MARCView and MARConvert:

Documentation for the open-source version

June 18, 2009
Revised June 22, 2009

MARCView and MARConvert were developed by

Stephen Toney
Systems Planning
4915 Redford Road
Bethesda, MD 20816 USA
(301) 652-1231
info@systemsplanning.com
http://systemsplanning.com

Table of Contents

INTRODUCTION 6

The Code 7

Tools 7

Additional Information 7

OVERVIEW 9

Definitions 9

Structure of Code 9

MARCView 9

SP_MARC 10

Important Variables 10

Retrieving Records, Fields, and Subfields 10

Return Codes 11

Results Messages 12

Returning Longs from SP_MARC 12

Searching: MV/MC 12

Normalization 13

Highlighting search results 13

Settings 13

Character sets 13

Maximum records 14

Maximum records in Navigation Grid 14

Navigation Grid columns 14

Unblock 14

Statistics 14

Basic 14

Leader 14

Extended statistics (future) 15

Character Sets 15

Character set for display 15

Displaying Unicode 16

Record Terminator Detection 16

For client LLL 16

For MV 17

Online Help 17

Context-sensitive help 17

UNIMARC Support **17**
MARCXML **18**
Reading records 18
Character conversion 18
Displaying records 18
Statistics 19
MARCSQL 19
Splash Screen **19**
MARCONVERT **20**
Conversion Policies **20**
Command-Line Functionality **20**
Client-Specific Processing **21**
New Primitives **21**
MCClient **21**
Interface **22**
Browse button 22
Import **22**
How import works -- tagged-text files 22
How import works -- delimited files 23
Import Translation Table 23
Fixed-field issues 24
Import translation nodes 24
Trans Index 24
More on Constants and Attachments 25
Constant problems (pun intended) 26
Indicators 26
Repeating fields or subfields 27
008 generation 27
Export **27**
2003 Modifications 27
How export works 28
Delimited files 28
Multiple output files 28
Multiple output files for MMM 29
Translation nodes 29
Export Translation Table 29
Build button 30
Constants 30
Writing to a database 31
Loading with ODBC 31
Bulk loading 31
Summary of parameters 31
MarcToMarc Conversion **32**

MV/MC CHARACTER CONVERSION....33
Goals....33
Subroutines....33
Conversion Methods....33
Method 1: Symbols.cpp....33
Method 2: Codetables.xml....34
Implementation considerations....34
Implementation....35
Use of Symbols.cpp for MARC-8 and UTF-8....35
MARC-8 to UNIMARC....35
MARC-8 to UTF-8....35
Symbols....36
Codetables....36
UTF-8 to Latin....36
UTF-8 to MARC-8....36
Symbols....36
Error Handling....37
MARCView....37
MARConvert....37
MC/MV DEVELOPMENT....38
Problems....38
CLIENT-SPECIFIC NOTES....39
How to Add a New Client....39
AAA....39
Purpose....39
BBB....39
Purpose....39
Releases....39
CCC....40
Purpose....40
Releases....40
DDD....40
Purpose....40
EEE....40
Purpose....40
Releases....40
Catalog data....41
Copy data....41
Implementation notes....41

FFF 43
Purpose 43
Problems 44
Releases 44
GGG 44
Purpose 44
HHH 44
Purpose 44
JJJ 44
Purpose 44
KKK 44
Purpose 44
Releases 44
LLL 45
Purpose 45
Releases 45
MMM 45
Purpose 45
Releases 45
NNN 45
Purpose 45
PPP 46
Purpose 46
QQQ 46
Purpose 46
RRR 46
Purpose 46
SSS 46
Purpose 46
EXPERIMENTAL FEATURE 47
MARCSQL 47
Query syntax 47
Other features 48
Implementation 48
Multiple-term searches 48
Nesting 49
Future ideas 49
KNOWN PROBLEMS 50
MARC-TO-MARC 50
Nodes being wasted 50
INSTALLATION 50

MARCSQL.....50

Introduction

MARCView and MARConvert are two products generated from the same C++ codebase. The IDE used to develop the code was Sybase Optima++ 1.5. We offer this code and documentation to the community free and without obligation, under the Apache 2.0 license.

