

NEWCONVERT

Conversion Program

Draft Version

Reference Manual

Table of Contents

CHAPTER 1 INTRODUCTION	1
Glossary	3
MARC FORMAT	3
MARC COMMUNICATIONS FORMAT	3
HUMAN-READABLE FORMAT	3
INPUT FILE.....	4
OUTPUT FILE	4
LOG FILE.....	4
RULE TABLE	4
STATIC RULES.....	4
REPLACEMENT RULES (REPLACE { }).....	5
EXPANSION RULES (EXPAND { })	5
SUBSTITUTION DICTIONARY (DICT { }).....	5
CATALOGUE DEFINITION GRAMMAR (CDG).....	5
REGULAR EXPRESSIONS	5
REGULAR EXPRESSION SYNTAX.....	5
REPLACEMENT PATTERN	5
R_TAG	5
I_TAG.....	5
S_TAG.....	6
I_PAT	6
S_PAT.....	6
TRIM	6
SWITCH.....	6
Structure of this document.....	7
Conventions	7
Regular expression syntax.....	8
SEARCH PATTERN	8
REPLACEMENT PATTERN	10

CHAPTER 2 THE CONVERSION PROCESS	11
Conversion parameters	11
Where.....	11
<RULETABLE>	11
<SKIP>	11
<INPUT FILE>.....	11
“-C” SWITCH	11
“-J” SWITCH.....	12
Conversion Commands	12
RENAMING THE OUTPUT FILE	12
“SKIPPING” PROBLEMATIC RECORDS IN AN INPUT FILE AND CONVERTING REMAINING FILE.....	12
“CHAINING” TOGETHER INPUT FILES CONVERT.....	12
CHAPTER 3 RULE TABLE STRUCTURE.....	13
Introduction	13
Static rules	14
INFORMAT.....	14
OUTFORMAT.....	14
CHARSET	15
Catalog definition grammar (CDG)	16
REPLACEMENT RULES	16
<r_tag>	17
<i_tag>	18
<s_tag>.....	19
<s_pat>.....	19
<trim>	20
<i_pat>	20
<switch>	21
REPLACEMENT RULE SWITCHES.....	21
/tocc=curr (or: tag occurrence = current).....	21
/tocc=first (or: tag occurrence = first).....	23
/tocc=notfirst (or: tag occurrence not = first)	23
/tocc=last (or: tag occurrence = last).....	24

/mocc=curr (or: match occurrence = current)	24
/mocc=first (or: match occurrence = first)	25
/mocc=notfirst (or: match occurrence = not first).....	25
/mocc=last (or: match occurrence = last)	26
/upcase	26
/ignore	26
/reject=rec	28
/reject=rule	28
/reject=itag.....	28
/nodups.....	28
/log=().....	29
/s_substr=().....	32
/i_substr=().....	33
SUBSTITUTION DICTIONARY	33
<s_string>	34
<r_string>	34
EXPANSION RULE	35

CHAPTER 4 CONVERSION ERROR MESSAGES38

**CHAPTER 5 SAMPLE BIBLIOGRAPHIC RECORD
BEFORE AND AFTER CONVERSION43**

Before	43
After	44

**CHAPTER 6 CONVERSION CATALOGUE DEFINITION
GRAMMAR EXAMPLES45**

Examples of the replace rule used in simple replacement patterns	45
Example of the uses of replace and expand rules.....	48

Chapter 1

Introduction

This conversion program manipulates bibliographic records in MARC communications format. It creates an output file in either a human-readable format or in MARC format.

With the conversion program, it is possible to:

- Combine MARC and non-MARC fields in a single load file, thus eliminating the necessity of loading in separate procedures.
- Retain all data from the MARC source files, effecting minimal modifications (e.g. maintain all 24 data characters in the leader), when the MARC source files contain either standard MARC (US or CANMARC) tags, or the human-readable source file.
- Edit the human-readable output file, and then reformat this file in standard MARC communication format.
- Modify, delete and generate specific fields and subfields in the MARC record, and/or delete the entire record, based on specific parameters.
- Increase the parameters offered in the program, hence allowing a more flexible conversion of records from currently supported sources such as Bibliofile, OCLC and ISM, and new sources such as DOBIS and generic MARC source files.
- Provide greater flexibility by permitting the processing of multiple source files in the same job, as well as providing a way in which "abnormal" records causing a conversion to fail, can be skipped and processed at a later date.
- Create an exhaustive journal file in which problem cases are clearly illustrated.

A record in the human-readable file may consist of any combination of the following:

- The entire MARC leader from the source (input) record
- MARC tags
- Generated non-MARC tags

The conversion program uses a rule table to convert the bibliographic data. The rule table contains **STATIC** rules and, optionally, **REPLACEMENT**, **DICTIONARY** and **EXPANSION** rules. The replacement, dictionary and expansion rules compose what is called the Catalog Definition Grammar (CDG) and are written using regular expression syntax.

The CDG **REPLACEMENT** rules are defined by a single syntax. These rules can add any new tag or field to a record, modify any existing tag or field, or delete any existing tag or field. This allows the user to:

- Delete information which the user feels is extraneous
- Update tags coded according to obsolete conventions
- Generate non-MARC tags or subfields, all without running additional utilities after conversion

For maximum flexibility, the V8 Standard for regular expressions is used to define existing tags and data to search for, and to define new data to add.

The CDG **EXPANSION** rules are also defined by a single syntax, but they can only be applied to tags with subfields. The function of expansion rules is to parse a single tag for subfields which indicate the quantity of entities (such as local holdings) which the tag represents, and generate a single tag for each of those entities. So, for instance, if the input record has a 090 tag which indicates that location A has three volumes of an item and two copies of each volume, and that location B has only two of these volumes and one copy of each, then an expansion rule could be designed which would generate eight 852 tags to represent each of the items coded into the single 090 tag. For simple coding, an expansion rule can be used to generate the complete item tag. More often, an expansion rule will be used to **EXPAND** a composite tag into several simple temporary tags, and then replacement rules would use these new tags in conjunction with other tags to generate the actual item load tags (852).

GLOSSARY

MARC Format

MARC stands for Machine Readable Cataloging. The MARC format is a standard cataloging format which codes bibliographic and holdings information in specific tags, also called fields, and subfields. Each tag and subfield has its own inherent meaning. The purpose of a standard cataloging format is for the exchange of bibliographic information between automated library systems.

MARC Communications Format

MARC source data is a standard machine-readable format.

Example:

```
00779nam 22002410
45000010011000000050017000110080042000
28011002200070035001700092040001400109046000700123050001
50013008200180014509000670016309900220023024500840025225
00019003362600066003553000033004215040029004546500034004
8370000200051703-000102519920205160034.0831128c19721968n
yu a bd eng d| a 68010023 //r80
a88482098.C.. aQMDCbeng aLC 0aH40.A2bI50
a300/.3/21z190 aH/40/.A2/I
5/1972bREFd1/2,3/4,5/6,7/8,9/10,11/12,13/14,15/170 aH 40
.A2 I 5 197200aInternational encyclopedia of the social
sciences /cDavid L. Sills, editor. -- aReprint ed. --0
aNew York :bThe Macmillan Co. & The Free Press,c1972,
c1968. a17 v. in 8 :bill. ;c26 cm. aIncludes
bibliographies. 0aSocial sciencesxDictionaries10aSills,
David L.
```

Control characters in the record cannot be printed and are represented by the ^_ symbol.

