Guide to USEMARCON

Creating and editing the system
files and rules files

Version 1.12
27 November 2001

Contents

Introduction *3*

CHAPTER 1 : USEMARCON System Files *4*

Initialization File (.ini) **4**
Format 5

Format Checking Table (.chk) **6**
Format 7

Character Set Translation Table (.trs) **8**
Format 8

Configuration File (.cnf) **10**
Format and Example 10

Rules Files (.rul) **11**

CHAPTER 2 : Creating USEMARCON rules *12*

Basic rule construction **12**

Some notes about 'S' **13**

Use of temporary fields **14**

Instructions **14**
Strings 15
Date and time variables 16
Conditions 16
Exists 17
NextSub 17
Precedes and Follows 18
Replace 18
Delete 19
ReplaceOcc 19
Len 20
From, To, Between 20
Strict 21
BFirst, EFirst, BLast, ELast 21
Lower, Upper 22
Val 22
Adding data from one subfield to the end of another 22
Memory instructions 23
Redo 23

Management of occurrences **23**

Comments **25**

Lookup Tables **25**
Format 26

CHAPTER 3 : The Error log file *29*

Overview **29**

Error Definitions **29**

Error Messages **30**

Introduction

This is a guide to the USEMARCON system files. It explains what the various files are used for and how to create the rules files which are required to carry out conversions from one MARC format to another.

It duplicates much of the information given in the USEMARCON Technical Manuals for both the Unix and Windows versions. The Technical Manuals are based on the one produced for the original GUI version of the software which was the final result of the USEMARCON Project, which was really written for programmers

This guide is written in simpler language and is aimed at non-programmers who wish to create USEMARCON rules and run conversions. Importantly, it also includes details on creating rules which utilise functions which were not available in the original application. This information is not included in the Technical manuals.

The examples given in the text are taken from a variety of conversions, not just from the sample files which are available in the download from the Web pages.

CHAPTER 1 : USEMARCON System Files

This section describes the format of the initialisation file, or .ini file and the formats for each of the system files which may loaded via the .ini file. Details of how to create the conversion rules are given in the next chapter.

All the files are simple text files which may be created using a text editor such as MS Windows Notepad, or Gedit in Linux.

Initialization File (.ini)

The non-GUI versions of USEMARCON may be run in 'batch' mode or in 'on-the-fly' mode. Batch mode is used to process a specified input file of MARC records e.g. to convert the file from one MARC format to another. 'On-the-fly' mode may be used within another application such as a Z39.50 client which is set up to automatically convert a selected MARC record from its orignal format to another.

The USEMARCON program uses a file to store a set of system files and parameters which enable the user environment and system information to remain uniform between sessions. This information is held in the initialization, or .ini file. This file is used in all versions of USEMARCON running in batch mode and as such, it must be included in the command line when launching the application. For example, to launch the USEMARCON NT application from your root directory and run the conversion from UNIMARC to UKMARC the following command line may be used:

```
c:\usemlib\uni2uk\usemarcon.exe c:\usemlib\uni2uk\uni2uk.ini
```

When setting up a Windows shortcut for the GUI version of USEMARCON a command line like the one above must be specified in the Target box of the Properties window.

When running the non-GUI version, if you are already in the c:\usemlib\uni2uk\ directory you would simply type:

```
usemarcon.exe c:\usemlib\uni2uk\uni2uk.ini
```

Note that even though you are launching the application form the same directory that holds the .ini file, you still need to include the the whole path to the .ini file in the command.

When running the Linux version (for which there is no GUI interface), if you put the .ini file and the usemarcon executable in the same directory you would navigate to that directory and type:

```
usemarcon convert.ini
```

Note that under Linux the whole path to the .ini file is not necessarily required.

The values contained in the .ini file may be modified in the "Initialization parameters of the .INI File" dialog box, which is accessed via the Options Menu in the Windows GUI version, or the file may be opened up and edited using a simple text editor.

In the .ini file you specify the full path and name of each of the files you wish to load in order to carry out the conversion. The files which are loaded via the .ini file are as follows:

The rules file (.rul)
The character set translation table (.trs)
The checking table for the input file(.chk)
The checking table for the output file
The input file
The output file
The configuration table for the input file (.cnf)
The configuration table for the output file
The error log file (.txt)

The input, output and error log file names are mandatory; all the other files may be used as needed. For example, you may simply specify a character set translation table if you wish to use USEMARCON to change the character set of a MARC file. Alternatively, if you wish to check the integrity of a MARC file you chould simply specify a checking table in either input or output.

Format

The .ini file has a typical MS Windows structure. The .ini file for the Windows versions would look something like this:

```
[DEFAULT_FILES]
ErrorLogFile=c:\usemlib\uk2us\errlog.txt
RuleFile=c:\usemlib\uk2us\uk2us.rul
TranscodingCharacterTable=c:\usemlib\uk2us\uk2us.trs
InputFormatCheckingTable=c:\usemlib\uk2us\uk.chk
OutputFormatCheckingTable=c:\usemlib\uk2us\us.chk
MarcInputFile=c:\usemlib\uk2us\testrecs\infile.mrc
MarcOutputFile=c:\usemlib\uk2us\testrecs\outfile.mrc
InputMarcEditConfigurationFile=c:\usemlib\uk2us\uk.cnf
OutputMarcEditConfigurationFile=c:\usemlib\uk2us\us.cnf

[DEFAULT_MARC_ATTRIBUTES]
IsInputBlockSegmentedChecked=
IsInputLastBlockPaddedChecked=
InputMarcSizeBlock=2048
InputMarcMinDataFree=5
InputMarcPaddingChar=5E

IsOutputBlockSegmentedChecked=FALSE
IsOutputLastBlockPaddedChecked=FALSE
OutputMarcSizeBlock=2048
OutputMarcMinDataFree=5
OutputMarcPaddingChar=5E

[DEFAULT_VALUES]
MaxErrorsToBeEncoutered=2000
```

```
OrdinalNumber=

[DEFAULT_STATES]
IsInteractiveExecutionModeChecked=
IsVerboseExecutionModeChecked=
IsShowCommandsActivated=

[DEBUG]
IsDebugExecutionModeChecked=
```