MARCView is an easy-to-use Windows program to view, search, and print any file of MARC21, UNIMARC, or MARCXML bibliographic or authority records. Records are formatted for easy viewing. Although MARCView handles files of any number of records, searching is linear, so becomes slower for large files; about a million records is the practical limit for searching. The size of the file must be less than 2 GB.

MARCView a simple file viewer. As such, it tries to read anything readable and does not ask the user much. For example, it ignores characters between records if it can. It does not report character conversion problems since the user can't do anything about them and they won't interfere with viewing. It does not report minor validation errors.

This simplicity means that certain limitations are hard to fix.

- Whole-word searching is hard because MV just looks through the records as converted to plain characters. Word-ending characters such as spaces, punctuation, delimiters, etc. are not used in searching.

- Highlighting is not perfectly consistent with the search; see **Highlighting search results**.

MV can read records in MARC-8, UTF-8, or Latin-1 character encodings; although Optima in theory can handle UTF-8, we could never get it working; therefore MV converts all characters to Latin-1 for display; thus the display characters are limited to that set.

MARConvert is a Windows application used to make custom conversions, especially when other programs were not customizable enough. Typically Systems Planning would modify the code for each client. Since we could modify the source code, virtually any kind of conversion was possible.

MARConvert has been used for various kinds of conversions:

1. importing records into MARC from text files, including those output from Ovid, Techlib, etc..
2. exporting records from MARC to text files or to update an ODBC database.
3. converting character encodings of MARC records
4. converting from MARC21 to UNIMARC
5. creating blocked records
6. unblocking RLIN files

MC has been customized for approximately 20 clients over the past few years. Most of these customizations have been highly idiosyncratic -- otherwise other solutions may have been preferred to MC. However, we want to enable you to build on our work, and at the same time protect the privacy of our clients. Therefore we have referred to clients by codes such as "AAA", "BBB", etc., both in the programs and documentation. In addition, this documentation describes what each client required for their conversion, so that you can use as a starting point the requirements most like your own.

The Code

The MARCView and MARConvert products are built from a single codebase. The code may be compiled for either MARCView or MARConvert based on **#define MARCONVERT** in the Application class. MARConvert includes just about all the features of MARCView, plus of course the conversion features.

In this document we refer to MARConvert (or MC) when we are discussing features specific to conversions. We refer to MARCView (or MV) when we are discussing searching and display features (which are shared by both products).

The executables consist of MARCView.exe for the interface and control, and SP_MARC.dll with functions to handle the MARC record. If we compile with **#define MARCONVERT** on, we rename the MARCView.exe to MARConvert.exe before delivery.

MARCView was converted from Delphi Pascal to C++ in 1996. As with any codebase that is 13 years old, it is not uniform in approach, techniques, or quality. Stability was preferred to elegance, so working code was left alone unless new requirements required refactoring. Therefore, techniques may be described herein that have not yet been uniformly applied to the code, but were intended to be applied during refactoring.

Tools

These products were built under and rely on Sybase Optima++, which consists of a C++ compiler and library. Although no longer supported, Optima++ (later renamed Power++) is an excellent environment for building C++ code quickly; it has been compared favorably to Delphi. The specific version used at Systems Planning is "Optima++ version

1.5 Professional Build 1129".

We have written several times to Sybase to ask if this product is still under copyright; if it were not, we would provide it for download. Sybase has never responded in any way. Therefore we cannot offer an opinion on whether you would be allowed to install Optima legally.

Some useful sites for Optima are:

- www.neatech.ch/powerpp/
- www.neatech.ch/powerpp/ListArchive/

(As of June 22, these sites do not seem to be working. They were up until a few months ago.)

In addition, we have extensive notes about using Optima that we can make available to those who need it. Contact us at info@systemsplanning.com.

The Export class also uses one class from the Virtual Windows Class Library (VWCL) to sort a linked list of strings. We believe this is no longer supported, but the portions used by MV/MC are quite simple.

MC and MV installation files are built using the open-source NSIS Nullsoft installer program.

The online help is an HTML file.

Additional Information

See the Systems Planning website at systemsplanning.com for information about MARCView and MARConvert capabilities.

MARCView was first developed in 1992 as a Pascal program for DOS. It was converted to Windows and C++ in 1996. MARConvert was first developed in 1999. Both products were enhanced and maintained until 2009, when the source code was given to the community.

