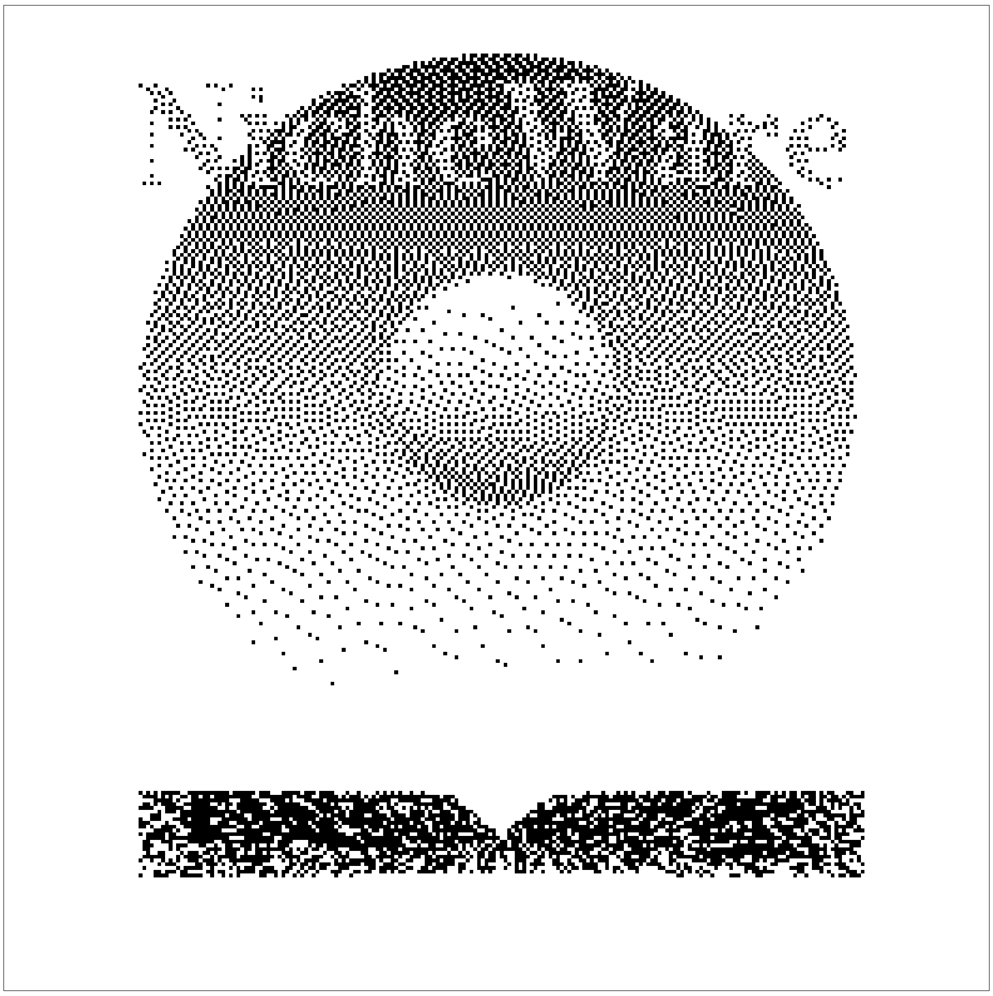

MARC RTP

MARC Record Translation Program

Version 1.4.9

User Manual

Table of Contents

What is MARC ?....................1
About MARC RTP....................1
Installing the software....................1
Getting started....................2
  Finding out what is in your MARC file....................2
Using the software....................2
  Generating a listing....................3
    Listing format....................3
  Generating a statistics analysis....................4
    Statistical format....................4
  Extracting MARC data....................5
    Importable data file format....................5
    The relationship between MARC records and importable output records....................6
The Requests file....................6
  Why do I need a Requests file?....................6
  What is an extraction request?....................6
Writing the requests file....................7
  Configuration definitions....................7
  Table definition....................9
  Row definition....................9
  Column definition....................10
Extraction requests....................11
  Simple requests....................11
    Literal text request....................11
    Tag requests....................11
    Sub field requests....................12
  Composite requests....................13
    Alternative (or OR) requests....................13
    Linked (or AND) requests....................14
    Linked (AND) requests within alternate (OR) requests....................14
    Linked requests are constrained....................14
  Located requests....................15
    Requesting specific instances of repeated tags and sub fields....................15
    Tag occurrence specifiers....................15
    Subfield occurrence specifiers....................16
  Fixed field and more general selections....................16
    Extracting information from the record leader....................17
    Extracting variable tag field indicators....................17
  Wildcard requests....................18
  Index functions....................19
  Defer function....................20

Qualifier functions....21
Alternative (or OR) requests revisited....22
Tips and hints....22
Assistance in designing your database....22
When you add to or modify your database....23
Be as specific as possible....23
Who is NicheWare?....24

What is MARC ?

The MARC formats are standards for the representation and communication of bibliographic and related information in machine-readable form.

``MARC is an acronym for MAchine Readable Catalogue or Cataloguing. This description, however, is rather misleading as MARC is neither a kind of catalogue nor a method of cataloguing. In fact, MARC is a standard format for representing information in a catalogue record in machine-readable form, i.e. for computer processing. MARC was primarily developed to meet the needs of libraries but has since been adopted by the wider information community as a convenient way of storing and exchanging records."
- British Bibliographic Network

About MARC RTP

MARC RTP was especially developed so that catalogue data contained in MARC format files could be converted, and selectively imported, into databases built with general-purpose applications.

RTP will work with various MARC formats including spanned, unspanned, blocked, or unblocked formats. The program supports formats obtained from a number of bibliographic sources including the United States of America Library of Congress and the Australian Bibliographic Network. These are often referred to as USMARC, LCMARC, and AUSMARC files, to name a few.