Human-readable Format

This format is called a human-readable format, because it can be read and edited. It can be reformatted in a MARC communication format by this conversion program.

Example:

```
00779nam 22002410 4500
001 03-0001025
005 19920205160034.0
```

```

008 831128c19721968nyu a bd eng d|
011 $a 68010023 //r80
035 $a88482098.C.B.
040 $aQMDC$beng[B
050 0$aH40.A2$bI5
0820 $a300/.3/21$z19
0900 $aH/40/.A2/I
5/1972$bREF$d1/2,3/4,5/6,7/8,9/10,11/12,13/14,15/17
0990 $aH 40 .A2 I 5 1972
24500$aInternational encyclopedia of the social sciences
/$cDavid L. Sills, &editor. --
250 $aReprint ed. --
2600 $aNew York :$bThe Macmillan Co. & The Free
Press,$c1972, c1968.
300 $a17 v. in 8 :$bill. ;$c26 cm.
504 $aIncludes bibliographies.
650 0$aSocial sciences$xDictionaries
70010$aSills, David L.

```

Input File

A file to be processed by the conversion program. The file must follow either the MARC standard structure for bibliographic information, or the human-readable format generated by the conversion program.

Output File

File of converted bibliographic records.

Log File

File of logged messages indicating the presence or absence of data.

Rule Table

A rule table contains a set of rules that can add any new tag or subfield to a record, or modify any existing tag or subfield or delete any existing tag, subfield, or record.

A rule table contains static rules and optionally, replacement, dictionary and expansion rules. The conversion program to convert and process MARC source records uses this file.

Static Rules

Rules implicit to the program.

Replacement Rules (REPLACE { })

Rules in which a single syntax can be used to define any tag or field of a record. Once defined, the tag/field can be added to, modified, deleted, or used to create other tag(s). It can also be used to eliminate or validate bib records.

Expansion Rules (EXPAND { })

Rules in which a single syntax can be used to process any tags in the MARC record containing subfields. These rules are used primarily to extract and expand upon condensed information within a tag's subfields (e.g. holdings information) and to create separate data elements, e.g. item tags.

Substitution Dictionary (DICT { })

Is used to translate one character string to another.

Catalogue Definition Grammar (CDG)

An ordered set of replacement and expansion rules used to build a conversion table.

Regular Expressions

A way of defining a pattern in which special control characters will find and/or replace strings of information.

Regular Expression Syntax

Used to define a regular expression

Replacement Pattern

In a regular expression, when a search pattern has been matched, it will be replaced by the replacement pattern.

R_tag

The replacement tag controls the execution of a replacement rule. It is a tag to be replaced in a record.

I_tag

Is a tag to be inserted, i.e. created, in a replace rule.

S_tag

Is a tag to be searched for in order to build a new <i_tag>.

I_pat

Is the pattern of data used to replace the S_pat

S_pat

Is the string of data to be searched for in the data of the S_tag to build the new I_tag.

Trim

Is an indication to the conversion program to "trim", i.e. cut a string of characters, e.g. spaces, etc. from the end of a subfield.

Switch

Is a qualifier that overrides some default mode of operation for a <s_tag> line.

STRUCTURE OF THIS DOCUMENT

This manual comprises six chapters. Each chapter has its own pagination.

CONVENTIONS

This manual uses the following convention:

Italics Program examples

It is assumed that the reader is familiar with the Standard MARC Record Structure and Contents.

REGULAR EXPRESSION SYNTAX

Search Pattern

In the SEARCH PATTERN <r_tag>, <s_tag>, <s_pat>, </s_substr> and /i_substr:

<	marks the beginning of a subfield (subfield indicator).
>	marks the end of a subfield.
<<	matches a single less-than (" < ") character in the text.
>>	matches a single greater-than (" > ") character in the text.
^	at the beginning of a string, marks the start-of-string position.
[^]	As the first character inside square parentheses (i.e. [^<char list>]) matches on a single character NOT in <char list> .
\$	matches only at the end of a character string. The \$ can be used to find a character string that ends in a period, for instance. That is, "\." will match on any of the three periods in the string " Sentence1. Sentence 2. Sentence 3. ". But "\.\$" will only match on the period which is the last character in the whole string.
\<char>	escapes <char> , so that <char> is taken literally, even if it is normally a special character in regular expressions, as in "\^" below.
\^	matches an actual caret character (" ^ ") in the text.
\\$	matches a dollar sign (" \$ ") in the text.
\\	matches a single backslash (" \ ") in the text.
[<char list>]	will match any single character in <char list> . For example, "[aAbBcC]" will match a single upper- or lower-case a, b, or c. A range of characters can be specified, such as "[a-z]" for all lowercase letters, "[A-F]" for uppercase A through F, or "[a-zA-Z0-9]" for any uppercase or lowercase letter, or any digit. If the hyphen ("-") is not between the limits of a range, then it is taken as a hyphen. For example, "[A-] " will match "A" or "-" but "[A-Z] " will match any uppercase letter [^<char list>] will match any character NOT in <char list> .
.	will match any single character.
\.	will match an actual period (" . ") in the text.

*	will match 0 or more occurrences of the preceding character. For example, "a*" will match as many contiguous A's as it can. "[a-z]*" will match as many contiguous lowercase letters as it can.
*	will match a single asterisk ("*") in the text.
.*	will match one or more characters.
+	will match 1 or more occurrences of the preceding character.
\+	will match a single plus sign ("+") in the text.
?	will match exactly 0 or 1 occurrences of the preceding character.
\?	will match a single question mark ("?") in the text.
(<pattern>)	identifies a sub-pattern within a search pattern. For instance, "\\$([0-9]+)\.([0-9]+)" will match on a dollar sign followed by one or more digits, followed by a period, followed by one or more digits. If this search pattern was matched against the text " price \$10.95 Cdn ", then the replacement pattern "&" would include the entire match string, i.e., " \$10.95 ". However, the first and second sets of digits are in parentheses, so they can be identified separately. Thus, the replacement pattern "\1" would return the match from the first set of parentheses, " 10 ", and "\2" would return the match from the second set of parentheses, " 95 ", so the replacement pattern "& ` \1 dollars and \2 cents " would return " \$10.95 ` 10 dollars and 95 cents ". Up to 9 sub-patterns can be used within a search pattern.

Replacement Pattern

In the REPLACEMENT PATTERN:

&	returns the entire matched string, as mentioned, append as is.
\1	returns the substring of the match that was enclosed by the first set of parentheses in the search pattern, as mentioned above
\2 or \3	returns the substring of the second or third match that was enclosed by the second or third set of parentheses in the search pattern, as mentioned above
<	returns a start of subfield marker
>	returns an end of subfield marker
<<	returns a single less-than ("<")
>>	returns a single greater-than (">")
\\	returns a single backslash

All other characters in a replacement pattern are taken literally.