We've had a lot of fun over the years, trying to fit new functionality into code that was never designed for it, and still trying to keep the kludges to a minimum! We recognize that there is lots of suboptimal code in the MARConvert functions, but (1) it works; (2) we were always working under time and budget constraints; (3) nearly always there were no modifications for a client after the initial delivery, so refactoring would have been pointless; (4) clients had the strangest requirements sometimes, so uniformity was not always possible. We hope you have as much fun and success with it as we did!

Note to Systems Planning clients:

We will continue to honor our support policies. For MARCView, we will support release 3.12 for at least two years after purchase. For MARConvert, support will be provided until five years after initial delivery. All support is free.

We thank all our clients for their business and suggestions over the years. Sixteen years is a long run!

We will continue to provide ways to search and view MARC files through our MWeb products. Please see systemsplanning.com/mweb/universal.asp

Overview

Definitions

Term	Product	Definition
Bad Record	MC	A record that cannot be processed.
Conversion*	Both	Generally, conversion of characters from one encoding to another, but sometimes used like "translation"
Detail Field	MC	An Export field that is output to a secondary table
Detail File	MC	An Export file holding many-to-many relationships (formerly called an aux file)
Export	MC	Conversion of records out of MARC
Import	MC	Conversion of records into MARC
MtM	MC	MARC-to-MARC; either a conversion from one MARC format to another, or a translation of character encodings while maintaining the same format, or both at the same time.
Pivot Field	MC	The Export field used as the foreign key to secondary tables
TLL	MC	Translation Linked List -- the internal representation of the TT
Translation*	MC	Generally, conversion of records from one format to another, but sometimes used like "conversion"
TT	MC	Translation Table, a text table used by MC for mapping between record formats

* Unfortunately we have not been consistent over the years in the use of the terms "conversion" and "translation". We apologize for any confusion.

Structure of Code

There are two modules: MARCView and SP_MARC.

MARCView

MARCView consists of about 15 classes. Some are extremely large. Most are more procedural than object-oriented. This module provides all the user-interface functionality.

In Optima, classes have the extension "wxc", unless the class has an on-screen form, in which case it has the extension "wxf". For example, MainForm.wxf has a form, but Utility.wxc does not.

The main classes for MV/MC are applic.wxc for the application, and MainForm.wxf. Most functions and variables that control the application as a whole are defined in those two classes.

The BibRec class provides functions for handling records, many of which are performed by calls to SP_MARC.dll.

Other important classes are Import, Export, and MarcToMarc, which include most of the MARConvert functionality.

SP_MARC

SP_MARC (the DLL code) provides MARC-record functionality. It consists of exported functions (all prefixed with "SP_"), plus some internal functions, global variables, defines, macros, etc.

SP_MARC was C (not C++) for most of its life, so is coded rather differently. However, it was converted to C++ a few years ago to take advantage of the Optima string class and other conveniences.

SP_MARC includes sp_val.cpp, which provides basic validation of MARC records, including sending messages about problems.

SP_MARC also includes Symbols.cpp, which provides crosswalks between several character encodings.

Important Variables

SP_MARC includes data structures for the current MARC record, field, and subfield, so calling functions do not need to store or pass these unless they want to modify them.

Badcount should be incremented only for records with problems the program cannot fix. In other words, a "bad record" is one that has or may have incorrect data in the output. For example, removing a diacritic before a 1E or 1F should **not** increment the badcount because MC has corrected the data (but should still generate a warning).

Retrieving Records, Fields, and Subfields

MV uses the sequential record number to identify a record. When a file is opened, SP_UseFile builds an array of record numbers and their offsets. This permits instantaneous navigation to any record. However, the array size must be fixed when MV is started.

MC doesn't count or scan the file before processing; this permits it to convert any number of records (up to the 2 GB limit on file size). So MC accesses each record in sequence by its offset instead of its position.

SP_MARC.dll has global variables to store the current MARC record, field, and subfield. A field can be obtained only after its record is read; likewise a subfield can be obtained only after its field is read. Only one record at a time is stored.

Here is how to use the functions in SP_MARC to do some common tasks. All tags can include the wildcard character as in 1xx, x10, etc.).