RTP allows you to select any part or parts of each MARC record, therefore you do not have to design a large unnecessarily complicated database. You need only include the data that is of interest to you. As an aid, RTP can produce a readable listing of all of the records from the MARC file, and a summary giving important information about which tags and fields exist in your MARC records.

Installing the software

The software and other files are contained on the supplied media specific to your system. For the following systems:

DOS

Unpack the ZIP file using WiZ or UnZip (free software from http://www.info-zip.org/pub/infozip/) or the commercial WinZIP product to a suitable place on your system. Read the file README.TXT just extracted.

Solaris

Change to an appropriate directory on your system and then extract the software with the following two commands:

```
uncompress mrtp-1.4.8-solaris-sparc-shared.tar.Z
tar xvf mrtp-1.4.8-solaris-sparc-shared.tar
```

Read the file README just extracted.

Linux

Change to an appropriate directory on your system and then extract the software with the following command:

```
tar zxvf  mrtp-1.4.8-linux-x86-shared.tar.gz
```

Read the file README just extracted.

Getting started

You've decided which database system you'll be using and you have your MARC records in a file on disk. All you need now is to fill the database with information from your MARC records. In order to prepare for this you need to:

- know which information you wish to include in your database.

  If you're building a new database, this may involve knowing precisely what information is in your MARC records.

- know how your database system places each piece of information into it's correct location within the database.

  You may need to know the ordering of the columns in your database table. This is dependent on your database system.

Finding out what is in your MARC file

One way to find out what is in your MARC file is to convert it from it's current format, which is designed to be read by your computer, to a format that you can read yourself. RTP can produce a listing for you.

NOTE: If your MARC file came on more than one floppy disk then you will need to join the contents of each floppy disk to make one complete MARC file.

A listing presents each record in turn in a format showing all tags, subfield codes, and their associated contents in an orderly readable style. This is an excellent way to quickly verify your records and to get an overview of the information that might be usefully included in your database. See the section *Output Formats/Listing Format* below.

Another useful thing RTP can do for you is look through every record in your MARC file and give you a compact summary of exactly what tags and subfields are present. From this information you will be able to determine useful features such as:

- the total number of records in the file
- which tags or subfields appear across the whole file
- which tags and subfields appear in every record, and which tags and subfields appear in more than one record
- the maximum number of characters needed to store the longest possible piece of information for each tag or subfield

Using the software

The program runs from a single command line and requires three parameters;

Parameter 1	a request file name or one of the keywords **-list** or **-statistics**
Parameter 2	the MARC file name
Parameter 3	the output file name (optional when extracting MARC data).

The command line will look like this:

```
marc parameter1 parameter2 parameter3
```

The program will output one of three different forms of output depending on what is given as parameter 1. The following subsections describe each of these.

Generating a listing

To list the MARC file *catalog.dat* and put the output into *catalog.lst* use one of:

```
marc -list catalog.dat catalog.lst
```

```
marc -l catalog.dat catalog.lst
```

Listing format

RTP listings are a human readable orderly format detailing each MARC record showing bibliographic tags, indicators, subfield codes and associated information content.

Sample listing

```
ST=n, TY=a, BL=m, EL= , DCF=a, LRR=
001    $          $abn92166827
008    $          $      s1992    vraa         f00000 eng d
019 1  $ a        $8994127
019    $ a        $abn92135252
020    $ a        $0646097237
035    $ r        $0646097237
040    $ acddd    $ADEET$ADEET$SUSA$NSL:M$WLB
043    $ a        $u-at---
082 0  $ a2       $374.994$20
245 00 $ abc      $Employment-related key competencies :$a proposal
                   for consultation /$the Mayer Committee.
260 0  $ abc      $Melbourne :$Mayer Committee,$1992.
300    $ abc      $71 p. :$ill. ;$29 cm.
500    $ a        $Cover title.
500    $ a        $"This paper presents a draft proposal for the set
                   of Key Competencies and the development of
                   nationally-consistent approaches to assessment and
                   reporting on young people's achievements in the
                   Key Competencies" -- p. 3.
610 20 $ abt      $Australian Education Council.$Review Committee.$
                   Young people's participation in post-compulsory
                   education and training.
650  0 $ az       $Post-compulsory education$Australia.
650  0 $ az       $Occupational training$Australia.
650  0 $ az       $Vocational education$Australia.
700 10 $ a        $Mayer, Eric.
710 20 $ ab       $Australian Education Council.$Mayer Committee.
```

The first line lists information contained in the US/MARC record leader. These are in order:

ST	Record Status
TY	Type of Record
BL	Bibliographic Level
EL	Encoding Level
DCF	Descriptive Cataloguing Form
LRR	Linked Record Requirement

Down the left side of the listing are the three digit MARC tag numbers followed by a space and then two characters of indicator characters. Separated from the tag numbers and indicator characters by a $ are the subfield codes that are present in the tag. Another $ starts the subfield data elements for each tag separated by a $ character.

```
082 0  $ a2       $374.994$20
```

Of course, the fixed fields don't have subfield codes.

