmax=<n>] | Limit the number of records that are added to the original record as a result of the *join* parameter for each key generated. This limit is applied for *join* parameter in the command line. |

**Example:**

```
mx cds join=author,2,3=mhu,(v70/) create=newcds -all now
```

## Description

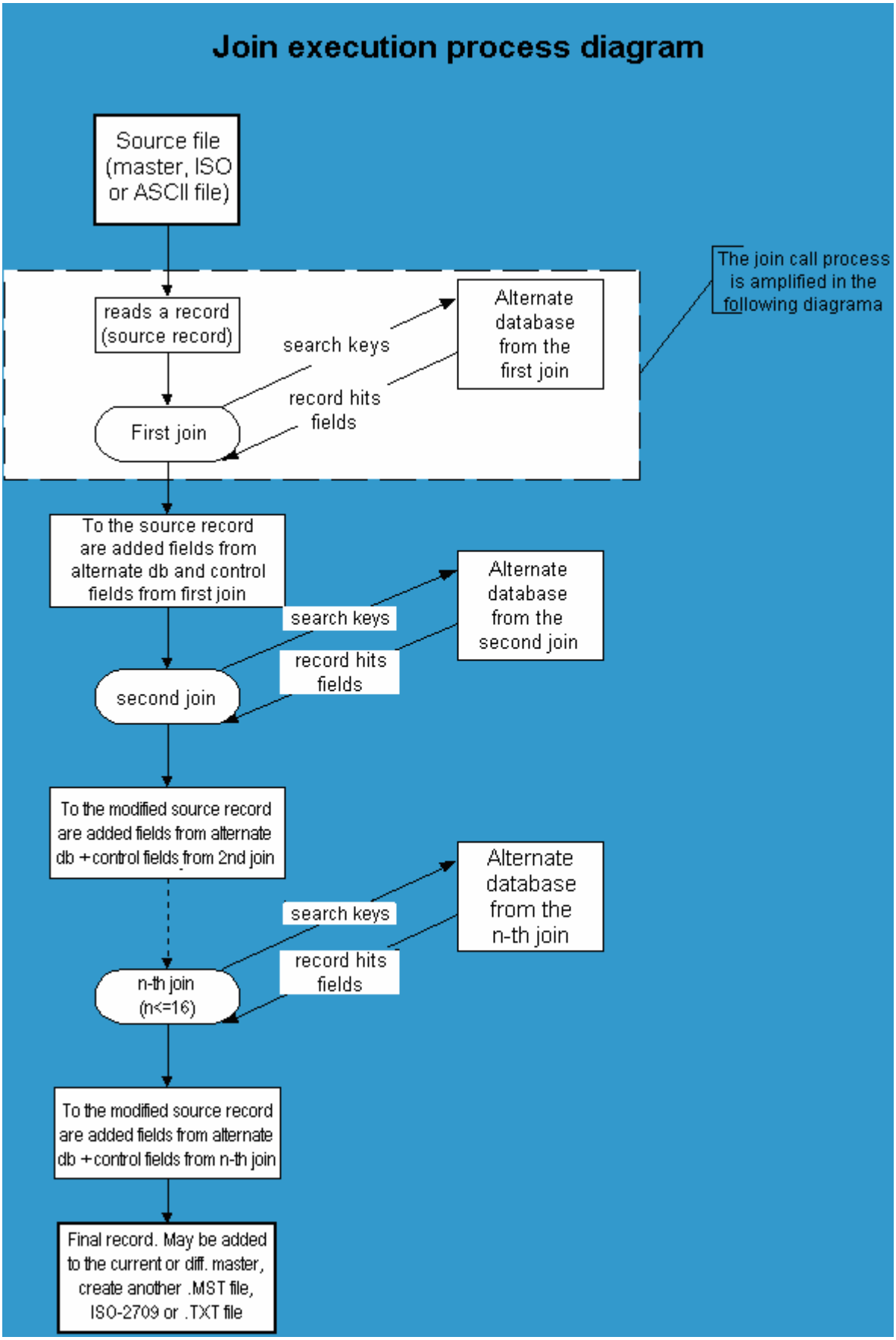
1. Take each record from the database *cds*.

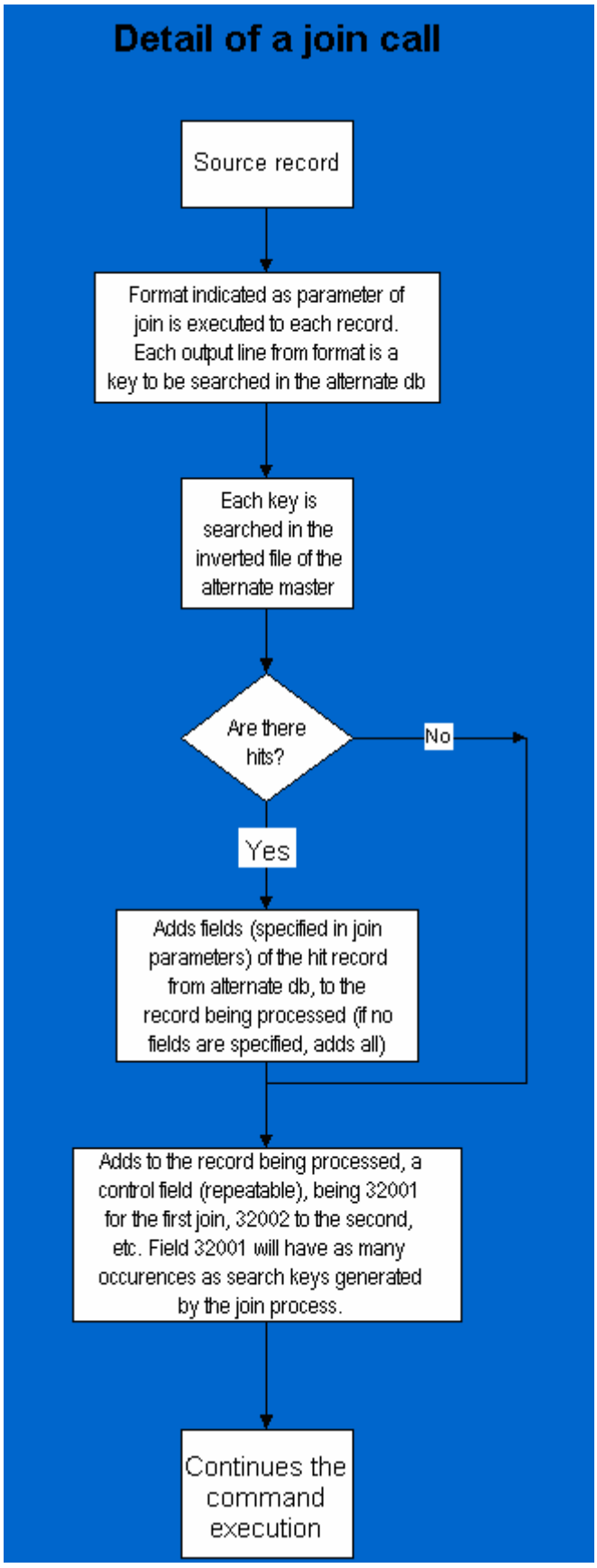
For each author generated by the format specification mhu, (v70/) it carries out a search in the database *author*.

If it finds any record in the database *author*, it adds a control field and adds the fields 2 and 3 to this in the record that is being processed.

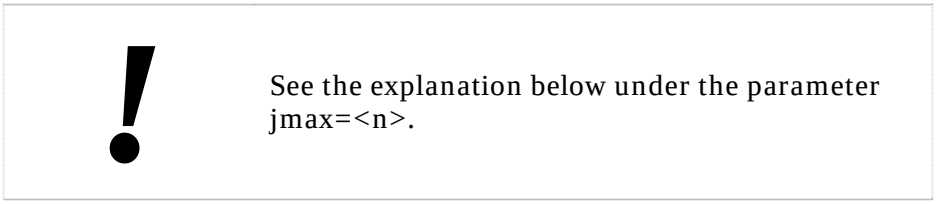
Finally, it adds the record to the database *newcds*.

The following diagrams show the process that occurs when MX applies the *join*:


Join execution process diagram
Source file (master, ISO or ASCII file)
reads a record (source record)
search keys
record hits fields
Alternate database from the first join
First join
The join call process is amplified in the following diagrama
To the source record are added fields from alternate db and control fields from first join
search keys
record hits fields
Alternate database from the second join
second join
To the modified source record are added fields from alternate db +control fields from 2nd join
search keys
record hits fields
Alternate database from the n-th join
n-th join (n<=16)
To the modified source record are added fields from alternate db +control fields from n-th join
Final record. May be added to the current or diff. master, create another .MST file, ISO-2709 or .TXT file


Detail of a join call
Source record
Format indicated as parameter of join is executed to each record. Each output line from format is a key to be searched in the alternate db
Each key is searched in the inverted file of the alternate master
Are there hits?
No
Yes
Adds fields (specified in join parameters) of the hit record from alternate db, to the record being processed (if no fields are specified, adds all)
Adds to the record being processed, a control field (repeatable), being 32001 for the first join, 32002 to the second, etc. Field 32001 will have as many occurences as search keys generated by the join process.
Continues the command execution

It is possible to specify up to 128 *joins* in an MX command line. Each successive *join* will act on the record it its current situation, so if a *join*, or any previous procedure in the process, modifies the original input record, the next *join* will act on the current data.




**Example:**

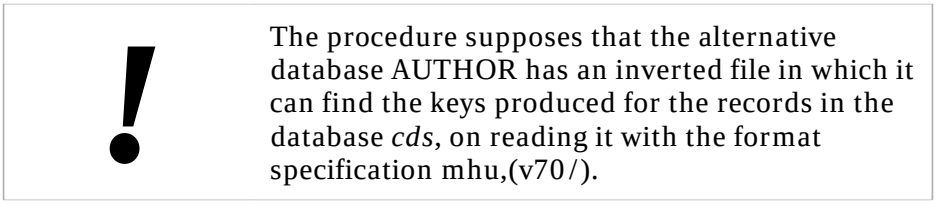
- Coming from a database AUTHOR that contains normalised data for authors, as is shown next:

```
mfn= 1
1 «Magalhaes, A.C.»
2 «Ing. Agrónomo»
3 «1935-1990»
mfn= 2
1 «Franco, C.M.»
2 «Bioquímico»
3 «1940-»
...
```

The following operation is carried out on the database *cds*, whose field 70 records the authors.

```
mx cds join=AUTHOR=mhu,(v70/)
```

Then: all the fields from the database author are added to the database *cds*, the control fields 32001 are added. The changes only display on the screen since no output is indicated.




The *join* process involves the following steps:

| **Processing of the parameter *join*** | |
|---|---|
| **Description** | **In the example** |
| 1) Reads the record from the source database cds. | Source database: cds |
| 2) Applies the specified format in the *join* to the record read.<br>The format mode MHU, or MPU, is necessary in general to get the keys, because the standard file ISISUC.TAB converts lowercase to | mhu,(v70/) |

| | |
|---|---|
| uppercase for the generation of the inverted file. The record read is the record originating from the database. | |
| 3) For each key obtained – that is, for each line generated by the specified format – it looks to extract the MFN of the next *posting* found in the inverted file of the alternative database. | Alternative database: AUTHOR |
| 4) Add to the original record a control field (repeatable) 32001 to the first *join*, 32002 to the second *join*, etc. Field 32001 will have as many repeats as different keys that have been produced in step 2. | |
| 5) Reads in the alternative database the record corresponding to the MFN obtained and adds the fields selected in the list <*tags*>. If this is not specified all the fields will be taken. | |
| 6) Returns to step 2. | |

## Selection list and renumbering of fields <tags>

- Selection:

  You can select fields from the alternative record in the following ways:

  1. Indicate the field numbers separating them with commas, e.g.

```
mx cds join=AUTHOR,1,2,3=mhu,(v70/)
```

  2. Indicate a **range of fields** using low value/high value, e.g.

```
mx cds join=AUTHOR,1/3=mhu,(v70/) copy=cds
```

- Renumbering:

  A new field number is specified as **new number:old number**

```
mx CDS "join=AUTHOR,100:1=mhu,(v70/)"
```

In all the cases a field will be copied with all its repeats.

## Content of the control fields (32001, 32002, etc.)

Fields with tag 32001, 32002, etc. are generated by MX to register the process of the *join*.

**Example:**

```
32001 «AUTHOR^l2^kFRANCO, C.M.^o2^m2»
```

| SUBFIELDS OF FIELD 32001 | |
|---|---|
| **Field** | **Content** |
| | First data, without subfield indicator: name of the alternative database |
| ^l[1\|2] | Subtree of the inverted file that the key was extracted from: 1 for terms up to 10 characters long and 2 for terms between 11 and 30 characters long |
| ^k | Key that the search was made with |

| SUBFIELDS OF FIELD 32001 | |
|---|---|
| **Field** | **Content** |
| ^o | Occurrence number of the key |
| ^m | MFN of the alternative database record that the data come from that are added to the original record.<br>If the term is not found in the alternative database dictionary this subfield is not generated. |

**Examples:**

- The following example will produce the output shown below. Note that field 1 of the alternative record is renumbered as 100 when it is copied to the original record.

```
mx cds join=AUTHOR,2,3,100:1=mhu,(v70/)
mfn= 1
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
50 «Incl. bibl.»
69 «Paper on: <plant physiology><plant transpiration> <measurement and
instruments>»
24 «Techniques for the measurement of transpiration of individual plants»
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
32001 «AUTHOR^l2^kMAGALHAES, A.C.^o1^m1»
100 «Magalhaes, A.C.»
2 «Ing. Agrónomo»
3 «1935-1990»
32001 «AUTHOR^l2^kFRANCO, C.M.^o2^m2»
100 «Franco, C.M.»
2 «Bioquimico»
3 «1940-»
..
```

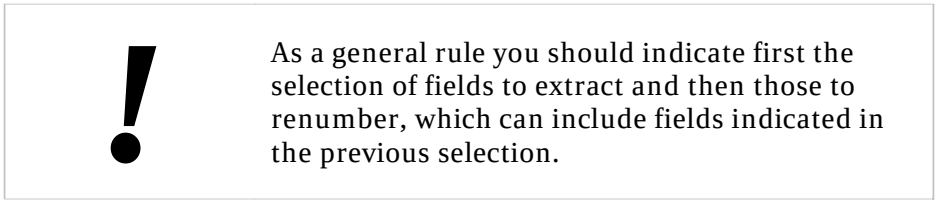
Note that two occurrences of 32001 occur, corresponding to each key generated by the original record, and after each one the fields extracted for each key of the alternative base appear.

Also the output was not specified, therefore, the modifications only appear on the screen.

The same output could be produced by:

```
mx cds join=AUTHOR,1/3,100:1=mhu,(v70/)
```

where field 1 is included in the range 1/3, that is then renumbered as 100.




It is important to bear in mind that these fields will become database fields. To delete them you can use the parameter *proc*. For example:

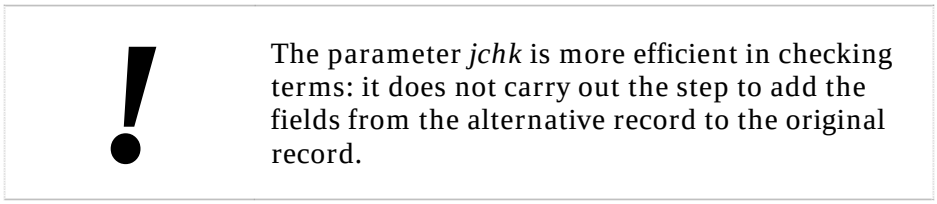
```
mx cds proc='d32001' copy=cds -all now
```

- To produce a list of non-valid authors:

```
mx CDS join=AUTHOR=mhu,(v70/) pft=@check.pft –all now
```

The archive check.pft has the following format specifications:

```
if p(v32001) then (if a(v32001^m) then mfn,x2,v32001^k | Does not exist| / fi)
fi
```




## Join by record number

It is possible to make a *join* by MFN number. In this case the format specification should include, at the beginning, the character string *mfn* followed by the record number that should be found.

The procedure using *join* by MFN does not require the existence of an inverted file.

**Examples:**

- Suppose that a file called NUMS contains a list with the record numbers that you want to extract from the database CDS.

```
mx seq=NUMS join=CDS='mfn=',v1 create=EXPORT now -all proc='d1/1d32001'
```

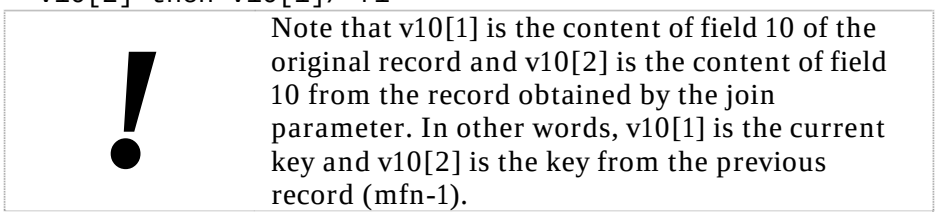
The resulting database EXPORT has two fields that are not present in the CDS records. Field 1 that comes from the file NUMS, where each line is considered a record of only one field, and field 32001 that is generated to record the *join* procedure. These two fields are finally deleted with the *proc*.

Supposed that the master file called SORTED contains records sorted by field 10 (non- repeatable).

```
mx SORTED "join=SORTED=if mfn > 1 then 'mfn=',f(mfn-1,1,0) fi" pft=@unavez.pft
-all now
```

The following format specification prints the different contents of field 10 once:

```
if v10[1] <> v10[2] then v10[1]/ fi
```

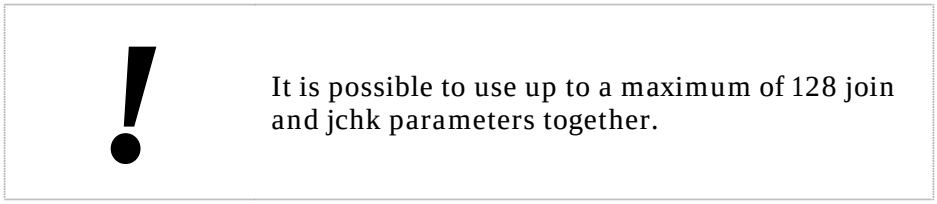


Note that v10[1] is the content of field 10 of the original record and v10[2] is the content of field 10 from the record obtained by the join parameter. In other words, v10[1] is the current key and v10[2] is the key from the previous record (mfn-1).

## Parameter [jmax=<n>]

The parameter *jmax*=<*n*> limits the number of records that are added to the original record resulting from the action of the *join* parameter for each key generated. This limit is applied form each *join* parameter in a command line.

If more than one *jmax* is specified, only the last of these will be valid. Therefore it is recommended that you indicate it once after detailing all the *joins* and *jchks* necessary for the process. By default the *jmax* value is infinite.



It is possible to use up to a maximum of 128 join and jchk parameters together.

## Compare databases with inverted files

**jchk=<if>[+<stwfile>]=<upkey_fmt>}**

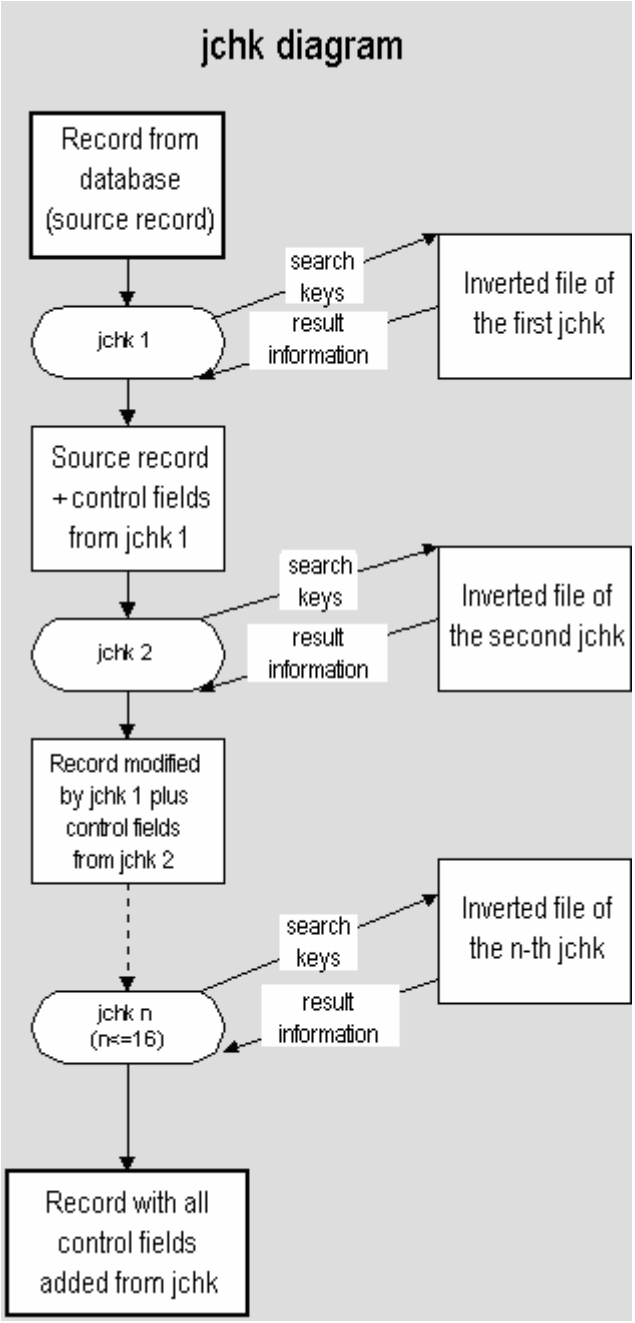
The parameter *jchk* is used to compare records (coming from a master file, an ISO-2709 or an ASCII file) against the terms in an inverted file.

The parameter *jchk* can pass the following arguments:

| **ARGUMENTS OF THE PARAMETER *jchk*** | |
|---|---|
| **Argument** | **Description** |
| <if> | An alternative inverted file in which to look for the keys extracted from the processed record |

| ARGUMENTS OF THE PARAMETER *jchk* | |
|---|---|
| **Argument** | **Description** |
| [+<stwfile>] | A list of words to remove (*stopword file*), optional. The use of this option causes the keys to be extracted according to the indexing technique *word by word* (IT 4) and each key generated is filtered through the *stopwords* file. |
| <upkey_fmt> @<file> | Format with which the input record is read to extract keys with which to search terms in the inverted file. It can be an explicit format or reference to an external file. |

The following diagram shows how the *jchk* call is carried out:



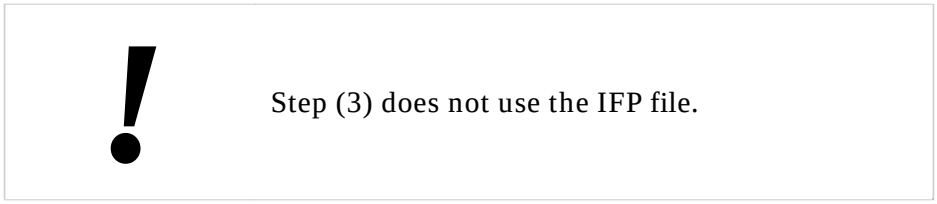

The examples will use the same database *AUTHOR* that was used with the *join*.