Chapter 2

The Conversion Process

CONVERSION PARAMETERS

NewConvert -a<ruletable>[[-s<skip>]<input file>...][-c<output file>]] [-j<journal file>]]

WHERE

<ruletable>

Is the file of conversion rules.

<skip>

Is the number of records from the start of the following <input file> to skip before processing any records.

<input file>

Is an input MARC file. Any number of <input file>s may be listed, each with an optional “-s” switch. A missing <input file> or an <input file> of “-” redirects input from standard input.

“-c” switch

Redirects output to <output file>. If <output file> is “-” or if it is missing, standard output is used.

“-j” switch

Redirects the log file to <journal file>. If <journal file> is “-” or if it is missing, standard error output is used.

CONVERSION COMMANDS

Renaming the Output File

The example below will allow you to specify the output filenames, specific paths, and even devices in which you wish to have the converted files appear, as opposed to the default filenames is the name of the table used with the extension .cX for the output file and .jx for the journal file.:

```
NEWconvert -a<ruletable><inputfile> -c<output filename wished> -j<journal filename wished>
```

“Skipping” Problematic records in an Input File and Converting Remaining File

To do this, you must enter the record number to skip. If, for example, the 52nd record has caused the program to “crash”, the command:

```
NEWconvert -a<ruletable><input file> -s52 <input file>
```

Will skip records 1 to 52 and start the conversion at record number 53.

“Chaining” Together Input Files convert

The following command allows you to consecutively convert more than one input file within the same command file. For example, the catalog output file (t_marc.cX if rule table was t_marc.tbl) will contain converted records from all three input files below:

```
NEWconvert -a<ruletable><input file> <input file> <input file>
```

Chapter 3

Rule Table Structure

INTRODUCTION

The algorithm for deciding which rule table will be chosen when processing input files with the conversion program is discussed in **Chapter 2 The Conversion Process**. The following section acts as a reference guide to the contents of the rule tables.

The rule table is built of two types of rules:

- Static rules which define various parameters and modes of operation.
- Replacement, Expansion, and Dictionary Translation rules of the Catalogue Definition Grammar (CDG).

All rules are optional.

Static rules always apply because they are implicit to the conversion program; they have default values if they are absent from the table.

If none of the CDG rules are present, then the output file will contain exactly the same data as the input file, except that the format will be a human-readable file instead of a MARC communications format. Diacritic characters will have been translated from the ALA character set.

The comment delimiter is "#". All characters on a line following this delimiter are ignored as comments.

STATIC RULES

A rule table must be given to NEWconvert. This rule table includes different rules that will edit the record to be treated. Each static rule included in the table must exist on its own line. Comments may follow the rule, but no other rule can begin on the same line.

The rule table must have the .tbl extension and as a convention we start the name with t_ for example: t_convert.tbl.

Following is a detailed explanation of each static rule.

Informat

Indicates the physical format of the input file.

- 0 (default) MARC communication format
- 1 Human-readable format (Important : this is not the same as the flat format when extracting from Unicorn. It could be a file that has already passed through NEWconvert with an outformat=1).
- 2 Microlif format

Outformat

Indicates the desired format of the output file.

- 0 (default)MARC communication format
- 1 Human-readable format (this format should be used for verifications only, you cannot import data in this format).

Charset

This parameter is mandatory if you want to convert a file from a human-readable format to a file in MARC communication format. (If you have a INFORMAT=1 and OUTFORMAT=0, you need to put CHARSET=1). Otherwise, do not use the CHARSET parameter, the default value will be set accordingly to your INFORMAT and OUTFORMAT.

CHARSET = 0 or 1 or 2

0: To keep the same format as the input file.

1: To export records in the ALA character set.

2: To export records in the ISO (European) character set. (Only for European sites.)

CATALOG DEFINITION GRAMMAR (CDG)

The Catalog Definition Grammar (CDG) is composed of replacement rules (REPLACE), expansion rules (EXPAND), and the substitution dictionary (DICT).

Replacement Rules

The CDG replacement rules are defined by a single syntax. The basic function of a replacement rule is to replace an existing tag with zero or more new tags. Following this logic:

- A tag can be deleted by simply replacing it with zero new tags.
- A tag's content can be modified by replacing it with a tag having the same tag name and the modified contents.

For example, to merge subfields "\$a" and "\$h" of a 100 tag, the original tag (with the separate subfields "\$a" and "\$h") will be deleted and replaced by a new 100 tag (with merged subfields "\$a" and "\$h").
- A tag can be renamed by deleting the existing tag and replacing it with a tag having the desired tag name and the original contents. Further, the subfields of the original tag can be divided into multiple insertion tags.
- A new tag can be generated by "deleting" the NULL tag, "", (a non-existent tag which acts as a placeholder in a replacement rule) and replacing it with the new tag.

For example, a non-MARC tag entitled "Media" could be created by deleting the NULL tag, "", and replacing it with a "MEDIA" tag whose contents are generated based on certain positions in the 008 tag. The 008 itself would not be modified by this rule.

Code REPLACEMENT RULES in the rule table as follows:

```
REPLACE      {
  "<r_tag>"          {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
      }
  }
}
```

where :

- /<switch> may occur multiple times for a single <s_tag>
- the entire line beginning with <s_tag> may occur multiple times for a single <i_tag>
- <i_tag> (and its associated <s_tag> lines) may occur multiple times for a single <r_tag>
- <r_tag> (and its associated <i_tag>'s) may occur multiple times within a single REPLACE set
- REPLACE sets may occur multiple times within a rule table

<R_TAG>

```

REPLACE    {
  "<r_tag>"      {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
  }
}

```

An <r_tag> is a tag to be replaced in the record. It is expressed as a regular expression search pattern, and the associated replacement rule is executed once for each tag in the record that matches this pattern.

The record is searched for occurrences of this pattern, and the rule is executed for each match. For each match that results in the generation of a new <i_tag>, **the matched tag is deleted from the record** (that is, it is replaced). If <r_tag> is coded as the NULL tag, "", the rule is executed exactly once, and no tag is deleted upon completion.

Any <r_tag> after the first in a REPLACE set may be preceded by the word *else*, in which case the rule is executed only if the preceding rule failed to generate any new <i_tag>'s.

Internally, the conversion program views a tag of a record as a single string of characters, beginning with the tag name, followed immediately by the indicators, followed in turn by the tag data. Because of this, the regular expression in <r_tag> can include the tag's indicators, or even the tag data. For example, the <r_tag> "090" will match on all 090 tags in the record, but the <r_tag> "09010" will match only on 090 tags with indicators "10", and "09010.*<bREF" will match only on 090 tags with indicators "10" and a subfield \$b that begins with "REF".

The tag name "BANNER" matches on the MARC leader. Although it is not incorrect to use "BANNER" as an <r_tag>, it is considered "bad form", since it is the <i_tag> (see below), which will direct the new data to the MARC leader. In this

sense "BANNER" is a special case, since it is the only field of which the conversion program will allow only one occurrence in a record.