Generating a statistics analysis

To produce a statistical analysis of the MARC file *catalog.dat* and put the output into *catalog.tbl* use one of:

```
marc -statistics catalog.dat catalog.tbl
```

```
marc -s catalog.dat catalog.tbl
```

Statistical format

As an aid in building your database, RTP can provide a simple analysis of the occurrences of tags and sub fields in the MARC file. For example, this information might allow the database designer to decide how many *General Note* (ie. tag 500) fields to provide in the database and the minimum size of the field required to hold the data. The output is a table listing of each tag, one per line, with the following information:

- **Tag**
  the MARC numerical code given to indicate specific type of information

- **Count**
  the total number of times this tag appears in the file

- **Min**
  the minimum number of times this tag may appear in a record,

- **Max**
  the maximum number of times this tag may appear in a record,

- **MaxAgg**
  the largest amount of space in characters required to hold all of the information associated with this tag in any record,

- **Subfields**
  for each sub field code that appears for this tag it gives:

  - **Code**
    the sub field code,

  - **Min**
    the minimum number of times this sub field code appears for this tag (across all repeated instances of the tag) in any record,

  - **Max**
    the maximum number of times this sub field code appears within this tag (across all repeated instances of the tag) in any record,

  - **Max Len**
    the length of the longest data element associated with this sub field code for this tag in any record

Sample statistics output

The sample below illustrates the output generated by the *statistics* request option of RTP.

```
Tag   Count Min Max MaxAgg  Subfields: format is code/min/max/maxlen ...
------------------------------------------------------------------------
001    2515   1   1     21
008    2515   1   1     41
010     129   0   1     47  a/1/1/19 z/0/2/12
015     623   0   1     27  a/1/2/20
019    5030   2   2     73  a/2/2/34 z/0/5/8
020    2723   0   5    150  a/0/5/46 c/0/3/36 b/0/1/6 z/0/2/25
022      21   0   1     14  a/1/1/9
030       4   0   1     11  a/1/1/6
032       4   0   1     17  a/1/1/6 b/1/1/4
035    2614   1   5    103  a/1/5/17 r/1/1/12
039     435   0   1     21  x/0/1/1 a/0/1/5 b/0/1/2 c/0/1/1 d/0/1/1 e/0/1/1
040     907   0   1     37  d/0/5/6 a/0/1/19 c/0/1/7 b/0/1/3
 ...
 ...
 ...
505      79   0   1   1361  a/1/1/1356
510      42   0  12    607  a/10/12/111 x/10/11/9 b/3/5/12
515       8   0   1     49  a/1/1/44
 ...
 ...
 ...
810      42   0   2    169  a/1/2/71 t/1/2/63 v/0/2/21 b/0/3/51 p/0/1/28
830     117   0   2    134  a/1/2/120 v/0/1/16 p/0/1/37 l/0/1/31
840       2   0   1     91  a/1/1/86 v/0/1/7
------------------------------------------------------------------------
The total number of MARC records read was 2515.
```

Where a tag or sub field minimum value is 0 (zero), it simply means that the tag does not appear in every record or the sub field does not appear in every occurrence of the tag. Conversely, if it is not 0 then it does appear in every record or tag occurrence respectively.

Extracting MARC data

To translate the MARC file named *catalog.dat* into the importable file or files with name suffix *.imp*, selected according to the requests specified in *myrequests* use:

```
marc myrequests catalog.dat .imp
```

Note that the third parameter is optional and may be used to specify a suffix for all output file names. The requests file will supply the first part of the filename, which is taken directly from the table names given there. If the third parameter is not included then no suffix will be appended and the table names in the requests file will be the complete file names for each table. See later under the heading *Table Definition*.

Importable data file format

MARC RTP can output extracted data in various ways determined by the requests file described later. Most commonly, the data will need to be output as **delimited text files**. This is a file where the individual pieces of information are arranged as separate fields grouped on one line. Each field is separated from the next by a special character, such as a comma or a tab. Each line in the file contains the same number of fields so that the file may be logically visualised as a table, with rows and columns, as illustrated below. However, the information in individual fields may be of varying length (number of characters). Most database software will read and ``import" data in this format.

record 1/field 1	record 1/field 2	record 1/field 3	record 1/field 4
record 2/field 1	record 2/field 2	record 2/field 3	record 2/field 4
record 3/field 1	record 3/field 2	record 3/ field 3	record 3/field 4

Example output record (in CSV format - comma separated values)

This following example shows one record of three fields separated by the default field delimiter, which is a single _tab_ character.

```
abn92166827tabMayer, Eric.tabAustralian Education Council.
```

The relationship between MARC records and importable output records

Through the requests file described in the remainder of this manual, the format of the output produced by MARC RTP can be manipulated to suit many different applications. For example, tag and/or subfield data extracted from each MARC record can be placed in separate fields in a single row in a single file, or in multiple rows in multiple files, or a mix of these.

The Requests file

Why do I need a Requests file?

- To select tags and fields for extraction.

  The tags and subfields contained in each MARC record differ widely between records. As well, you may only require some of the data available in each record. The requests file allows you to specify which fields and/or sub fields to take from each record.

- To control output structure.

  The final written output needs to be arranged and presented in a way which suites the system that will eventually import it, and so in addition to selection, the requests file also defines the format and structure of the resulting importable data files.

What is an extraction request?

The *extraction request* provides a generalised method of telling RTP what to look for in each MARC record. A request is a description of a particular field, or data element, in each MARC record which should be copied to the output.

A MARC record data element will have one or more of the following identifiable characteristics that can be described within any request:

- a bibliographic tag;

- a subfield identifying letter or digit used by variable tag data;

- a starting and ending character position range if subfields aren't used and the entire tag data isn't required. This is particularly useful in the case of fixed tag data where there are no subfields, the 008 tag for example;

- A specific instance reference. For example, when it is necessary to specify a particular instance of possibly several multiply occurring, or repeated, tags in the record.

Often MARC tag data is organised into separate subfields but is often displayed together as one item. Therefore, RTP allows a group of requests to be linked together such that the result of all of them is written to the output assembled as one data item.

As the complement of request linking, if a request fails to match any candidate data in a particular MARC record then alternative requests can be given such that RTP will attempt to match requests in turn until it finds some data. If ultimately no match is found then nothing will be written for that particular request group.

Linking and alternative requests are described in detail under the section *Composite Requests*.