**Example**

```
mx CDS jchk=AUTHOR=mhu,(v70/)
```

The implementation of *jchk* is made up of the following steps:

| IMPLEMENTATION OF *jchk* | |
|---|---|
| **EXPLANATION** | **IN THE EXAMPLE** |
| Take an original record from the input database | A record from *cds* |
| Implement the indicated format as an argument of *jchk* on the original record and consider each line produced by this format as a search key. The format mode MHU, or MPU, is necessary in general to obtain the keys, because the standard ISISUC.TAB file converts lower-case into upper-case for the generation of the inverted file. | Format: mhu,(v70)/<br>Original record: record that comes from the *cds* database. |
| For each key obtained in the previous step search the alternative inverted file. | Alternative inverted file: AUTHOR |
| Add to the original input record a field (repeatable) 32001 for the first *jchk*, 32002 for the second *jchk*, etc. The field 32001 will have as many occurrences as the different keys that are produced in step 2. | The control fields are added to the record being processed. |



Step (3) does not use the IFP file.

If the key is present in the inverted file, *jchk* assumes that it is a valid key and in step (4) it will add to the output in field 32001 a subfield ^m with a value 1: on the other hand field 32001 will not have a subfield ^m.

**Example:**

This example produces the output as follows.

```
mx cds jchk=AUTHOR=MHU,(v70/)
mfn= 1
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
50 «Incl. bibl.»
69 «Paper on: <plant physiology><plant transpiration><measurement and
instruments>»
24 «Techniques for the measurement of transpiration of individual plants»
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
32001 «AUTHOR^l2^kMAGALHAES, A.C.^o1^m1»
32001 «AUTHOR^l2^kFRANCO, C.M.^o2^m1»
..
```

## Content of the control fields (32001, 32002, etc.)

Field 32001 and the successive fields are produced to record the results of the *jchk* process, and they are composed of the following subfields:

**Example:**

```
32001 «AUTHOR^l2^kFRANCO, C.M.^o2^m1»
```

| SUBFIELDS OF FIELD 32001 | |
|---|---|
| **FIELD** | **CONTENT** |
| | First data, without subfield indicator: name of the alternative inverted file |
| ^l[1/2] | Subtree of the inverted file where the key is extracted from, 1 for terms up to 10 characters long and 2 for terms between 11 and 30 characters |
| ^k | Key that the search is made with |
| ^o | Number of the field occurrence in the original record that the key comes from |
| ^m1 | If the key exists a subfield *m* is created that contains: "1".<br>If the key does not exist in the alternative inverted file, the subfield ^m is not created. |

**Example:**

1. Produce a list of invalid authors:

```
mx CDS jchk=AUTHOR=mhu,(v70/) pft=@check.pft –all now
```

The file *check.pft* has the following format specifications:

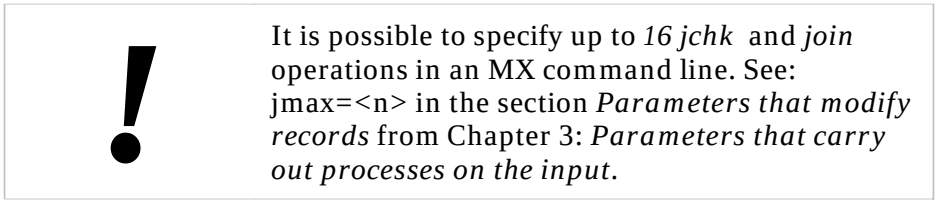
```
if p(v32001) then (if a(v32001^m) then mfn,x2,v32001^k | Does not exist | / fi)
fi
```

Note that this example is the same as using a *join* but, because it is not necessary to carry out a *ref* for each output and to recover the record, the run time is less.

## Advantages of *jchk* over *join*

The run time is much quicker, since it is not necessary to access each record; it is sufficient to know that the term exists.

Another advantage is that *jchk* does not use the file .IFP. In the case of distributing validation files (*authority files*) it is not necessary to give this file, which saves disk space.




## Tables for character conversion

**convert=ansi**

The parameter *convert=ansi* converts decimal ASCII characters into alphanumeric characters defined in an internal table. By default CISIS uses ASCII characters defined for the IBM PC, according to the codepage installed on the computer.

## Tables for definition of alphanumeric characters

**[actab=<file> |ansi ]**

Using the parameter *actab* you can indicate the table of decimal ASCII codes for the alphanumeric characters to be considered as part of a word. This table is used for the indexing technique (IT) word-by-word (IT 4, 8, 1004, 1008). *actab=ansi* uses an internal table for this procedure.

If this parameter is not indicated, CISIS by default uses the standard corresponding CDS/ISIS table (ISISAC.TAB) defined for the IBM PC.

## Tables for the conversion of a characters to uppercase

**[uctab=<file>|ansi]**

Using the parameter *uctab* you can indicate the conversion table of 256 ASCII characters and their upper-case equivalents.

If this parameter is not indicated, CISIS by default uses the corresponding standard CDS/ISIS table (ISISAC.TAB) defined for the IBM PC. *uctab=ansi* uses an internal table for this procedure.

## Field selection table – generation of keys - fst

**fst[/h]={<fst>|@[<file>]} [stw=@[<file>]]**

**ln{1|2}=<out_file> [+fix[/m]]**

The parameter *fst* extracts the record keys according to the specification in the table *FST* and the optional *stopword* file. If the arguments ***ln1*** and/or ***ln2*** are indicated, the keys generated are kept in these files. If these arguments are not indicated the keys are added as new fields to the record that is being processed, with *tag* equal to the identifier of the line in the *FST* where they are extracted from.

Where *tag* already exists, they are added as new occurrences of the field.

**Example:**

```
mx cds "fst=24 4 v24"
mfn= 1
24 «Techniques for the measurement of transpiration of individual plants»
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
24 «TECHNIQUES^m1^o1^c1^l1»
24 «FOR^m1^o1^c2^l1»
24 «THE^m1^o1^c3^l1»
24 «MEASUREMENT^m1^o1^c4^l2»
24 «OF^m1^o1^c5^l1»
24 «TRANSPIRATION^m1^o1^c6^l2»
24 «OF^m1^o1^c7^l1»
24 «INDIVIDUAL^m1^o1^c8^l1»
24 «PLANTS^m1^o1^c9^l1»
```

Content of the subfields:

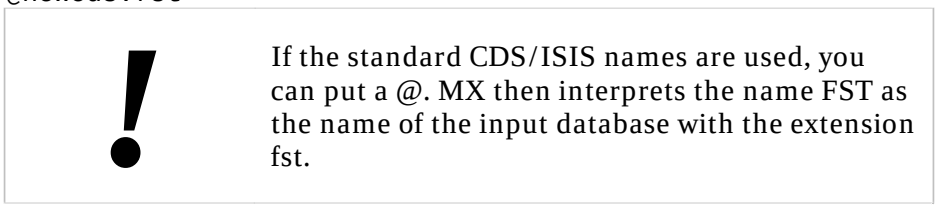
| Subfield | Content |
|---|---|
| KEY | Extracted with the FST indicated. |
| ^m | MFN that the key is extracted from. |
| ^o | Occurence of the field where the key appears. |
| ^c | Sequence of this key within the field. |
| ^l[1/2] | File where the key is stored: 1 for output of up to 10 characters long, 2 for terms of 11 to 30 characters long. |

### Reference to a selection table of external fields

**fst[/h]=@[<file>]**

An external file can be referenced instead of writing the FST in the command line:

```
mx CDS fst=@newcds.fst
```



If the standard CDS/ISIS names are used, you can put a @. MX then interprets the name FST as the name of the input database with the extension fst.

**Example:**

```
mx CDS fst=@cds.fst
```

is equivalent to:

```
mx CDS fst=@
```

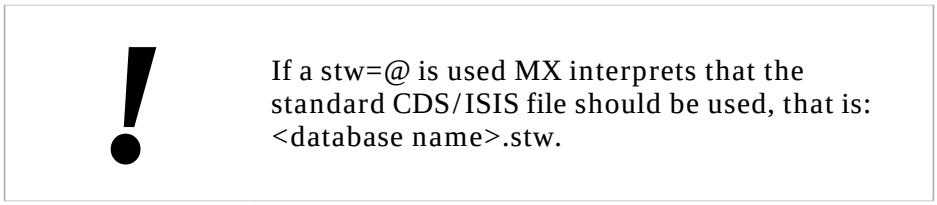
The parameter /h allows the format that extracts the keys to be not limited to 30 (60) characters. All the keys are generated in field 33000, generating an occurrence of this fields for each line extracted from the FST, not from an occurrence of the original data.

## Stopword file

**stw=@[<file>]**

Using a file of non-significant words (*stopwords*) avoids the generation of non significant terms (pronouns, prepositions, etc.).

A stop word file is an ASCII text file with one word per line.



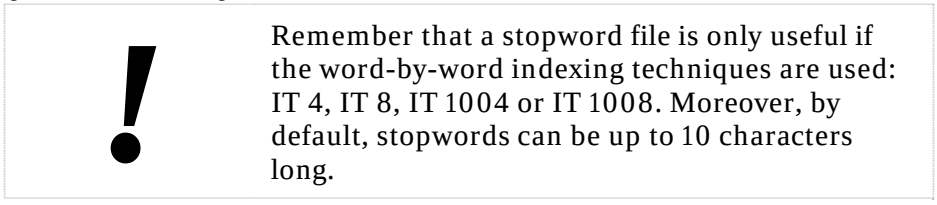
If a stw=@ is used MX interprets that the standard CDS/ISIS file should be used, that is: <database name>.stw.

**Example:**

```
mx cds fst=@cds.fst stw=@
```

is equivalent to:

```
mx cds fst=@cds.fst stw=@cds.stw
```



Remember that a stopword file is only useful if the word-by-word indexing techniques are used: IT 4, IT 8, IT 1004 or IT 1008. Moreover, by default, stopwords can be up to 10 characters long.

**Example:**

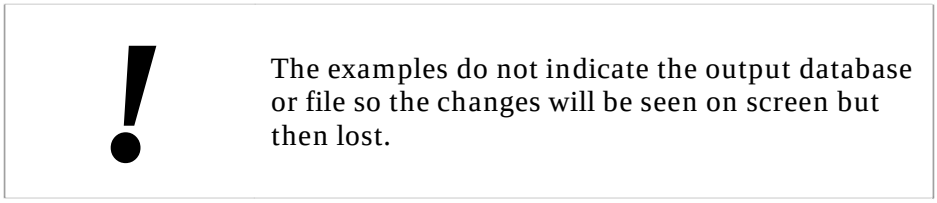
The following example uses a file of *stopwords*.

The example supposes the existence of a *stopwords* file (cds.stw*)* with the following content:

```
A
AN
AND
AS
BY
FOR
FROM
IN
INTO
ITS
OF
ON
THE
TO
UPON
WITH
```

The command line:

```
mx CDS "fst=24 4 v24" stw=@cds.stw
Mfn= 1
24 «Techniques for the measurement of transpiration of individual plants»
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
24 «TECHNIQUES^m1^o1^c1^l1»
24 «MEASUREMENT^m1^o1^c4^l2»
24 «TRANSPIRATION^m1^o1^c6^l2»
24 «INDIVIDUAL^m1^o1^c8^l1»
24 «PLANTS^m1^o1^c9^l1»
..
```



The examples do not indicate the output database or file so the changes will be seen on screen but then lost.

## Indexing techniques 1-8 / 1000 - 1008

The CISIS interface accepts all the standard indexing techniques (IT) of CDS/ISIS, (IT 0 to IT 8), plus eight new methods (IT 1000 - IT 1008).

The techniques IT 1000...1008 generate a key made up of:

```
<id>   1000+IT   '/', <mfn_fmt>, '/',  <fmt>
```

1. Opening delimiter character.
2. The MFN.
3. Closing delimiter character (the same that is used for opening).
4. The normal key extracted according to the corresponding IT 0 to 8.

Examples:

```
1 1004  '|', f(l->idif(id.10),1,0), '|', (v83^*)
1 1004  '|', v98 '|', v83
```

By this technique it is possible to control programmatically the MFN component of the posting which is generated.

**Example:**

1. Supposing that, the value 23 is found in field 999 of the first record in the *cds* database, and that record 2 does not have field 999. Then the command:

```
mx CDS to=2 "fst=1 1000 '/', if p(v999) then v999 else mfn fi,'/',,,(|AU=|v70/)
"
```

generates the output:

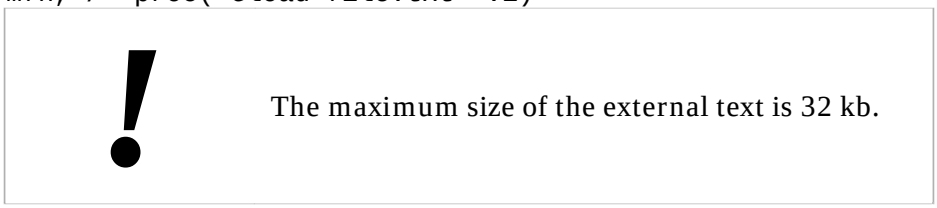
```
mfn= 1
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
50 «Incl. bibl.»
69 «Paper on: <plant physiology><plant transpiration><measurement and
instruments>»
24 «Techniques for the measurement of transpiration of individual plants»
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
999 «23»
1 «AU=MAGALHAES, A.C.^m23^o1^c1^l2»
1 «AU=FRANCO, C.M.^m23^o1^c2^l2»
..
mfn= 2
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
50 «Incl. bibl.»
69 «Paper on: <plant evapotranspiration>»
24 «<The> Controlled climate in the plant chamber and its influence upon
assimilation and transpiration»
26 «^c1965»
30 «^ap. 225-232^billus.»
70 «Bosian, G.»
1 «AU=BOSIAN, G.^m2^o1^c1^l2»
..
```

Note that in record MFN=1 the postings for the authors are shown as coming from MFN=23 whereas MFN=2 generates a standard posting.

Example

- Index of an associated external text to a record

```
1 1004 '/',mfn,'/' proc('Gload=file.txt'=v1)
```



The maximum size of the external text is 32 kb.

## Generation of link files (links)

**ln1=<out_file>/ ln2=<out_file>**

If the parameter *fst* has the arguments *ln1* and/or *ln2* then a corresponding link file is generated.

If you want the files to have the standard CDS/ISIS extensions you can indicate this specifically. CISIS files can have the standard ISIS extensions (pft, fst, ln1, ln2, etc.) or not.

**Examples:**

```
mx CDS "fst=24 4 v24" ln1=cds.ln1 -all to=10 now
mx CDS fst=@cds.fst ln2=cds.ln2 -all to=10 now
mx CDS fst=@otra ln1=cds.ln1 ln2=cds.ln2 -all to=10 now
mx CDS fst=@otra ln1=otro.1 ln2=otro.2 -all to=10 now
```

## Link files of fixed length

**Option +fix[/m]**

The option *+fix[/m]* generates files .ln[1/2] of fixed length, not standard CDS/ISIS ones, that can only be processed with CISIS utilities. Files of fixed length require more disk space, but they can be sorted more quickly.

The standard *CDS.LN1* output, recognised by *CDS/ISIS* is:

1 24 1 1 TECHNIQUES
1 24 1 8 INDIVIDUAL
1 24 1 9 PLANTS

The special fixed length output has the structure:

KEY MFN TAG OCC CNT

```
mx CDS "fst=24 4 v24" ln1=cds.ln1 +fix
```

That produces the following file:

```
TECHNIQUES    1        24      1        1
INDIVIDUAL    1        24      1        8
PLANTS        1        24      1        9
```

With the option *+fix/m* fixed length files .ln[1/2] are generated with only the KEY and the MFN of the posting.

The link files with KEY and MFN, without the TAG and OCC components, are used for applications that handle more than one inverted file generated for a master file.

# Output

## Execution of an external program

**sys[/show]={<sys_fmt_spec> | @<file>}**

The parameter *sys=* initiates the execution of an operating system command, that should be specified as a format instruction. The result of this format should produce valid commands for the operating system.

Each line generated by the format will execute as an operating system command.

**Example:**

Supposie that field 777 of the database record contains the name of a text file (alltext1.txt) that you want to display, and it contains ASCII text (the full text of a document, for example):

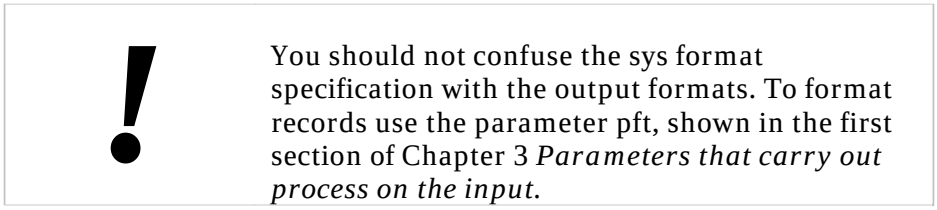
Add to database record 1 the field 777 with the content *alltext1.txt*

```
mx cds to=1 proc='a777#alltext1.txt#'  copy=cds
```

Create a file *alltext1.txt* with some content.

Execute MX indicating that it should display the file *alltext1.txt* with the parameter *sys*

```
mx cds "sys=if p(v777) then 'more 'v777 fi"
```




The following example is incorrect:

```
mx cds "sys=if v24:'water' then mfn/  'dir *.mst'  fi"
```

It should be entered as:

```
mx cds "pft=if v24:'water' then mfn/ fi"  "sys=if v24:'water' then 'dir *.mst '
fi"
```

### Option */show*

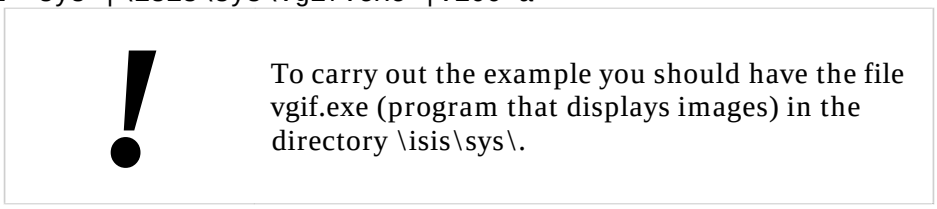
The option */show* stops the execution of the command indicated in *sys*. It shows the command that will be executed, and waits for the intervention of the operator to continue.

```
mx cds "sys/show=if v24:'water' then ‘dir *.mst' fi"  to=20 now
```

One application of the parameter *sys* could be to execute an external program on the record data, for example, an image whose file name is stored in a field.

**Example:**

```
mx cds -all "sys=|\isis\sys\vgif.exe |v100^a"
```



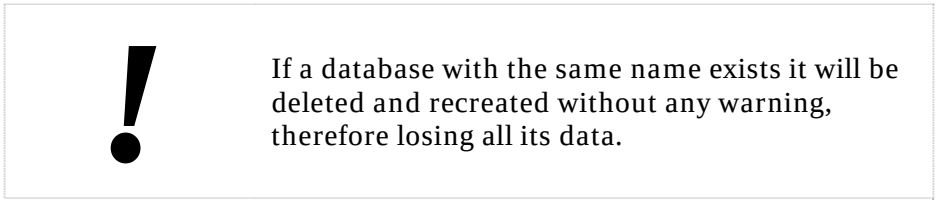

## Parameters that create/modify databases

**{create | copy | append | merge | updatf}=<out_dbn>**

### Creation of a master file

**create=<out_dbn>**

The parameter *create* creates and initializes unconditionally a master file with the name assigned in <out_dbn>, the master file in which the records resulting from the process are stored.

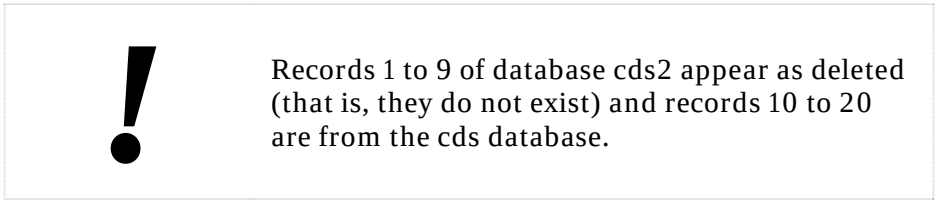


If a database with the same name exists it will be deleted and recreated without any warning, therefore losing all its data.

**Examples:**

- The following example reads records 10 to 20 from the *cds* database and writes them in a database *cds2*;

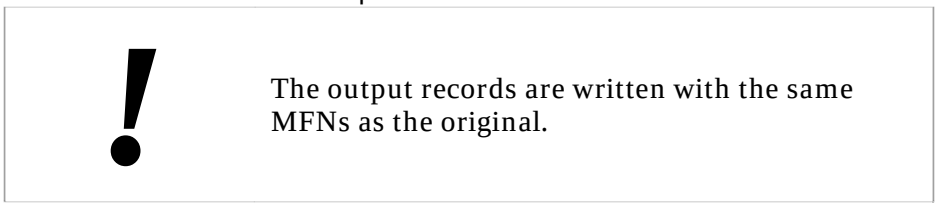
```
mx cds from=10 to=20 create=cds2 now -all
```



Records 1 to 9 of database cds2 appear as deleted (that is, they do not exist) and records 10 to 20 are from the cds database.

- Create a database with records resulting from a search:

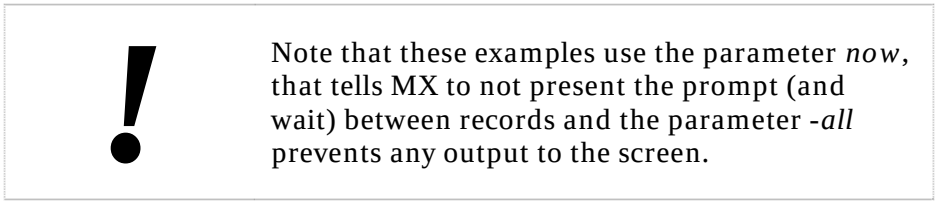
```
mx cds "plants and water" create=plawat now -all
```



The output records are written with the same MFNs as the original.

- Create a database from an ISO-2709 file (the file cds.iso must exist):

```
mx iso=cds.iso create=cds3 now -all
```



Note that these examples use the parameter *now*, that tells MX to not present the prompt (and wait) between records and the parameter *-all* prevents any output to the screen.

## Copy records to a master file

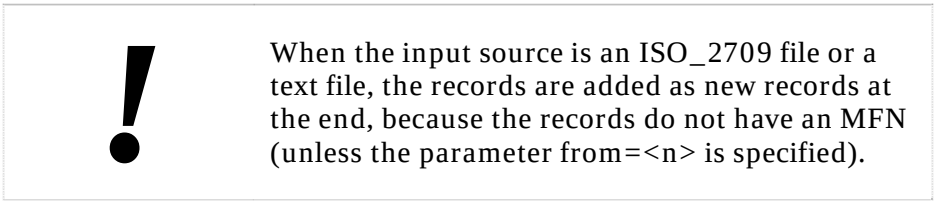
**copy=<file>**

The parameter *copy* writes the processed records to an output master file <out_dbn> with the MFN that the record had when it was read. If a record with this MFN already exists in <out_dbn> its content will be lost and if the record does not exist it will be created.

When the output database (<out_dbn>) is the same as the input, *copy* will modify the records, since they are read, modified and recorded in the same database with the same mfn.

Unlike *create*, *copy* does not reinitialize the destination database.

If the destination database does not exist, it is created. In this case *copy* functions exactly the same as *create*.



When the input source is an ISO_2709 file or a text file, the records are added as new records at the end, because the records do not have an MFN (unless the parameter from=<n> is specified).