The changes associated with a given replacement rule, i.e. the matched <r_tag>s which are to be deleted and the new <i_tag>s which are to be inserted, do not take effect in the record until the current replacement rule has finished executing.

Furthermore, the changes associated with a given replacement rule, once applied, will be understood by the following replacement rules, which allows for step-by-step modification of the data, particularly useful for complex processing.

<I_TAG>

```
REPLACE    {
  "<r_tag>"      {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
      }
  }
}
```

An <i_tag> is a tag to be inserted. It is expressed as a regular expression replacement pattern, with respect to the associated <r_tag>.

In the case of a NULL <i_tag>, "", no new tag is inserted into the record, but the conversion program still goes through the motions of executing the rule. This can be used as a mechanism for logging the presence of some data of interest in the journal file without changing the record.

An <i_tag> of "BANNER" will replace the MARC leader.

Any <i_tag> after the first in each replacement rule can be preceded by the word "else", in which case NEWconvert will attempt to build the <i_tag> only if the preceding <i_tag> was not created.

<s_tag>, <s_pat>, <trim>, <i_pat> and /<switch>

```
REPLACE    {
  "<r_tag>"      {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
      }
  }
}
```

Together, these fields are used to build the contents of the <i_tag>. These fields always occur as a complete set, i.e., for each <s_tag>, there is one associated <s_pat>, one <trim>, one <i_pat>, and any optional /<switch>s.

<S_TAG>

```
REPLACE    {
  "<r_tag>"      {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
  }
}
```

An <s_tag> is a tag to be searched to build the new <i_tag>.

As with <r_tag>, <s_tag> contains the name of a tag expressed as a regular expression search pattern. For each <s_tag> in turn, the entire record is searched for matching tags.

An <s_tag> of "BANNER" matches on the bibliographic record leader, so that a pattern in the leader can be used to build other new tags.

Any <s_tag> after the first for a particular <i_tag> can be preceded by the word "else", in which case it will be searched for only if the preceding <s_tag> through /<switch> combination (i.e. the preceding line) found no matches.

<S_PAT>

```
REPLACE    {
  "<r_tag>"      {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
  }
}
```

An <s_pat> is a regular expression search pattern to be searched for in the data of <s_tag> to build the new <i_tag>.

Just as the entire record is searched for occurrences of <s_tag>, the data of a matched <s_tag> is searched for occurrences of the pattern in <s_pat>.

After each match is processed, the rest of the tag data (from the end of the match, onwards) is searched for more matches, if not instructed otherwise (see **switch/mocc**).

<TRIM>

```
REPLACE    {
  "<r_tag>"      {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
    }
  }
}
```

A <trim> is a string of characters, each of which is trimmed from the end of a matched pattern.

After a matched <s_pat> has been applied to <i_pat>, the last character in the string to be added to the new <i_tag> is compared to each character in <trim>. If the character is found, then it is deleted from the string. This is repeated until the last character in the string is not found in <trim>.

A <trim> is useful for removing trailing spaces from subfields, or redundant trailing punctuation.

<I_PAT>

```
REPLACE    {
  "<r_tag>"      {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
    }
  }
}
```

An <i_pat> is a pattern to be accumulated as the tag data for <i_tag>. It is expressed as a regular expression replacement pattern relative to <s_pat>. That is, each string of characters which is matched against <s_pat> is applied against the regular expression replacement pattern in <i_pat>. Trailing characters in the resulting string which are found in <trim> are deleted, and then the string is appended to the new <i_tag>.

<SWITCH>

```
REPLACE {
  "<r_tag>" {
    "<i_tag>":
      "<s_tag>" "<s_pat>" "<trim>" "<i_pat>" /<switch>
  }
}
```

A **<switch>** is a qualifier that overrides some default mode of operation for a **<s_tag>** line.

The following section illustrates the Replacement Rule Switches.

Replacement Rule Switches

The **/tocc** switches limit which tags in the record will successfully match against the **<s_tag>**, even after the pattern in **<s_tag>** is matched.

/TOCC=CURR (OR: TAG OCCURRENCE = CURRENT)

Allows the **<s_tag>** to match a tag in the record only if the occurrence number of the **<s_tag>** is the same as the current occurrence number of the **<r_tag>**.

E.g.: A bibliographic record contains three 090 tags. We will need to create three corresponding 852 tags. The contents of the first 090 will be deposited in the first 852, the second 090 in the second 852 and the third 090 in the last 852.

Data before treatment with the conversion program:

```
.
.
090 $first call number$bfirst location code
090 $asecond call number$bsecond location code
090 $athird call number$bthird location code
.
.
.
```

The rule created in the rule table:

```
"090"      {
#the 090 will become a new 852 by copying its content
"852":
"090"      ".*"      " "      "&" /tocc=curr
}
```

The results after running the conversion program with the previous rule:

```
.
.
.
852  $first call number$bfirst location code
852  $asecond call number$bsecond location code
852  $athird call number$bthird location code
.
.
```

Explanation:

Using the example shown, we need to ensure that when the <r_tag> matches on the first 090 tag in the record, the <s_tag> will only match on the first 090 tag in the record, and when the <r_tag> is matched on the second 090 tag, then the <s_tag> will only match on the second 090 tag, etc... . Without this switch, the <s_tag> would have matched on all three 090 tags in the record each time the <r_tag> matched on a record, so the result would have been:

```
.
.
.
852  $first call number$bfirst location code$asecond call number$bsecond
location code$athird call number$bthird location code
852  $first call number$bfirst location code$asecond call number$bsecond
location code$athird call number$bthird location code
852  $first call number$bfirst location code$asecond call number$bsecond
location code$athird call number$bthird location code
.
```

Each time an <r_tag> matches against a tag in the record, its occurrence number is incremented. Similarly, each time an <s_tag> matches against a tag in the record, its own occurrence number is incremented. Note, then, that the /tocc=curr switch does not necessarily ensure that the same tag matched by <r_tag> will be the one matched by <r_tag> **unless the pattern in each is the same in the <r_tag> and the <s_tag>**.

/TOCC=FIRST (OR: TAG OCCURRENCE = FIRST)

Allows the <s_tag> to match only on the first tag in the record that matches its pattern.

E.g.: A bibliographic record contains the price in tag 020 subfield \$c. This needs to be copied to the tag 852, subfield \$2. In case where multiple 020's exist, take the price from the first occurrence of 020 that matches.

```
""      {
        "852":
        "020" "<c([>]+)" " "          "<2\1" /tocc=first
      }
```

/TOCC=NOTFIRST (OR: TAG OCCURRENCE NOT = FIRST)

Limits the process to all occurrences of <s_tag> that match the pattern except to the very first one.

E.g.: A bibliographic record may contain duplicate 1XX tags. We would move all 1XX tags, except the first, to 7XX.