The .ini file for the Linux version would look something like this:

```
[DEFAULT_FILES]
ErrorLogFile=/home/mydir/usemarcon_batch/runtime/errlog.txt
RuleFile==/home/mydir/usemarcon_batch/runtime/uni2uk.rul
TranscodingCharacterTable==/home/mydir/usemarcon_batch/runtime/iso2blec.trs
InputFormatCheckingTable==/home/mydir/usemarcon_batch/runtime/uni.chk
OutputFormatCheckingTable==/home/mydir/usemarcon_batch/runtime/uk.chk
MarcInputFile==/home/mydir/usemarcon_batch/runtime/UNI.MRC
MarcOutputFile==/home/mydir/usemarcon_batch/runtime/uk.mrc
InputMarcEditConfigurationFile==/home/mydir/usemarcon_batch/runtime/uni.cnf
OutputMarcEditConfigurationFile==/home/mydir/usemarcon_batch/runtime/uk.cnf

[DEFAULT_MARC_ATTRIBUTES]
IsInputBlockSegmentedChecked=
IsInputLastBlockPaddedChecked=
InputMarcSizeBlock=2048
InputMarcMinDataFree=5
InputMarcPaddingChar=5E

IsOutputBlockSegmentedChecked=FALSE
IsOutputLastBlockPaddedChecked=FALSE
OutputMarcSizeBlock=2048
OutputMarcMinDataFree=5
OutputMarcPaddingChar=5E

[DEFAULT_VALUES]
MaxErrorsToBeEncoutered=19999
OrdinalNumber=

[DEFAULT_STATES]
IsInteractiveExecutionModeChecked=
IsVerboseExecutionModeChecked=
IsShowCommandsActivated=

[DEBUG]
IsDebugExecutionModeChecked=FALSE
```

These paths and directory names are examples only. You can set up your own paths and directory names to suit yourself.

Format Checking Table (.chk)

A MARC Format Checking Table (.chk) describes the valid features of a particular MARC format. The table lists all the fields allowed in the record together with their indicator values and subfield codes. It also specifies whether the fields and subfields are mandatory or not and whether they are repeatable or not.

For each MARC format conversion two tables may be used; one to check the format of the input records and one to check the format of the output records.

Format

A Format Checking Table is a simple text file. The first line may contain a comment such as the name of the format. The table itself <u>must</u> start on the third line of the file.

Each following line can be either an include line or a description line.

An include line enables you to load another table of coded data as part of the current file. The first element is #, followed by the word 'include' which is followed by the file name in double quotes.

A description line describes the field, its indicator values and it subfield codes. Each line must have the following structure :

1. A field tag e.g. 245
2. A character from the following list:

_	means that the field is mandatory and not repeatable,
+	means that the field is mandatory and repeatable,
?	means that the field is not mandatory and not repeatable,
*	means that the field is not mandatory but is repeatable.

3. Space or tab, pipe character (|), space or tab.
4. I1=
   This should be followed by a list of valid values for the first indicator. Blank should be represented by _. The fill character (|) is represented by a space then the hex value i.e. 0x7C
5. Space or tab, pipe character (|), space or tab.
6. I2=
   This should be followed by a list of valid values for the second indicator. Blank should be represented by _. The fill character (|) is represented by a space then the hex value i.e. 0x7C
7. Space or tab, pipe character (|), space or tab.
8. A list of the subfield delimiters allowable for the tag.
   The delimiters are introduced by a control character, usually represented by $. This is followed by a letter or digit identifying the subfield e.g. $a.
   The delimiter is then followed by a character from the list at 2 above, to indicate its repeatability, etc.
   The delimiters must be separated from each other by space or tab, pipe character (|), space or tab.
9. An optional text comment preceded by two solidi (//). This may also be inserted above the field tag.

This file verifies the occurrence of fields or subfields, but does not verify that a particular field or subfield may only be repeated a particular number of times e.g. twice. This file only verifies fixed structures and if the occurrence of a field/subfield is linked to particular Indicator values these cannot be checked.

Each field found in a record is compared with this table, if an error is detected, it will be logged in the Error Log File.

Example:

```
UNIMARC format
…
#include  deb_uni.fmt
…
001_  |      I1=   |      I2=   |

100_  |  I1=_    |  I2=_      |  $a_

101_  |  I1=012 0X7C  |  I2=_ |$a*  |  $b*  |  $c*  |  $d*  |  $e*  | $f*  |  $h*
|  $i*  |  $g?  // Mandatory, not repeatable. I1 can be fill character. $g optional
not repeatable, rest optional & repeatable
…
200_  |  I1=01  |  I2=_      |  $a+ | $b* | $c* …  // Title
700*  |  I1=_    |  I2=012 |  $a_ | $b? | $c? | $d? | $f? … // Author
```

Character Set Translation Table (.trs)

Different MARC formats usually require the use of different character sets. The USEMARCON system therefore incorporates the ability to convert between different specified character sets. This process is handled by the use of Character Set Translation Tables (.trs) which map input to output characters. The character set table need only deal with non-standard characters as conventional ASCII characters (e.g A-Z, a-z, 1-9 etc.) can be specified in a default character set table (standard.trs).

Format

The first and second lines of the file may contain comments such as the name of the file, or a description of its function. The table itself <u>must</u> start on the third line of the file.

Each line of the table can be either an include line or a translation line.

An include line enables you to load another table of coded data as part of the current file. The first element is #, followed by the word 'include' which is followed by the file name in double quotes.

e.g. `#include "standard.trs"`

A translation line contains the the characters in the input record and the characters to which they are to be translated in the output record, separated by a pipe character.

The format of the translation line is as follows:

Character (or sequence of characters), space or tab, pipe character (|), space or tab, character (or sequence of characters). Optionally, a comment may be added to the end of the line , which must be preceded by two solidi (//).

i.e. `<characters in>   |     <characters out>  //<comment>`

A character can be described as its ASCII representation, as its hexadecimal (Hex) value or its decimal value. However, numbers must be described by their decimal or hexadecimal value e.g. 0x31 for 1, because '1' does not mean number 1 but the character whose ASCII value is 1. Hex values must be preceded by 0x e.g. 0x31, 0xE2. Blanks or spaces must be represented by Hex 20 i.e. 0x20.

Example:

```
…
0x30 | 0x30 //  0
0x31 | 0x31 //  1
0x32 | 0x32 //  2
0x33 | 0x33 //  3
0x34 | 0x34 //  4
0x35 | 0x35 //  5
0x36 | 0x36 //  6
0x37 | 0x37 //  7
0x38 | 0x38 //  8
0x39 | 0x39 //  9
…
```

Each character or sequence to be translated will either match or not match one character or sequence in the first column of the table. The translation software searches for the longest matching sequence. In order to prevent ambiguity with shorter sequences, the longest one will be chosen.

Example:

```
BL Exchange set to iso5426

#include "standard.trs"

0x22      | 0xAA  // Left High Double Quotation
0x22      | 0xB8  // Double Prime
0x22      | 0xBA  // Right High Double Quotation mark
0x24      | 0xA4  // Dollar Sign
0x27      | 0xA9  // Left High Single Quotation
0x27      | 0xB9  // Right High Single Quotation mark
0x49 0x6A | 0xE6  // IJ
…
```

If a character or sequence in the input record matches a character or sequence in the first column of the table, it will be replaced in the output record by the character or sequence in the second column of the table. If there is no character or sequence in the second column, then the input character will not appear in the output record. This is useful if you need to eliminate diacritics from a file or delete a string of characters throughout a file.