**Examples:**

- To copy records 30 to 40 from the database *cds* to database *cds2*:

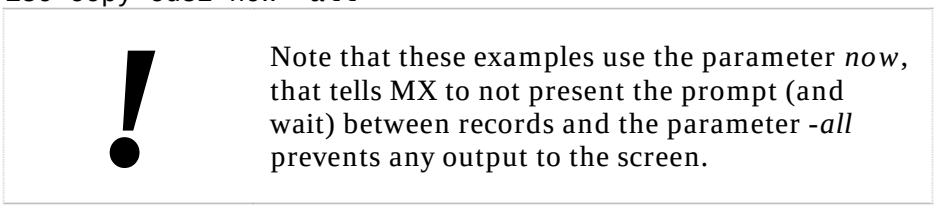
```
mx cds copy=cds2 now from=30 to=40 -all
```

- To copy the result of a search to a new database:

```
mx cds water copy=cds2 now -all
```

- To read records from an ISO-2709 file and write them in a database that already exists:

```
mx iso=cds.iso copy=cds2 now -all
```



Note that these examples use the parameter *now*, that tells MX to not present the prompt (and wait) between records and the parameter *-all* prevents any output to the screen.

## Add records to a database

**append=<dbn_out>**

The parameter *append* adds the processed records as new records to the output database at the end. Unlike *create* and *copy*, the records processed lose their original mfn number.

If the destination database does not exist it is created.

**Examples:**

- Add records 30 to 40 from the database cds to the database *cds2*:

```
mx cds append=cds2 now from=30 to=40 -all
```

- Add the result of a search to a new database:

```
mx cds "plants * water" append=cds2 now -all
```

- Read records from an ISO-2709 file and write them in a database that already exists:

```
mx iso=cds.iso append=cds2 now -all
```

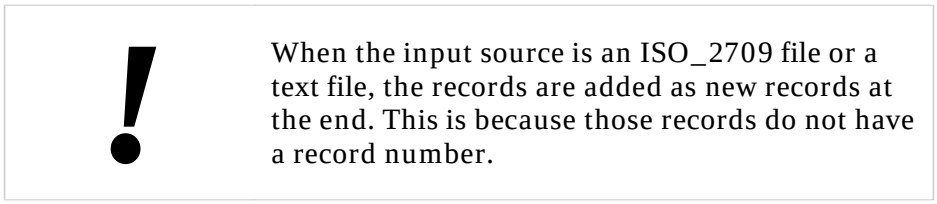
## Mix/Merge records

**merge=<outdbn>**

The parameter *merge* writes the processed records in the output database only if those records do not exist in <out_dbn>.

Supposing that in the database<out_dbn> all the records between MFN=1 and MFN=20 exist with the exception of record MFN=10, and that <in_dbn> is composed of 15 records with MFN 1 to 15. If we apply a merge, the resulting database will be composed of 20 records, of which 10 are from <in_dbn> and the rest are those which were already in <out_dbn>.

If this output database <out_dbn> does not exist, it is created.



When the input source is an ISO_2709 file or a text file, the records are added as new records at the end. This is because those records do not have a record number.

**Examples:**

- To insert the *cds* records, with mfn between 1 and 20, that are not in *cds2*:

```
mx cds merge=cds2 now from=1 to=20
```

- To insert the records returned by a search made in *cds* that are not in *cds2:*

```
mx cds merge=cds2 energy$ -all now
```

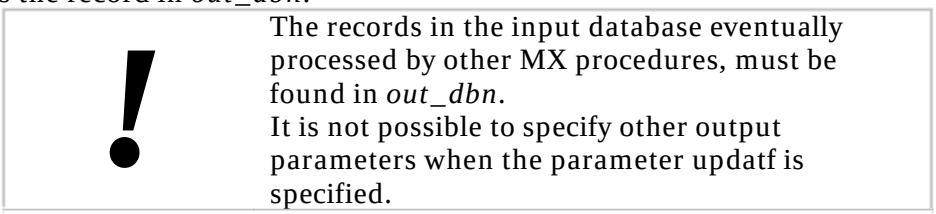
## Updating fields

**updatf=<out_dbn>**

The parameter *updatf* (*update fields*) takes the input record data and substitutes the fields of the output record with it, where the tags coincide.

*out_dbn* is the database that has the records without updated fields and that will be rewritten. The process of updating that MX carries out on *out_dbn* is the following:

1. It reads a record from the input database and executes the *gizmo*, *join*, *proc* and *fst* processes if they are specified
2. It searches for a record with the same mfn in *out_dbn*, with the following result:
   - all the fields of the record from *out_dbn* that exist in the record of the input database are replaced by these
   - all the fields of the record from *out_dbn* that do not exist in the record of the input database are maintained
3. It rewrites the record in *out_dbn*.



The records in the input database eventually processed by other MX procedures, must be found in *out_dbn*.
It is not possible to specify other output parameters when the parameter updatf is specified.

The result of the execution of the parameter updatf is not displayed by MX, i.e. the record which is rewritten in *out_dbn* is not shown, but the record which goes to update the record of *out_dbn* is shown.

**Examples:**

Suppose there are two databases:

| **INPUT** | **CDS** |
|---|---|
| Mfn= 1<br>1 «field 1 new»<br>2 «field 2 new»<br>4 «field 4 new» | Mfn= 1<br>44 «Methodology of plant eco-physiology: proceedings<br>69 «Paper on: <plant physiology><plant transpiration><br>24 «Techniques for the measurement of transpiration<br>26 «^aParis^bUnesco^c-1965»<br>30 «^ap. 211-224^billus.»<br>70 «Magalhaes, A.C.»<br>70 «Franco, C.M.»<br>1 «field1»<br>1 «field 1b»<br>2 «field 2»<br>2 «field2b»<br>2 «field 2c»<br>3 «field3» |

The following example carries out updating on the *CDS* record fields taking as the source the records from the database *INPUT:*

```
mx INPUT updatf=CDS -all now
```

```
mx CDS
```

This is the result of a record from the *CDS* database with the changes:

```
mfn= 1
44 «Methodology of plant eco-physiology: proceedings
69 «Paper on: <plant physiology><plant transpiration>
24 «Techniques for the measurement of transpiration
26 «^aParis^bUnesco^c-1965»
30 «^ap. 211-224^billus.»
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
3 «field3»
1 «field 1 new»
2 «field 2 new»
4 «field 4 new»
```

Note that the fields 1 and 2 of *CDS* were substituted by the original ones from the database INPUT, but the other fields (including field 3) were not modified, and in addition field 4 was added.

## Generate a ISO_2709 file

**[out]iso[={marc|<n>}]=<out_isofile> [outisotag1=<tag>]**

The MX program can read and write files in the ISO-2709 format and apply the same procedures as on the .MST/XRF files (excepting those processes that require the use of an inverted file or dictionary).

The option *iso=<out_isofile>* or *iso=marc|<n>=<out_isofile>* allows the creation of ISO-2709 files. In the same way that a master file can be created, an ISO-2709 file can be created.

With the option *iso=<n>=<out_isofile>* you can indicate a fixed length for the lines. This option is used to interchange PC .ISO files with computers such as an HP running MINISIS. The field separator and the record separator in the file generated by MX are the # character. Indicating *iso=0=<filename>* the lines in the records will be of variable length.

The option *iso=marc=<out_isofile>* will generate output lines with the following characteristics, compatible with the MARC definition:

- Lines of variable length
- The field and record separators will be the ASCII characters '\029' '\030'

- The lines will finish with only '\012' and will not include the '\013' (CR).

**Examples:**

- This example produces an ISO-2709 file from the CDS database

```
mx CDS iso=output.iso -all now
```

- Take as input a database (CDS), carry out a search (plants and water) and create an ISO-2709 output file (*cds.iso*) from the records retrieved.

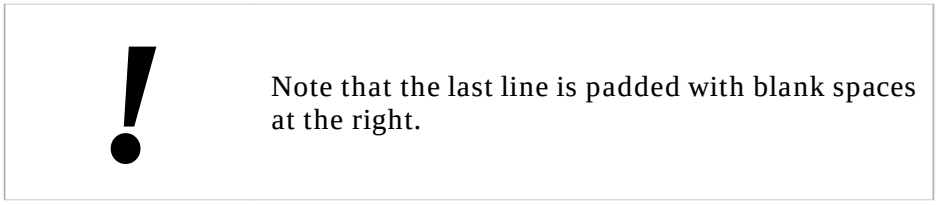
```
mx cds "plants and water" iso=cds.iso now -all
```

- This example generates an ISO-2709 file with record 1 from the database CDS, with a fixed line length = 40 characters.

```
mx cds iso=40=output.iso to=1 now -all
```

It creates an output of the type:

```
00409000000000109000450004400780000000690
07900078024006900157026002300226030002l0
02490700016002700700013002864Methodology
of plant eco-physiology: proceedings of
the Montpellier Symposium4Paper on: <pl
ant physiology><plant transpiration><mea
surement and instruments>4Techniques for
the measurement of transpiration of ind
ividual plants4^aParis^bUnesco^c-1965#^a
p. 211-224^billus.4Magalhaes, A.C.4Franc
o, C.M.##
```



Note that the last line is padded with blank spaces at the right.

Remember that an ISO-2709 file can be read with MX, indicating that it is the input source:

```
mx iso=cds.iso
```

## Generate an ASCII file with separators

**fix=<out_file>**

Displays each record in a single line in pure ASCII format, separating all the fields by a pipe ( | ). All the field tag information is lost. The maximum length of the line is determined by the operating system.

**Example:**

- Using a database called *ADMIN* that contains publisher, title and price of each book, the following instructions generate an output file *PRICES*:

```
mx admin fix=prices now -all
```

PRICES file**:**

```
Publisher 1|Title 1|124
Publisher 1|Title 2|100
Publisher 2|Title 3|89
Publisher 2|Title 4|99
Publisher 2|Title 5|101.50
```

This file can be imported by other programs that read field data separated by delimiters. For example, it can be imported by an spreadsheet such as Excel to carry out calculations for statistical analysis, economics, etc.

## Interchanging data from the leader of the record

**[[out]isotag1=<tag>]**

MARC records store data in positions 5-8 and 17-19 at the start of the record (leader). The CISIS format instructions do not have direct access to these positions, but it is possible to convert these bytes into conventional fields of the master record in the import or export of records.

In the conversion process, the data from the leader is loaded in a consecutive series of fields from a base value + the position of the byte in the leader. For example, if 3000 is designated as a base, then byte 5 from the leader is loaded in field 3005. During the export, the values will be taken from those fields that are registered in the positions corresponding to the leader. If there are no data in the field, a blank space will be put in the corresponding leader position.

**Example**

```
mx mrclte
mfn=     1
  1  «00089048230 /AC/r91»
  3  «DLC»
  8  «891101s1990    maua   j      001  0eng  »
 10  «##^a###89048230 /AC/r91»
 20  «##^a0316107514»
 40  «##^aDLC^cDLC^dDLC»
 50  «00^aGV943.25^bB74 1990»
 82  «00^a796.334/2^220»
100  «##^aApter, David Ernest^d1919-1999»
245  «10^aMake the team.^pSoccer :^ba heads up guide to super soccer! /^cRi J.
Brenner.»
250  «##^a1st ed.»
260  «##^aBoston :^bLittle, Brown,^cc1990.»
300  «##^a127 p. :^bill. ;^c19 cm.»
3005  «c»
```

```
3006  «a»
3007  «m»
3017  «5»
3018  «i»
```

## Load elements generated by an FST

### Function fullinv

**Option /dict**

The option */dict* creates only the dictionary from the file <out_inf>: it does not create content in out_inf.ifp.

```
mx CDS fst=@ ifupd/create/dict=cds now from=10 to=30
```

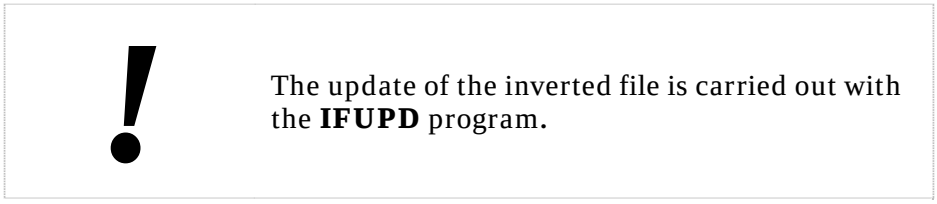
An inverted file created with the option /dict can be used for MX processes of type jchk.

**Option /m**

extracts the keys (updates the inverted file) only with the MFN component of the posting.

**Option /ansi**

When **ansi** is indicated, an internal pattern table is used for the ANSI character set, removing the need to use the parameters **actab** and **uctab**.



The update of the inverted file is carried out with the **IFUPD** program.

## Tabulation of frequency

**tab[/lines:100000/width:100/tab:<tag>]=<tab_fmt>**

generates a frequency table from the content of a field/subfield. The tabulated value is the difference from 1,000,000,000 (*high-value*), which allows sorting in ascending or descending order.

It writes the content of the tag <tag> in a file of the name <mfn>.<tag>

**Example**

```
mx cds  "tab=(v26^c/)" now lw=0 | sort | more
999999949|50|1966
999999962|37|1976
999999974|25|1965
999999991|8|1973
999999994|5|1975
999999995|4|1974
999999997|2|1968
999999997|2|1971
999999997|2|27 Aug. 1976
999999998|1|11 Dec. 1975
999999998|1|14 June 1976
999999998|1|15 June 1976
999999998|1|17 Sept. 1976
999999998|1|1983
999999998|1|23 Oct. 1975
999999998|1|25 June 1976
999999998|1|6 Feb. 1976
```

# Initialization parameters / setup variables

These parameters allow you to change the values that MX takes by default such as: the maximum size of a record, the output format, the definition of setup variables or of logical names, etc.

## CISIS parameters file

**cipar=<file>**

The file CIPAR is a tool that provides the functions of SYSPAR.PAR and <dbn>. par in the standard CDS/ISIS . It also adds other functions.

CIPAR is a pure ASCII file that – among other functions – relates defined logical names in CIPAR to the actual physical names of the files: in this way it becomes independent of the real name and its location.

A detailed explanation of this parameter is given in the Appendix “CIPAR File”.

Example:

If the archive *\dir\filename* contains:

```
cds.*=\cisis\databases\cds.*
cds1.pft=\cisis\databases\cds1.pft
```

the command:

```
mx cipar=\dir\filename cds pft=@cds1.pft
```

is equivalent to:

```
mx \cisis\databases\cds pft=@\cisis\databases\cds1.pft
```

## Maximum record size

**mfrl=<n>**

This parameter determines the maximum size that a record can have in reading, processing and writing operations, including *join proc and fst*. The set value for MX is 32767 bytes, whereas in CDS/ISIS for MS-DOS it is 8,192 bytes and in Winisis it is 64Kb

It is necessary to bear in mind that when records are processed by the different internal MX procedures (such as *join* or *proc*) they can end up with temporary lengths larger than this limit.

Examples:

```
mx mfrl=32000 CDS a$ join=AUTHORS=mhu,(v70/) create=OUT
```

## Maximum size for the result of a format

**fmtl=<n>**

This parameter determines the maximum size of the internal area where the results of all formats are stored (*pft*, *sys*, *join*, etc). The set length is equal to the length of the record by default.

Example:

```
mx mfrl=20000 fmtl=20000 TITLES join=ISSUES=v30 pft=@holdgins.pft
```

# General parameters

## Parameters that control output to the screen

**{+ | -}{control | leader | xref | dir | fields | all }**

These parameters determine the output that the *dump* procedure produces in the standard output (by default, the screen) of the data, that is used to modify the .MST and .XRF records.

The set output is +*fields*, which implies unformatted output of all the record fields.

Deleted records are included in the dump and are indicated by the message [DELETED].

### Parameter +

Activates the options associated with the procedure to output to the screen. It cancels the predefined options the first time that it is used.

### Parameter -

Deactivates the options associated with the procedure to output to the screen. It activates all the other predefined options the first time that it is used.

**Examples:**

```
mx CDS +xref
mx CDS -xref
mx CDS +control +leader +dir
mx CDS +all
```

Note that using *mx CDS -xref* is equivalent to *mx CDS +control +leader +dir +fields*.
Also, *mx CDS -xref -fields* is equivalent to *mx CDS +control +leader +dir*.

The parameters refer to the following parts of .MST and .XRF, that do not have direct access to standard CDS/ISIS.

| Option | Displays |
|---|---|
| control | MST control record, identified as MFN=0 |
| leader | Segement of fixed length of 18 bytes at the beginning of each MFN |
| dir | Directory within the record that contains the data field indexes |
| fields | The data contained in the record fields |
| xref | Content of the file .XRF |
| all | Activates or deactivates all the previous options |

To understand better the use of these parameters it is essential to understand the structure of ISIS records.

A record with an ISIS structure has two special characteristics that offer great versatility for the management of textual information: repeatable fields and variable length.

Since the records do not have a predetermined length and the fields do not have a fixed length, or a predetermined number of occurrences, it is not possible to have direct access to any portion of data in the database.

Access to a record is done indirectly, through pointers in an auxiliary file with the extension .XRF, and within the record data are accessed through pointers in a directory.

The file .XRF contains all the information necessary to find the start point of the record in the .MST.

For more details see the Appendix “Structure of records in an ISIS database”.

# Parameters for multi-user environments

## Process options

**[mono | mast | full]**

If a database is updated in a multi-user environment it should be indicated before the parameter *mast* or *full*. The value *mono* (single use) is the default value.

### Single use mode: mono

Is the default mode. It does not verify consistency or access to the data. It supposes a single user.

This parameter corresponds to the syspar parameter 14=0 of CDS/ISIS.

### Limited access to data: mast

This parameter allows the simultaneous retrieval and updating of the master file. It ensures that the inverted file is not updated.

This parameter corresponds to the syspar parameter 14=2 of CDS/ISIS.

### Complete access: full

When this parameter is used it can simultaneously retrieve and modify, in the master file as well as the inverted file. On the other hand, the system becomes slower.

This parameter corresponds to the syspar parameter 14=1 of CDS/ISIS.

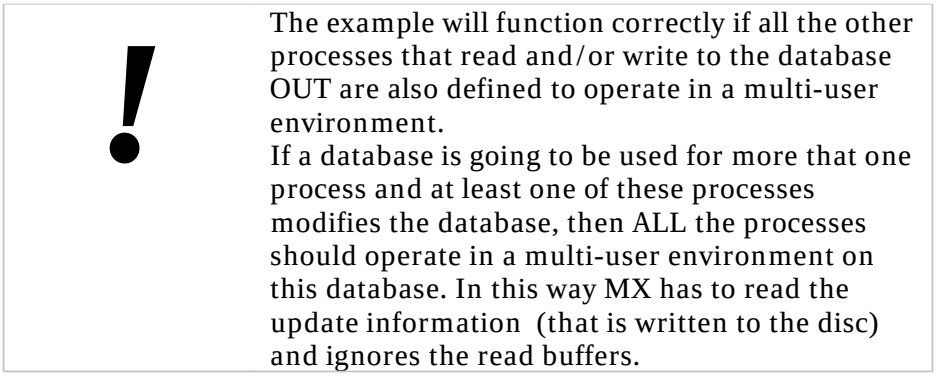
**Examples:**

- Put two databases in a multi-user environment, *CDS* and *OUT*:

```
mx mast CDS from=120 append=OUT now -all
```

To place only one database in a multi-user environment: the database *CDS* will be single-user, whereas the database *OUT* will be multi-user.

```
mx CDS from=120 mast append=OUT now -all
```




## Other parameters

### Delimiters

**?**

The equal sign used as a delimiter can be substituted by the question mark ?

```
mx CDS from=10    is correct
mx CDS from?10    is equivalent to the above
```

### Prompts

**p1|p2=<prompt>**

It is possible to change the MX prompts during an MX session.

```
parameter p1=<prompt1>   change .. to <prompt1>
parameter p2=<prompt2>  change -> to <prompt2>
```

**Example:**

To change prompts in such a way that between records MX says: press <enter> for the next record or <x> to exit. (*prompt1*) and when the record finishes displaying the message reads: press <x> to exit or type a search expression (*prompt2).*

```
mx cds "p1=<enter> for the next record or <x> to exit" "p2=press <x> to exit or
type a search expression:" plants
```

Viewing one of the records from the search (not the last):

```
mfn= 13
24 «Experience with three vapour methods for measuring water potential in
plants»
26 «^c1965»
30 «^ap. 369-384^billus.»
```

```
44 «Methodology of plant eco-physiology: proceedings of the Montpellier
Symposium»
69 «Paper on: <plant physiology><water><pressure><measurement and instruments>»
70 «Barrs, H.D.»
70 «Slatyer, R.O.»
50 «prueb»
<enter> for the next record or <x> to exit
```

Viewing the last record from the search:

```
mfn= 68
24 «Some important animal pests and parasites of East Pakistan»
26 «^c1966»
30 «^ap. 285-291^billus.»
44 «Scientific problems of the humid tropical zone deltas and their
implications: proceedi+»
50 «Incl. bibl.»
69 «Paper on: <pests><parasites><biology><ecology><plants><agriculture><public
healt»
70 «Yosufzai, H.K.»
press <x> to exit or type a search expression:
```

## Parameter *trace*

**trace**

The parameter *trace* activates the output of internal information from any process.

## Parameter *mfrl*

**mfrl**

The parameter *mfrl* shows the maximum length of the record that has been processed up to this moment.

**Example:**

```
mx CDS now -all mfrl
MFN=1 -> MFRmfrl=408
MFN=3 -> MFRmfrl=450
MFN=21 -> MFRmfrl=452
MFN=27 -> MFRmfrl=822
MFN=104 -> MFRmfrl=980
```

# MX: execution return code

MX finishes its execution with an output value of 1 or 0 depending on whether an error has occurred or not. This return code can be used to control batch processes (*errorlevel* in MS-DOS).

# Master file utilities

## MXF0 - Program

The program MXF0 analyses all the records of a master file, producing information about their fields and about the frequency of the characters.

The result of the execution of MXF0 is a master file with a record which contains the following information:

- Name of the database, date, time, number of records, number of active records, number of records logically deleted and number of records physically deleted.
- For each *tag* of a distinct field found in the records of the input file, it produces an occurrence of a repeatable field which contains: the label (*tag*), the frequency, total occurrences, maximum and minimum length, and the total number of characters.
- A repeatable field with an occurrence for each distinct character found in the input data, its hexadecimal code and the number of times that it appears.

This utility program can be used to produce a list of the field tags present in a database. It also serves to verify whether:

- mandatory fields are present
- non-repeatable fields occur more than once
- fields of fixed length (like normalized dates) have the correct length, etc.

# MXF0 - Presentation

The following line processes the master file *cds* located in the directory \cisis\cds, reinitializes the master file *x* (in the working directory) and stores the result in the first record, as indicated below:

```
mxf0 \cisis\cds\cds create=x 0
```

The resulting record in the database *x* is

```
mfn=     1
1001  «cds»
1003  «20051014 09:44:40 Fri»
1009  «      149»
1010  «      149»
1011  «        0»
1012  «        0»
1013  «      150»
1020  «^t024^d      148^o      148^l    6^u  179^n      9589»
1020  «^t025^d        5^o        7^l    2^u   22^n        55»
1020  «^t026^d      147^o      148^l    6^u  101^n      2897»
1020  «^t030^d      146^o      146^l    6^u   36^n      2484»
1020  «^t044^d       80^o       80^l   22^u  112^n      7525»
1020  «^t050^d       99^o       99^l   11^u  209^n      2043»
1020  «^t069^d      148^o     1134^l    3^u   36^n     16318»
1020  «^t070^d      121^o      163^l    8^u   35^n      2493»
1030  «^tall^x20^n      4516»
1030  «^tall^x22^n         8»
1030  «^tall^x27^n        29»
...
1030  «^tall^xa1^n        10»
1030  «^tall^xa2^n        10»
..x
```

This record states that the master file \cisis\cds\cds contains 149 active records, and 8 different data fields: tags 24, 25, 26, 30, 44, 50, 69 and 70.