Data before treatment with the conversion program:

```
.
.
100  $first author,$b(personal)
100  $asecond author,$b(personal)
110  $athird author,$b(corporate)
.
.
```

The rule created in the rule table:

```
"1(..)" {
    "7\1":
    "1.." ".*" " " "&" /tocc=notfirst
}
```

The results after running the conversion program with the previous rule:

```
.
.
100 $first author,$b(personal)
.
.
700 $asecond author,$b(personal)
710 $athird author,$b(corporate)
.
.
```

/TOCC=LAST (OR: TAG OCCURRENCE = LAST)

Limits the process to the last occurrence of <s_tag> that matches the pattern.

E.g.: A bibliographic record contains more than one 050 tag. We are asked to take the last 050 and place it in tag 099.

```
"" {
    "099":
    "050" ".*" " " "&" /tocc=last
}
```

/MOCC=CURR (OR: MATCH OCCURRENCE = CURRENT)

Limits the process of the match of <s_pat> to the matched occurrence of the replace <r_tag>. The /mocc switches limit which matches of <s_pat> will be considered successful.

E.g.: A bibliographic record contains a 049 tag in which one subfield with multiple volume information appears as follows:

049 \$dv.1, v.2, v.3. We wish to copy this information to match three corresponding 852 item tags. (We assume in the following example the required 852 tags have already been created.)

```
"852" {
    "852":
    "852" ".*" " " "&" /tocc=curr
    "049" "v\[.,,]+ " " "<j&" /mocc=curr
}
```

/MOCC=FIRST (OR: MATCH OCCURRENCE = FIRST)

Limits the process of the match of <s_pat> to its first occurrence

E.g.: A bibliographic record contains an 020 tag with more than one price (found in subfield \$c). We wish only the first occurrence of \$c price added to the tag 852 subfield \$2. (We assume in the following example the required 852 tags have already been created and are ready to receive a price, i.e. no other information will be added to the tag.)

```
"852" {
    "852":
    "852" ".*" " " "&" /tocc=curr
    "020" "\$( [0-9]+ )\.( [0-9]+ )" " " "<2\1\2" /mocc=first
}
```

/MOCC=NOTFIRST (OR: MATCH OCCURRENCE = NOT FIRST)

Limits the process of the match of <s_pat> to each occurrence of the replace <r_tag> except the first match

E.g.: When building a tag, multiple subfield \$z's have been created during manipulation. We need to concatenate these \$z's within the same subfield \$z.

```
"852" {
    "852":
    "852" "<[^z][^>]+" " " "&" /tocc=curr
    "852" "<z([^>]+)" " " "&" /tocc=curr/mocc=first
    "852" "<z([^>]+)" " " " \1" /mocc=first/tocc=curr
}
```

/MOCC=LAST (OR: MATCH OCCURRENCE = LAST)

Limits the process of the match of <s_pat> to its last occurrence.

E.g.: A bibliographic record contains a 020 tag with more than one price (found in subfield \$c). We wish only the last occurrence of \$c price added to the tag 852 subfield \$2. (We assume in the following example the required 852 tags have already been created and are ready to receive a price, i.e. no other information will be added to the tag.)

```
"852" {
    "852":
    "852" ".*" " " "&" /tocc=curr
    "020" "\${[0-9]}\.([0-9]+)" " " "<2\1\2" /mocc=last
}
```

/UPCASE

After a match has been made, ensures that all characters in <i_pat> are in uppercase.

E.g.: Source data contains location codes in both upper and lower case in 852 \$z. We will place these location codes in the tag 852 subfield \$c and then ensure they all appear in uppercase characters in the <i_pat>.

```
"852" {
    "852":
    "852" "<[^c][^>]+" " " "&" /tocc=curr
    " " " " " " "<c"
    "852" "<z([^>]+)" " " "\1" /tocc=curr/upcase
}
```

/IGNCASE

When searching the data of <s_tag> for the pattern in <s_pat>, ignore case. Output copied from <s_pat> will all appear in uppercase, but lowercase literal characters in <i_pat>, or lowercase characters returned from the substitution dictionary will remain lowercase.

E.g.: Data before treatment with the conversion program:

```
.  
.   
.   
852          $hReference  
.   
.   
.
```

The rule created in the rule table:

```
"852"  {  
      "852":  
      "852"  "<h(ref)"  ""  "<c\1"  /ignore/tocc=curr  
      }  
.   
.   
.
```

The results after running the conversion program with the previous rule:

```
.  
.   
.   
852          $cREF  
.   
.   
.
```

/REJECT=REC

If the <s_tag>, <s_pat> and any other switches result in a successful match, then this record will be sent to the journal file instead of the regular output file.

E.g.: We want to ensure that no serial records are included in the load file. We will identify serials as records with "s" in leader position 7.

```
""      {
        "":
        "BANNER""^.....s" "" "" /reject=rec
      }
```

/REJECT=RULE

If the <s_tag>, <s_pat> and any other switches result in a successful match, then the current replacement rule (ie., everything controlled by the current <r_tag>) is abandoned.

Since the changes associated with the current rule do not take effect until it has successfully completed execution (see the definition of <r_tag>), changes already caused by previous matches to the current <r_tag> are also abandoned. No further attempts to match tags in the record against the current <r_tag> will be made.

/REJECT=ITAG

If the <s_tag>, <s_pat> and any other switches result in a successful match, then the new <i_tag> being built is abandoned.

/NODUPS

Only one instance of each unique <i_pat> will be appended to the new <i_tag>, that is no duplicate.

E.g.: Suppose we have some locally defined category that we have coded in tag 590 subfield \$a that we will store as a local characteristic. Suppose also, that the existing system was judged to be too specific, so we will use fewer codes and therefore, several of the original codes may map to a single local characteristic. If a record can have multiple 590 tags (i.e., a record can fall into several categories), then we can easily map the old code into a local characteristic with the translation dictionary, and use the /nodups switch to ensure that each local characteristic will appear only once for each record:

```
""      {
        "ZONEN*":
        #The local characteristic field is coded "ZONEN*"
        "590"    "<a([>]+)" "" \1"/dict/nodups
      }
```

/LOG=()

If the <s_tag>, <s_pat> and any other switches result in a successful match, then the message declared by the /log switch will be written to the journal file (e.g.: t_marc.j1). The syntax is as follows:

```
/log="( <message> ", <variable> )
```

Where:

- There must be at least 1 occurrence of <message>, which is expressed in the syntax of the formatted message of the "printf" function of the C programming language. Specifically:
- String variables are represented in the text of <message> by %s. Each occurrence of <variable> is substituted in turn for occurrences of %s in <message>. The C minimum and maximum string length options are allowed (i.e., /%<min>.<max>s).
- The C character constants :

```
\n    produce a newline (ie., a linefeed)
\b     produce a backspace
\f     produce a formfeed
\r     produce a carriage return
\t     produce a horizontal tab
\v     produce a vertical tab
```

- The double-quote character, ", must be escaped by a backslash, i.e., \", to distinguish it from the double quote characters delimiting "<message>".
- There must be one occurrence of <variable> preceded by a comma, for each %s embedded in <message>. The available string variables are:

TAG: The name of the current tag matched against <s_tag>.