Writing the requests file

The *Requests File* is a plain text file which can be edited using any editor capable of saving work in plain text format. The requests file provides a template for the selection of data elements from each record processed from the MARC file.

When the program is run, RTP reads a MARC record from the MARC file and then processes every request in the requests file. The next record is then read and the requests are re-applied. This process is repeated for every record in the MARC file.

The requests file has the following structure:

```
# A comment (will be ignored)

BEGIN

  DEFINE configurable-item IS value

  TABLE tablename IS (columnname1, ..., columnnameN)
    ROW IS
      columnname1:  request1   # Another comment (ignored)
      ...
      columnnameN:  requestN

    <More row definitions>

  <More table definitions>
END
```

First of all, as shown in the example above, any text from a # to the end of a line is a comment. Otherwise, the file has no special restrictions on style or layout. For example, the indentation shown above is not mandatory and is used only to make the file more readable by reflecting the hierarchical relationships between tables, rows and columns.

The definitions part of the requests file starts with the BEGIN keyword, and finishes with the END keyword.

Immediately following the BEGIN you may include any configuration definitions (DEFINEs).

One or more TABLE definitions may be included after any DEFINEs.

Each TABLE definition may include one or more ROW definitions.

Each ROW definition may be followed by one or more column definitions.

As MARC RTP reads the requests file, the following rules apply:

- Any text between a # character and the end of a line is taken as an annotation and is ignored.
- Blank lines are ignored.
- All other formatting such as indentation and line wrapping is entirely arbitrary.

Configuration definitions

RTP can be configured by including statements at the beginning of the requests file before the BEGIN word. Statements appear on separate lines beginning with the word DEFINE.

DEFINE configname IS value

The following supported configuration options can be included:

OUT-COLUMN-DELIMITER-CHARACTER

- column separator character. Default is <tab>. This character is automatically escaped in normal data.

e.g. DEFINE out-column-delimiter-character IS ","

OUT-ROW-DELIMITER-STRING

- row separator (or end-of-line) string. Default is a single newline character. All character in this string are automatically escaped in normal data.

e.g. DEFINE out-row-delimiter-string IS "%cr;%nl;"

OUT-ESCAPE-STRING

- character/s to insert immediately before any special character in the output. For example, if the character used as the field separator is found in the data it must be differentiated from the actual field separator. Default is "\".

e.g. DEFINE out-escape-string IS "&esc;"

OUT-ENCLOSE-CHARACTER

- enclose all extracted data between this character. Default is nothing. Sets both out-enclose-left and out-enclose-right to the same character.

e.g. DEFINE out-enclose-character IS "&dq;"

OUT-ENCLOSE-LEFT

- use this character as the left-hand enclosing character. This character is automatically escaped in normal data.

e.g. DEFINE out-enclose-left IS "`"

OUT-ENCLOSE-RIGHT

- use this character as the right-hand enclosing character. This character is automatically escaped in normal data.

e.g. DEFINE out-enclose-right IS "'"

OUT-SUBSTITUTE-CHARACTER

- whenever the first character is seen in the data, replace it with the second character.

e.g. DEFINE out-substitute-character IS "$" TO "#"

OUT-SUBFIELD-LEADER

- use the specified character instead of the default underscore (_) character to prefix subfield identifying characters in raw tag field output. This character is automatically escaped in normal data.

e.g. DEFINE out-subfield-leader IS "$"

For example, by including the two statements at the start of your requests file:

```
DEFINE out-column-delimiter-character IS ","
DEFINE out-escape-string  IS "/"
```

You are telling RTP that the output field delimiter will be a *comma* (,), and then, if a *comma* appears within the data itself, RTP must insert the single escape character / immediately before it. Additionally, if any of the characters in the escape string appear within the data, they will also be escaped.

For example, in the following examples, the input text on the left will produce the output on the right:

Input text	**Output**
`Cover title.`	`Cover title.`
`Mayer, Eric.`	`Mayer/, Eric.`
`a proposal for consultation /`	`a proposal for consultation //`

Table definition

The requests file may contain one or more TABLE definitions. All output generated by a TABLE is written to a file whose name consists of a common suffix as given by the optional third parameter on the MARC RTP command line and a first part which is given by *tablename*.

'*tablename*' may be either:

- a single word starting with a letter and consisting only of the characters A-Z, a-z, 0-9, or a -, e.g. `Table-1`, `Authors-and-Publishers.` This is the original style for table namesthat has been superceded by arbitrary strings in quotes as described below.

- any text enclosed in double quotation marks, e.g. `"C:\SQL Imports\Authors and Publishers"`. If *tablename* does not represent a valid filesystem path on the host system then an error will be returned.

The maximum number of tables you may have in your requests file will be limited by the number of files you may have open at the same time on your computer system.

```
BEGIN
TABLE tablename IS (columnname1, ..., columnnameN)
  ROW IS
    columnname1: request1
    ...
    columnnameN: requestN
END
```

Each table has a fixed number of columns determined by the number of column labels given within parentheses in the definition. The data extracted for each column will be written in the same order as given in this list which may be different to the order of appearance of the actual column definitions described later.

Row definition

Each *table* may have one or more ROW definitions. A row definition introduces and groups the column definitions for a new row in the table:

```
BEGIN
TABLE tablename IS (columnname1, ..., columnnameN)
  ROW IS
    columnname1: request1
    ...
    columnnameN: requestN
END
```

Optionally, you can use ROW IS REPEATED UNTIL <condition> to tell RTP to repeatedly process the row until a terminating condition is met. The only terminating condition available at this time is DONE. For example:

```
BEGIN
TABLE tablename IS (columnname1, ..., columnnameN)
  ROW IS REPEATED UNTIL DONE
    columnname1: request1
    ...
    columnnameN: requestN
END
```

Rows will continue to be output while any request in any column in the row definition succeedes in selecting new data from the MARC record. Consequently, a row with a REPEATED qualifier can produce zero or more rows of output while a non-repeated row definition will produce exactly one row of output, even if all columns fail to select any data.