The field with tag 24 (title) occurs once in each of the 149 records (the shortest occurrence is 6 characters, the longest 179, and the total of these occurrences comes to 9589 bytes).

The field 70 (author) occurs in 27 records, and has in total 163 ocurrences.

Considering all the data fields of \cisis\cds\cds, the character code *20* (the ASCII code for space in hexadecimal, in decimal: 2x16 + 0 = 32) ocurrs 4,516 times. In contrast, the character code *a1* in hexadecimal (i with acute accent, in ASCII decimal: 10x16+1=161) , only 10 times.

# MXF0 - Syntax

```
mxf0 <dbname> [create=]<dbnout> [<tnrecs>] [noedit] [tell=<n>]
```

The parameters should be given in the order indicated, for example:

```
mxf0 cds create=x noedit 0
```

gives an error. The correct form is:

```
mxf0 cds create=x 0 noedit
```

## Mandatory parameters

**<dbname> <dbnout>**

### Name of the input master file

**<dbname>**

Master file on which the analysis is to be performed

### Name of the outout master file

**<dbnout>**

Output master file, the master file which contains the table generated. If a master file with this name already exists, the records are added.

### Create output master file

**create=<dbnout>**

The output master file is created. If a master file with this name already exists, all its information is lost.

### Approximate number of records

**<tnrecs>**

Number of records in the input master file. In case of uncertainty, if specified, it should be set to zero. When the parameter is non-zero, the program finishes

execution with the return code equal to zero only if the number of records read is equal to the value of the parameter.

## Optional parameters

**[noedit] [tell=<n>]**

### Elimination of blank spaces

**noedit**

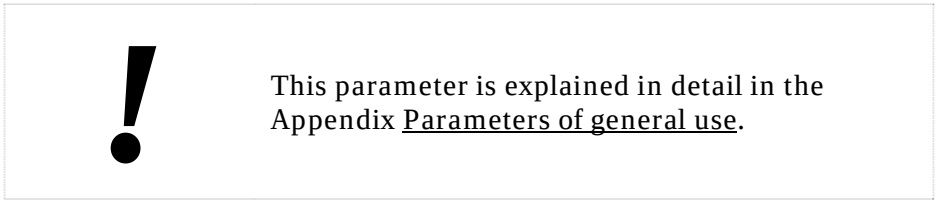
When present, it supresses all the spaces to the right in the fields of output data.

When this parameter is not present (default value), the program uses fixed lengths for the fields, padded with spaces to the right.

### Information about the execution of the process

**tell=<n>**

Sends a message to the *stderr* (standard error) each <*n*> records.



This parameter is explained in detail in the Appendix Parameters of general use.

# MXF0 - output

The output record of MXFO has the following fields:

| TAG | CONTENTS |
|---|---|
| 1001 | Name of the input master file |
| 1003 | Date and time |
| 1009 | Number of records processed |
| 1010 | Number of active records |
| 1011 | Number of records logically deleted |
| 1012 | Number of records physically deleted |
| 1013 | Next MFN to be assigned |

| | |
|---|---|
| 1020 | ^tTAG ^dDOCS ^oOCCS ^lMINLEN ^uMAXLEN ^nDATA BYTES |
| 1030 | ^tall ^xCHRCODE ^nCHRFREQ |

The repeatable field 1020 has an ocurrence for each different field tag present in the input records:

| SUBFIELD | CONTENTS |
|---|---|
| TAG | Field number |
| DOCS | Number of records which contain TAG |
| OCCS | Total of ocurrences |
| MINLEN | Shortest length |
| MAXLEN | Greatest length |
| DATA BYTES | Total of characters |

The repeatable field 1030 has an ocurrence for each different character contained in the totality of fields of input data:

| SUBFIELD | CONTENTS |
|---|---|
| CHRCODE | Code of the character, in hexadecimal format (x00-xFF) |
| CHRFREQ | Frequency of the code |

# MXTB - Program

The program MXTB allows counting of the contents of the fields, for example, the number of times that each author appears, the number of that each descriptor appears, the number of times that an author and journal appear together, etc.

The result of the execution of MXTB is a master file with a record for each different word or phrase found. Each record contains, amongst other data, a string of characters, frequency, etc.

MXTB is a program that tabulates in multiple columns the records of the master file, which defines the tabulation keys by means of the formatting language.

After the records are read, the format provided is executed and the resulting data take values according to the corresponding tabulation keys.

All the possible n-tuples are tabulated and, finally, are output as individual records of the master file together with their frequencies.

The process of tabulation is carried out in memory, using a *hash technique*. This requires an amount of memory sufficient to load all the n-tuples generated, their frequencies, plus additional space. The more space, the quicker the execution. (This is seen more clearly when the parameter *class* is explained later in this chapter).

This utility program can be used to produce combined tables of some data stored in the master file. For example: a list of authors containing their frequencies in a bibliographic database, or a list of these divided by year of publication, or other data.

Optionally, MXTB takes an additional format specification to obtain a value for the process of tabulation. In this case, the numeric output information is the sum of these values for a particular n-tuple.

# MXTB - Presentation

```
mxtb \cisis\cds\cds create=freqaut 40:(v70/)
```

This command line writes a record in the master file *freqaut* for each different occurrence of the field 70 in *\cisis\cds\cds*, as indicated below:

```
MFN 1
1 «Magalhaes, A.C.»
998 «999999997»
999 «2»
MFN 2
1 «Franco, C.M.»
998 «999999998»
999 «1»
...
```

Fields of the records of *freqaut*:

| TAG | CONTENTS |
|---|---|
| 1 | String of characters to count, in the example, an author generated by the format (v70/).<br>(category) |
| 998 | Maimum possible ocurrences of a category minus the frequency found.<br>(999999999 - frequency) |
| 999 | Number of times that the contents of field 1 of freqaut occur in the input file \cisis\cds\cds.<br>(frequency) |

The field 998 provides a simple way to produce lists in descending order of the categories found (for example, with the command MSRT freqaut 10 v998).

MXTB can also take an external value to be tabulated, which means that instead of adding the value for each ocurrence found it can add a value entered by means of a format. This may not only be a fixed value but also a field of the same or another database, where the contents of field 999 will be the sum of the values found in each case.

## Example of the use of MXTB

If the cost of the documents, date of acquisition, and area to which they belong have been entered in the database, you can produce a table which contains – for example – the money spent on documents in the last year for each area.

Supposing that among the data entered in the records are the cost of the document and the publisher, with MXTB you can easily construct a table that contains the publisher and the total you have paid them.

```
mxtb holdings x 40:(v26^b/) tab=v90
```

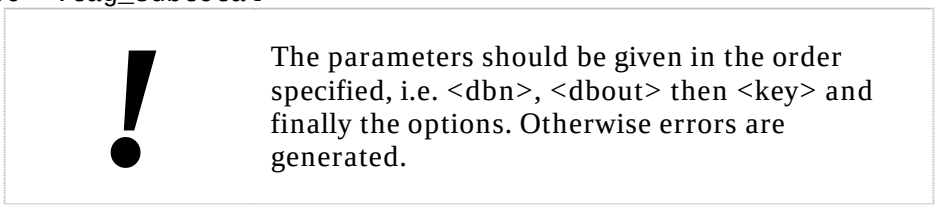
For the first 40 characters of each distinct publisher stored in subfield ^*b* of field *26* in the holdings database, write a record in the *x* database, as shown below:

```
MFN 1
1 «Unesco Press»
998 «999993999»
999 «6000»
MFN 2
1 «ESCAP»
998 «999999969»
999 «30»
MFN 3
1 «Praeger Publishers»
998 «999999599»
999 «400»
...
```

The example puts into field 999 the sum of the contents of field v90 of each category, then – instead of adding 1 for each element found – sums what is indicated in field 90. To use the parameter *tab*, field 90 of all the records should contain a numeric value: the records whose fields 90 do not contain a numeric value are not taken into account.

# MXTB - Syntax

```
mxtb <dbn> [create=]<dbnout> <key> [<key> [...]] [<option> [...]]
keys:    keylen:key_fmtspec

options: {from|to|loop|count|tell|btell}=<n>
         tab=<tab_val_fmt>
         class=1000
         bool=<expr>
         {min|max}{avg|freq}=<n> -
          uctab={<file>|ansi}

Ex: mxtb in out len1:fmt1 len2:fmt2 len3:fmt3

     out = 1  key/key1_value (max len1 chars)
           2  key/key2_value (max len2 chars)
           3  key/key3_value (max len3 chars)
         998  999999999 - key_frequency
         999  key_frequency

Ex: mxtb in out len:fmt tab=Vtag

     out = 1  key_value (max len chars)
         998  999999999 - Vtag_subtotal
         999  Vtag_subtotal
```




## Mandatory parameters

**<dbn> <dbnout> <key>**

### Name of the input master file

**<dbn>**

Name of the master file from which MXTB obtains the data to generate the table.

### Name of the output master file

**<dbnout>**

The output master file is the master file which contains the table generated. If there already exists a master file with this name the records are added.

### Create output master file

**create=<dbnout>**

The output master file is created. If a master file with this name exists, all its information is lost.

### Key

**<key>**

The parameter <key> applies to each of up to 8 tabulation keys, and has the following form:
`<length of the key>:<format which specifies the key>`

### Maximum length of the key

**<keylen>**

<keylen> is the maximum number of characters returned by the format specification <key_fmtspec>. This means that if the string of characters generated by <key_fmtspec> is more than <keylen>, only the first <keylen> characters are taken into account.

### Format which specifies the key

**Parameter <key_fmtspec>**

is the format which generates the tabulation keys. For example, to tabulate by editor the format should generate the editors, one per line.

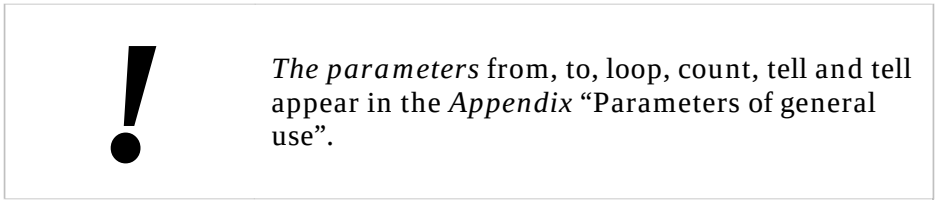
## Optional parameters

**bool=<exp>**

**{from|to|loop|count|tell|btell}=<n>**

**tab=<tab_val_fmt>**

**class=<n>**



*The parameters* from, to, loop, count, tell and tell appear in the *Appendix* “Parameters of general use”.

## Processing the results of a search

**bool=<exp>**

You can tabulate the results of a search, indicating the parameter *bool* and the boolean expression. This permits, for example, to count all books with a certain property: all the journals from the publisher X or all the books from an area and the total cost, etc.

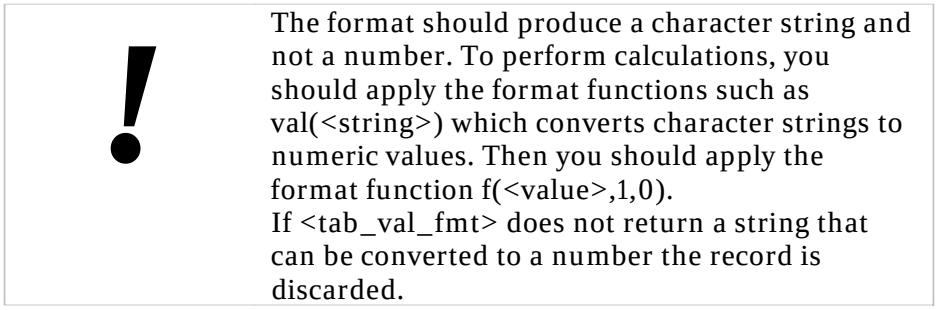
## Tabulation of the format results

**tab=<tab_val_fmt>**

The default value for tabulation (if you do not specify *tab*) is 1. This means that MXTB adds the value 1 to a category for each ocurrence found.

*tab* allows changing the default value according to a format, which can specify a constant value (e.g. '5'), or can specify a field from which to take the value (for example the price field of a database of books).

*tab* allows you to produce tables which contain, for example, money thay the library invests in books and journals for each area.



The format should produce a character string and not a number. To perform calculations, you should apply the format functions such as val(<string>) which converts character strings to numeric values. Then you should apply the format function f(<value>,1,0).
If <tab_val_fmt> does not return a string that can be converted to a number the record is discarded.

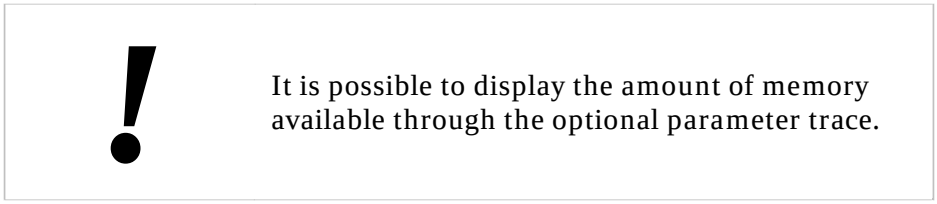
### Number of categories

**class=<n>**

asisgns space for up to <n> categories to be tabulated.

The default value for this parameter is 1000.

To allow for the effects of *hashing*, you are recommended to specificy a parameter *class* with a value equal to between 2 and 3 times the expected number of categories.




## MXTB - Output

The output records of MXTB comprize the following fields:

| MFN n | |
|---|---|
| **TAG** | **CONTENTS** |
| 1 | Value produced by <key_fmtspec1>, up to <len1> [characteres] |
| 2 | Value produced by <key_fmtspec2>, up to <len2> [characteres] |
| 3 | Value produced by <key_fmtspec3>, up to <len3> [characteres] |
| 998 | Value 999999999 minus the value of the field 999 |
| 999 | Frequency of the category formated by the fields 1, 2, 3, ... |

When you use the option *tab=<tab_val_fmt>* the field 999 contains the sum of the values produced by the format specification *<tab_val_fmt>* for each record belonging to that category.

The records that do not generate output from the primary specification <key> are discarded and therefore not included in the tabulation.

## MXCP - Presentation

The command:

```
mxcp \cisis\cds\cds create=x clean undelete
```

reinitializes the master file ***x*** (in the working directory) and copies all the records of the master file *cds* located in the directory \cisis\cds.

No data field in the master file ***x*** has blank spaces at the beginning or end of the field due to specifying the option *clean*. Furthermore, each non-pritable character which appears in the input fields will be converted to a space character in the output.

Due to specifying the option *undelete*, all the logically deleted records in the master file \cisis\cds\cds will be active in the master file *x*.

MXCP can convert into repeatable fields those which are not, in all cases or when they have a delimiter character in the data.

In the following example MXCP converts field with tag 70 containing the sign "**;**" as the delimiter to repeated occurrences.

```
mxcp in create=out repeat=;,70
```

For example, let us suppose that records 1 and 2 of the database *in* have in field 70:

```
MFN 1
70 «Magalhaes, A.C.; Franco, C.M.»
MFN 2
70 «Bosian, G.»
```

Then in *out* (the resulting master file) the records will be of this form:

```
MFN 1
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
MFN 2
70 «Bosian G.»
```

When you use the parameter *repeat*, MXCP also performs the process of "cleaning".

If you do not specify a tag in the parameter *repeat*, all the fields present will be included:

```
mxcp in create=out repeat=%
```

You can also specificy a list of fields:

```
mxcp in create=out repeat=%,70,99,100,200
```

a range:

```
mxcp in create=out repeat=%,99/105
```

or a mix of both:

```
mxcp in create=out repeat=%,70,99/105,110
```

Another feature of MXCP is to suppress the final punctuation signs. The following example removes the final full-stop in field 70:

```
mxcp in create=out period=.,70
```

For example, suppose that records 1 and 2 of the database *in* have in field 70:

```
MFN 1
70 «Magalhaes, A.C.»
70 «Franco, C.M.»
MFN 2
«Bosian G.»
«xxx.»
```

Then in *out* (the resulting master file) the records will be:

```
MFN 1
70 «Magalhaes, A.C»
70 «Franco, C.M»
MFN 2
«Bosian G»
«xxx.»
```

MXCP also offers the facility of a change of patterns, allowing the contents of the input fields specified to be changed at the same time that they are read.

Typing exactly what is shown below (ending each line with <enter>) creates a conversion table to replace the angle brackets by the percent sign (%):

```
mx seq=con create=xtable -all now
<|
>|
><|%
> <|%
> <|%
Paper on: <|
<Ctrl>Z
```

Then if the fields with tag 69 of the database *cds* contain:

```
MFN 1
69 «Paper on: <plants><water><plant transpiration>»
MFN 2
69 «Paper on: <plant physiology><plant transpiration><measurement and
instruments»
```

the command:

```
mxcp cds create=out gizmo=xtable repeat=%,69
```

will cause the records with tag 69 of *cds* to be transformed in the following manner in *out*:

```
MFN 1
69 «plants»
69 «water»
69 «plant transpiration»
MFN 2
69 «plant physiology»
69 «plant transpiration»
69 «measurement and instruments»
```

# MXCP - Syntax

```
mxcp {in=<file>|<dbin>} [create=]<dbout> [<option> [...]]
options: {from|to|loop|count|tell|offset}=<n>
         gizmo=<dbgiz>[,tag_list>]
         undelete
         clean [mintag=1] [maxtag=9999]
         period=.[,<tag_list>]
         repeat=%[,<tag_list>]
         log=<filename>

 Ex: mxcp in create=out clean period=.,3 repeat=;,7

      in = 3 «   Field 3 occ 1. »
           3 «Field 3 occ 2 . »
           7 « Field 7/1;Field 7/2 ;Field 7/3.»
     out = 3 «Field 3 occ 1»
           3 «Field 3 occ 2»
           7 «Field 7/1»
           7 «Field 7/2»
           7 «Field 7/3.»
```

The parameters present must be in the order stipulated, firstly <dbin> then <dbout> and lastly the options, or else an error will be generated.

## Name of the input master file

**<dbin>**

Master file of input.

## Name of the output master file

**<dbout>**

The output master file is where the records coming from *dbin* are copied.

If *dbout* does not exist an error is produced; to create it you should use the parameter *create*, which is explained below.

### Create output master file

**create=<dbnout>**

When the output master file does not exist it can be created through this parameter.

If *dbout* exists you will lose all its data, since the output database is reinitialized before copying the records.

## Optional parameters [option]

**{from|to|loop|count|tell|offset}=<n>**

**undelete**

**gizmo=<dbgiz>[,tag_list>]**

**period=.[,<tag_list>]**

**repeat=%[,<tag_list>]**

**clean [mintag=1] [maxtag=9999]**

**log=<filename>**

*The parameters* from, to, loop, count, tell *and* offset can be seen in the *Appendix* "Parameters of general use."

### Recovery of deleted records

**undelete**

recovers records of the master file. (It activates the records of the input file which are logically deleted.)

```
mxcp cds newcds undelete
```

## Global change of patterns

**gizmo=<dbgiz>[,<tag_list>]**

applies a gizmo procedure to the input data, using the master file *dbgiz* as the gizmo table. It can apply to all the fields of a record or only to the fields specified in <tag_list>.

The list <tag_list> can be specified in the following manner:

- A tag
- A list of tags separated by commas
- A range of tags in the form *lower value / higher value*
- A combination of these options

If you specify more than one gizmo=<dbgiz>[,<tag_list>], the second is executed on the record that results from the processing of the first gizmo, and so on.

```
mxcp cds newcds gizmo=xtable
mxcp cds newcds gizmo=xtable,24,50/70
```

The parameter gizmo is explained more fully in the Appendix “Parameters of general use”.

Suppresion of punctuation characters

```
period=<char>[,<tag_list>]
```

MXCP provides the facility of suppressing the character <*char*> as the final punctuation sign. It can be applied to all the fields of a record or only to the fields present in <tag_list>.

The list <tag_list> can be specified in the following manners:

- A tag
- A list of tags separated by commas
- A range of tags in the form *lower value /higher value*
- A combination of these options

```
mxcp cds newcds period=;
mxcp cds newcds period=;,24,50/60,70
```

## Converting repeatable fields

**repeat=<char>[,<tag_list>]**

Converts the input fields which have <*char*> as delimiter to repeatable fields. It can be applied to all the fields of a record or only to the fields present in <tag_list>.

The list <tag_list> can be specified in the following manners:

- A tag
- A list of tags separated by commas
- A range of tags in the form *lower value* / *higher value*
- A combination of these options

```
mxcp cds newcds repeat=/
mxcp cds newcds repeat=;,24,50/60,70
```

## Suppression of blank spaces

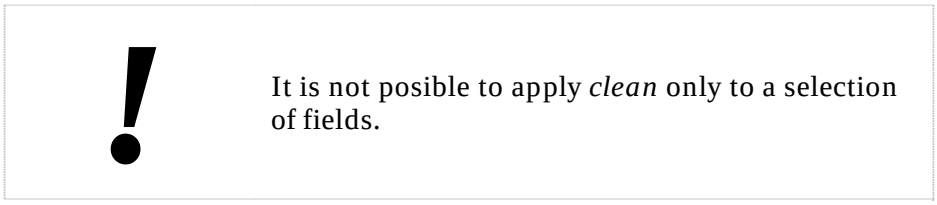
**clean**

suppresses all the blank spaces at the beginning and end of the field and replaces all the non-printing characters with blank spaces.

If you use *repeat* or *period* the option *clean* is initiated automatically.

The option *clean* is applied to all the fields of the record.

```
mxcp cds newcds clean
```



It is not posible to apply *clean* only to a selection of fields.

## Elimination of fields by tag

**mintag=<n>**

deletes all the ocurrences of the fields with tag less than <n>. By default n=1

```
mxcp cds newcds mintag=10
 maxtag=<n>
```

deletes all the ocurrences of the fields with tag greater than <n>. By default n=9999

```
mxcp cds newcds maxtag=70
```

### Record of events

**log=<filename>**

directs the control and status messages to the file <*filename*>.

## MXCP - Output

MXCP does not produce outputs from empty records. The input or processed records that do not have fields are discarded. The logically deleted records are processed only if you use the option *undelete*.

You can use the same master file for the input as for the output, although it is only possible to reorganize a master file copied to a new one. The option *create*=<*dbout*> ensures that <*dbout*> will be created or reinitialized.

The command:

```
mxcp in create=out period=.,3 repeat=;,7
```

for the master file *in:*

```
MFN 1
3 «Field 3 occ 1. »
3 «Field 3 occ 2 .»
7 «Field 7/1; Field 7/2; Field 7/3.»
MFN 2
...
```

creates *out* with:

```
MFN 1
3 «Field 3 occ 1»
3 «Field 3 occ 2»
7 «Field 7/1»
7 «Field 7/2»
7 «Field 7/3»
MFN 2
...
```

and produces the following messages:

```
*** mfn 1 tag=3/1 -> rejected char
```

```
*** mfn 1 tag=3/1 -> rejected char
*** mfn 1 tag=3/1 . -> rejected char
*** mfn 1 tag=3/2 -> rejected char
*** mfn 1 tag=3/2 . -> rejected char
*** mfn 1 tag=3/2 -> rejected char
*** mfn 1 tag=7/1 -> rejected char
```

# MSRT - Program

The program MSRT sorts a master file, in ascending order, following the sort keys generated by a given format specification.