Example:

```
…
0xBA |  //  Eth
0xE0 |  //  High tone*
0xE1 |  //  Grave*
0xE2 |  //  Acute*
0xE3 |  //  Circumflex*
0xE4 |  //  Tilde*
0xE5 |  //  Macron*
0xE6 |  //  Breve*
0xE7 |  //  Dot above*
0xE8 |  //  Umlaut* (diaeresis)
0xE9 |  //  Hacek*
0xEA |  // Degree
…
```

If no character or sequence in the input record matches a character or sequence in the first column, then the current intput character is written in the output record.

Configuration File (.cnf)

This file describes fields that have no indicator values in the corresponding MARC file. There is one configuration file for the Input MARC file and one for the Output MARC file.

A wildcard character (?) can be used to replace one character. For Example 00? means any field with a tag beginning with 00.

Format and Example

```
00?
010
998
999
```

If no configuration file is provided, a MARC field is treated as having no indicators if a subfield delimiter ($) is found within the first two positions of the field contents. If the tag is of the format 00X i.e. is a control field, and no subfield delimiter appears in the field contents, it is treated as having no indicator. If the tag is of the 00X format but a subfield delimiter is found after the first two positions of the field contents, the first two positions are treated as being indicators. Every field which begins with anything other than 00X and which does not have $ in the first two positions of the field contents, are treated as having indicators.

Rules Files (.rul)

The rules governing a specific MARC conversion are held in a single rules file (.rul). The rules file describes precisely how the fields and subfields of the input MARC records should be converted into their counterparts in the output MARC format. Rules files are dealt with in detail in the next section.

CHAPTER 2 : Creating USEMARCON rules

Basic rule construction

USEMARCON rules are created in a non-formatted text file. The first and second lines may contain a comment such as the name of the format. The rules themselves <u>must</u> start on the third or subsequent line of the file. Rules are constructed as follows:

```
<input> | <output> | <instruction>
```

There may be any number of spaces or tabs on either side of the pipe symbol, but there must be at least one.

The input and the output may consist of

♦ a tag number,
e.g. `001 | 001 | S`

♦ a tag number and indicator, (indicators are represented by I1 and I2)
e.g. `245I1 | 245I1 | 1`

♦ a tag number and subfield code,
e.g. `245$a | 245$a | S`

♦ a tag number and character position(s)
```
e.g.  008/30/ | 008/30/ | S
      008/25-28/ | 008/25-28/ | S
```

♦ a tag number, subfield code and character position
e.g. `008/1-6/ | 100$a/3-8/ | S`

Remember that tags beginning with '00' are control tags which do not have indicators. All other tags have indicator values which must be dealt with when creating rules to convert an entire field. Tag '000' is used to represent the record label.

The following rule produces the most simple coversion of all:

```
001 | 001 |
```

It simply puts the contents of the 001 field in the input record into the 001 field of the output record. As 001 is a control field, no indicator values are required. The same rule can also be written thus:

```
001 | 001 | S
```

The following rule produces the most simple conversion for a non-control field:

```
245I1 | 245I1 |
245I2 | 245I2 |
245   | 245   |
```

This produces exactly the same in the output as there was in the input. Note that to do this you cannot simply write the following:

```
245 | 245 |
```

This would produce a 245 field in the output with no indicator values.

Very often a rule consists of a number of instructions. The instructions are separated from each other by a semicolon (;).

Usually it is necessary to write a rule or set of conversion rules for each subfield in a field (nnn represents tag number):

```
nnnI1 | nnnI1 | <instruction>
nnnI2 | nnnI2 | <instruction>
nnn$a | nnn$a | <instruction>; <instruction>
nnn$b | nnn$e | <instruction> ; <instruction>; <instruction>
nnn$c | nnn$c | <instruction>
etc.
```

Note that the order that the subfields are dealt with in the rules dictates the order that they will appear in the output. Thus the subfields in the example above will appear in the output in the order $a, $e, $c.

A rule consisting of several instructions may be written as one long string but this makes editing difficult. It is better to write the rule on multiple lines in which case the last character of each line of all but the last line must be a semicolon.

Some notes about 'S'

S represents the input Source. Where the instruction consists only of S then the contents of the output is exactly the same as the input.

Example:
```
001 | 001 | S
```

S may be combined with punctuation by using the plus (+) sign. This is particularly useful when converting from a MARC format which does not use embedded punctuation such as UKMARC and UNIMARC, to one which does, such as MARC21. Marks of punctuation must be surrounded by single quotes.

Examples:

`S + '.'`	Full stop is placed after the data
`S + ','`	Comma is placed after the data
`S + ' '`	A single space is placed after the data
`' ' + S + ','`	A space is placed before the data and a comma is placed after it
`'(' + S + ')' + ','`	The data is placed in parentheses, followed by a comma

The contents of the output may be dependant on the value of the input in which case S can be combined with a value.
Example:
```
08/29/ | 008/29/ | If (S='1') Then 'o'
```
This means:
If character position 29 in 008 in input is '1' then cp 29 in 008 in output is 'o'.

Example:
```
040$a  | 040$a | If (S='OX/U-1') Then 'UkOxU'
```
This means:
If the contents of 040$a in input is 'OX/U-1' then 040$a in output contains 'UkOxU'

You can also use `If (S='')` to specify 'If the source is empty', and `If (S!='')` to specify 'If the source is not empty'.

Use of temporary fields

Quite often the conversion of a field, or even a subfield, requires several stages. It is possible to process the contents of the input using one set of rules, place the contents into a temporary field, process the new contents using another set of rules before finally placing the data in output. This process of cycling the data through temporary fields can be carried out as often as is necessary.

The temporary tag may be numeric, alphabetic or a combination of both but it must be preceded by '<' in output.

The following is a simple example of the use of a temporary field:

```
084   | <08A  | If (@084$2='rubbk') Or (@084$2='rugastni') Then S
08A   | 686I1 | ' '
08A   | 686I2 | ' '
08A$a | 686$a | S
08A$b | 686$a | S
08A$2 | 686$s | S
```

In this example field 084 is to be converted to field 686 but only if subfield $2 contains either 'rubbk' or 'rugastni'. If $2 does contain either of these strings then the whole 084 field is converted to temporary field 08A and the contents of that is then placed in 686. If $2 did not contain either of the specified strings then temporary field 08A would not be created and 686 would not exist in the output.