See also the description of the REPEAT function under Qualifier functions.

Column definition

Each ROW definition may contain one or more COLUMN definitions and may define fewer columns than listed in the TABLE definition. A column definition begins with a column name matching any of the column names already defined for the table and is followed by a data extraction request. Column definitions may be listed in any order different to the order already specified by in the TABLE definition. Columns not defined for a row but defined for the table will be left empty when written to the output.

```
BEGIN
TABLE tablename IS (columnname1, ..., columnnameN)
  ROW IS
    columnname1: request1
    ...
    columnnameN: requestN
END
```

The alternative verbose style shown below is also permitted.

```
BEGIN
TABLE tablename IS (columnname1, ..., columnnameN)
  ROW IS
    COLUMN columnname1 IS    request1
    ...
    COLUMN columnnameN IS    requestN
END
```

Each extraction request may be composed of any of the following request types. Each of these is described in the next section. They are:

Request type	**Examples**
simple requests	
literal requests	`"Literal text"`
tag requests	`~550`
subfields	`~550:a`
fixed position ranges	`~LDR[5..23]`

composite requests	
linked request lists	`"This text " AND "goes with this text"`
alternate request lists	`~550:a OR "No 550:a subfield was found"`
located requests	
tag occurrences	`~550{2}, ~550{2}:a`
subfield occurrences	`~550:a{2}, ~550{2}:a{2}`
wildcard requests	
tag only wildcard	`~any`
tag and/or subfield wildcard	`~any:z, ~550:any, ~any:any`
index functions	
RECNO	recno
ROWNO	rowno
COLNO	colno
LOOPNO	loopno
LASTTAG	lasttag
LASTSUBFIELD	lastsubfield
deferred functions	
CHARACTERS-WRITTEN	DEFER characters-written LENGTH IS 5
qualifier functions	
REPEAT	repeat(~100:a)

Extraction requests

Please note: wherever an example output record is shown, the output field delimiter used is the default *tab* character displayed as ***tab*.**

Simple requests

Literal text request

The simplest request possible is a literal text request. A literal request is text that you want included in an output field that did not come from the MARC record. The text is contained within double quotes as shown below:

```
columnname: "Photograph"
```

This request results in the text

```
Photograph
```

being written to the output.

Tag requests

Use a MARC tag with a ~ (tilde) prefix if you want to extract all the information for that tag. For example, here are two tag requests:

```
columnname1: ~001
columnname2: ~008
```

Using the column requests above and a MARC record with the following tag and field information:

```
001    $           $abn92166827
008    $           $      s1992    vraa         f00000 eng d
040    $ acddd     $ADEET$ADEET$SUSA$NSL:M$WLB
043    $ a         $u-at---
700 10 $ a         $Mayer, Eric.
710 20 $ ab        $Australian Education Council.$Mayer Committee.
```

RTP will produce the following output record:

```
abn92166827tab      s1992    vraa         f00000 eng d
```

Multiple tag occurrences

Often a tag can appear more than once in a MARC record. To complicate matters further, the same tag will often appear a different number of times in different MARC records.

To make it possible to select more than one instance of the same tag, RTP automatically remembers where in the MARC record data was last selected for any given tag.

This makes it easy to select multiple occurrences of the same tag but, be warned, there are traps for the unwary. Fortunately there is also a way to avoid the traps. Later on, in the section *Requesting Specific Instances of Repeated Tags and Sub Fields*, you'll also see how you can extract any one of a repeated tag contained within the MARC record.

By simply including repeated requests in the requests file you can select multiple occurrences of the same tag. For example, given the following requests file:

```
columnname1: ~650
columnname2: ~650
```

RTP will select and write the first, and then the second, "650" tag field. Repeated requests don't have to be grouped together either, they can be scattered throughout the requests file.

Sub field requests

All fields other than fixed MARC fields contain subfields. For these fields, using a tag alone as a request will select all the data for that tag, including all subfield data and their subfield codes.

The data in any subfield can be extracted by appending the subfield code to the request tag. A colon (:) separates the subfield code from the tag, as shown below.

```
columnname1: ~500:a
```

In this example, only the data in the "a" subfield of the 500 tag will be extracted. The subfield code will not be included with the data.

Multiple sub field occurrences

As for simple tag requests, RTP remembers where in the MARC record a tag and subfield was last selected so that repeated instances can be selected by including repeated *identical* requests.

Note that RTP regards tags with and without subfield codes as distinct. Look closely at this next example. While you may think that the following requests would select from separate 500 fields, they will actually select information from the same tagged field in the MARC record.

```
columnname1: ~500:a
columnname2: ~500
```

The first output field will contain only the "a" subfield information (without the subfield code), while the second output field will contain the entire tagged field information including all subfield information, including the leading subfield codes.

To illustrate further, look at the requests below:

```
columnname1: ~040:a
columnname2: ~040
```

and the following MARC tag data.

```
040    $ acddd     $ADEET$ADEET$SUSA$NSL:M$WLB
```

RTP will output a record with the two fields:

```
ADEETtab_aADEET_cADEET_dSUSA_dNSL:M_dWLB
```

Note that RTP converts the normally unprintable MARC record subfield code leader character to _ by default. That is, subfield code character pairs are output as _a, where "*a*" is the subfield code letter from the tag field in the MARC record. A character other than _ can be output via the OUT-SUBFIELD-LEADER define.