After a master file is sorted, the record with mfn=1 contains the lowest sort key, the record with mfn=2 the second lowest sort key, and so on.

# MSRT - Presentation

The command:

```
msrt \cisis\cds\cds 60 mhu,v24
```

sorts the master file *cds* located in the directory \cisis\cds according to the first *60* characters of field *24* (converted appropriately to upper-case).

The command:

```
msrt mxtb_out 9 v998
```

sorts the master file *mxtb_out* according to the values stored in field 998 (presumed to be of fixed length and with zeros to the left).

Whereas the command:

```
msrt mxtb_out 9 f(999999999-val(v999),9,0)
```

sorts the master file *mxtb_out* in descending order, according to the numeric value stored in the field 999.

# MSRT - Syntax

```
msrt <dbname> <keylen> <keyfmt> [-mfn] [tell=<n>]
```

The parameters present must be in the order stipulated, i.e. <dbname>, <keylen>, then <keyfmt> and finally the options, otherwise an error is produced.

## Mandatory parameters:

**<dbname> <keylen> <keyfmt>**

### Name of the input master file

**<dbname>**

Master file to be sorted.

### Maximum key length

**<keylen>**

Maximum number of characters to be compared.

### Generation key

**<keyfmt> | tag=<n>**

Format which is applied to the records to generate the sort keys.

To specify a data field as a sort key, without the necessity of executing a format, the parameter <keyfmt> should be **tag=<n>**.

## Optional parameters

**-mfn**

**+of the**

**tell=<n>**

The parameter tell is explained in detail in the Appendix “Parameters of general use”.

### Mantaining original MFNs

**-mfn**

Saves the original *MFN*. By default, the records are renumbered after sorting.

### Deleting identical keys

**+del**

Deletes identical keys of consecutive records after sorting.

## MSRT - Output

When a master file is sorted, the program MSRT updates only the .xrf, changing the corresponding entries of each pair of records which are to be swapped.

At the end of this process, if the option –mfn is used, it updates the .mst file, assigning the record numbers to the mfn field of the record leader.

This update is carried out with the records remaining in the same place, which means that the records are modified in their own positions, therefore, the organization of the master file processed is not changed. In this way the time is saved that would be needed to rewrite all the sorted records in the master file.

## RETAG - Program

The program *RETAG* changes the tag of the fields of the master file, according to a renumbering table. Also it is possible to perform an *unlock* of the master file.

The operations that perform RETAG are carried out on the same record, which means that the records are modified in their original positions and, therefore, the organization of the master file processed is not changed.

# RETAG - Presentation

The command:

```
retag \cisis\cds\cds xtable
```

processes all the active records of the master file *cds* located in the directory \cisis\cds, rewriting the segment of the directory of the records corresponding to a tag-renumbering table *xtable* located in the working directory.

A renumbering table is a sequential file which has one line for each tag to be renumbered, as shown below:

```
24 240
70 700
69 690
```

The previous command changes the tag 24 to 240, the tag 70 to 700, and the tag 69 to 690.

In this example, only the records which contain at least one ocurrence of the tag 24, 70 or 69 are rewritten.

In addition, the program *RETAG* can be used to undo the effect of blocks of the master file.

The command:

```
retag \cisis\cds\cds unlock
```

instead of specifying a renumbering table of tags, specifies the keyword *unlock,* which removes the *exclusive write lock* or *data entry locks* and all the *record locks* existing in the master file *cds* located in the directory \cisis\cds.

# RETAG - Syntax

```
retag <dbname> {<retag.tab>|unlock} [<option> [...]]
```

## Mandatory parameters

**<dbname> <retag.tab>**

### Input master file

**<dbname>**

Master file to be retagged or *unlocked*.

### Renumbering table

**<retag.tab>**

Name of a sequential file which provides a retagging table, or the keyword *unlock*.

If <retag.tab> is *unlock*, this indicates to the program that it should perform an unlocking of the master file.

A retagging table has the format:

```
<tag> <new tag>
```

It may contain up to 5,461 entries in the standard version.

## Optional parameters

**from=<n>**

**to=<n>**

**tell=<n>**

**shift=<n>**

The parameters *from, to, shift* and *tell* can be seen in the Appendix “Parameters of general use”.

# RETAG - Output

The RETAG operations are performed on the same physical record (*in-place*), whether the request is a retagging or an unlocking.

In the retagging operation, only the directory segment of the records processed is rewritten.

In the unlocking operation, only the control record of the master file is rewritten and the *leader* segment of the records processed.

# CTLMFN - Program

The program CTLMFN opens and updates the control record of a master file.

You should use it before executing the program MKXRF to restore all the active records in a master file logically reinitialized, to establish the next number to be assigned in the master file and to define the number of data to be analysed.

# CTLMFN - Presentation

The command:

```
ctlmfn \cisis\bases\cds
```

reads the control record of the master file \cisis\bases\cds.mst and allows each of the fields of this record to be updated. These fields are:

| FIELD | CONTENTS |
|---|---|
| nxtmfn | MFN to be assigned to the next record created in the database |
| nxtmfb | Last block assigned in the master file (the first block is 1) |
| nxtmfp | Following position available in the last block of the master file |
| mftype | 0 for user database, 1 for system message files |
| reccnt | Reserved |
| mfcxx1 | Reserved |
| mfcxx2 | Number of users in the process of inputting data (*Data Entry Lock*). |
| mfcxx3 | *Exclusive Write Lock*. |

If you write an invalid value for *nxtmfb* you are informed of the total number of blocks of the master file.

The commands

```
ctlmfn \cisis\bases\cds
```

and

```
mkxrf \cisis\bases\cds
```

permit changing one or more control fields in \cisis\bases\cds.mst (data records) and then to create, reinitialize and write the cross-reference file \cisis\bases\cds.xrf.

# CTLMFN - Syntax

**ctlmfn <dbname> [ok]**

## Name of the input master file

**<dbname>**

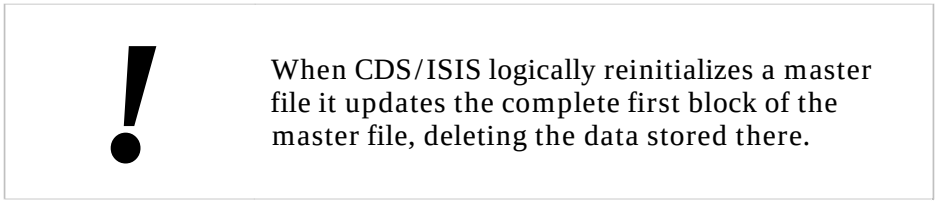
Mandatory parameter which indicates the master file to be processed.

## Confirmation prompt

**ok**

# CTLMFN - Output

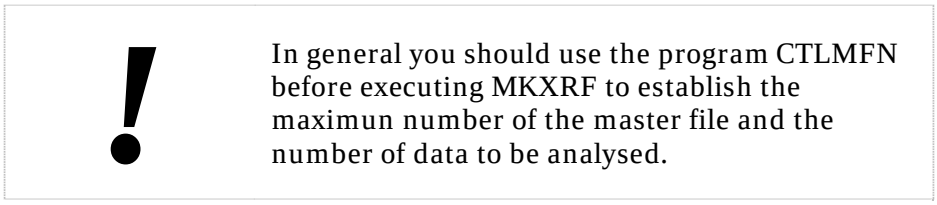
The program CTLMFN writes only the first bytes (the bytes of the control record) of a .mst file. The remaining bytes of the first block of the master file are not changed.




## MKXRF - Program

The program MKXRF reads a .mst file and creates its corresponding .xrf file.

Thus it can be used to restore all the active records in a master file logically reinitialized.




## MKXRF - Presentation

The command:

```
mkxrf \cisis\bases\cds
```

creates, reinitializes and writes the cross-reference file \cisis\bases\cds.xrf corresponding to the master file \cisis\bases\cds.mst.

The program MKXRF can restore a master file reinitialized by accident, provided that its control record has the following control fields adjusted to the appropriate form:

| FIELD | CONTENTS |
|---|---|
| nxtmfn | MFN to be assigned to the next record created in the database |
| nxtmfb | Last block assigned in the master file (the first block is 1) |
| nxtmfp | Next position available in the last block of the master file |

These values can be established using the program CTLMFN, as is shown in the following commands:

```
ctlmfn \cisis\bases\cds
mkxrf \cisis\bases\cds
```

These commands allow changing one or more control fields in the file \cisis\bases\cds.mst (data records) to create, reinitialize, and write the cross-reference file \cisis\bases\cds.xrf.

Description of the execution of CTLMFN in the example:

a) If *nxtmfn* is unknown, you should establish the maximum mfn *possible* (see the explanation below); the real *nxtmfn* will be obtainable later by executing the program *MXF0* (field 1013)
b) *nxtmfb* is the number of blocks of the master file \cisis\bases\cds.mst which is to be analysed; if an invalid value is written then you are informed of the total number of blocks.
c) *nxtmfp* is the last position in the block *nxtmfb* of the master file which is to be analysed; a value of 512 forces *MKXRF* to process all the contents of the block nxtmfb of the master file.

# MKXRF - Syntax

```
mkxrf <dbname>
```

## Input master file

**<dbname>**

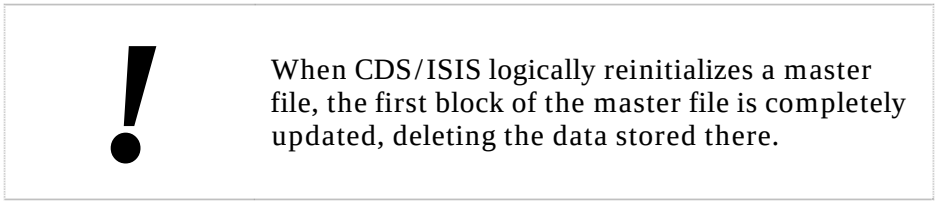
Name of the master file to be processed (mandatory parameter).

## MKXRF - Output

The program MKXRF reads the control record and then examines the input file .mst up to the place *nxtmfb* and *nxtmfp*. In this process it identifies a master file record if the following conditions apply to the components of the *leader*:

a) MFRmfn is in the range from 1 to (nxtmfn-1).
b) MFRmfrl is in the range LEADER to MAXMFRL.
c) MFRbase is up to MFRmfrl.
d) MFRmfbwb points to some previous valid direction.
e) MFRmfbwp is in the range from 0 to MSBSIZ.
f) MFRnvf is zero or positive.
g) MFRstatus is ACTIVE or DELETED.
h) MFRbase is equal to LEADER + MFRnvf*sizeof(DIRSTRU).

The location of each record of the master file identified is written in the .xrf file which is created, and includes the versions of old records.



When CDS/ISIS logically reinitializes a master file, the first block of the master file is completely updated, deleting the data stored there.

## Restoring a damaged database

Below are detailed procedures for restoring a database according to various problem states:

| State | | | Action |
|---|---|---|---|
| **MST** | **XRF** | **Information** | |
| OK | Damaged | MaxMFN unknown | Use MKXRF |
| Reinitialized | OK | Size of the XRF | Estimate maxMFN and use CTLMFN |
| Locked | OK | | Use CTLMFN, write zeros in the fields mfcxx2, mfcxx3 |

## Approximate calculation of the MaxMFN

An estimate of the maximum number of records (*maxMFN*) is obtained from:

```
(size_in_bytes_of_XRF/512) * 127
```

If the value of *maxMFN* which is established is more than the real number of records, no records are generated either in the .mst nor in the .xrf. The .xrf file is written only for the records which can be read in the .mst. The database will be able to be read without problems but it cannot be updated (it generates the error *fatal: recxref/read*). The *maxMFN* should be corrected following one of these two forms:

Take the value "x recs" which is indicated at the end of the execution of *MKXRF* and later store x + 1 in the *nxtmfn* through *CTLMFN*.

Create another master file (with *create*, not with *copy*) starting from the one you have, with the effect of correcting the control record in the new master file.

# ID2I - Program

The program I2ID receives an ASCII file, with determined structure and generates a master file.

# ID2I - Presentation

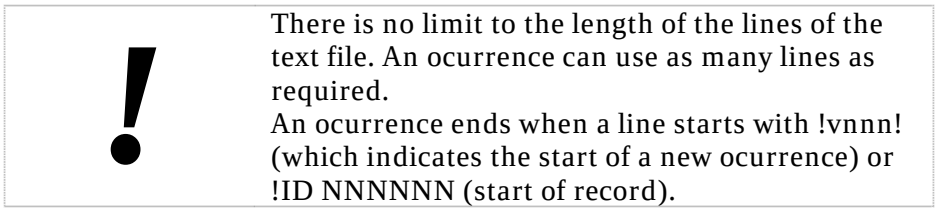
The program receives an ASCII file and returns a master file.

## Estructura of the file ASCII:

- !ID nnnnnn Mark of start of record with mfn=nnnnnn
- !vnnn Mark of start of an ocurrence of the field with tag nnn.

The file has the form:

```
!ID nnnnnn
!vXXX!...contents of the tag XXX.............
!vYYY!...contents of the tag YYY.............
...
!ID nnnnnj
!vXXQ!...contents of the tag XXQ.............
!vYYQ!...contents of the tag YYQ.............
...
```



There is no limit to the length of the lines of the text file. An ocurrence can use as many lines as required.
An ocurrence ends when a line starts with !vnnn! (which indicates the start of a new ocurrence) or !ID NNNNNN (start of record).

**Example**:

```
id2i x.txt create=newcds
```

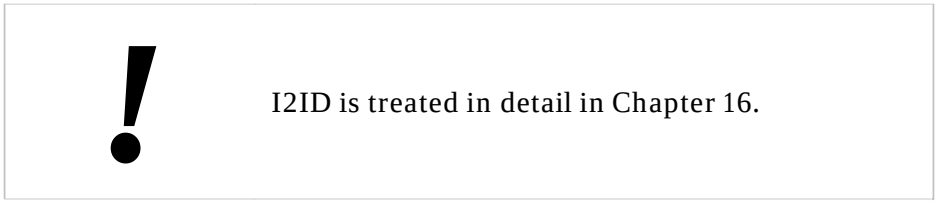
The idea is to use *ID2I* jointly with *I2ID*, which takes a master file and returns a text file (editable and modificable).

By means of *ID2I* you can convert the modified text file into a master file:

```
I2ID cds >x.txt
```

Edit x.txt (Edit the contents permitting creating, modifying and deleting records and ocurrences)

```
ID2I x.txt create=cds
```



I2ID is treated in detail in Chapter 16.

# ID2I - Syntax

```
id2i <filein> [create[/app]=]<dbout> [option [option] ... ]
options: {from|to|loop|count|offset|tell|id}=<n>
```

## Mandatory parameters

**<filein> <dbout>**

### File ASCII of input

**<filein>**

is an ASCII file with the structure seen in the Presentation.

### Name of the master file of output

**<dbout>**

Output master file of output, where the records coming from *filein* are copied.

If *dbout* does not exist an error is produced: to create it you should use *create*. If *dbout* exists, you will lose all its data, since it is reinitialized before copying the records.

### Create output master file

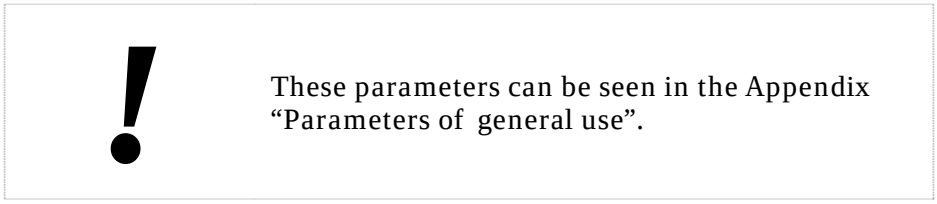
**create/app=<dbnout>**

Creates the master file of output; if *dbout* exists, you will lose all its data, since it is reinitialized before copying the records. The option **/app** signifies that the numbering is done sequentially.

Moreover it does not consider the MFN information in the start markers of the records (!ID nnnnnn) and it accepts markers of !ID 000000.

### Optional parameters

**{from|to|loop|count|tell|offset|id}=<n>**



These parameters can be seen in the Appendix "Parameters of general use".

## I2ID - Program

The program I2ID receives a master file and generates a text file.

## I2ID - Presentation

The program receives a master file and returns a text file with the following structure:

```
!ID nnnnnn           Mark of start of record with mfn=nnnnnn
!vnnn                Mark of start of an ocurrence of the field with tag nnn.
```

Text file returned by *I2ID*:

```
!ID nnnnnn
!vXXX!...contents of the tag XXX.............
!vYYY!...contents of the tag YYY.............
...
!ID nnnnnj
!vXXQ!...contents of the tag XXQ.............
!vYYQ!...contents of the tag YYQ.............
...
```

There is no limit to the length of the lines of the text file. An ocurrence can use as many lines as required.

An ocurrence ends when a line starts with !vnnn! (which indicates the start of a new ocurrence) or !ID NNNNN (start of a record).

**Example:**

```
I2ID cds >x.txt
```

The idea is to use *I2ID* in conjunction with *ID2I*. The first returns a text file (which can be edited and modified) and then, by means of *ID2I*, returned to convert the text file into a master file.

```
I2ID cds >x.txt
```

edit x.txt (Edit contents – you can create, modify and delete records and ocurrences)

```
ID2I x.txt create=cds
```

ID2D is treated in detail in Chapter 15.

# I2ID - Syntax

```
i2id <dbn> [option [option] ... ]
options: {from|to|loop|count|offset|tell}=<n>
          1record=<tag>=<value>
```

## Mandatory parameters:

**<dbn>**

Input master file

**<dbn>**

Master file to be converted.

## Optional parameters

**{from|to|loop|count|offset|tell}=<n>**

These parameters can be seen in the Appendix “Parameters of general use”.

# CRUNCHMF - Syntax

```
crunchmf <dbn> <target_dbn> [<option> [...]]
```

converts the master file (.mst and .xrf) from one operating system to another, since the structure in binary is not compatible between the diferent systems. The program detects automatically the system in which it is operating.

options:

```
 {from|to|loop|count|tell}=<n>
 target={pc|linux|hpux|sun|alpha|vax|unisys|mpe|cdc|same} default: linux
 format={isis|cisisX}                                     default: isis
 mstxl={0|1|2|4}                                          default: as <dbn>
```

# Inverted file utilities

## IFKEYS - Program

The program IFKEYS displays the terms of the inverted file and the corresponding totals of *postings*, optionally broken down by the tags from which they are extracted.

IFKEYS accepts a range of terms as parameter and can be used to display a set of the terms from the inverted file.

## IFKEYS - Presentation

The command:

```
ifkeys \cisis\bases\cds from=plant to=plants
```

reads the inverted file *cds* located in the directory \cisis\bases beginning with the term *plant* up to the term *plants* and desplays those terms preceded by their numbers of *postings*:

```
8|PLANT
4|PLANT ECOLOGY
1|PLANT EVAPOTRANSPIRATION
1|PLANT PHOTOSYNTHESIS
20|PLANT PHYSIOLOGY
6|PLANT TRANSPIRATION
```

```
8|PLANTS
```

Meanwhile the command:

```
ifkeys \cisis\bases\cds from=plant to=plants +tags
```

produces the same information and, additionally, (a) includes the tag (*Field ID*) of the field from which these terms were generated and (b) generates a different line for each combination of term and tag, as shown below:

```
8| 24|PLANT
4| 69|PLANT ECOLOGY
1| 69|PLANT EVAPOTRANSPIRATION
1| 69|PLANT PHOTOSYNTHESIS
20| 69|PLANT PHYSIOLOGY
6| 69|PLANT TRANSPIRATION
6| 24|PLANTS
2| 69|PLANTS
```

The parameters *from* and *to* do not distinguish upper-case from lower-case: from=plant produces the same output as from=PLANT.

# IFKEYS - Syntax

```
ifkeys <dbname> [from=<key>] [to=<key>] [+tags] [tell=<n>]
```

## Input inverted file

**<dbname>**

Inverted file to be processed.

## Optional parameters

**from=<term>**

**to=<term>**

**+tags**

**tell=<n>**

The parameter tell is explained in detail in the Appendix “Parameters of general use”.

### First term to be listed

**from=<term>**

starts the list from the term <term>.

### Last term to be listed

**to=<term>**

ends the list with the term <term>.

### Show information about tags

**+tags**

adds to the list the tag of the field from which the term was extracted.

## IFKEYS - Output

If you use the option *from=<term>* and the starting term specified does not exist, the list commences with the following term of the inverted file. If you employ the option *to=<term>* and the final term final specified does not exist, the list stops with the previous term of the inverted file.

The output of IFKEYS can be redirected to a file for later processing by the program MX. For example, the commands:

```
ifkeys cds +tags >x
mx seq=x "pft=if val(v1)=1 and val(v2)=24 then v3/ fi" now
```

display the words of the title (field 24 in the database *cds*) which occur exactly once.

# IFLOAD - Program

The program IFLOAD loads an inverted file using files of links ordered according to the processing options. Other formats are accepted, as well as the standard link file format of CDS/ISIS.

It also allows creating only the inverted file of the dictionary.

The program IFLOAD can also be used to create and reinitialize an inverted file.

# IFLOAD - Presentation

The command:

```
ifload \cisis\bases\cds \isis\work\cds.lk1 \isis\work\cds.lk2 tell=99
```

loads the inverted file *cds* located in the directory \cisis\bases using the link files of short keys and long keys cds.lk1 and cds.lk2, located in the directory \isis\work.

The option *tell=99* produces a message about progress each 99 records of links processed, showing the current key which is being loaded.

Suppose that the original link files are in the standard format of links of CDS/ISIS, which have the following format:

```
102 24 1 1 ABOUT
42 24 1 9 ABSENCE
6 24 1 10 ABSORPTION
87 24 1 5 ACCOUNT
136 69 1 1 ACCOUNTING
40 24 1 6 ACID
101 24 1 5 ACTION
49 24 1 6 ACTIVITIES
130 24 1 7 ACTIVITIES
23 24 1 5 ACTUAL
```

The format is records of variable length, with fields that identify the origin of the key and the key itself.