Instructions

In most of the examples above the instruction has simply consisted of no text at all or of S, both of which produce the same in the output as existed in the input.

Strings

An instruction may consist of a string, and the string may consist of one character . In the following rule the instruction is to make both indicator values 0:

```
050I1 | 050I1 | 0
050I2 | 050I2 | 0
050$a | 050$a |
```

The string may be a blank. In the following, the instruction is to make both indicator values blank:

```
043I1 | 043I1 | ' '
043I2 | 043I2 | ' '
043$a | 043$a |
```

Here you will note that the blank is enclosed by single quotes.

Be sure that you do not use smart quotes when creating rules files.

A string may be empty. In such a case it is represented by two quotes without a space between them i.e. `''`

Example:
```
If (S!='') Then S
```
This means:
If the input is not empty then enter the input data.

Here is another example of the use of strings:

```
001 | 801I1 | ' '
001 | 801I2 | 3
001 | 801$a | 'US'
001 | 801$b | 'CStRLIN'
001 | 801$c | Year+Month+Day
```

This rule will place US in 801$a and CStRLIN in 801$b. The contents of $c is dealt with in the next section.

Note also that, in this example, the field that is being created in the output is completely different from the field specified in input. Very often it is simply necessary to specify one tag in the input and an equivalent but different tag in the output, for example when converting from UKMARC to UNIMARC the data in the 245 in input is placed in the 200 field in output. In this case, however, here is *no* field in the input which is the equivalent of the field that is required in output so the the field in output has to be created, and as there must always be an input and an output in the rule, a field which will exist in all records has been used for the input i.e. 001. But note that *no data* is actually transferred from field 001 to 801.

A field which exists in the record but which has no rule applied to it is simply ignored and is omitted from the output.

Date and time variables

There are six date and time variables:

Year	the current year e.g. 2000
Month	the current month e.g. 10
Day	the current day e.g. 27
Hour	the current hour e.g. 17
Minute	the current minute e.g. 20
Second	the current second within the minute e.g 48

As many of the variables as necessary may be used with a plus sign between each e.g.

```
// Date and time of latest transaction
001 | 005 | Year+Month+Day+Hour+Minute+Second+'.0'
```

Conditions

'If … Then …' and 'If … Then … Else …' may be used in the construction of a rule and may be used in combination with the Boolean operators 'And', 'Or' and 'Not'. Conditions may be applied to indicators, fields or subfields, in other words, wherever they are required. But remember that the condition you specify always refers to what is in the input. If a tag number is not specified in the rule then the tag in the input column is assumed.

Example:
```
350I1 | 037I1 | If (@008/39/='p') Then ' '
```
This means:
If character position 39 in the 008 in input is 'p' then 037 indicator 1 is blank.

Note here also that the @ symbol appears before 008. This is to specify that 008 is a tag not just a number.

Example:
```
240I2 | 130I1 | If (I1=3) Then S
```
This means:
If the value of 240 indicator 1 in input is '3', then the current value of 240 indicator 2 becomes the value of 130 indicator 1

Example:
```
350I1 | 020I1 |  If (@008/39/!='p') And (@000/18/!='8') Then ' '
```
This means:
If character position 39 in 008 in input is not 'p', and if character position 18 in the record label in input is not '8', then 020 indicator 1 in output contains blank

Example:
```
000/6/ | 000/6/ | If (S='f') Or (S='p') Then 'a' Else S
```
This means:
If character position 6 of the record label is 'f' or 'p' then character position 6 in the output record label contains 'a'. Otherwise, the contents of output is the same as the input.

If the conditions specified do not apply then the rule is not processed

Exists

In all the examples in the section above 'equal to' and 'not equal to' has been used in all the conditions. The output may be dependant on whether a field, a subfield or a bit of data *exists* in input. The following example is one of many rules used to insert punctuation in a field in the UK to MARC 21 conversion.

Example:
```
111$k | 111$d | If Exists($i) And Exists($j) Then S + ' :'
```
This means:
If subfield $i and subfield $j exists in field 111 in input, place the contents of 111$k into 111$d in output and add space colon at the end.
Again, Exists may be used with Boolean operators And, Or and Not and with Else.

Example:
```
243$a | 240$a | If Exists($o) Or Exists($r) Or Exists($s) Then S + '.' Else
S
```

Example:
```
440$a | 440$a |If Not (Exists ($b) Or Exists ($e) Or Exists ($k)) Then S
```

Example:
```
710$k | 710$d | If Not Exists($i) And Not Exists($j) Then '(' + S + ')'
710$k | 710$d | If Exists($i) And Exists($j) Then S + ' :'
710$k | 710$d | If Exists($i) And Not Exists($j) Then S + ')'
710$k | 710$d | If Exists($j) And Not Exists($i) Then '(' + S + ' :'
```

In the example above, the data in 710$k in input is placed in 710$d in output, but the punctation in the field in output is dependant upon the existence or otherwise of other subfields in the input.

NextSub

NextSub means Next Subfield and is used in the rules in a similar way to Exists i.e. it may be used with If … Then … Else and with And, Or and Not. Next Subfield always means 'the next subfield in input'.

Example:
```
100$a | 100$a | If NextSub='c' Then S + ', '
```
This means:
If the next subfield after $a in the input is $c then place the contents of the input in 100$a in output and place a comma at the end.

Example:
```
100$a | 100$a | If NextSub='c' Then S + ', '
100$a | 100$a | If NextSub='e' And Not Exists ($h) Then S + ' '
100$a | 100$a | If NextSub='e' And Exists ($h) Then S + ', '
100$a | 100$a | If NextSub='f' Then S + ','
100$a | 100$a | If NextSub='h' Then S + ', '
100$a | 100$a | If NextSub='l' Then S + ' '
```

The example above is taken from the UK to MARC 21 rules conversion. The rules place the correct punctuation at the end of subfield $a in output which depends upon which subfield follows $a in input.

Precedes and Follows

Precedes and Follows is used to refer to subfield positions in the input.

Example:
```
U45 | 245$a | If (($k Precedes $b) And Not ($b Precedes $k)) Then …
```

Replace

Replace may be used to replace a subfield code with another subfield code, a subfield code with a mark of punctuation, a mark of punctuation with a subfield code or one string of text with another. The syntax used is as follows:

```
Replace ('x' By 'y')
```

Example:
```
008/9-15/ | 008/9-15/ | S; Replace('?' By 'u')
```

It may be used as part of an 'If … Then ..' condition

Example:
```
…If Exists($d) And Not Exists ($e) Then Replace ('$d' By ' (');…
```

It is extremely useful for tidying up punctuation.