Composite requests

Alternative (or OR) requests

Where it is possible that a request tag won't be found in the MARC record, it may be useful, or necessary, to extract some alternative information. This may be a different tag field or perhaps some literal text. To accomplish this, multiple requests may be provided, each separated by the keyword "OR". These *alternative requests* will be processed one at a time starting with the first and continuing until one can be satisfied. That is, by finding a matching tag in the MARC record. For example:

```
columnname1: ~020 OR ~035
```

In this example, if the MARC record has no "020" tag then RTP will look for the "035" tag data. If neither tag is found then an empty output field will result. If a record contains both tags then RTP will stop searching after it finds the "020" tag.

Alternative requests can include any type of request including literal text requests. So it may be useful to write the request above as:

```
columnname1: ~020 OR ~035 OR "No Information"
```

As might be expected, a literal text request will always be selected if encountered.

Linked (or AND) requests

Request linking makes it possible to combine data from separate requests into a single cell (same table, same row, same column) in the output.

Linked requests form a single request, which is only output if all of the requests in it succeed.

Each request in a linked request is separated from the next by the keyword "AND". You may leave the "AND" keyword out because RTP can deduce from the surrounding context that a group of requests forms a linked request.

For example, say you want to link data from two fields in the MARC record, separating them with a literal semi colon "/". The following equivalent requests are equivalent:

```
columnname1: ~650:a AND "/" AND ~650:z


columnname1: ~650:a "/" ~650:z
```

When applied to the following MARC tag data:

```
650  0 $ az        $Occupational training$Australia.
```

Will produce the single cell output shown below:

```
Occupational training/Australia
```

Linked (AND) requests within alternate (OR) requests

Linked requests can be built up into alternative requests using the "OR" keyword as before.

Remember, for a linked request to produce any output, all its individual requests must be produce output. Linked requests will often be used to build up alternative requests covering all possible permutations of subfields. For example, the example above would probably be rewritten as the alternative request below. Note the stylistic freedom allowed in the layout for easier reading.

```
columnname1:      ~650:a AND "/" AND ~650:z
               OR ~650:a
               OR ~650:z
```

This request will ensure that any MARC records containing a 650:a and 650:z subfield will produce the same output as the previous example, but those with only a 650:a or 650:z subfield will also be selected.

Linked requests are constrained

Linked requests allow concatenation of data from two or more individual requests into the same column of the same row (cell) of the output table.

If subfields are linked (with the same tag identifier), then it is usually expected that the data will come from subfields in the same tag field. For example:

```
     Title:  ~245:a AND ~245:b
```

MARC RTP versions up to and including version 1.4.7 had a misfeature that could cause subfields from separate fields that have the same tag identity, (eg. any 650 tags) to be linked together.

Version 1.4.8 added constraints on the selection of subfield data where the tag occurrence is not specified. For example, in the following linked request:

```
     ~650:a AND ~650:x AND ~650:z
```

each of the subfields 'a', 'x', and 'z', must exist in the same 650 field for the request to succeed (and output to be written).

Mixed linked requests

The individual requests in a linked request do not all have to be of the same tag identity, but can be mixed. For example:

```
~650:a AND ~700:a AND ~650:x AND ~700:b
```

will constrain the 650 requests to the same 650 field and separately constrain the 700 requests to the same 700 field.

Located requests

Requesting specific instances of repeated tags and sub fields

A particular occurrence of a tag or sub field within a MARC record may be requested by including instance numbers in the request, referred to here as *occurrence specifiers*.

Tag occurrence specifiers

Simply append the instance number in curly braces to the tag part of the request as shown in the following example:

```
columnname1: ~650{2}
```

This request has no subfield code. It will select the second (entire) 650-tagged field in each MARC record if there is one, including all subfield data elements and subfield codes present. That is, it would contain all the text (including any subfield codes):

```
_aCriticism.
```

Compare the above request, 650{2}, with the request:

```
columnname1: ~650{2}:a
```

This request will search for an "a" subfield from the second 650-tagged field in the MARC record. To illustrate, given the request above and the following tagged data:

```
650    $ axx       $Literature$History and Criticism$Theory, etc.
650    $ a         $Criticism.
```

the output field will contain the text:

```
Criticism.
```

Subfield occurrence specifiers

Instance numbers can also be used with sub fields by appending the number in curly braces to the sub field part of the request. To illustrate, look again at the MARC record fields:

```
650    $ axx       $Literature$History and Criticism$Theory, etc.
650    $ a         $Criticism.
```

and at what the following two requests select.

```
columnname1:     ~650:x{2}
columnname2:     ~650{2}
```

Notice that the first request has now selected the second "x" subfield from the *first* tagged field, while the second request (with no subfield part) yields the same result as for the earlier example.

```
Theory, etc.tab aCriticism.
```

Incorporating both tag and subfield occurrence numbers, the request below will request the *second* occurrence of an "a" subfield in the *second* 500-tagged field, if there is one.

```
columnname1:     ~500{2}:a{2}
```

Using the following MARC record fields:

```
500   $ aaa          $abc$def$ghi
500   $ aaa          $rst$uvw$xyz
```

the request:

```
columnname1:     ~500{2}:a{2}
```

will select the following data

```
uvw
```

RTP keeps a separate note of the **last selected** *tag* and **last selected** *subfield* occurrences of each **uniquely specified request** in the requests file. For example, the two requests: 500 and 500:a are **distinct,** and RTP will keep separate note of their last occurrence locations.

Fixed field and more general selections

Fixed MARC tags do not have subfields and contain information at specific character locations within the field. Examples of fixed fields are the 006, 007 and 008 fields and also the record leader information. Other MARC tags also contain **indicator-fields** which may need to be extracted.

To allow this information to be extracted, numbers representing the number of characters to count up to, but not including, the first and last character of the desired information can be included in the request.