The first 4 fields are the components of the *posting*.

| Field | Contents |
|---|---|
| MFN | Number of record (master file record number) |
| TAG | Identifier of field, assigned by the FST (field identifier) |

| Field | Contents |
|---|---|
| OCC | Number of ocurrence of the field |
| CNT | Sequential number of the term in the field |

To allow the use of standard sorting programs, IFLOAD accepts link files with records of fixed length, as shown below:

```
ABOUT          102   24    1     1
ABSENCE        42    24    1     9
ABSORPTION     6     24    1     10
ACCOUNT        87    24    1     5
ACCOUNTING     136   69    1     1
ACID           40    24    1     6
ACTION         101   24    1     5
ACTIVITIES     49    24    1     6
ACTIVITIES     130   24    1     7
ACTUAL         23    24    1     5
```

The following command loads the inverted file *cds* located in the directory \cisis\bases using the link files of fixed length cds.lk1 and cds.lk2 located in the directory \isis\work:

```
ifload \cisis\bases\cds \isis\work\cds.lk1 \isis\work\cds.lk2 +fix
```

A procedure to generate an inverted file ***x*** using the default Field Select Table (FST) and Stopword file is:

```
mx x fst=@ stw=@ ln1=x.ln1 ln2=x.ln2 +fix tell=100
del *.$$$
mys 37 x.ln1 x.lk1
del x.ln1
del *.$$$
mys 57 x.ln2 x.lk2
del x.ln2
ifload x x.lk1 x.lk2 +fix tell=1000
del x.lk1
del x.lk2
```

When you create various inverted files for a given master file, and proximity searching is not required, only the MFN component of the *posting* is necessary. The link files can be in the format:

```
ABOUT          102
ABSENCE        42
ABSORPTION     6
ACCOUNT        87
ACCOUNTING     136
ACID           40
ACTION         101
ACTIVITIES     49
ACTIVITIES     130
ACTUAL         23
```

and loaded with the command:

```
ifload \cisis\bases\cds \isis\work\cds.lk1 \isis\work\cds.lk2 +fix/m
```

The program IFLOAD allows loading only the dictionary of the inverted file, using the option *-posts*:

```
ifload authority x.lk1 x.lk2 -posts
```

Después que se carga a file invertido, por defecto, the señal of *I/F update is pending* se reinicializa in all the records asociados of the master file.

When various inverted files are created for a given master file, you should specificy that the marker *I/F update is pending* is maintained, as indicated below:

```
ifload au au.lk1 au.lk2 reset=0
ifload ti ti.lk1 ti.lk2 reset=0
```

If it is necessary to reinitialize the marker *I/F update is pending* and the name of the inverted file to be loaded is different from the associated master file, you should use the option *master*

```
ifload kw kw.lk1 kw.lk2 master=\cisis\bases\cds
```

# IFLOAD - Syntax

```
ifload <dbname> {<file_lk1>|void} {<file_lk2>|void} [<option> [...]]
```

## Mandatory parameters

**<dbname>**

**<file_lk1>**

**<file_lk2>**

### Input inverted file

**<dbname>**

Inverted file to be loaded. The inverted file <*dbname*> is created or reinitialized if it already exists.

### Link file of short keys

**<file_lk1|void>**

Link file of short keys, sorted.

### Link file of long keys

**<file_lk2|void>**

Link file of long keys, sorted.

## Mandatory parameters

**master=<name>**

**- {reset | posts | balan}**

**tell=<n>**

**+fix[/m]**

### Reinitializing the marker of update pending

**master=<mst_name>**

Reinitializes the marker *I/F update is pending* in the master file <*mst_name*>; by default it processes the master file <*dbname*>.

### Update pending

**-reset**

Do not reinitialize the marker *I/F update is pending* in the master file <dbname> or <*mst_name*>.

## Do not balance the dictionary

**-balan**

Does not rebalance the dictionary (the B* trees)

## Do not load postings

**-posts**

Loads only the dictionary. Does not load the *postings* in the file *.ifp*.

## Information about the execution of the process

**tell=<n>**

Produces a message about progress in the *standard error* each <n> records of links loaded.

## Files of fixed-length links

**+fix[/m]**

## Load files of fixed length

**+fix**

Loads files of links with records of fixed length with the format:

```
KEY MFN TAG OCC CNT
```

## Load file of links with reduced format

**+fix/m**

Loads files of links with records of fixed length with the format:

```
KEY MFN
```

# IFLOAD - Output

The inverted file consists of six physical files, five of which contain the dictionary of search terms (organized as a B* tree) and the sixth contains the list of postings associated with each term. To effect the optimizing of storage on disk, the B* trees (a data structure that permits storing classified information) are maintained separately, one for terms of up to 10 characters and the other for terms of more than 10 characters and up to a maximum of 30 characters. Both B*trees are structured as pages of fixed length, with complete keys with blank spaces to the right.

You can obtain a further optimization of disk space using the Utility Program of CISIS, MYZ (chapter 21), which compresses the dictionary of the inverted file for each one of the B* trees, as indicated below:

```
myz ifn 1 ifn_z tell=10
myz ifn 2 ifn_z tell=10
```

The dictionary of the resulting inverted file, *ifn_z,* is made up of the files *ifn_z.cnt, ifn_z.n01, ifn_z.l01, ifn_z.n02 and ifn_z.l02.*

After the commands have been executed, the file *ifn.ifp* should be renamed as *ifn_z.ifp*, where *ifn_z* is the compressed version of the inverted file *ifn*.

# IFUPD - Program

The program IFUPD updates an inverted file, in accordance with the contents of:

- The Field Select Table (FST)
- Stopwords
- The records of the master file
- The processing options

The program IFUPD can also be used to create or reinitialize an inverted file.

# IFUPD - Presentation

The command:

```
ifupd \cisis\bases\cds fst=@ stw=@
```

reads the master file *cds* located in the directory \cisis\bases and updates the corresponding inverted file, according to the default Field Select Table (FST) and the *Stopword*s file *-cds.fst* and *cds.stw-* located in the same directory.

The following command uses a different Field Select Table and specifies that the keys of the inverted file are to be extracted without using *Stopwords*:

```
ifupd \cisis\bases\cds fst=@\cisis\bases\another.fst
```

IFUPD accepts specifications in the input line, such as:

```
ifupd \cisis\bases\cds "fst=70 0 (v70/); 69 2 v69"
```

IFUPD allows several invetred files to be updated for a given master file. In this case, in all the updates (except the last) it should be specified that the marker *I/F update is pending* is maintained, as indicated below:

```
ifupd au fst=@au.fst master=\cisis\bases\cds reset=0
ifupd ti fst=@ti.fst master=\cisis\bases\cds reset=0
ifupd kw fst=@kw.fst master=\cisis\bases\cds
```

In the last line of the example, reset=0 does not appear because all the inverted files are updated, so it is not necessary to mantain the marker of update pending.

# IFUPD - Syntax

```
ifupd [mono|full] [create=]<ifname> [<option> [...]]

[mono|full]                          → single/multi user operation
[create=]                            → to delete+create <ifname>
<ifname>                             → output inverted file

options:

 fst=<fstspec>|@[fstfile]            → field select table
 stw=<stwspec>|@[stwfile]            → stop words
 -posts                              → (init and) do not load .ifp
 master=<name>                       → alternate master file
 actab=<file>                        → alphabetic chars table
 uctab=<file>                        → upper case chars table
 from=<n>                            → initial mfn
 to=<n>                              → final mfn
 count=<n>                           → max mfns
 tell=<n>                            → tell <n>% loaded
```

## Mandatory parameters

**<ifname>**

### Inverted file to be updated

**<dbname>**

Inverted file to be updated.

If you specificy *create=<dbname>*, the inverted file <*dbname*> is created and if it already exists it is reinitialized.

## Field Select Table

**fst=<fstspec>|@[fstfile]**

### Default FST

**fst=@**

Uses the Field Select Table (FST) by default.

### Loading FST from an external file

**fst=@<file>**

The Field Select Table (FST) is supplied in the file <file>.

### Specification of FST in line

**fst=<fstspec>**

The Field Select Table (FST) is supplied in the input line as a list separated by full stop and commas.

## File of non-significant words

**stw=<stwspec>|@[stwfile]**

### Default STW file

**stw=@**

uses the *Stopword* file by default.

### Load STW list from an external file

**stw=@<stwfile>**

The list of non-significant words (*stopwords*) is supplied in the file <file>.

## Mantaining the indication of update pending

**reset=0**

Mantains the marker *I/F update is pending*; by default reinitializes all the records processed.

## Do not load postings

**-posts**

only updates the dictionary. It does not load the postings in the file *.ifp*.

## Alternative master file

**master=<name>**

By default it assumes that the master file is the file that has the same name as the inverted file which is to be processed, but if you indicate *master=<name>*, *ifupd* it will use the master file called <name>.

# IFUPD - Output

Only the records of the master file which have the marker of *I/F update is pending* are processed.

The parameter *reset=0* allows the records processed from the master file to be used later to create or update another inverted file.

# MYS - Syntax

```
mys {link1|link2|<preclen>} <fileln> <filelk> [<option> [...]]

options:  tell=<n>
          +fix/m

Ex: mys 37 x.ln1 x.lk1
     mys link1 x.ln1 x.lk1 tell=0

Ex: mys 57 x.ln2 x.lk2
     mys link2 x.ln2 x.lk2 tell=0
```

# MYS – Output

performs a sort of the links file to create the inverted file.

# IFMERGE - Syntax

```
ifmerge <out> <if1>[,n1] <if2>[,n2] [...] [<option> [...]]
 <out>                          → output inverted file
 <if1>[,n1]                     → input inverted file #1, maxmfn#1
 <if2>[,n2]                     → input inverted file #2, maxmfn#2

options:
+mstxrf                         → create <out> M/F and its mstxrf <out>.pft
-posts                          → do not load .ifp
-balan                          → do not rebalance the dict
tell=<n>                        → tell <n> keys have been loaded
```

## IFMERGE – Output

combines various inverted files of different master files into a single inverted file, with a procedure to retrieve the records from the source master files.

## MKIY0 - Syntax

```
mkiy0 <dbn> [-ifp] [-v] [blksize=32768]
```

## MKIY0 - Output

combines the six files which comprize the inverted file into a single physical file.

## CRUNCHIF - Syntax

```
crunchif <dbn> <target_dbn> [<option> [...]]

options:
-ifp                              → don't crunch .ifp file
/ifp                              → crunch .ifp file if needed
tell=<n>                          → tell <n> records processed

target={linux|hpux|sun|alpha|vax|unisys|mpe|cdc|pc}  default: linux
```

## CRUNCHIF - Output

converts the inverted file from one operating system to another, for example from Windows to Linux.

# Bibliographic references


1. UNESCO. *Mini-micro CDS/ISIS:* Reference manual (version 2.3). Organized by Giampaolo Of the Bigio. Paris: United Nations Educational, Scientific and Cultural Organization, 1989. 286 p. ISBN 92-3-102-605-5.

2. BUXTON, Andrew, HOPKINSON, Alan. *The CDS/ISIS for Windows Handbook* [online]. Paris: United Nations Educational, Scientific and Cultural Organization, 2001 [cited 30 August 2006]. 164 p. Available from internet: <http://bvsmoof theo.bvs.br/download/winisis/winisis-handbook-en.pdf>.

3. SUTER, Tito. “Prehistoria” e historia of the MicroISIS [online]. In: *Manual para instructores of Winisis*. Buenos Aires: Centro Atómico Constituyentes (CAC), Comisión Nacional of Energía Atómica (CNEA), 1999 [citado the 30 Agosto 2006]. p. 21-26. Disponible in internet: <http://www.cnea.gov.ar/cac/ci/isis/isidams.htm>.

# Glossary

- **Application**. Program used to execute tasks in connection with an application, such as the creation or edition of texts, drawings, animations, layout, etc. E.g.: text processor, database manager, Internet browser, etc.
- **Backup**. Procedure used to duplicate one or more files and/or directories in another storing device (tape or disc), thus producing a backup copy that may be restored in the event of accidental deletion or physical damage to the original data.
- **Bibliographic Database**. Electronic version of a catalog or bibliographic index.
- **Browser**. Internet page navigator, such as Internet Explorer and Netscape Navigator.
- **CDS/ISIS – MicroISIS**. Software program developed and maintained by UNESCO to manage bibliographic data.

- **CGI**. The Common Gateway Interface is a standard for interfacing external applications with information servers, such as HTTP or Web servers.

- **Database**. Collection of data that are structured to be easily accessed and handled. It is formed by units called records whose attributes are represented by fields. For example, in a file called "customer base", each customer is a record, with several fields such as "NAME", "CUSTOMER CODE", "TELEPHONE" etc.

- **Field**. Record element that provides storage of specific information. See Database.

- **File**. In computing, a set of data that may be saved into some type of storing device. The data files are created by applications, such as a text processor for example.

- **ISO Format (of files)**. Standard established by the ISO to allow the exchange of data between institutions, networks and users.

- **LILACS Format**. A bibliographic description format established by BIREME, based on the UNISIST Reference Manual for Machine-readable Bibliographic Descriptions.

- **Posting**. It is the address of a key extracted from the master file.

- **Presentation format**. Set of commands that defines the data output of an ISIS database.

- **Record**. Set of structured data aimed to store a specific subject. *See Database*.

- **Subfield**. Element that contains the tyniest piece of information in a field, whose meaning may be unclear if it is not analysed outside the scope of a group of elements.

- **UNISIST**. Intergovernmental program designed to foster cooperation in the field of scientific and technological knowledge.

- **URL**. Standard defined for the addressing of data contents via the TCP/IP protocol. Internet browsers use the URL to access Web pages.

# Appendix I - Parameters of general use

## Relating to the standard output:

**now, tell, -all, >, >>**

### Disable prompt between records

**now (nowait)**

This parameter disables the prompt which MX presents between record and record.

The parameter *now* or *nowait* permits the processing of all the records without operator intervention and suppresses output of the program prompt.

Thus you obtain the complete output in immediate form without stopping between record and record with the prompt of two full stops ..

```
mx cds from=24 to=50 pft=mfn/ now
```

## Inform every n records

**tell=<n>**

The parameter *tell=n* produces a brief message in the error outout device (*stderr*, normally the screen) every <n> records.

Although the output from processes may be redirected to a printed output or to a file, the messages produced by *tell=n* cannot be redirected and continue to go to the screen.

You are recommended to use this parameter in code with the parameter *-all* which suppresses the output to the screen in the list of records.

**Example:**

```
mx cds from=1 to=150 tell=30 now -all
```

produces on the computer screen only the following messages:

```
C:\>mx cds from=1 to=150 tell=30 now -all
+++ 30
+++ 60
+++ 90
+++ 120
+++ 150
C:\>
```

## Disabling dumping of information to the screen

**-all**

The parameter *-all* means that no type of information is sent to the screen, except error messages or information which generates the parameter *tell* (which uses the *standard error* to send its messages).

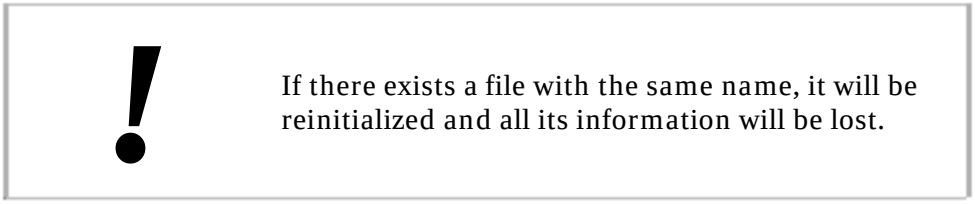
## Redirecting the standard output

**> <file> | >> <file>**

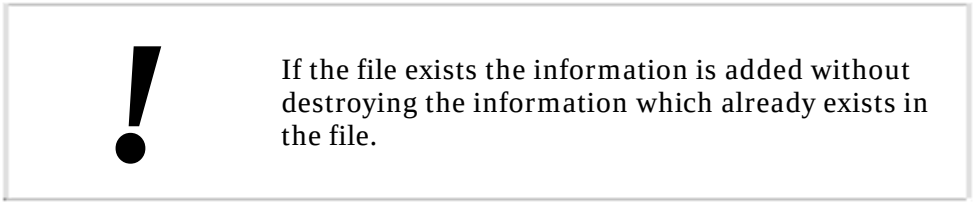
All the information presented on the screen (standard output by default) can be redirected to a file, printer, etc.

It is possible to direct the output to a file or to a printer using the redirection attributes of the operating system, > or >>.

If you add to the end of the command line the instruction >***file***, MX creates a file with the name ***file*** and puts in it all the information that would be directed to the standard output (screen).




If you place >>***file*** in the MX call, this opens the file with name ***file*** and adds all the information that would be directed to the standard output (screen). If the file does not exist it is created.



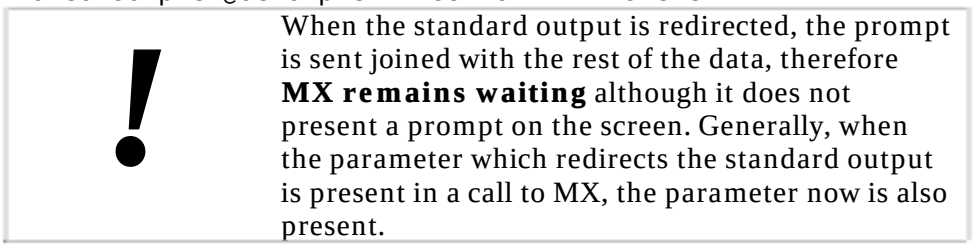

**Examples:**

- Send the list of the results of a search to the printer:

```
mx cds plants pft=@cds.pft now > LPT1
```

- Send the list of a range of records to a file:

```
mx cds from=10 to=30 pft=@otro.pft lw=35 now > file.txt
```




Note that if you add the parameter ***-all*** and there is no output, then the file to which you direct the output will remain empty.

# Relating to the selection of records:

**<from> <to> <count> <loop>**

## Start at record n

**from=<n>**

commences the processing at record <n> of the input database. If not specified, the process commences at the first record of the master file.

## Finish at record n

**to=<n>**

ends the processing at record <n> of the input database. If not specified, the process ends at the last record of the master file.

## Process each *nth* record

**loop=<n>**

jumps <n> records for each record processed.

For example, if this parameter is present and the value after the equals is 5, the records processed will be 1, 6, 11, etc.

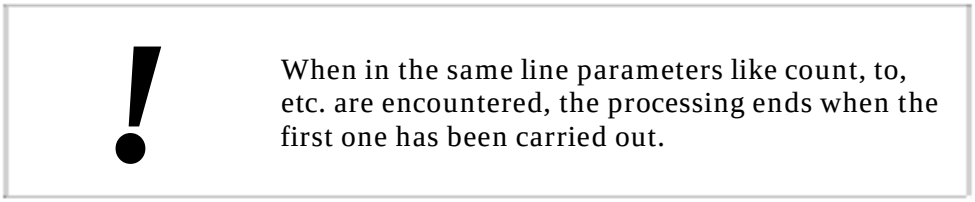
## Select *n* records

**count=<n>**

The parameter *count=n* selects exactly *n* records from an initial point. If you do not indicate the start record, it begins at the first record.

| **Example with MX** | **Output** |
|---|---|
| `mx cds from=24 to=50 loop=5 pft=mfn/ now` | `000024`<br>`000029`<br>`000034`<br>`000039`<br>`000044`<br>`000049` |
| `mx cds from=24 to=50 count=3 loop=5 pft=mfn/ now` | `000024`<br>`000029`<br>`000034` |
| `mx cds from=24 to=50 count=9 loop=5 pft=mfn/ now` | `000024` |

| Example with MX | Output |
|---|---|
| | 000029<br>000034<br>000039<br>000044<br>000049 |




# Relating to the output records:

**<offset>**

## Add n to the record numbers

**offset=<n>**

adds <n> to the MFN. Thus the MFN which is held in *dbout* is the *MFN* of the input record of input plus *offset*.

```
mxcp cds newcds offset=1000
```

In the example, by indicating offset=1000, you transfer the records of the database cds into the database newcds with mfn 1000, 1001, 1002, etc.

# Global change of patterns

**gizmo**

The parameter *gizmo*= allows making global changes to the contents of the fields of a CDS/ISIS database, converting a string of characters (or a single character) to another, and so to make modifications, coding/decoding, compression of data, etc.

These changes can be made against all the records of the database or to a set of records (selected by means of a search, a range, etc.). In turn, the changes can apply to the whole record or only to certain fields. For example, to change the signs < > to / /, or the string of characters *plants* to *PLANTAS*, etc.

To effect these changes it is necessary to prepare a *gizmo* master file. This master file contains in principle two fields: field 1 contains the data to change, and field 2 the new value. Each pair of data is a record in the *gizmo* database.

Each input record is submitted to the change procedure established in the *gizmo* file. At the beginning of the execution of MX, the data of the gizmo file are loaded as a table in memory and are ordered alphabetically by the value of field 1 and by their length. (In this way the longer strings of characters are converted before the short ones).

**Examples:**

A database called TEST is created using the known parameters of MX and the data are entered directly from the keyboard:

```
mx seq=con create=test -all now
<|/
>|/
plants|PLANTAS
<ctrl>Z (or <F6>)
```

Produces the following records:

```
mfn= 1
1 «<»
2 «/»
mfn= 2
1 «>»
2 «/»
mfn= 3
1 «plants»
2 «PLANTAS»
```

The contents of the title and descriptors fields of the record MFN=1 are:

```
mx cds to=1 "pft=mfn/v24/v69"
000001
Techniques for the measurement of transpiration of individual plants
Paper on: <plant physiology><plant transpiration><measurement and instruments>
```

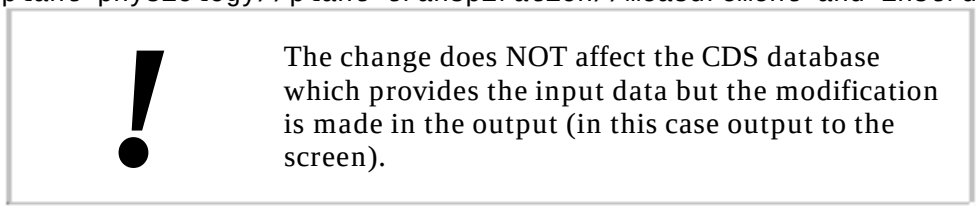
In the following example the parameter gizmo is applied:

```
mx cds gizmo=test to=1 "pft=mfn/v24/v69"
```

giving the result:

```
000001
```

```
Techniques for the measurement of transpiration of individual PLANTAS
Paper on: /plant physiology//plant transpiration//measurement and instruments/
```




To actually change the records you should send the results of the process to the same master file, as in the following example:

```
mx cds gizmo=test to=1 copy=cds –all now
```

If in the change you want to generate a new database, it is necessary to specify:

```
mx cds gizmo=test to=1 create=output –all now
```

It is possible to restrict the change to a specific field of the record, by indicating after the parameter *gizmo* the tag or tags to which the change is to be made. It is also possible to indicate a range of tags separated by /.

```
mx cds gizmo=test,69,24 to=1 create=output -all now
mx cds gizmo=test,35/56 to=1 create=output -all now
```

# Appendix II - CIPAR file

The CISIS programming interface provides a tool called CIPAR, which takes the place of SYSPAR.PAR and <dbn>.par of standard CDS/ISIS. It adds many other functions special to CISIS. CIPAR is a pure ASCII file that activates mechanisms for the search of names of logical files, names of databases, setting parameters of the work environment, etc.

A CIPAR parameter file consists of lines of commands, one command per line, with instructions about logical equivalences. A logical equivalence is a sentence of assignation where the value to the right of the = sign represents the setting of parameters to the left of the sign.

CISIS reads the lines of the CIPAR file and holds them in memory as a table of references. Each line is read until the end of the file or until the combination of /* (stroke, asterisk), whichever ocurrs first. The text that follows /* is considered as comment and documentation about CIPAR.