Example:
```
… Replace ('pbk' by 'pbk.'); Replace ('pbk..' by 'pbk.'); Replace (':(' by
'(')
```

It is also very useful for preserving the original order of the data in the output where multiple occurrences of a subfield may appear in any order in the input. A common example of this in UKMARC and MARC21 is the 260 (Imprint field) in which the subfields could appear in the order $a, $a, $b, $a, $b, $c, for example. It is obviously necessary that this order is preserved when the record is converted, so in the US to UKMARC conversion the following rules are used:

```
260 | 260 | Replace ('(' By '$d'); Replace(' ;$a' By '$a');
            Replace(',$a' By '$a'); Replace(' :$b' By '$b');
            Replace(' ;$b' By '$b'); Replace(' ; ' By '$a');
            Replace(' : ' By '$b'); Replace(',$b' By '$b');
            Replace(',$c' By '$c'); Replace(' $e(' By '$i');
            Replace(',$f' By '$j'); Replace(' :$f' By '$j');
            Replace(',$g(' By '$k'); Replace('$g(' By '$k')
```

These rules combine replacing the MARC21 punctuation with no punctuation, and replacing MARC21 subfield codes with UKMARC codes.

It is also possible to specify that the replacement is to take place at the end of the field or subfield using the following:

```
Replace ('x' By 'y', At Ending)
```

Again this is most useful for tidying up punctuation.

Example:
```
Replace ('..' By '.', At Ending)
```

Delete

Delete is used in a similar way to Replace and may be used to delete a string of text, a mark of punctation or a subfield code, though this last is rarely used. The syntax is as follows:

```
Delete ('x')
```

Example:
```
… Delete(')'); …
```

Example:
```
… Delete (' : CIP entry'); …
```

You may also specify that you wish the deletion to take place at the end of the field or subfield by using the following:

```
Delete ('x', At Ending)
```

Again, this is very useful for getting rid of extraneous punctuation:

Example:
```
…; Delete('.', At Ending)…
```

ReplaceOcc

ReplaceOcc is used to replace multiple occurrences of the same subfield with different things in output. This is used mostly in the UK to MARC 21 conversion since subfields which are repeatable in UKMARC may not be repeatable in MARC 21. This usually means that occurrences of the subfield after the first must be replaced by marks of punctuation.

The syntax is as follows:

```
ReplaceOcc ('x' By 'y','>1')
```

This means replace x with y if the occurrence of x is greater than 1.

```
ReplaceOcc ('x' By 'z','=1')
```

This means replace x with z if this is the first occurrence of x.

Note that when combining the two, `'ReplaceOcc ('x' By 'y','>1')'` must be placed *before* `'ReplaceOcc ('x' By 'z','=1').`

In UKMARC subfield $b in the 245 is repeatable. In Marc21 subfield $b in 245 is not repeatable so after the first occurrence of $b other subtitles must be separated from each other by ' : '. In order to achieve this the following rule is used:

```
… ReplaceOcc('$b' By ' : ','>1'); ReplaceOcc('$b' By ' :$b','=1'); …
```

This function will work with =, !=, >=, <=, <, >.

Len

This function returns the length of whatever is specified. It may be used with =, >=, <=, <, >. It is generally used with 'If… Then …'

Example:
```
010$a | 010$a | If (Len(S)=8) Then S
```
This means:
If the length of the data in 010$a in input is 8 characters then place it in 010$a in output.

From, To, Between

'From' is used to specify the point in a subfield from which the processing of the field must start.

Example:
```
001 | 001 | From(4)
```
This means:
Place the contents of 001 in input into 001 in output, starting at character position 4.
If the input contains 12345678, output will contain 45678

'To' is used to specify the point up to the processing of the field must end.

Example:
```
001 | 001 | To(6)
```
This means:
Place the contents of 001 in input into 001 in output, ending at character position 6.
If the input contains 12345678, the output will contain 123456

'Between' is used to specify the points between which the processing must take place.

Example:
```
041$a | 101$a | If (Len(S)=6) Then Between(1,3) + '$a' + Between(4,6)
```
This means:
If the input is six characters long, place characters 1-3 in output, followed by '$a', followed by characters 4-6. Thus

```
041$aengger
```
becomes:
```
041eng$ager
```

Strict

'Strict' may be used in conjuction with From, To, Between, Replace and Delete and is used to handle blank spaces.

When using 'From', if 'Strict' is not specified and blank characters appear at the beginning of the part to be processed, they are skipped. If 'Strict' is specified, blank spaces are retained.

When using 'To' if 'Strict' is not specified and blank characters appear at the end of the part to be processed, they are skipped. If 'Strict' is specified, blank spaces are retained.

When using 'Between' if 'Strict' is not specified and blank characters appear at the boundaries of the part to be processed, they are skipped. If 'Strict' is specified, blank spaces are retained.

Example:
```
001 | 001 | From(5, Strict)
```
This means:
Place the contents of 001 in input into 001 in output, starting at character position 5.
If the input contains ###1234567 the output will contain 234567 (# represents blank). If 'Strict' is not specified the initial blanks would be ignored and the output would contain 567.

BFirst, EFirst, BLast, ELast

These functions are usually used in combination with 'From', 'To' and 'Between'.

BFirst means 'Beginning of first …', EFirst means 'End of first …', BLast means 'Beginning of last …' and ELast means 'End of last…'.

Example:
```
S; To (BFirst('-'))
```
This means:
Place everything before the first '-' in the input into output.

Example:
```
S; From (EFirst('-'))
```
This means:
Place everything after the first '-' in the input into output.

Example:
```
S; To (BFirst('$p'))
```
This means:
Place everything before the first $p in the input into output.

Example:
```
S; Between (BLast(String),ELast(String))
```
This means:
Place everything between the beginning and the end of the last string in the input into output.

Example:
```
S; To (BFirst(Number))
```
This means:
Place everything before the first number in the input into output.

Lower, Upper

'Lower' is used to change upper case characters to lower case and 'Upper' changes lower case characters to upper case.

Example:
```
…; Lower(Between(1,1)) + (From(2))
```
This means:
Make the first character lower case, then continue with the rest of the word.

Val

'Val' returns the value of whatever is specified.

Example:
```
Val(S); If S>99999 Then S
```
This means:
Return the value of the input; if it is greater than 99999, place it in output.

Adding data from one subfield to the end of another

It is sometimes the case that data in two different subfields in the input has to appear in a single subfield in the output. To do this a plus sign is placed at the beginning of the rule

Example:
```
082$a | 082$a | S
082$b | 082$a | + '/' + S
```

This adds the data in 082$b in input to the end of 082$a in ouput. The data is preceded by slash. Thus:

```
082.00$a822.3$b3
```
becomes:
```
082.04$a822.3/3
```

The plus sign must be always be placed at the beginning of the rule, even if the rule contains conditions.