The form of the fixed field request is shown in this example:

```
columnname1:     ~008[7..14]
```

In this request the tag is followed by the character range specifier. In the example, the "7" and "14" are interpreted as the number of characters to count **before** coming to the first and last characters respectively of the information required. That is, all characters starting from the 8th through to and including the 15th will be extracted.

Single character information can be extracted using either of the following forms.

```
columnname1:     ~008[6..6]
columnname2:     ~008[6]
```

That is, a single offset specifier is equivalent to having the same first and last number.

As we have just shown, this request format is very general in that it can be applied to any MARC field, not just the fixed fields. However, range specifiers cannot be used in any request that includes subfield specifiers. For example, you cannot have the request 035:a[3]. You can, however, combine them with a tag occurrence specifier as described in the section *Extracting Variable Tag Field Indicators* below.

Extracting information from the record leader

Since the leader does not have a tag like other MARC fields do, RTP recognises the special pseudo request tag LDR. This tag can be combined with character range specifiers as for fixed fields. For example, the following request will extract the ST Record Status value, which is the first character of the Leader field in each MARC record.

```
columnname1:     ~LDR[5]
```

Extracting variable tag field indicators

MARC records allow for indicator characters to appear at the beginning of every variable tag field. To extract these you use range specifiers with tag requests. For example, because the indicators occupy the first two character positions in the tag field, the following consecutive requests will extract the *first* and *second* indicators from the 035-tagged field.

```
columnname1:     ~035[0]
columnname2:     ~035[1]
```

As before, where a tag appears multiple times within a MARC record, occurrence specifiers may be combined with the request. The same rules apply to occurrence specifiers here as they do elsewhere. These rules are described in detail under the section *Selecting Particular Tags or Sub Fields from Multiple Occurrences*. Also as before, repeated *identical* requests can be used to select subsequent occurrences of the same tag.

The following example will select the *first* indicator from both the first and second 082-tagged field in each record.

```
columnname1:     ~082[0]
columnname2:     ~082[0]
```

And just to show the flexibility of occurrence specifiers once again, the following example will reverse the order of the tags. That is, the first request will select the given indicator from the *second* 082 field, and the following request will then select the corresponding indicator from the *first* 082 field.

```
columnname1:     ~082{2}[1]
columnname2:     ~082{1}[1]
```

Wildcard requests

A *wildcard* request matches any tag or subfield or both. For example:

```
~any
```

Will select the next unselected tag starting the search from the last selected tag. Similarly,

```
~550:any
```

Will match the next unselected 550 tag subfield. And finally,

```
~any:any
```

Matches the next unselected tag:subfield.

This is a very powerful request as the following example shows. This request file will dump an entire file of MARC records into a single table as raw tag fields in a single column:

```
BEGIN
TABLE everything IS (data)
  ROW IS
    data:  ~ldr[5..23]       # The record leader
  ROW IS REPEATED UNTIL DONE
    data:  ~any              # All other tag fields
END
```

The resulting output will look something like this, with each tag field on a separate row (line):

```
nam  2200265 a 4500
abn92166827
      s1992    vraa         f00000 eng d
1 _a8994127
  _aabn92135252
  _a0646097237
  _r0646097237
  _aADEET_cADEET_dSUSA_dNSL:M_dWLB
  _au-at---
0 _a374.994_220
00_aEmployment-related key competencies :_ba proposal for
consultation
0 _aMelbourne :_bMayer Committee,_c1992.
  _a71 p. :_bill. ;_c29 cm.
  _aCover title.
  _a"This paper presents ... in the Key Competencies" -- p. 3.
20_aAustralian Education Council._bReview Committee.
 0_aPost-compulsory education_zAustralia.
 0_aOccupational training_zAustralia.
10_aMayer, Eric.
20_aAustralian Education Council._bMayer Committee.
```

This particular output is not very useful, as it doesn't include the tag information along with the data. It is also arranged vertically in a single column, which once combined with the data from other MARC records will be even less satisfactory. What is needed when arranging data in a single column is indexes. If we could add two more columns to the above example, one to contain information about which MARC record the data belongs to and another for the tag the data belongs to, then we would have a very usable database. The capability is what is described in the next section.

Index functions

As you saw above in the section on wildcard tags, an ability to include indexing information alongside MARC data is essential in many applications.

Index functions provide access to various internal MARC RTP counters and other information. Index functions can be used wherever a request is accepted, including within composite requests.

The following counters are available:

RECNO	- returns the sequence number of the current MARC record being processed.
ROWNO	- returns the sequence number of the current row generated from the current MARC record for the current table.
COLNO	- returns the sequence number of the current column (in process order, not table order) for the current table.
LOOPNO	- returns the sequence number of the repeated row generated from the current MARC record for the current table.

The following information can also be accessed:

LASTTAG	- returns the three-digit tag of the last selected tag or subfield request.
LASTSUBFIELD	- returns the single character subfield of the last selected subfield request, or "-" if the last selected request was a tag request without a subfield.

Currently there exists a simple formatting capability for integer index functions. You can add a format control string in parentheses appended to the index function name. The format control string specifies the number of character positions that the number will occupy when output, whether the number is right or left justified, and whether the number is padded with zeros on the left. For example:

Function	**Value**	**Output**

RECNO("%d")	99	`99`	equivalent to just RECNO
RECNO("%5d")	99	`   99`	right justified, space padded
RECNO("%-5d")	99	`99   `	left justified, space padded
RECNO("%05d")	99	`00099`	right justified, zero padded

Here's an example. The requests file:

```
BEGIN
TABLE everything IS (record, tag, data)
  ROW IS
    record:      RECNO("%05d") # Current MARC record number
    data:        ~ldr[5..23]   # The record leader
    tag:         LASTTAG

  ROW IS REPEATED UNTIL DONE
    record:      RECNO("%05d")
    data:        ~any          # All other tag fields
    tag:         LASTTAG       # Ask RTP for the tag info
END
```