- To get first instance of specific field:
  ```
  SP_ResetRecord
  SP_Field("650", ppfld, plen, pptag, convert)
  ```
- To get next instance of specific field
  ```
  SP_Field("650", ppfld, plen, pptag, convert)
  ```
- To get next field
  ```
  SP_Field("xxx", ppfld, plen, pptag, convert)
  ```
- To get first field
  ```
  SP_ResetRecord
  SP_Field("xxx", ppfld, plen, pptag, convert)
  ```
- To get first instance of specific field+subfield
  ```
  SP_ResetRecord
  SP_Field("650", ppfld, plen, pptag, convert)
  SP_Subfield('z', ppsubf, plen, char, convert)
  ```
- To get first instance of specific subfield in current field
  ```
  SP_ResetField
  SP_Subfield('z', ppsubf, plen, char, convert)
  ```
- To get next instance of specific subfield in current field
  ```
  SP_Subfield('z', ppsubf, plen, char, convert)
  ```
- To get next subfield in current field (actually this gets the entire field from the current position of subfieldoffset)
  ```
  SP_Subfield('$', ppsubf, plen, char, convert)
  ```
- To get first subfield in current field (actually this gets the entire field)
  ```
  SP_ResetField
  ```

```
SP_Subfield('$', ppsubf, plen, char, convert)
```

- To get first subfield in next field (actually this gets the entire field)
```
SP_Field("xxx", ppfld, plen, pptag, convert)
SP_Subfield('$', ppsubf, plen, char, convert)
```

Return Codes

Throughout the codebase, the following return codes are used:

1	Perfect success
0	Benign exception (such as a normal end of file)
<0	Processing problems (see list below)
>1	Validation problems (2 is minor, 3 is fatal). The specific problems are passed using a message

This enumeration is unfortunate, as codes cannot be checked for problems with a single comparison such as

```
    if result < 0)
```

Instead you must do

```
    if ((result < 0) || (result == 3))
```

Here are the possible codes:

E_FATALVALERROR	3	// file ok but record has fatal problem
E_MINORVALERROR	2	// file ok but record has minor problems
E_PERFECTSUCCESS	1	// both file and record ok
E_EXCEPTION	0	
E_GENERAL	-1	
E_FILENOTFOUND	-2	
E_FILENOTOPEN	-3	
E_UNEXPECTEDEOF	-4	
E_NONMARC	-5	
E_MORETHANMAX	-6	
E_FUTUREFEATURE	-7	
E_NOMEMORY	-8	
E_SECURITY	-9	
E_CONVERSION	-10	
E_BLOCKED	-11	
(no name)	-50	//in sp_subfield, unknown charset

There is another variation in SP_UseFile; it returns 1 if the file is MARC21 and 0 if UNIMARC.

Results Messages

In order for MV/MC to display a message generated in SP_MARC, it passes a string-reference (WString&) to the DLL subroutine and lets the subroutine add whatever it needs to. This is known as the "rMessage technique".

As code is refactored these string-references are added as the first parameter in all calls.

A similar technique is used for messages from MC subroutines Export, Import, and MarcToMarc. However, in these cases the program may stay in the MC subroutine during processing of the entire file; this would result in a huge string with all error messages for the file. In this case we print the relevant messages at the end of each record and then clear the message string.

Returning Longs from SP_MARC

In order for MV/MC to receive a long integer value from SP_MARC, it defines the long and a reference and passed the reference to SP_MARC. For example, the caller should define both the long and the reference as follows:

```
long         somelong;
long&        rsomelong = somelong;
```

SP_MARC subroutines set the value of the reference at end of processing, like this:

```
rsomelong = anotherlong;
```

Searching: MV/MC

Formerly, a search on 100$aJohn would find John anywhere in the 100$a, but $aJohn will find John only as first word. This anomaly was solved by this new algorithm:

- We noticed that (1) searching specifying a subfield and tag=xxx is about 100 times faster than (2) searching specifying a subfield but no tag. Since case 1 avoids the anomaly, MV now supplies tag=xxx for case 2, converting it to case 1, thus eliminating the anomaly and speeding up the search by many times.

- Likewise, the same holds true when neither tag nor sfc is specified. searching for a value with tag=xxx is much faster than searching without a tag. thus we supply tag=xxx if none specified. This means the leader and directory are no longer searched. The leader can be searched by specifying it as the tag, but the directory can no longer be searched at all.