Example:
```
300$e | 020$a | + If (@021$b='m') And Exists (@350$a) Then ' (' + S + ')' +
' :' Else ' (' + S + ')'
```

Memory instructions

Memory instructions are used to store intermediate results within complex rules. 100 memories are available numbered from 0-99. The memory functions most commonly used are 'Sto' and 'Mem'.

Sto(n) stores the current value of S in the nth memory.
Mem(n) returns the content of the nth memory.

Example:
```
00x | <00z | Sto(0); Between(3,3); If S='0' Then Mem(0); From(4)
```
This means:
`Sto(0);` – Store contents of temporary field 00x in memory 0;
`Between(3,3)` – return the character in position 3;
`If S='0' Then Mem(0); From(4)` - if the character is 0 place the contents of memory 0 in output starting from character position 4

The contents of the memories are not cleared automatically, therefore it is possible to exchange information between rules and even between records. However, if you store a result in a memory in one rule and store a result in the same memory in a later rule, the later memory will overwrite the earlier one.

Redo

Redo is used where a conversion of 'one to many' is required. Data can be taken from one subfield in input, split up and placed in more than one subfield in output.

Example:
```
041$a | 101$a(no) | Sto(0); To(3); Redo; Mem(0); From(4)
```

In order to ensure $a is repeated (no) must be used (see 'Management of occurrences', below).

In the first pass the rule places the first three characters of 041$a into 101$a and stores what is left in 014$a in memory. In the next pass it creates another $a in output and puts the first three characters of what is stored in memory in the new $a. What is left is 041$a is then stored in memory. The cycle contines until nothing stored in memory.

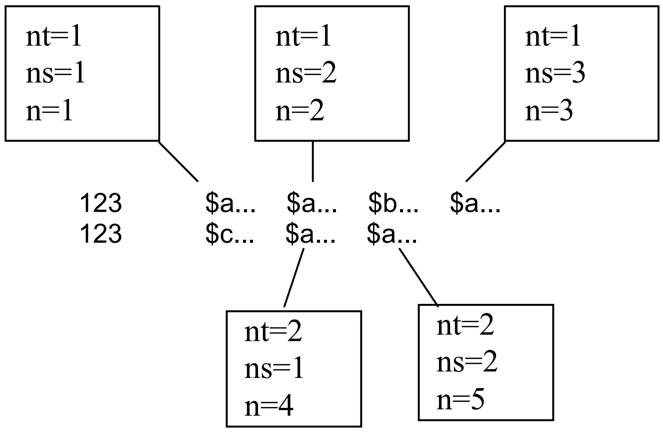
Thus:
```
041.00$aengfrespa
```
becomes:
```
101.  $aeng$afre$aspa
```

Management of occurrences

The management of occurrences is very complex and is only described in brief in this guide. For full details see the USEMARCON Technical Manual.

Occurrences are managed by using the terms n, ns, nt, no, nso, and nto.

nt (no. of tag) represents the ocurrence of the input tag
ns (no. of subfield) represents the occurence of the input subfield
n represents the input, whatever it is
nto represents the occurrence of the output tag
nso represents the occurrence of the ouput subfield
no represents the out put, whatever it is



Basically, no, nto and nso are used to ensure a new field or subfield is written in the output, where required, and prevents existing fields or subfields with the same tag or code being overwritten. They may only be used in association with output tags and subfield codes.

`nnn$? \| nnn(no)$? | …`	New field is created
`nnn$? \| nnn(nto)$? | …`	New field is created
`nnn$? \| nnn$?(no) | …`	New subfield is created
`nnn$? \| nnn$?(nso) | …`	New subfield is created

Example:
```
949I1 | 949(nto)I1 |
949I2 | 949(nto)I2 |
949$a | 949(nto)$a |
```

(nto) was added to these rules to ensure that 949 fields containing multiple $a subfields would produce instead, multiple 949 fields containing a single $a subfield

Thus:
```
949.00:0/0 $aTJ 843 BURwLCi72535620hRC3#
949.00:0/1 $aTJ 843 BURwLCi73016751hRC3$aTJ 843 BURwLCi73016760hRC3#
```
becomes:
```
949.00:0/0 $aTJ 843 BURwLCi72535620hRC3#
949.00:0/1 $aTJ 843 BURwLCi73016751hRC3#
949.00:0/2 $aTJ 843 BURwLCi73016760hRC3#
```

It is also possible to specify a particular occurrence of a tag or subfield in the input within a rule by using nt, ns and n.

Example:
```
500$r | 240$d     | If (n=1) Then S
500$r | 740(no)$d | If (n>1) Then S
```
This means:
Place the first occurence of 500$r in 240$d. Place any other occurrences of 500 $r in 740$d. Note that **no** has been used with 740 to ensure several 740 fields are created in output if there are several occurrences of the 500 field in input.

Example:

<u>Input record:</u>
```
650.00$aLocal area networks (Computer networks)$xManagment
660.10$aComputers$aNetworks$aManagement
```

<u>Required output record:</u>
```
606.  $aLocal area networks(Computer networks)$xManagement$slc
606.  $aComputers$xNetworks$xManagement$2COMPASS
```

<u>Rules to achieve this:</u>
```
650I1 | 606I1       | ' '
650I2 | 606I2       | ' '
650$a | 606$a       |
650$x | 606$x       |
650   | 606$s       | 'lc'
660$a | <66A$a      | If (ns=1) Then S
660$a | <66A$x(nso) | If (ns>1) Then S
660   | <66A$2      | 'COMPASS'
66a   | 606(nto)    |
66A   | 606(nto)I1  | ' '
66A   | 606(nto)I2  | ' '
```

Comments

Any rule may be preceded or succeeded by a comment. A comment must be preceded by two forward slashes i.e. //. A comment may be place above a rule, below a rule or at the end of a line. It is good practice to place the name of the field to be processed in a comment above the rules and it can be very useful indeed to use comments to annotate particularly complex rules or simply to remind yourself why you wrote the rule as you did e.g.

```
// Source of control number
001 | 003 | 'Uk' // THIS MAY HAVE TO BE AMENDED LATER
```

Lookup Tables

Look Up Tables (.tbl) are used to hold large amounts of tabular information (e.g. ISO Standard country codes) which are required by a conversion rules file. Data is held separately in a table to allow easier maintenance of the main rules file. A rule can therefore be written which will ‘call’ the table rather than actually containing the information in the table.

Example:
```
008/16-17/ | 102$b | Table ('Region.tbl')
```

Unlike other conversion files, Look Up Tables are commonly named to reflect their function e.g. ill.tbl for a table of illustration codes.