Produces the output below ("...") replaces some rows for clarity):

```
00001   LDR     nam  2200265 a 4500
00001   001     abn92166827
00001   008           s1992    vraa         f00000 eng d
00001   019     1 _a8994127
00001   019       _aabn92135252
00001   020       _a0646097237
00001   035       _r0646097237
00001   040       _aADEET_cADEET_dSUSA_dNSL:M_dWLB
00001   043       _au-at---
00001   082     0 _a374.994_220
00001   245     00_aEmployment ... :_ba proposal/_cthe Mayer
Committee.
...

00002   LDR     nam  2200253 a 4500
00002   001     anb73062460
00002   008           s1992    vra          f00010 eng d
00002   019     1 _a8680093
00002   019       _aabn92026929
00002   020       _a0730624609 :_cprice unknown
00002   035       _r0730624609
00002   040       _dNU
00002   043       _au-at---
00002   082     04_a374.994_220\\
00002   245     00_aEmployment ... and training :_ba discussion paper
/
...
```

Defer function

The DEFER function is activated at the conclusion of processing the current MARC record for the current table. The general format of each function is:

```
columnname1:     DEFER function LENGTH IS n
```

Where *n* is the number of characters that will be required to hold the value returned by *function*.

MARC RTP records the position at which the functions returned value must be placed in the output and ensures that the requested space in the output is reserved for it. The DEFER function, like index functions, can be used wherever a request is valid, including within composite requests.

There is only one function available at this release of MARC RTP (version 1.4):

CHARACTERS-WRITTEN

- return a count of the characters written to the current table for the current MARC record. The count does not include end-of-line characters (the new-line character on Unix systems or the carriage-return and new-line pair on MS DOS and MS Windows systems).

  e.g. DEFER characters-written LENGTH IS 5

In the example below, the deferred function is used to insert in the first row a count of actual characters output for each MARC record.

```
BEGIN
TABLE everything IS (tag, data)
  ROW IS
    data:  DEFER characters-written LENGTH IS 5
       AND ~ldr[5..23]       # The record leader
    tag:   LASTTAG           # Ask RTP for the tag info

  ROW IS REPEATED UNTIL DONE
    data:  ~any              # All other tag fields
    tag:   LASTTAG           # Ask RTP for the tag info
END
```

In the example above the number of characters written for each MARC record is to be inserted at the front of the leader information. This is shown below in **bold** font. Because it isn't possible to know the total number of characters written until all of the rows associated with a record have been output, MARC RTP must remember where in the output the number must eventually go, and then return to update it at the end of the record. This is why it is termed a *deferred* function.

```
LDR     00942nam  2200265 a 4500
001     abn92166827
008           s1992    vraa         f00000 eng d
019     1 _a8994127
019       _aabn92135252
020       _a0646097237
035       _r0646097237
...

LDR     00707nam  2200253 a 4500
001     anb73062460
008           s1992    vra          f00010 eng d
019     1 _a8680093
019       _aabn92026929
...
```

Qualifier functions

There is currently only one qualifier function, which is the *repeat* function.

The *repeat* function is used with a single request and causes RTP to not record the occurrence of the selected data. Consequently, when the request is repeated for subsequent rows in the same table, or in a ROW IS REPEATED UNTIL <condition> set, the same data will be selected for each subsequent row. In the following example, tag 100:a data is used as a key value to identify data from the same MARC record.

```
BEGIN
TABLE everything IS (key, data)
  ROW IS
    key:   repeat(~100:a)
    data:  ~ldr[5..23]       # The record leader
  ROW IS REPEATED UNTIL DONE
    key:   repeat(~100:a)    # The same tag and field as
                             # above used in each new row.
    data:  ~any              # All other tag fields
END
```

Alternative (or OR) requests revisited

Now that all of the options relating to requests have been described, a recap of the method of specifying alternative request tags in a single request is appropriate. Now a complete request tag can include the specification of its occurrence.

A single request can look like any of the following.

```
columnname1:     "This is some text"
columnname2:     ~500
columnname3:     ~500:a
columnname4:     ~500{2}
columnname5:     ~500:a{2}
columnname6:     ~500{1}:a{2}
columnname7:     ~650:a "/" ~650:x
columnname8:     ~LDR[5]
columnname9:     ~082[1]
columnname10:    ~082{3}[1]
columnname11:    RECNO
```

A complex request might look like this:

```
columnname1:       ~500:a AND "/" AND 500:b
                OR ~500:a
                OR ~500:b
                OR ~500      # Fall back to the whole field
                OR "No Information"
```

Or by leaving out the ANDs, which are implied when omitted, the following is equivalent:

```
columnname1:       ~500{2}:a{3} ":" 500{2}:a
                OR ~500:a
                OR ~500      # Fall back to the whole field
                OR "No Information"
```

Tips and hints

Assistance in designing your database

Remember that RTP provides both full listings and some basic statistics from the MARC file. These can be obtained before writing a requests file.

The statistics will help you to plan your database by giving you information about the number of occurrences of tags and the size of the tag and sub fields within the MARC file records.

When you add to or modify your database

Most database packages allow merging of databases and so new MARC records can be imported by simply re-running RTP with the same requests file over the additional MARC file.

If you add, delete or re-arrange fields within your database and want to import new data from the MARC file, you will probably have to reflect those changes in the requests file.

Be as specific as possible

Whilst RTP tries to interpret your intentions as naturally as possible, specifically with respect to repeated requests, you will be less prone to errors if you use occurrence specifiers in requests wherever possible.

Who is NicheWare?

NicheWare is two people:

Lynnette Taylor
Information Management Consultant

Ross Johnson
Consultant Programmer
eMail: rpj@ise.canberra.edu.au