INDICATORS: The indicators of the current tag matched against <s_tag>.

TEXT: The text of the current tag matched against <s_tag>.

MATCH: The matched string applied to <i_pat>.

E.g.: To determine whether the Christian name of a personal name heading is being coded in subfield \$h (an obsolete coding practice), we can build a rule to report occurrences of subfield \$h in tags 100, 600 and 700 as follows:

```

" "      {
#Delete no tag
" ":
#Add no new tag
"[167]00"<h([>]+)" " "\1"
/log=( "%s: subfield h in tag %s: \"%s\" \"n\",numenot , tag, match)
}

```

In the above example, there are three occurrences of %s in <message>. The first is replaced by numenot value (record number) , the second is replaced by the tag matched to <s_tag>, either 100, 600 or 700 (tag) and the third is replaced by the \$h subfield in the <i_pat>, (match). The last of these is surrounded by escaped double quotation marks, so that the message in log file will have the \$h subfield in quotation marks.

E.g.: To simulate the "old" *LIS*tape journal file:

```

REPLACE  {
#Open the Replace rule

#This rule simulates the JRN rule of the existing LIStape
# i.e., for each record that is processed, a journal entry is
#made in the following format: a record number (preceded
# by "N") starts the line, followed by the 001 tag, if one exists
# (maximum of 11 characters), followed by up to 25 characters

```

of tag 245 subfield \$a. This information is double-spaced

```
"" {
# No tag to delete

"":
# No tag to add or create

"" "" "" "" /log=(" N%s ",numenot)
# Start with the record number

"001" "" "" "" /log=(" %11.11s ",TEXT)
# Log tag 001

else "" "" "" "" /log=(" ")
# Or else enter spaces if no tag 001 exists.

"245" "<a([>]+)"",./: " " \1"/log=(" %.25s\n\n",MATCH)
# Use the title (25 char.) from tag 245 subfield a
}

# Close the Replace rule
}
```

Note : In the log messages, the notation %11.11s or %.25s indicates the length of the string that we want to print out. The value before the "." is the minimum length of the string and the one after is the maximum. For example if we had %11.25, it would mean that we want a string that would have a minimum length of 11 characters and a maximum of 25.

/S_SUBSTR=()

Limits the text of string to be searched for <s_pat> to a substring of the tag's text. The syntax is as follows:

```
/s_substr="<pattern>"
```

Where <pattern> is expressed as a regular expression search pattern. When <s_tag> is matched against a tag in the record, the text of the tag is matched against <pattern>, and the resulting match is the string in which <s_pat> is searched for. If <pattern> contains parentheses to define a substring, then only the first substring is searched for <s_pat>. After looping over all occurrences of <s_pat> in a matched substring, <pattern> is searched for again in the rest of the tag's text until the text is exhausted.

E.g.: We would like to extract all prices in 020 \$c into a note field (e.g. 500 \$a). We need to restrict our search to subfield \$c, but if we include the subfield delimiters in <s_pat>, then we restrict the match to either the first or last occurrence of a price in the subfield. To loop over all prices listed in a single subfield \$c, we can move mention of the subfield delimiters out of <s_pat> and into /s_substr, as follows:

```
" " {
    "500":

    # All prices will go into the same $a subfield, so start
    # the subfield independent of any prices matches
    "" "" "" "<a"

    # We identify a price as a dollar sign followed by one or
    # more digits followed by a decimal point, followed by
    # exactly 2 digits.

    "020" "$[0-9]+\.[0-9][0-9]" "" "&" /s_substr="<c([>]+)"
}
```

The parentheses in <pattern> of /s_substr mean that only the contents of the \$c subfield will be searched for occurrences of prices. Each occurrence of subfield \$c in the 020 tag will be looped over, in turn.

Note that /s_substr is just a quicker alternative to first extracting the desired substring into its own temporary tag, and then searching for the patterns of interest in two separate replacement rules.

/I_SUBSTR=()

Reduces <i_pat> to a substring of the complete inserted string. The syntax is as follows:

```
/i_substr="<pattern>"
```

Where <pattern> is expressed as a regular expression search pattern.

Before <i_pat> is appended to the new <i_tag>, it is matched against <pattern> defined in */i_substr*. If <pattern> contains parentheses to define a substring, then only the first substring is appended to the new <i_tag>; otherwise, the entire string matched by <pattern> is used.

E.g.: We have decided that the original 008 tags in our records are unreliable, so we are building new ones with information found in other tags of the record. Suppose that we have accumulated this new information in a temporary tag called NEW008*. We want to ensure that the new 008 tag is padded to 40 characters with trailing spaces. The length of NEW008* will depend on how much of the information we need was found in the rest of the record. Thus, we can guarantee that the new tag will be exactly 40 characters by padding NEW008* with an extra 40 spaces, and then extracting only the first 40 characters of the new string, as follows:

```
"NEW008*" {  
"008":  
"NEW008*" ".*" "" "& "  
/i_substr="^....."  
}
```

Note that */i_substr* is just a quicker alternative to first building the larger string in a temporary tag, and then extracting the desired substring into the final tag in two separate replacement rules.

Substitution Dictionary

The substitution dictionary is used to translate one character string into another. The original string can potentially be deleted altogether if its translation is the NULL string, "".

The syntax for building a translation dictionary is as follows:

```
DICT {  
    "<s_string>" "<r_string>"  
}
```

where:

- "<s_string>" "<r_string>" pairs can occur multiple times in a single DICT set.
- DICT sets may occur multiple times within a rule table.

<S_STRING>

An <s_string> is a string to be matched to the <i_tag>. It is expressed as an absolute string of characters, not as a regular expression, so the two strings must match exactly.

<R_STRING>

An <r_string> is a string with which to replace a matched <s_string>. If <r_string> is the NULL string, "", then the searched string is simply deleted.

Specifically, if an <i_pat> in a replacement rule is qualified with the **/dict** switch, then the <i_pat> is searched for as an <s_string> in the dictionary. If it is found, then the <r_string> replaces the <i_pat> in the new tag being built by the replacement rule. As mentioned, if <r_string> is the NULL string, "", then the original string is simply deleted, that is, nothing is added to the new <i_tag> being built for the current <i_pat>. This is a useful method of implementing a stopword list.

If the <i_pat> is not found as an <s_string> in the substitution dictionary, then the original <i_pat> is appended to the new tag.

The /dict will translate a known search <s_pat> pattern into a known output using the substitution dictionary table. That is, if the <i_pat> is found in the substitution dictionary, then its substitution from the table is appended to the new <i_tag> instead of the original <i_pat>. If the <i_pat> is not found in the substitution dictionary, then it is just appended to the new <i_pat> as usual.

If the <i_pat> is found in the substitution dictionary and if its substitution is the NULL string, "", then the <i_pat> is disregarded and nothing is appended to <i_tag>.