**Examples:**

Suppose that the file is called *DATA.CIP* and contains the following lines:

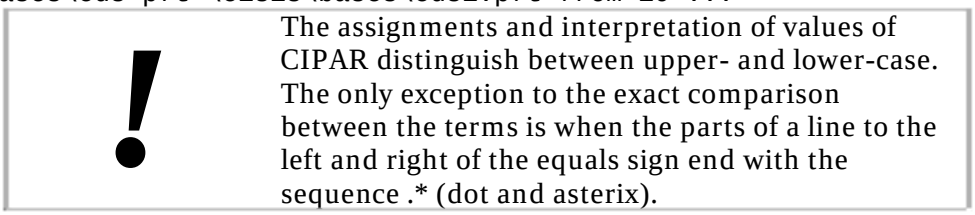
```
CDS.*=\cisis\bases\cds.*
CDS1.PFT=\cisis\bases\cds1.pft
```

Then the instruction

```
mx cipar=data.cip CDS pft=@CDS1.PFT from=10 ...
```

is equivalent to:

```
mx \cisis\bases\cds pft=\cisis\bases\cds1.pft from=10 ...
```




In the following example:

```
dbxtrace=y
14=1
cds.*=c:\cisis\bases\cds.*
cds1.pft=c:\cisis\bases\cds1.pft
lilacs.xrf=c:\lilacsok.xrf
lilacs.*=d:\bases\lilacs\lilacs.*
```

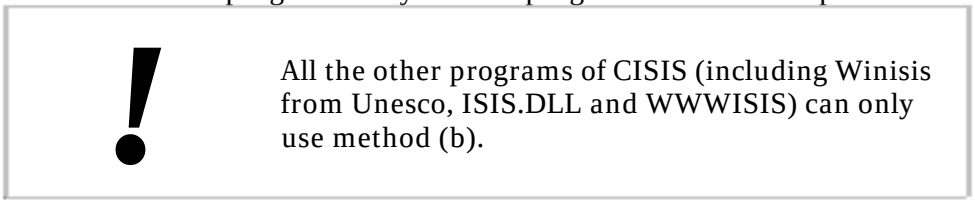
the terms to the left are converted or not to the terms on the right according to whether they comply or not with the exact comparison. If the MX program encounters a sequence of characters (which could be the name of a database, of a format, etc.), it searches in the lines of CIPAR and performs the transformations which correspond if they meet the line of equivalence.

| **Encounters** | **Converts to:** |
|---|---|
| dbxTRACE | Does not convert |
| dbxtrace | y |
| 14 | 1 |
| cds.mst | c:\cisis\bases\cds.mst |
| lilacs.xxx | d:\bases\lilacs\lilacs.xxx |
| lilacs.xrf | c:\lilacsok.xrf |
| LILACS.xxx | Does not convert |

Besides the definition of logical names of files it is possible to assign values of environment variables globally to the applications.

There are two ways to activate CIPAR:

a) Using it directly within the command line of MX by means of the parameter *cipar*=<*file*>, where <*file*> specifies the name of file which is to be used as CIPAR during the execution of the program. Only the MX program can use this option.



b) As an environment variable of the operating system. For this you use the command *set CIPAR=<name>* where *<name>* is the CIPAR file which is used in all the subsequent executions of the CISIS programs.

Example:

```
c:\>set cipar=\cipar.par
```

This instruction creates an environment variable (a global variable of the operating system) calleed CIPAR whose content is *c:\cipar.par*. All the CISIS applications which are executed from now take this file as CIPAR, no matter where the programs are run from.

It is possible to change the parameter to another file, reassigning the environment variable with the instruction set=<new file>.

```
c:\> set cipar=\other.par
```

To remove this environment variable from the operating system, it is sufficient to assign a null value:

```
c:\> set cipar=
```

The facility to assign global value sof the operating system is not exclusive to MS-DOS; other operating systems such as Unix also offer this possibility, athough using other commands:

- SunOs and UNIX/CSH use

```
setenv cipar=x
```

- HP-UX, UNISYS 6000 and UNIX/ksh use

```
cipar=x; export cipar
```

If an environment variable exists in CIPAR and also you declare the parameter *cipar=* in the command line of MX, this latter takes precedence for the specified execution.

**Example:**

```
C:\> set cipar=c:\cipar.par
C:\> mx cipar=other.par CDS from=10 to= ... etc.
```

The execution by MX of this command uses the definitions in the file **other.par**.

# Parameters which can be included in CIPAR

## Parameters only for MX

**[cgitag=<tag>] [cgipfx=<pfx>] cgi={<fmt>|mx}**

receives parameters of a call to the CGI.

It stores the CGI data in a repeatable field with the tag defined by *cgitag* - by default the field 2000. Each name/value pair of data is saved in the subfields ^n and ^v. MX reads each line by means of a format (<fmt> or mx.pft) which must be included as an additional parameter.

| | |
|---|---|
| cgitag=<tag> | Tag of field for the subfields n, v (by default 2000) |
| cgipfx=<pfx> | Prefix of the name of the field tag (by default **tag**) |
| cgi=<fmt> | Specifies the parameters of mx by means of a format |
| cgi=mx | Specifies the parameters of mx as the corresponding names |

**Examples**:

- In the command line you define the variables REQUEST_METHOD=GET and REQUEST_METHOD=GET, then you call mx indicating the format to be interpreted in the CGI mode.
  - In Linux and Windows:

```
set REQUEST_METHOD=GET
```

  - In Windows

```
set "QUERY_STRING=db~cds&count~2&now&btell~0&bool~plants*water&pft~mfn/"
```

  - In Linux

```
set QUERY_STRING="db~cds&count~2&now&btell~0&bool~plants*water&pft~mfn/"
```

  - In Linux and Windows:

```
mx cgi=mx
000004
000011
```

In order for the format to interpret the variable *QUERY_STRING*, you must use the reserved character & as the separator of occurrences and the character ~ as separator of subfields. The specified format must process the subfields *n* (before ~ in the case that it exists) and *v* (after ~ in the case that it exists) of the tag defined.

- In the command line the variables REQUEST_METHOD=GET and QUERY_STRING are defined, then mx is called, forcing the reading of data in the determined field (11000 in the example) and indicating the format to be interpreted for the CGI mode.
  - In Linux and Windows:

```
set REQUEST_METHOD=GET
```

  - In Windows

```
set "QUERY_STRING=db~cds&count~2&now&btell~0&bool~plants*water&pft~mfn/"
```

  - In Linux

```
set QUERY_STRING="db~cds&count~2&now&btell~0&bool~plants*water&pft~mfn/"
```

  - In Linux and Windows:

```
mx cgitag=11000   "cgi=(if v11000^n='db' then v11000^n,'=',v11000^v/ fi)"
mfn=1
 24 «Techniques for the measurement of transpiration of individual plants»
 26  «^aParis^bUnesco^c-1965»
```

```
 30  «^ap. 211-224^billus.»
 70  «Magalhaes, A.C»
 70  «Franco, C.M.»
 44  «Methodology of plant eco-physiology: proceedings of the Montpthelier
Symposium»
 ...
```

- In this example the parameters are passed through the CGI environment and not through the environment variables. In a web server, with the program mx and the format mx.pft written in the CGI area, the following page is loaded.

```
<html>
   <head>
      <title>Example of calling MX in the CGI environment</title>
   </head>
<body>
   <form method="post"  action="/cgi-bin/mx.exe/cgi=mx" >
      Execute MX with  'bool=' and 'count='  attributing the following values: <hr>
      Search expression (bool=): <input type="text" name="bool"
value="$"></input> <br/ >
      Number of records to display (count=): <input type="text" name="count"
value="10">
         </input> <br/ >
      <input type="hidden" name="db" value="d:\httpd\bases\cds\cds”></input>
      <input type="hidden" name="now"></input>
      <input type="hidden" name="btell" value="0"></input>
      <input type="hidden" name="pft"

value="lw(8000),newline('<br>'),@d:\httpd\bases\cds\cds”.pft"></input>
      <input type="submit" value="search & display"></input>
   </form>
</body>
</html>
```

In the example, the standard format *mx.pft* has been used, as can be seen in the attribute *action* of the form.

### cipar: ci_tempdir=<path>

defines the drive and/or directory where the temporary work files are created. If you do not define the parameter ci_tempdir, then the temporary files are created in the directory indicated by the environment variable TEMP or TMP.

```
echo ci_tempdir=c:\work >xcip
mx cipar=xcip cdromdb

@set ci_tempdir=c:\work
mx cdromdb
```

### cipar: maxmfrl=<nbytes>

specifies the size of the MFRmfrl (master file record length). By default it has a length of 32767 bytes, which is the maximum standard MFRmfrl.

```
echo maxmfrl=32767 >xcip
mx cipar=xcip bigrecsdb
```

**cipar: ci_fststrip=<maxlen>**

eliminates any marker of the type <text> or </text> up to a maximum length of <maxlen> characters for the data fields at the beginning of a executing an FST. (The data fields are padded with blank spaces.)

```
echo ci_fststrip=21 >xcip
mx cds
mfn= 1
69 «Paper on: <plant physiology><plant transpiration><measurement and instruments>»
mx cipar=xcip cds "fst=690 4 v69"
mfn= 1
69 «Paper on: <measurement and instruments> »
690 «PAPER^m1^o1^c1^l1»
690 «ON^m1^o1^c2^l1»
690 «MEASUREMENT^m1^o1^c3^l2»
690 «AND^m1^o1^c4^l1»
690 «INSTRUMENTS^m1^o1^c5^l2»
```

# Parameters of MX and applications programmed for a multiuser environment

**{netws | 14} | <dbn>.net ={0 | single | MONONETS} |**

**{1 | full | FULLNETS} |{ 2 | master | MASTNETS }**

The values of the parameter correspond with the values 0, 1, 2, of parameter 14 of the SYSPAR.PAR file. The default value is MONONETS. Also it is possible to assign the network parameter of a database through <*dbn*>.*net*=<*n*>

**Examples**:

```
14=1
netws=MASTNETS
cds.net=FULLNET
```

# Parameter maxmfrl

**maxmfrl=<n>**

performs the same specification as the parameter *mfrl=<n>* of the command line of the MX program.

# Parameter mstxl in CIPAR

**mstxl=<n>**

**mstxl | <dbn>/mstxl=<n>**

n=0 ... 4 (0 is the default value, predefined in the CISIS Interface)

Determines the maximum size that the file .MST can have. This parameter is read and considered only during the creation of the database.

| **<n>** | **Maximum .MST** | **Record in bytes multiple of** |
|---|---|---|
| 0 | 512 Mb | 2 (default value) |
| 1 | 1 Gb | 2 |
| 2 | 2 Gb | 4 |
| 3 or 4 | 4 Gb | 8 |

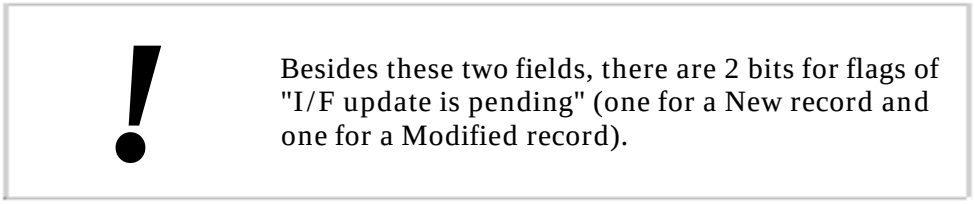
**Example:**

```
cds/mstxl=3
```

These parameters apply also to WinISIS, ISIS.DLL and WWWISIS and to all the aplications developed with the CISIS Interface.

## How to exceed the limit of 512 Mb for the master file:

The pointer to each record of the .MST is stored in the .XRF. This pointer is divided into two fields: the number of the block and the displacement (*offset*) within the block. These two fields occupy 20 and 9 bits respectively.




The pointer occupies 4 bytes and is processed as entered with a sign (a negative value indicates physically/logically deleted).

In this form the limit is 512 Mb, as indicated below:

```
2**20-1 = 1048675 blocks of 512 (offset from 0 up to 2**9-1 = 511)
```

The implementation of the MSTXL in the CISIS Interface uses *offset* for blocks of 512 bytes in units of 2, 4 and 8 bytes (using therefore 8, 7 and 6 bits for storing the offset) and block number in 21, 22, and 24 bits, respectively, for mstxl = 1, 2, and 3 (or 4).

The indication of a master file in the MSTXL format is stored in the leftmost byte of the MFTYPE field of the control record of the .MST, which is reinitialized when it is loaded in memory at the end of processing. This indication is "remembered" by CISIS during later reading or writing until this master file is closed.

It is sufficient to create a master file with a CIPAR which contains *mstxl*=<*n*> or <*dbn*>/*mstxl*=<*n*> because the following processes respect this initialization independently of the CIPAR.

# Parameter dbxtrace=y

**dbxtrace=y**

displays messages in the standard output (*stdout*):

```
dbxopen - <dbn><.ext> fd=<n> [R]
dbxopen - <dbn><.ext> fd=<n> [RW]
```

| | |
|---|---|
| <n> | file descriptor number |
| [R] | file was opened for read |
| [RW] | file was opened for read/write |

dbxtrace=y same as "trace=dbx" of the MX (like trace=rec, trm, giz, b40, par, mul, etc)

# Parameter mstload=<n>

**mstload=<n>**

loads into memory and closes the master files (default=0).

```
mstload=<n>  same as "load=<n>" of MX, for master fle
```

# Parameter invload=<n>

**invload=<n>**

Loads in memory and closes the inverted files (default=0).

```
invload=<n>  same as "load=<n>" of MX, for inverted file
```

# Parameter mclose={y|n}

**mclose={y|n}**

closes all the master files (default=n).

```
mclose=y    only leaves open a master file
```

# Parameter iflush={y|n}

**iflush={y|n}**

dumps all the inverted files (default=n).

```
iflush=y       only leaves open an inverted file invertido with its "data base
               descriptor" in memory
```

# Parameter mflush={y|n}

**mflush={y|n}**

dumps all the master files (default=n).

```
mflush=y       only leaves open a master file with its "data base descriptor" in
               memory. (data base descriptor includes input/output buffers and
               gizmos)
```

# Parameter what={y|n}

**what={y|n}**

displays the version of CISIS (default=n).

```
what=y         same as "what" of the MX (called as the unique parameter of MX)
```

**Example**

- To create a master file without records or reinitialize an existing one

```
mx seq=NUL create=NEW
```

If a database with this name already exists, MX does not advize that it will be cleared irreversably.

- To compact a database

This procedure recovers the space lost in the MST by successive updates of records.

```
mx DATA -all now create=DBN_AUX tell=100
xcopy DBN_AUX.* DATA.*
del DBN_AUX.*
FULLINV DATA DATA.FST DATA
```

- To verify duplicated elements

You wish to verify in the book catalogue that no duplicate inventory numbers exist.

The inventory numbers are recorded in field 7 and and are indexed with technique 0 with the prefix INV=, i.e. |INV=|v7.

```
mx LIBROS "pft=(if npost(|INV=|v7) > 1 then mfn,x3,v7/ fi)" tell=100 now >
duplic.lst
```

The procedure requires that the inverted file is present.

- To obtain a rapid analysis of the fields used in a database

The programs `MX.EXE` and `MXF0.EXE` are necessary and these two format files:

```
DMXF0A.PFT
'Analysis of the data of the database ',v1001/#
" 1001 = input master file name ................. "v1001/,
" 1003 = date & time stamp ...................... "v1003/,
" 1009 = total number of records ................ "v1009/,
" 1010 = number of active records ............... "v1010/,
" 1011 = number of logically deleted records .... "v1011/,
" 1012 = number of physically deleted records ... "v1012/,
DMXF0B.PFT
/#
"TAG DOCS OCCS"d1020/
"---------------------"d1020/
,(v1020^t,v1020^d,v1020^o/),
With this you prepare a .bat file, like the following:
MYSCAN.BAT
REM %1 = <dbn_name> %2 = <number of records estimated>
REM
mxf0 %1 create=list %2 tell=100
mx list pft=@dmxf0a.pft now > %1.lst
mx list pft=@dmxf0b.pft now >> %1.lst
```

The execution of *MYSCAN.BAT* requires two input parameters, the database with its path and the estimated number of records it contains.

As a result, it generates a file with the name of the database and extention .lst.

**Example:**

```
myscan c:\dbisis\cds\cds 150
```

- To eliminate duplicate terms in a repeatable field

Suppose that the descriptors are held in the field v87, as a repeatable field.

```
mx DATA fmtl=20000 proc=@CLEAN from=1 to=100 now -all create=OUT
```

The file *CLEAN* contains the following format specification:

```
proc('d870d871'),
( if v870[1] : s(v87|~|)
then
these proc('D870A870|'v870[1],v87'~|','A871|'v87'|')
fi ),
proc('d870'),
proc('d87d871',|A87~|v871|~|),
```

- Quality control of the data

A simplified model of a .bat file is offered to perform various controls on a database.

For this example a database called *TEST* is used to carry out the controls indicated in the menu, and a base *THES* (thesaurus) against which the descriptores of *TEST* will be validated.

```
CHK.BAT
@echo off
:BEGIN
cls
echo ------------------------------------------------------------
echo QUALITY CONTROL
echo ------------------------------------------------------------
echo.
echo E - Delete invalid characters (clean records)
echo O - Check mandatory fields
echo D - Check Descriptors
echo X - Exit
echo.
choice /c:EODX /N Select one option:
if errorlevel 4 goto END
if errorlevel 3 goto DESCRIPTORS
if errorlevel 2 goto MANDATORY
if errorlevel 1 goto CLEAN
:CONTINUE
echo.
echo (Press any key to continue...)
pause > nul
goto BEGIN
:CLEAN
call OPC_CLN. BAT
goto CONTINUE
:MANDATORY
call OPC_OBL.BAT
goto CONTINUE
:DESCRIPTORS
call OPC_DES.BAT
goto CONTINUE
:END
```

The procedure is completed with a series of .bat files which perform each one of the options presented.

The files are called *OPC_xxx*, where xxx represents one of the options of the menu. Each *OPC_xxx* activates a CISIS program that calls a file *in=<text_file>* which contains the parameters necessary for the validation function. Associated with this *<text_file>* there is a format file with the same name and with the extension *.pft*, which is used for the validation of the data.

If the validation is carried out against an inverted file, the *in=<text_file>* will call a procedure of *jchk*.

```
OPC_CLN.BAT
@echo off
mxcp \dbisis\TEST \dbisis\TEST clean > garbage
OPC_OBL.BAT
@echo off
echo Wait..
mx in=chk00 >> errors.tmp
CHK00
\dbisis\test
pft=@chk00.pft
-all
now
CHK00.PFT (example of a .PFT of validation)
if a(v02) then mfn,c8,'02->FATAL ERROR: field #2 missing!' fi/
if a(v01) then mfn,c8,'01->ERROR: field #01 missing!' fi/
if a(v923) and p(v23) then mfn,c8,'923->ERROR: field #923 missing!' fi/
if v31<>'ENG' then mfn,c8,'31->ERROR: field #31 invalid!' fi/
OPC_DES.BAT
@echo off
echo Wait...
mx in=chk620 >> errors.tmp
 CHK620 (verifies field 620 against a thesaurus)
\dbisis\test
jchk=Thes=mhu,(v620/)
pft=@chk620.pft
-all
now
 CHK620.PFT
if p(v32001) then ( if a(v32001^m) then mfn,c8,|620->|v32001^k,c65,'*invalid'/ fi )
fi
```

- Special index of titles

In indexing the title field (series or monographs) with technique 0 (whole field), the alphabetic order in the dictionary does not respect the normal bibliographic convention of not considering the words such as A, An, La, Le and The.

In CDS/ISIS it is possible to obtain the printed output in the correct form if you enclose such words between <..>, but this solution is not available for the inverted file. The following example resolves the problem.

The fields v24 (title) is indexed, between the records 15 and 20, in three ways to compare the results. Examples 1 and 2 show what you get with standard CDS/ISIS, and example 3 shows the solution of the problem.

The following FST is used: 240 0 v24

```
mx CDS from=15 to=20 now -all "fst=240 0 v24" ifupd/create=cds
ifkeys cds +tags
1| 240|<A> METHOD OF DETERMINING EVAP
1| 240|<THE> HEAT RESISTANCE OF PLANT
1| 240|<THE> MEASUREMENT OF DROUGHT R
1| 240|<THE> ROLE OF DEW IN PINE SURV
1| 240|GAUGES FOR THE STUDY OF EVAPOT
1| 240|MEASUREMENT OF DROUGHT RESISTA
```

All the titles that have <...> appear at the beginning of the dictionary because the character < has an ASCII value less than the letter "A".

The following FST is used: 240 0 mhu,v24

```
mx CDS "fst=240 0 mhu,v24" ifupd/create=cds now -all from=15 to=20
```

The result is presented in the following dictionary:

```
Ifkeys CDS +tags
1| 240|A METHOD OF DETERMINING EVAPOT
1| 240|GAUGES FOR THE STUDY OF EVAPOT
1| 240|MEASUREMENT OF DROUGHT RESISTA
1| 240|THE HEAT RESISTANCE OF PLANTS,
1| 240|THE MEASUREMENT OF DROUGHT RES
1| 240|THE ROLE OF DEW IN PINE SURVIV
```

Because of the use of MHU, the <..> do not appear in the dictionary. The sorting considers the words enclosed as not significant.

The solution is obtained using a *gizmo*= WORDS, whose content is:

```
mfn= 1
1 «<The> »
mfn= 2
1 «<A> »
mfn= 3
1 «<An> »
mfn= 4
1 «<La> »
mfn= 5
1 «<Le >»
mfn= 6
```

```
1 «<L'> »
mfn= 7
1 «<Les> »
..
```

There is no field 2 in the gizmo, because the conversion transforms the data of field 1 into a null output. Also a blank space is included at the end of each field 1 so that the title fields, after the gizmo is applied, do not begin with a blank space.

```
mx CDS from=15 to=20 now -all "fst=240 0 v24" ifupd/create=cds gizmo=words,24
ifkeys CDS +tags
1| 240|GAUGES FOR THE STUDY OF EVAPOT
1| 240|HEAT RESISTANCE OF PLANTS, ITS
2| 240|MEASUREMENT OF DROUGHT RESISTA
1| 240|METHOD OF DETERMINING EVAPOTRA
1| 240|ROLE OF DEW IN PINE SURVIVAL I
```

# Appendix III - Structure of the records of an ISIS database

A record with the ISIS structure has two special characterstics which give great versatility for the management of textual information: repeatable fields and variable length fields.

Because the records do not have a predetermined length, nor do the fields have a fixed length or a predetermined number of occurrences, it is not possible to have direct access any piece of data within the database.

Access to the record is obtained indirectly, by means of pointers in an auxiliary file with the extension .XRF, and within the record you get to the data through pointers in a directory.

The .XRF file contains all the information necessary to finf the starting point of the desired record within the .MST.

All the examples which follow use the record with MFN=1 in the database CDS, whose complete contents are the following:

| \cisis\bases\cds | Database |
|---|---|
| Nxtmfn nxtmfb nxtmfp t reccnt mfcxx1 mfcxx2 mfcxx3 RC<br>152 123 13 0 0 0 0 0 0 | Control record |
| Mfn= 1\|comb= 1\|comp= 64\| N\| ... \| 1+000=00000c40 | xref |

| | |
|---|---|
| Mfn= 1\|mfrl= 370\|mfbwb= 0\|mfbwp= 0\|base= 66\|nvf= 8\|status= 0\| 0 | leader |
| Mfn= 1\|dir= 1\|tag= 44\|pos= 0\|len= 77<br>Mfn= 1\|dir= 2\|tag= 50\|pos= 77\|len= 11<br>Mfn= 1\|dir= 3\|tag= 69\|pos= 88\|len= 78<br>Mfn= 1\|dir= 4\|tag= 24\|pos= 166\|len= 68<br>Mfn= 1\|dir= 5\|tag= 26\|pos= 234\|len= 22<br>Mfn= 1\|dir= 6\|tag= 30\|pos= 256\|len= 20<br>Mfn= 1\|dir= 7\|tag= 70\|pos= 276\|len= 15<br>Mfn= 1\|dir= 8\|tag= 70\|pos= 291\|len= 12 | dir |
| Mfn= 1<br>44 «Methodology of plant eco-physiology: proceedings of the Montpellier Symposium»<br>50 «Incl. bibl.»<br>69 «Paper on: <plant physiology><plant transpiration><measurement and instruments>»<br>24 «Techniques for the measurement of transpiration of individual plants»<br>26 «^aParis^bUnesco^c-1965»<br>30 «^ap. 211-224^billus.»<br>70 «Magalhaes, A.C.»<br>70 «Franco, C.M.»<br>.. | fields |

# The CONTROL record

An ISIS database has a special record at the beginning (MFN=0) to which CDS/ISIS does not provide access. This record has a different structure from the rest of the records of the master file (MST).