- The reason for this speedup is unclear. It would appear to be doing more work, not less.

Normalization

MV does not normalize the user's search terms, such as removing leading/trailing blanks, removing punctuation, or removing diacritics. Because MV searches by doing a substring comparison, the user's search term has to include the **punctuation** that is in the record or there will be no match. This can be a positive feature, depending on what the user is looking for: "Japan :" may be exactly what the user is trying to find.

MV does, however, provide a way to ignore **diacritics** in the record by converting it to its plain equivalent before trying to find the search term. This means that if the user enters diacritics in the search term, the term is not found.

Also it offers the option to ignore **case**, as Optima offers this natively.

Possible future features might be to permit diacritics in the search term, to ignore punctuation, to ignore multiple embedded spaces, etc. The NACO normalization rules (www.loc.gov/catdir/pcc/naco/normrule.html) might be a good starting point.

Highlighting search results

When a search term is found, it is highlighted everywhere it appears in the record; this is a feature. When the text in the record contains diacritics in cannot be highlighted, so MV displays a "found xxx but cannot highlight" message.

If the user searches for a tag and/or subfield with no term, we cannot highlight the tag or subfield code. Likewise with a term AND a tag or subfield, we highlight the term but not the tag or subfield code. We considered trying to highlight the tag or subfield code in these two cases, but there are these problems:

- The record is already displayed before the highlighting occurs; highlighting is done on the RecordView richtext, which has no idea of syntax. So trying to highlight a tag like "650" would also highlight "650" when in occurred in field data. Not so bad, but for subfield code $a, it would highlight every "a" in the record.
- If we preceded the subfield code with the subfield delimiter and then highlighted, it would be OK if there were no tag specified; but if the user specifies the 500$a, s/he doesn't want to see the $a subfields in other tags highlighted.
- If there was a term and subfield but no tag specified, the user doesn't want to see every $a highlighted, just the one containing his term. This would be impossible to achieve.

Settings

Character sets

(See more below under **Character Sets**)

The user can choose from four choices of input character set:

- Read each Leader byte 09 to see what charset each record uses. Leader/09 is blank if the charset is MARC-8 and 'a' for UTF-8. This the default because it handles both legal charsets. The user can override this to force the entire file to be treated as:
- MARC-8 (applies to entire file)
- Latin-1 (applies to entire file)
- UTF-8 (applies to entire file)

Note that if Leader/09 is chosen, every record must be checked as a file may contain records of both charsets; although we later found out this is supposedly invalid, MV already handled it so it was left in.

This setting does not apply to UNIMARC files, which use the UNIMARC charset.

This setting does not apply to MARCXML files, which use the UTF-8 charset.

Maximum records

The maximum records to view sets the size of the array of offsets which is created by scanning the file when it is first opened. This is pretty much irrelevant now, as everyone has enough memory to store an array big enough for millions of records.

The user's input is checked for non-numeric input or out-of-range values. If detected, the default is used -- with no warning.

Maximum records in Navigation Grid

When MV displays a MARC file, it creates a Navigation Grid which is clickable and scrollable. This is a costly procedure, and becomes slower as the number of records grows (it is an Optima built-in class, so we can't fix this). The practical limit is around 5,000 records.

The user's input is checked for non-numeric input or out-of-range values. If detected, the default is used -- with no warning.

Navigation Grid columns

Users can specify which columns they want to see in the Navigation Grid. They may specify different columns for MARC21 bib and authority records, and UNIMARC bib and authority records. MARCXML files use the MARC21 settings.

Unblock

This function creates a file of standard MARC record from an RLIN snapshot tape. The RLIN online documentation is contradictory about whether a block can contain parts of more than one record. A note in the code says they may, but MC doesn't handle it yet because no test data was available.

This is NOT the same format as LC's segmented record format, which uses blocks of 2048, whereas RLIN's are 2048 data plus 32 for block data, or 2080.

Statistics

Basic

The SP_Statistics function uses an array of longs to store its results. These are processed by the caller (MV) to produce the report. Works on MARC21, UNIMARC, and MARCXML.