E.g.: In this example, we are building the tag 852 from the contents of tags 090. First, we create the subfield \$bMAIN for all 852 tags. Second, we use a dictionary rule for translating the old contents of subfield \$f from the tag 090 to the new contents of subfield \$c for the new tag 852. At the same time, we add another subfield, the circulation category, \$1 for all items whose category is X or C.

```

REPLACE {
#Open the REPLACE rule

"090" {
  "852":
  "" "" "" "<bMAIN"
  "090" "<f([^\>]+)" "" "<c\1" /dict/tocc=curr
}

#Close the REPLACE rule
}

DICT {
#Open the DICTIONARY rule

"<cact" "" "<cGENER<1P"
"<cmain" "" "<cGENER<1P"
"<creference" "" "<cREF<1P"
"<caudiovisual""<cAV<1P"

# Close the DICTIONARY rule
}

```

Expansion Rule

The following is an example on how to use the EXPAND rule. The purpose is to separate multiple holdings within the same tag, e.g. 949, and create a temporary tag (~949) that will be used later to generate multiple 852 tags (item tags).

The original data before the EXPAND rule:

.
. .
.
949^^\$a30296002001970\$bwmr\$g127.00\$v2\$a30296002001988\$bwmr
&\$g127.00\$v3\$a30296002001996\$bwmr\$g127.00\$v4
. .
.

The EXPAND rule to use:

EXPAND {

#949 \$a has to exist once (1 occurrence)

#For each subfield other than 949 \$a, e.g. \$b, \$g, etc.

#the conversion program must encounter 0 or 1 occurrences.

#If no \$a exists, the EXPAND rule will not apply.

```
"949"          {  
  "~949":  
  1            "a"  ".*"  "&"  
  0-1          "b"  ".*"  "&"  
  0-1          "g"  ".*"  "&"  
  0-1          "m"  ".*"  "&"  
  0-1          "v"  ".*"  "&"  
  0-1          "l"  ".*"  "&"  
  0-1          "c"  ".*"  "&"  
  0-1          "n"  ".*"  "&"  
  }  
}
```

The conversion results after using the EXPAND rule:

.
. .
. .
~949 \$a30296002001970\$bwkmr\$g127.00\$v2
~949 \$a30296002001988\$bwkmr\$g127.00\$v3
~949 \$a30296002001996\$bwkmr\$g127.00\$v4

Chapter 4

Conversion Error Messages

*** NEWconvert: conversion parameters

Application: XXXXXXXX data

Rule table: XXXXXXXX

Record loading mode: X

- Records will be built according to MARC tag delimiters in the record body
- Records will be built according to the MARC directory
- NEWconvert: /I_SUBSTR cannot be NULL, "", in file NEWconvert.tbl
- NEWconvert: /S_SUBSTR cannot be NULL, "", in file NewConvert.tbl
- NEWconvert: <s_subfld> must be a single character in file NewConvert.tbl
- NEWconvert: DICT replacement word cannot be NULL, "", in NewConvert.tbl
- NEWconvert: ELSE condition at beginning of list in file NewConvert.tbl
- NEWconvert: EXPAND fields cannot be NULL, "", in file NewConvert.tbl
- NEWconvert: NULL, "", r_tag has no current occurrence
- NEWconvert: attempt to define same leader position in file NewConvert.tbl
- NEWconvert: bad leader position number in file NewConvert.tbl
- NEWconvert: bad or missing argument
- NEWconvert: bad regular expression in NewConvert.tbl
- NEWconvert: bad switch in file NewConvert.tbl
- NEWconvert: cannot allocate memory for character string in file NEWconvert.tbl
- NEWconvert: cannot allocate memory for rule in NewConvert.tbl
- NEWconvert: cannot open XXXXXXXXXX for reading

- NEWconvert: cannot open journal file XXXXXXXXX
- NEWconvert: cannot open output file XXXXXXXXX
- NEWconvert: could not open a conversion rule table
- NEWconvert: duplicate DICT target word (%s) in file NEWconvert.tbl
- NEWconvert: expecting %c in file NewConvert.tbl
- NEWconvert: expecting number of subfield occurrences in file NewConvert.tbl
- NEWconvert: expecting quote character (") at end of XXXXXX in NewConvert.tbl
- NEWconvert: garbage in file NewConvert.tbl
- NEWconvert: illegal value for rule (X) in file NewConvert.tbl
- NEWconvert: incomplete log message in NewConvert.tbl
- NEWconvert: insufficient memory to add subfield XXXXXXXXXX to tag XXX
- NEWconvert: invalid data type
- NEWconvert: mismatched variables in log message
- NEWconvert: only one input file may be redirected from standard input
- NEWconvert: premature end-of-file within REPLACE set in file NewConvert.tbl
- NEWconvert: premature end-of-rule in file NewConvert.tbl
- NEWconvert: processing terminated at input file "XXXXXXXXXX"
** cannot open file for input
- NEWconvert: record number became negative at XXXXXXXX
- NEWconvert: tag XXX too large to add subfield XXXXXXXXXX
- NEWconvert: trailing garbage on line in file NewConvert.tbl
- USAGE: NEWconvert -a<ruletable> [-vvv] [[-s<skip>] <infile> ...] [-c[<loadfile>]] [-j[<journal>]]

where:

-<ruletable> is the conversion table to edit the records, and as a convention we name it by beginning with t_XXX.tbl (it must have a .tbl extension) : e.g. : t_tableconv.tbl. Also note that you must type in the complete name of the table with the extension.

"-v" causes only *fatal* problems to be reported to the journal file.

"-vv" (default) adds to the report problems which were successfully corrected.

"-vvv" adds regular expression diagnostics.

-<skip> is the number of records from the start of the following **<infile>** to skip before processing any records.

-<infile> is an input MARC file. Any number of **<infile>**'s may be listed, each with an optional **"-s"** switch. A missing **<infile>** or an **<infile>** of **"-"** redirects input from standard input.

"-c" switch redirects output to **<loadfile>**. If **<loadfile>** is **"-"** or if it is missing, standard output is used.

"-j" switch redirects the journal file to **<journal>**. If **<journal>** is **"-"** or if it is missing, standard error output is used.

- XX-XXXXXXXX: bad item number format - deleting tag: XXX
- error closing input file XXXXXXXXXXXXX
- record XXXXXXXX, tag XXX, subfld X: illegal # of subfield values
- record XXXXXXXX, tag XXX, subfld X: XXX is a bad /abs value
- record XXXXXXXX, tag XXX, subfld X: XXX is a bad /qty
- record XXXXXXXX, tag XXX, subfld X: XXX is a bad /range
- record XXXXXXXX, tag XXX, subfld X: illegal # of occur. of subfld X
- record XXXXXXXX, tag XXX, subfld X: illegal size for /allnums in subfld X
- record XXXXXXXX, tag XXX: END-OF-FILE in indicators
 - truncating record at this tag
- record XXXXXXXX, tag XXX: END-OF-RECORD in indicators
 - truncating record at this tag