*The information of this record is shown with the parameter* +control.

The structure is:

| **Structure of the CONTROL record** | |
|---|---|
| Ctlmfn | Always 0. This field does not apear with MX <dbn> +control. |
| Nxtmfn | MFN to be assigned to the next record |

| Structure of the CONTROL record | |
|---|---|
| Nxtmfb | Last block assigned in the .MST. The blocks are of 512 bytes. |
| Nxtmfp | First position free within the last block assigned. A record can begin in any free position between 0-498 and extend for one or more blocks. No record can start between the bytes 500 and 510. |
| Mftype | Type of database: 0 = user databasebase, 1 = database of system messages |
| Reccnt | Reserved |
| Mfcxx1 | Reserved |
| Mfcxx2 | Block of data entry. The values 0, 1...n, depend on how many records are being edited at that moment. |
| Mfcxx3 | Block of exclusive reading. Value 0 or 1. |

# The XREF record

The .XRF file is organized as a table of pointers to the master file (.MST). The first pointer corresponds to MFN=1, the second to MFN=2, etc.

Each pointer consists of two fields (4 bytes) which in the table of the example above indicate that the record MFN=1 commences in block 1 (*comb*) and within the bloque beginning at byte 65 (*comp*), and that it is not pending updating of the inverted file. The last value is the actual reference of the pointer expressed as a hexadecimal value.

Each block of the .XRF is a file of 512 bytes length and contains 127 pointers (data important for the reconstruction of the .MST as explained in the CTLMFN utility).

# The MST File Record

The records of the master file are stored consecutively, one after the other, each record occupying exactly *MFRL* bytes. Each file is stored as physical blocks of 512 bytes. A record can start at any point between position 0 and 498 and can extend for one or more blocks.

The MST record has a variable length and consists of three sections:

- A *leader* of fixed length
- A directory
- The data fields of variable length

## Structure of the LEADER

The *leader* consists of a block with a fixed length of 18 bytes.

| | |
|---|---|
| Mfn | Number of record |
| Mfrl | Total length of the record, including the three sections: *leader*, *directory* and data area. Always an even number. |
| Mfbwb | Pointer to the previous version of the record: number of the within the MST. Initially it is 0, and also after updating the inverted file. |
| Mfbwp | Pointer to the later version of the record. Displacement within the block. |
| Base | Position where the data area commences within the record. This value is the sum of the length of the *leader* and the length of the directory. |
| Nvf | Number of fields in the record, which is also the number of entries in the directory of the record |
| Status | Indicator of deletion (0 = active record; 1 = logically deleted) |

## Structure of the DIRECTORY

The *directory* of the record is an index to the contents of the record in the data segment. This index is made up of as many entries as the number of fields (nvf), permitting access to the data. Each entry in the *directory* is formed of three parts:

| | |
|---|---|
| Tag | Identifier of field or tag |
| Pos | Position in bytes of the first character in the data area corresponding to this ocurrence of the field |
| Len | Length of the field in bytes |

# Appendix IV - List of TAB files available

For use both in display and inversion of databases

## ASCII CODE PAGE 437 (CP437)

| | |
|---|---|
| ac437.tab | Valid characters of code ASCII CP437 |
| ac437n.tab | Valid characters of code ASCII CP437 incl. 0-9 |
| ac437XT.tab | Valid characters of code ASCII CP437 incl. 0-9 e &'()*+,-./: |
| ma437.tab | Conversion of characters to to upper-case (with accents) |
| mi437.tab | Conversion of characters to lower-case (with accents) |
| na437.tab | Eliminates accents and maintains the case of the characters |
| uc437.tab | Conversion of characters to upper-case (without accents) |
| lc437.tab | Conversion of characters to lower-case (without accents) |

## ASCII CODE PAGE 850 (CP850)

| | |
|---|---|
| ac850.tab | Valid characters of the code ASCII CP850 |
| ac850n.tab | Valid characters of the code ASCII CP850 incl. 0-9 |
| ac850XT.tab | Valid characters of the code ASCII CP850 incl. 0-9 e &'()*+,-./: |
| ma850.tab | Conversion of characters to upper-case (with accents) |
| mi850.tab | Conversion of characters to lower-case (with accents) |
| na850.tab | Eliminates accents mantaining upper-/lower-case of the characters |
| uc850.tab | Conversion of characters to upper-case(without accents) |

| | |
|---|---|
| lc850.tab | Conversion of characters to lower-case (without accents) |

# ANSI (Windows)

| | |
|---|---|
| acans.tab | Valid characters of the code ANSI |
| acansn.tab | Valid characters of the ANSI code incl. 0-9 |
| acansXT.tab | Valid characters of the ANSI code incl. 0-9 e &'()*+,-./: |
| maans.tab | Conversion of characters to upper-case(with accents) |
| mians.tab | Conversion of characters to lower-case (with accents) |
| naans.tab | Eliminates accents mantaining upper-/lower-case of the characters |
| ucans.tab | Conversion of characters to upper-case(without accents) |
| lcans.tab | Conversion of characters to lower-case (without accents) |

# GIZMOs available for conversion of the contents of databases

## Conversion of character codes

| | |
|---|---|
| g437ans | Conversion of ASCII CODE PAGE 437 to ANSI |
| gans437 | Conversion of ANSI to ASCII CODE PAGE 437 |
| g850ans | Conversion of ASCII CODE PAGE 850 to ANSI |
| gans850 | Conversion of ANSI to ASCII CODE PAGE 850 |
| gutf8ans | Conversion of UFT-8 to ANSI |
| gansutf8 | Conversion of ANSI to UFT-8 |

## ASCII CODE PAGE 437 (CP437)

| | |
|---|---|
| g437ma | Conversion to upper-case accented |
| g437mi | Conversion to lower-case accented |
| g437na | Removes accents mantaining upper-/lower-case |
| g437uc | Conversion to upper-case without accents (Upper case) |
| g437lc | Conversion to lower-case without accents (Lower case) |

## ASCII CODE PAGE 850 (CP850)

| | |
|---|---|
| g850ma | Conversion to upper-case accented |
| g850mi | Conversion to lower-case accented |
| g850na | Removes accents mantaining upper-/lower-case |
| g850uc | Conversion to upper-case without accents (Upper case) |
| g850lc | Conversion to lower-case without accents (Lower case) |

### ANSI (Windows)

| | |
|---|---|
| gansma | Conversion to upper-case accented |
| gansmi | Conversion to lower-case accented |
| gansna | Removes accents mantaining upper-/lower-case |
| gansuc | Conversion to upper-case without accents (Upper case) |
| ganslc | Conversion to lower-case without accents (Lower case) |

## Supplementary conversion of punctuation characters

| | |
|---|---|
| gentit | Conversion of characters to the corresponding HTML entity ("&'<>) |
| gchar | Conversion of the HTML entity to the corresponding characters ("&'<>) |

## Supplementary conversion of HTML entities

| | |
|---|---|
| Ghtmlans | Converts HTML entities to ANSI characters |
| ganshtml | Converts ANSI characters to HTML entities |
| ghtml850 | Converts HTML entities to ASCII characters CP850 |
| g850html | Converts ASCII characters CP850 to HTML entities |
| ghtml437 | Converts HTML entities to ASCII characters CP437 |
| g437html | Converts ASCII characters CP437 to HTML entities |

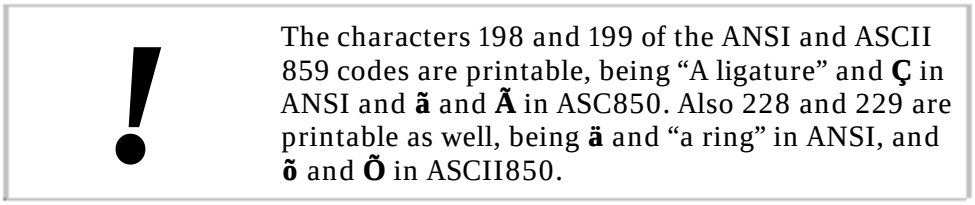
## How to recognize the character code in a CDS/ISIS database

If in the distribution of characters offered by `MXF0` there is an ABSENCE of characters with codes between 127(0x7f) and 159(0x9f) (inclusive), then they belong to the ANSI code.

If in the distribution of characters offered by `MXF0` there is a high concentration between the codes 128(0x80) and 167(0xa7) (inclusive) it is very probable that it is Code Page 437.

If in the distribution of characters offered by `MXF0` there is presence of elements in some of the following codes: 181(0xb5), 182, 183, 198, 199, 210, 211, 212, 214, 215, 216, 222, 224, 226, 227, 228, 229, 233, 234, 235, 236, 237 it is very probable that it is Code Page 859.

It is important to note the presence of characters which discriminate the code page: 198(0xc6) and 199(0xc6), which are **ã** and **Ã**, and also 228(0xe4) and 229(0xe5) which are **õ** and **Õ**.



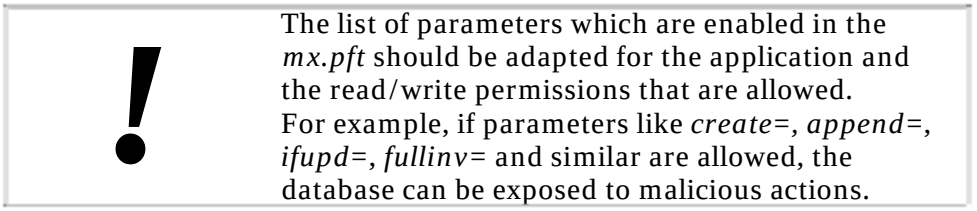

Complication: Windows 95 and 98 use different codes from Windows 98SE and later versions.

## NOTES

The gizmos for conversion between the character codes ANSI, ASCII CP437, and ASCII CP850 have as their objective to produce similar graphic characters.

The AUXILIARY CONVERSION gizmos are limited to accented characters only.

# Appendix V - MX.PFT: List of parameters which extract from the CGI environment




| **Parameter** | **Extraction format** |
|---|---|
| what | ,(if v2000^n='what' then v2000^n/ fi), |
| prolog= | ,(if v2000^n='prolog' then v2000^n'='v2000^v/ fi), |
| epilog= | ,(if v2000^n='epilog' then v2000^n'='v2000^v/ fi), |
| in= | ,(if v2000^n='in' then v2000^n'='v2000^v/ fi), |
| trace | ,(if v2000^n='trace' then v2000^n/ fi), |
| trace=dbx | ,(if v2000^n='trace=dbx' then v2000^n/ fi), |
| trace=rec | ,(if v2000^n='trace=rec' then v2000^n/ fi), |
| trace=dec | ,(if v2000^n='trace=dec' then v2000^n/ fi), |
| trace=trm | ,(if v2000^n='trace=trm' then v2000^n/ fi), |
| trace=b50 | ,(if v2000^n='trace=b50' then v2000^n/ fi), |
| trace=b40 | ,(if v2000^n='trace=b40' then v2000^n/ fi), |
| trace=fmt | ,(if v2000^n='trace=fmt' then v2000^n/ fi), |
| trace=fst | ,(if v2000^n='trace=fst' then v2000^n/ fi), |

| Parameter | Extraction format |
|---|---|
| trace=all | ,(if v2000^n='trace=all' then v2000^n/ fi), |
| cipar= | ,(if v2000^n='cipar' then v2000^n'='v2000^v/ fi), |
| mfrl= | ,(if v2000^n='mfrl' then v2000^n'='v2000^v/ fi), |
| fmtl= | ,(if v2000^n='fmtl' then v2000^n'='v2000^v/ fi), |
| load= | ,(if v2000^n='load' then v2000^n'='v2000^v/ fi), |
| pages= | ,(if v2000^n='pages' then v2000^n'='v2000^v/ fi), |
| uctab= | ,(if v2000^n='uctab' then v2000^n'='v2000^v/ fi), |
| actab= | ,(if v2000^n='actab' then v2000^n'='v2000^v/ fi), |
| -all | ,(if v2000^n='-all' then v2000^n/ fi), |
| -control | ,(if v2000^n='-control' then v2000^n/ fi), |
| -leader | ,(if v2000^n='-leader' then v2000^n/ fi), |
| -xref | ,(if v2000^n='-xref' then v2000^n/ fi), |
| -dir | ,(if v2000^n='-dir' then v2000^n/ fi), |
| -fields | ,(if v2000^n='-fields' then v2000^n/ fi), |
| +hits | ,(if v2000^n='+hits' then v2000^n/ fi), |
| cgiplushits | ,(if v2000^n='+hits''cgiplushits' then v2000^n'+hits'/ fi), |
| +fix | ,(if v2000^n='+fix' then v2000^n/ fi), |
| +fix/m | ,(if v2000^n='+fix/m' then v2000^n/ fi), |
| +all | ,(if v2000^n='+all' then v2000^n/ fi), |
| +control | ,(if v2000^n='+control' then v2000^n/ fi), |
| +leader | ,(if v2000^n='+leader' then v2000^n/ fi), |
| +xref | ,(if v2000^n='+xref' then v2000^n/ fi), |
| +dir | ,(if v2000^n='+dir' then v2000^n/ fi), |
| +fields | ,(if v2000^n='+fields' then v2000^n/ fi), |
| from= | ,(if v2000^n='from' then v2000^n'='v2000^v/ fi), |
| to= | ,(if v2000^n='to' then v2000^n'='v2000^v/ fi), |
| loop= | ,(if v2000^n='loop' then v2000^n'='v2000^v/ fi), |
| count= | ,(if v2000^n='count' then v2000^n'='v2000^v/ fi), |
| tell= | ,(if v2000^n='tell' then v2000^n'='v2000^v/ fi), |
| b50= | ,(if v2000^n='b50' then v2000^n'='v2000^v/ fi), |
| b40= | ,(if v2000^n='b40' then v2000^n'='v2000^v/ fi), |
| nb1= | ,(if v2000^n='nb1' then v2000^n'='v2000^v/ fi), |
| nbb= | ,(if v2000^n='nbb' then v2000^n'='v2000^v/ fi), |
| nb0= | ,(if v2000^n='nb0' then v2000^n'='v2000^v/ fi), |
| nb2= | ,(if v2000^n='nb2' then v2000^n'='v2000^v/ fi), |
| btell= | ,(if v2000^n='btell' then v2000^n'='v2000^v/ fi), |
| b50t= | ,(if v2000^n='b50t' then v2000^n'='v2000^v/ fi), |
| b40t= | ,(if v2000^n='b40t' then v2000^n'='v2000^v/ fi), |
| b40u= | ,(if v2000^n='b40u' then v2000^n'='v2000^v/ fi), |
| dupr= | ,(if v2000^n='dupr' then v2000^n'='v2000^v/ fi), |
| dupl= | ,(if v2000^n='dupl' then v2000^n'='v2000^v/ fi), |
| p1= | ,(if v2000^n='p1' then v2000^n'='v2000^v/ fi), |
| p2= | ,(if v2000^n='p2' then v2000^n'='v2000^v/ fi), |
| nowait | ,(if v2000^n='nowait' then v2000^n/ fi), |
| now | ,(if v2000^n='now' then v2000^n/ fi), |
| stderr=off | ,(if v2000^n='stderr=off' then v2000^n/ fi), |
| mono | ,(if v2000^n='mono' then v2000^n/ fi), |
| mast | ,(if v2000^n='mast' then v2000^n/ fi), |
| full | ,(if v2000^n='full' then v2000^n/ fi), |

| Parameter | Extraction format |
|---|---|
| db= | ,(if v2000^n='db' then v2000^n'='v2000^v/ fi), |
| dbn= | ,(if v2000^n='dbn' then v2000^n'='v2000^v/ fi), |
| dict= | ,(if v2000^n='dict' then v2000^n'='v2000^v/ fi), |
| k1= | ,(if v2000^n='k1' then v2000^n'='v2000^v/ fi), |
| k2= | ,(if v2000^n='k2' then v2000^n'='v2000^v/ fi), |
| cgi= | ,(if v2000^n='cgi' then v2000^n'='v2000^v/ fi), |
| null | ,(if v2000^n='null' then v2000^n/ fi), |
| tmp | ,(if v2000^n='tmp' then v2000^n/ fi), |
| seq= | ,(if v2000^n='seq' then v2000^n'='v2000^v/ fi), |
| iso= | ,(if v2000^n='iso' then v2000^n'='v2000^v/ fi), |
| isotag1= | ,(if v2000^n='isotag1' then v2000^n'='v2000^v/ fi), |
| bool= | ,(if v2000^n='bool' then if v2000^v>'' then v2000^n'='v2000^v/ fi fi),<br>if v2000:'^nbool^v' then 'bool='<br>(if v2000^n='bool' then if v2000^v>'' then v2000^v fi fi),/<br>fi, |
| mfbw | ,(if v2000^n='mfbw' then v2000^n/ fi), |
| tbin= | ,(if v2000^n='tbin' then v2000^n'='v2000^v/ fi), |
| tb= | ,(if v2000^n='tb' then v2000^n'='v2000^v/ fi), |
| join= | ,(if v2000^n='join' then v2000^n'='v2000^v/ fi), |
| jchk= | ,(if v2000^n='jchk' then v2000^n'='v2000^v/ fi), |
| jch0= | ,(if v2000^n='jch0' then v2000^n'='v2000^v/ fi), |
| jch1= | ,(if v2000^n='jch1' then v2000^n'='v2000^v/ fi), |
| jmax= | ,(if v2000^n='jmax' then v2000^n'='v2000^v/ fi), |
| jtag= | ,(if v2000^n='jtag' then v2000^n'='v2000^v/ fi), |
| proc= | ,(if v2000^n='proc' then v2000^n'='v2000^v/ fi), |
| convert= | ,(if v2000^n='convert' then v2000^n'='v2000^v/ fi), |
| decod= | ,(if v2000^n='decod' then v2000^n'='v2000^v/ fi), |
| gizp= | ,(if v2000^n='gizp' then v2000^n'='v2000^v/ fi), |
| gizmo= | ,(if v2000^n='gizmo' then v2000^n'='v2000^v/ fi), |
| giz1= | ,(if v2000^n='giz1' then v2000^n'='v2000^v/ fi), |
| giz2= | ,(if v2000^n='giz2' then v2000^n'='v2000^v/ fi), |
| putdir= | ,(if v2000^n='putdir' then v2000^n'='v2000^v/ fi), |
| getdir= | ,(if v2000^n='getdir' then v2000^n'='v2000^v/ fi), |
| sys= | ,(if v2000^n='sys' then v2000^n'='v2000^v/ fi), |
| sys/show= | ,(if v2000^n='sys/show' then v2000^n'='v2000^v/ fi), |
| text= | ,(if v2000^n='text' then v2000^n'='v2000^v/ fi), |
| text/show= | ,(if v2000^n='text/show' then v2000^n'='v2000^v/ fi), |
| lw= | ,(if v2000^n='lw' then v2000^n'='v2000^v/ fi), |
| pft= | ,(if v2000^n='pft' then v2000^n'='v2000^v/ fi), |
| invx= | ,(if v2000^n='invx' then v2000^n'='v2000^v/ fi), |
| iso= | ,(if v2000^n='iso' then v2000^n'='v2000^v/ fi), |
| outiso= | ,(if v2000^n='outiso' then v2000^n'='v2000^v/ fi), |
| outisotag1= | ,(if v2000^n='outisotag1' then v2000^n'='v2000^v/ fi), |
| fix= | ,(if v2000^n='fix' then v2000^n'='v2000^v/ fi), |
| ln1= | ,(if v2000^n='ln1' then v2000^n'='v2000^v/ fi), |
| ln2= | ,(if v2000^n='ln2' then v2000^n'='v2000^v/ fi), |
| fst= | ,(if v2000^n='fst' then v2000^n'='v2000^v/ fi), |
| fst/h= | ,(if v2000^n='fst/h' then v2000^n'='v2000^v/ fi), |
| stw= | ,(if v2000^n='stw' then v2000^n'='v2000^v/ fi), |

| Parameter | Extraction format |
|---|---|
| ifupd= | ,(if v2000^n='ifupd' then v2000^n'='v2000^v/ fi), |
| ifupd/create= | ,(if v2000^n='ifupd/create' then v2000^n'='v2000^v/ fi), |
| ifupd/dict= | ,(if v2000^n='ifupd/dict' then v2000^n'='v2000^v/ fi), |
| ifupd/create/dict= | ,(if v2000^n='ifupd/create/dict' then v2000^n'='v2000^v/ fi), |
| fullinv= | ,(if v2000^n='fullinv' then v2000^n'='v2000^v/ fi), |
| fullinv/dict= | ,(if v2000^n='fullinv/dict' then v2000^n'='v2000^v/ fi), |
| fullinv/keep= | ,(if v2000^n='fullinv/keep' then v2000^n'='v2000^v/ fi), |
| fullinv/m= | ,(if v2000^n='fullinv/m' then v2000^n'='v2000^v/ fi), |
| fullinv/ansi= | ,(if v2000^n='fullinv/ansi' then v2000^n'='v2000^v/ fi), |
| fullinv/dict/keep= | ,(if v2000^n='fullinv/dict/keep' then v2000^n'='v2000^v/ fi), |
| fullinv/dict/m= | ,(if v2000^n='fullinv/dict/m' then v2000^n'='v2000^v/ fi), |
| fullinv/dict/ansi= | ,(if v2000^n='fullinv/dict/ansi' then v2000^n'='v2000^v/ fi), |
| fullinv/dict/m/ansi= | ,(if v2000^n='fullinv/dict/m/ansi' then v2000^n'='v2000^v/ fi), |
| fullinv/ansi= | ,(if v2000^n='fullinv/ansi' then v2000^n'='v2000^v/ fi), |
| fullinv/m/ansi= | ,(if v2000^n='fullinv/m/ansi' then v2000^n'='v2000^v/ fi), |
| fullinv/keep/ansi= | ,(if v2000^n='fullinv/keep/ansi' then v2000^n'='v2000^v/ fi), |
| fullinv/keep/m/ansi= | ,(if v2000^n='fullinv/keep/m/ansi' then v2000^n'='v2000^v/ fi), |
| maxlk1= | ,(if v2000^n='maxlk1' then v2000^n'='v2000^v/ fi), |
| maxlk2= | ,(if v2000^n='maxlk2' then v2000^n'='v2000^v/ fi), |
| copy= | ,(if v2000^n='copy' then v2000^n'='v2000^v/ fi), |
| append= | ,(if v2000^n='append' then v2000^n'='v2000^v/ fi), |
| updatf= | ,(if v2000^n='updatf' then v2000^n'='v2000^v/ fi), |
| create= | ,(if v2000^n='create' then v2000^n'='v2000^v/ fi), |
| merge= | ,(if v2000^n='merge' then v2000^n'='v2000^v/ fi), |