Each Look Up table used in a Rules file is defined in a separate file. Each file bears the name of the table used in the Rules file.

Format

The first and second lines of the file may contain comments such as the name of the file, or a description of its function. The table itself must start on the third line of the file.

Each line of the table can be either an include line or a translation line.

An include line enables you to load another table of coded data as part of the current file. The first element is #, followed by the word 'include' which is followed by the file name in double quotes.

A translation line contains the characters in the input record and the characters to which they are to be translated in the output record, separated by a pipe character.
If a character or sequence in the input record matches a character or sequence in the first column of the table, it will be replaced in the output record by the character or sequence in the second column of the table.

The format of the translation line is as follows:

Character (or sequence of characters), space or tab, pipe character (|), space or tab, character (or sequence of characters).

i.e. `<characters in>   |     <characters out>`

Example:

```
pubcode.tbl

s | d
r | e
m | a
z | e
```

Optionally, a comment may be added to the end of the line, which must be preceded by two solidi (//).

Each character can be described as its ASCII representation, as its hexadecimal (Hex) value or its decimal value. However, numbers must be described by their decimal or hexadecimal value e.g. 0x31 for 1, because 1 does not mean ‘1’ but the character whose ASCII value is 1. Hex values must be preceded by 0x e.g. 0x31, 0xE2. Blanks or spaces must be represented by Hex 20 i.e. 0x20.

Example:

```
isds.tbl

0x20 | 0x20 0x20
0x31 | 0x20 0x31
0x32 | 0x20 0x32
0x33 | 0x20 0x33
0x34 | 0x20 0x34
0x35 | 0x20 0x35
0x36 | 0x20 0x36
0x37 | 0x20 0x37
0x38 | 0x20 0x38
0x39 | 0x20 0x39
u    | 0x7C 0x7C
```

When typing words in a lookup table each character of each word must be separated by a space, and a space between words must be represented by Hex 20 (0x20).

Example:

```
Freq.tbl

D a i l y                                      | d
T h r e e 0x20 i s s u e s 0x20 w e e k l y    | i
T w o 0x20 i s s u e s 0x20 w e e k l y        | c
W e e k l y                                    | w
T h r e e 0x20 i s s u e s 0x20 m o n t h l y  | j
F o r t n i g h t l y                          | e
I s s u e d 0x20 t w i c e 0x20 a 0x20 m o n t h | s
M o n t h l y                                  | m
```

Each character or sequence to be translated will either match or not match one character or sequence of the first column of the table. The translation software searches for the longest matching sequence. In order to prevent ambiguity with shorter sequences, the longest one will be chosen.

If no character or sequence in the input record matches a character or sequence in the first column, then the current input character is written in the output record.

A default may also be set up to take care of cases where no sequence in the input record matches a sequence in the input column. This is entered in the form:

```
#DEFAULT | <characters out>
```

Example:

```
colour.tbl

b | a
c | b
m | v
u | u
#DEFAULT | 0x20
```

In the example above, if the input record does not contain b,c,m or u then a blank will be in the appropriate position in output.

CHAPTER 3 : The Error log file

Overview

System messages describing errors detected during format checking or data conversion are held in the file 'errcodes.txt'. When running USEMARCON under Windows this file should be in the same directory as usemarcon.exe. When running under Linux, this file should be in the same directory as the .ini file.

When the software encounters an error it finds the appropriate error message in errcodes.txt and writes it to the end of the Error Log File. This is a simple ASCII text file which may be opened and viewed in any text editor.

In common with the other system files, the path and name of the Error Log File may be changed in the .ini file. The file is commonly given the name 'errlog.txt' but you can give it any name you wish. It should be noted, however, that the Error Log File is the only default file which is mandatory i.e. the software will not run at all if it is not named in the .ini file.

For each error, one line is logged in the report file; this enables easier filtering of the report file.

Each line begins with either WARNING, ERROR or FATAL.

- WARNING will not stop the program
- ERROR gives you the option to stop in Interactive mode but will not stop in Batch mode
- FATAL errors will not allow the software to proceed further.

The rest of the line consists of:

(<coded error number>), <the error description>, <the record control number>, <an indication of where the problem lies>

Example:
```
WARNING(2104)-Invalid or redundant subfield found in input record : Notice
'tgs90000001' : field '110' (subfield '$z')
```

Error Definitions

Errors are numbered 000-9999. The first digit describes the part of the process in which the error appears:

0xxx The error is due to bad interface use.
1xxx The error appears during reading of a MARC file, i.e. a format problem.
2xxx The error appears during MARC checking in input (comparison with the input checking table).
3xxx The error appears during character translation.
4xxx The error appears during coded data translation.

5xxx	The error appears during rule analysis or conversion.
7xxx	The error appears during checking of the MARC output.
8xxx	The error appears during writing of the MARC file.
9xxx	Other internal errors.

Examples:

```
WARNING(2001) - Invalid input format checking rule ( item expected ) …
ERROR(3001) - Character not transcoded (unable to find it in transco table)
: 0x90 in …
FATAL(7011) -....
```

Error Messages

A complete list of error messages used by the USEMARCON software is shown below.

501	-	Invalid rule pattern to search
502	-	No more patterns found
503	-	Maximum errors to be encountered has been reached
504	-	The last record to be converted has been reached
505	-	Type or browse a MARC file to open before
506	-	Pattern not found
507	-	Unable to evaluate an empty rule
508	-	No defined box to search. Please point to the box to search in
509	-	No search is available on this selection. Please select another one
510	-	Comment is too long and will be truncated
511	-	Line is too long and will be truncated
512	-	Unknown input format comment
513	-	Unknown output format comment
514	-	Please, select RI boundaries
515	-	Please, select No boundaries
516	-	Invalid RI boundaries
517	-	Invalid No boundaries
590	-	Specified file does not exist (please check path)
1001	-	Unable to reset a writing mode opened file
1002	-	Invalid SCW encountered when attempting to read a MARC notice
1003	-	Unable to go further in MARC file reading
1004	-	Invalid length encountered when attempting to read a MARC notice
1005	-	Unable to go further in MARC file writing
1006	-	Unable to flush the MARC output file
1007	-	Invalid MARC data location address
1009	-	Unable to set the content of the field
1101	-	Invalid MARC tag
1102	-	Invalid MARC indicators
1202	-	Unable to set the label
1501	-	MARC buffer allocation failure
1502	-	Error encountered when attempting to read the MARC file
2001	-	Invalid input format checking rule (item expected)
2002	-	Invalid or absent tag in input format checking rule