- record XXXXXXXX, tag XXX: END-OF-RECORD in indicators
-truncating record at this tag
- record XXXXXXXX, tag XXX: END-OF-TAG mark in indicators
-nuking tag
- record XXXXXXXX, tag XXX: END-OF-TAG mark in indicators
-nuking tag
- record XXXXXXXX, tag XXX: first subfield found in indicators
-using spaces to pad indicators
- record XXXXXXXX, tag XXX: first subfield found in indicators
-using spaces to pad indicators
- record XXXXXXXX, tag XXX: premature END-OF-RECORD
-nuking empty tag XXX
- record XXXXXXXX, tag XXX: premature END-OF-RECORD
-nuking empty tag XXX
- record XXXXXXXX, tag XXX: reported offset in directory `XXXXX, actual offset `XXXXX
- record XXXXXXXX, tag XXX: reported size of tag `XXXX, actual size `XXXX
- record XXXXXXXX, tag XXX: tag deleted due to insufficient memory
- record XXXXXXXX, tag XXX: tag does not end with a MARC end-of-tag
- record XXXXXXXX, tag XXX: truncate to XXXX characters from XXXX characters
- record XXXXXXXX, tag XXX: truncated to XXXX characters from XXXX characters
- record XXXXXXXX: END-OF-FILE found in MARC directory
- record XXXXXXXX: END-OF-RECORD found in MARC directory
- record XXXXXXXX: bad char at leader position XX (X)
-characteristic-syncing start-of-record
- record XXXXXXXX: corrupt directory
-rejecting record
- record XXXXXXXX: no end-of-record marker immediately after last tag in directory

- record XXXXXXXX: no end-of-record marker immediately after last tag in directory
- record XXXXXXXX: rejecting record due to corrupt directory
- record XXXXXXXX: reported size of record `XXXXXX` actual size `XXXXXX`
- record XXXXXXXX: tag XXX truncated to XXXX from XXXX chars
- skipping first XXX records

Chapter 5

Sample Bibliographic Record Before and After Conversion

BEFORE

00882pam 2200277

450000100110000000500170001100800420002801000130007002000150

0083035001700098039001800115040001800133049000900151050002400160082001800
1840900046002021000020002482450093002682600053003613000035004145040039004
49650002400488650001600512650004400528700003200572^^01-
0000005^^19911029141204.0^^840328s1984

njua b 00110 eng ^^ ^_a84006939^^ ^_a0138512701^^
^_aocm10696501^^0 ^_a2^_b3^_c3^_d3^_e3^^ ^_aDLC^_cDLC^_dSMU^^
^_aSMUU^^0 ^_aHD30.28^_b.H3881984^^0 ^_a658.4/012^_219^^
^_aHD30.28^_b.H388 1984^_mSMUU^_132922000000056^^10

^_aHAX, Arnolddo C.^^10^_aStrategic management :^_ban integrative perspective
/^_cArnolddo C. Hax, Nicolas S. Majluf.^^0 ^_aEnglewood Cliffs, N.J. :^_bPrentice
Hall,^_cc1984.^^ ^_axvii, 468 p. :^_bill. ;^_c24 cm.^^ ^_aIncludes bibliographies
and index.^^ 0^_aCorporate planning.^^ 0^_aManagement.^^10^_aNations
Unies^_xPolitique internationale^^10^_aMajluf, Nicolas S.,^_d1945-^^^]

AFTER

^LN01-1234567

00882pam 2200277 4500

001 01-0000005

005 19911029141204.0

008 840328s1984 njua b 00110 eng

010 \$a84006939

020 \$a0138512701

035 \$aocm10696501

0390 \$a2\$b3\$c3\$d3\$e3

040 \$aDLC\$cDLC\$dSMU

049 \$aSMUU

0500 \$aHD30.28\$b.H388 1984

0820 \$a658.4/012\$219

090 \$aHD30.28\$b.H388 1984\$mSMUU\$i32922000000056

10010\$aHax, Arnaldo C.

24510\$aStrategic management :\$ban integrative perspective /\$cArnaldo C. Hax,
Nic

&olas S. Majluf.

2600 \$aEnglewood Cliffs, N.J. :\$bPrentice Hall,\$cc1984.

300 \$axvii, 468 p. :\$bill. ;\$c24 cm.

504 \$aIncludes bibliographies and index.

650 0\$aCorporate planning.

650 0\$aManagement.

65010\$aNations Unies\$xPolitique internationale

70010\$aMajluf, Nicolas S.,\$d1945-

Chapter 6

Conversion Catalogue

Definition Grammar

Examples

EXAMPLES OF THE REPLACE RULE USED IN SIMPLE REPLACEMENT PATTERNS

1. Move the contents of field 001 to tag 035, subfield "\$a":

```

REPLACE      {
"001"        {
              "035":
"001"        ".*"          ""      "<a&"
              }
            }

```

2. Copy the contents of tag 001 into 035 (keeping a copy of the 001):

```

REPLACE      {
"001" {
"001":
"001"        ".*"          ""      "&"
"035":
"001"        ".*"          ""      "<a&"

```

```

    }
}

```

3. Multiple search tags, s_tags, (050, 090, etc.) to be used to create an insert, i_tag (852). In this example we assume there is only one 050 and one 090 in the record. We also assume no 852 tag has yet been created:

```

REPLACE {
    " " {
        "852":
            #take the location code from 090 subfield a and insert it into 852 $b
            "090"      "<a[^\>]+'"      " "      "<b\1"
            #call number is found in 050 subfield $a and will be inserted in 852 $h
            "050"      "<a[^\>]+'"      " "      "<h\1"
            #remaining part of call number is appended to $h above
            "050"      "<b[^\>]+'"      " "      "\1"
            #take all numbers from the 090 but leave out any "d pn" which precede
            #the numbers and put into subfield 852 $p
            "090"      "<xdpn([0-9]+x?)"  " "      "<p\1"
    }
}

```

4. Remove the slashes and replace them by spaces in the 090 field:

```

REPLACE {

    "090" {
        "090":
            "090" "([^\>/]+)/"      " "      "\1" /s_substr="<a[^\>]+>/tocc=curr
        else  "090" "<a[^\>]+'"      " "      "&" /tocc=curr
            "090" "/([^\>/]+)>"      " "      "\1" /s_substr="<a[^\>]+>/tocc=curr

        #Keep subfields other then $a's and $d's as is
            "090" "<[^\>ad][^\>]+'"      " "      "&" /tocc=curr
    }
}

```

```

#Separate prefixes and suffixes out of volume ranges in $d
"090" "<d([0-9/A-Za-z\ . ]+)([ \. ])?([0-9,-]+)" ""
"<p\1\2<d\3"/tocc=curr
else "090" "<d([0-9]+-[0-9]+)+([A-Za-z\ . ]+)" "" "<d\1<S\2"
/tocc=curr
else "090" "<d[">]+ " " " "&" /tocc=curr
}

}

```

EXAMPLE OF THE USES OF REPLACE AND EXPAND RULES

1. Expand 949 \$z111\$z112\$z113, into : 852 \$p111, 852 \$p112, and 852 \$p113.

```
EXPAND {
  "949" {
    "852":
      1   "z"   "[^>]+"   "\1"
    }
  }
REPLACE {

  "852" {
    "852":
      "852" "<([>]+)" "" "<p\1" /tocc=curr
    }
  }
}
```