2003	-	Format checking rule redundancy in input format checking file
2004	-	Invalid or absent first indicators list in input format checking rule
2005	-	Invalid or absent second indicators list in input format checking rule
2006	-	Invalid or absent subfield in input format checking rule
2101	-	Not repeatable but redundant field found in input record
2102	-	Invalid first indicator found in input record
2103	-	Invalid second indicator found in input record
2104	-	Invalid or redundant subfield found in input record
2105	-	Unexpected field found in input record
2106	-	Mandatory field expected in input record
2107	-	Mandatory subfield expected in input record
2108	-	Field without any subfield found in input record
2501	-	TControlField allocation failure when attempting to load a new input format checking rule
2502	-	TCtrlSubfield allocation failure when attempting to load a new input format checking rule
2503	-	First indicators list allocation failure when attempting to load a new input format checking rule
2504	-	Second indicators list allocation failure when attempting to load a new input format checking rule
3000	-	Memory allocation error
3001	-	Character not transcoded (unable to find it in transco table)
4001	-	Coded data not loaded
5000	-	Memory allocation error
5001	-	The selected rule file does not exist
5002	-	Unable to load the invalid rule
5003	-	Unable to find the label in any CD
5004	-	Only one indicator has been found
5005	-	Content of indicator is too long (>1)
5100	-	Rule analysis error
5101	-	A CDOut like TTT(no)... has an invalid subfield occurrence number (no, nso or nto)
5102	-	A CDOut like ...SS(no) has an invalid field occurrence number (no, nso or nto)
5103	-	A CDOut like TTT(nto)... has an invalid subfield occurrence number (no, nso or nto)
5104	-	A CDOut like ...SS(nso) has an invalid field occurrence number (no, nso or nto)
5200	-	Expected CD tag
5201	-	Invalid CD tag (three characters are required)
5202	-	Invalid CD tag (only numerics or letters are allowed)
5203	-	Invalid CD subfield (only two characters required)
5204	-	Invalid CD subfield (only '1' or '2' is expected behind a 'I' subfield)
5205	-	Invalid CD subfield (only numerics or letters are allowed behind a '$' subfield)
5206	-	Invalid CD subfield (only I1, I2 or $? are allowed)
5207	-	Invalid position settings
5208	-	Invalid CD tag occurrence number
5209	-	Invalid CD subfield occurrence number
5210	-	Misplaced rule : please insert this rule before the previous one

5211	-	Misplaced rule : please insert this rule after the next one
5212	-	Invalid character found in rule
5301	-	Invalid output occurrence number 'no'
5302	-	Invalid output tag occurrence number 'nto'
5303	-	Invalid output sub occurrence number 'nso'
5304	-	Invalid input occurrence number 'n'
5305	-	Invalid input tag occurrence number 'nt'
5306	-	Invalid input sub occurrence number 'ns'
5307	-	Unknown main input CD (please type or load it before evaluating)
5308	-	Unknown old output CD (please type it before evaluating)
5309	-	Unknown other input CD (please, type it before evaluating)
5501	-	TRule allocation failure when attempting to analyse the rule
5502	-	Unable to allocate space for setting text of analysed rule
5503	-	Unable to allocate space for setting comment of analysed rule
5504	-	TCD allocation failure when attempting to analyse the rule
5505	-	TCDLib allocation failure when attempting to deal with other input CDs
5506	-	Buffer allocation failure when attempting to split the rule
5507	-	TCD allocation failure when attempting to load CD from MARC record
7001	-	Invalid output format checking rule (item expected)
7002	-	Invalid or absent tag in output format checking rule
7003	-	Format checking rule redundancy in output format checking file
7004	-	Invalid or absent first indicators list in output format checking rule
7005	-	Invalid or absent second indicators list in output format checking rule
7006	-	Invalid or absent sub in output format checking rule
7101	-	Redundant field (not repeatable) found in output record
7102	-	Invalid first indicator found in output record
7103	-	Invalid second indicator found in output record
7104	-	Invalid or redundant subfield found in output record
7105	-	Unexpected field found in output record
7106	-	Mandatory field expected in output record
7107	-	Mandatory subfield expected in output record
7108	-	Field without any subfield found in output record
7501	-	TControlField allocation failure when attempting to load a new output format checking rule
7502	-	TCtrlSubfield allocation failure when attempting to load a new output format checking rule
7503	-	First indicators list allocation failure when attempting to load a new output format checking rule
504	-	Second indicators list allocation failure when attempting to load a new output format checking rule
8001	-	Unable to delete the Error Log File
9001	-	TRuleFile allocation failure when attempting to load the Rule File
9011	-	TCheckFile allocation failure when attempting to load the Input Check File
9012	-	TCheckFile allocation failure when attempting to load the Output Check File
9013	-	TTransFile allocation failure when attempting to load the Translation Character Table

9021	-	TMARCFile allocation failure when attempting to open the Input MARC File
9022	-	TMARCFile allocation failure when attempting to open the Output MARC File
9031	-	TMARCRecord allocation failure when attempting to load the Input MARC File
9032	-	TMARCRecord allocation failure when attempting to load the Output MARC File
9041	-	TMARCField allocation failure when attempting to load the notice into memory fields
9101	-	TRuleDoc not created
9102	-	TMARCDoc not created
9103	-	TDummyDoc not created
9104	-	TTransDoc not created
9105	-	TCheckDoc not created
9201	-	TCD allocation failure when attempting to search for another CD
9202	-	Label is mandatory and has not been converted
9203	-	TMARCField allocation failure when attempting to merge CDs into fields
9301	-	TCDLib allocation failure when attempting to split a field into CDs
9401	-	Find SLIST allocation failure when attempting to memorise precedent find/replace request
9402	-	Replace SLIST allocation failure when attempting to memorise precedent find/replace request
9403	-	TRule allocation failure when attempting to search/replace items
9404	-	TCD allocation failure when attempting to search/replace items
9501	-	Unable to open the ASCII mode file
9502	-	Unable to open the binary mode file
9503	-	Unable to delete the file
9504	-	Unable to get the next line of a binary file
9505	-	Invalid #include 'file' directive found in file
9506	-	Unable to read two first lines of an ASCII file
9601	-	Unable to open the MARC Input Window
9602	-	Unable to open the MARC Output Window
9603	-	Unable to open the Rule Edit Window
9604	-	Unable to open the Rule Eval. Window
9700	-	Two identical CDs found in a record !
9701	-	Unable to save the MARC edit configuration file
9703	-	Invalid tag field to add to the list of tags without indicator
9704	-	Unable to add the selected tag to the list of tags without indicator
9705	-	Unable to remove the selected tag from the list of tags without indicator
9706	-	(No) is not filled
9800	-	Unable to open the help file usemarco.csc/hlp
9999	-	Unknown error