Leader

000503 Added statistics from the leader. These are counts of the number of records with each value in bytes 5 - 9 and 17 - 19, eight in all. Some of them take numerous values, both numeric and alpha, plus the space. Alpha characters are all lowercase at present. To allow for all 37 values, and avoid problems when the range of values changes, the program creates a matrix for the eight values that allow them to take any of the 37. This is simply an array of 296 longs (8 x 37). A space is output to the user as #.

SP_Statistics could send back only the non-zero values, but this would mean identifying each one with its byte position and code. It is easier to pass all 296 to MV in a canonical order. (Note, this is different from the way field statistics are passed, in which only those fields in the file are passed.)

Statistics ignores illegal codes (anything other than the space, 0-9, and a-z).

Extended statistics (future)

Subfield-level statistics are a hard problem, especially for MARCXML files. The goal would be to provide the same information for subfields as MV does for fields. This is far more complex, as there are thousands of combinations, so an array or linked list could not be searched sequentially.

Some ideas for data structures for extended statistics are:

1. For numeric tags, an array of 36,000 records (one for each possible subfield code in each tag), each record consisting of
   - count (long)

- min length (int)
- max length (int)
- mean length (int)

-- for a total of 360,000 bytes

This array would be accessed using the tag plus the value of the subfield code (normalized so that sfcs 0-9 are 0-9, and a-z are 10-35).

2. For nonnumeric tags, an array or linked list of records, each record consisting of
   - the tag and sfc (char 4)
   - count (long)
   - min length (int)
   - max length (int)
   - mean length (int)

   -- for a length of 14 bytes per record

Another idea is to add statistics nodes as the file is scanned, instead of creating a 36,000-node array up front; perhaps a hash would be appropriate, or a sortable structure of some kind.

Character Sets

(For Settings, see above)

Character set for display

The default is to read the Leader byte 09 to determine how to convert the characters in the record for display as Latin-1. This byte is either blank for MARC-8 or 'a' for UTF-8. Some of the code assumes that byte 09 will be blank for records in Latin-1 also (which it probably would be). The check of byte 09 is done for each record to handle files of mixed encodings.

Although MV lets the user choose the encoding of the input file, there are some considerations:

- User may forget they overrode the default and later files may display improperly. Could put a warning in the MessageLabel whenever a file is opened and an override is used.
- Latin-1 files have no distinctive leader byte.

Displaying Unicode

Since Optima is supposedly able to display UTF-8 encodings, it would seem logical that MV could also. However, we've tried both UTF-8 and UCS-16 in the RecordView with no success. We tried prefixing the record with the signal bytes for UTF-8 and UCS (EFBBBF and FFFE, respectively) and loading formatted symbols in byte by byte. The

#define _UNICODE in Optima makes no difference either.

Another problem is that the double-dagger (hex 87) used for a subfield delimiter appears in UTF-8 codes as well. Thus when a field is parsed into subfields, garbage results.

A similar problem would arise if UNIMARC records were ever in UTF-8, since the start- and end-of-non-filing-characters symbols appear in UTF-8 codes as well.

For these reasons, MV displays all records in Latin-1.

Record Terminator Detection

Although SP_Val already checks for an RT inside a record (before the Record Length), or when the last byte according to the Record Length is not an RT, one MC client needed more than this. They have a lot of records whose RecLens are too short, so the RT is not found.

There are actually these possible scenarios:

1. RT where expected but a second one inside record. Already handled by validation routines.
2. RecLen too small, so RT comes after
3. RecLen too large, so RT comes before
4. Missing RT

For client LLL

MC handles scenarios 2-4 for client LLL, but these were not implemented for MV.

Here's how it was done: If the RT is not detected, MC goes back to the start of the record (point A) and scans forward until an RT is found (point B). It then discards everything between A and B. It then begins reading the next record at point B. MC has no way of knowing if it is skipping one record or more than one.

MC doesn't know if the RecLen is wrong, or the RT is just missing. It really doesn't matter to the algorithm. It should work in all cases, except that for scenario 4 MC could try looking for the next record at the expected offset, and maybe thus skip one fewer records. This hasn't been implemented.

The goal for client LLL was to implement the simplest possible algorithm. After all, the data coming in is wrong, and that is where it should be fixed. We are just trying to salvage the remainder of the file without elaborate algorithms.

This algorithm will fail on scenario 1, so we have to ensure that it fires only if an RT is not where RecLen says it should be.

For MV

Oddly, MV's scanning to build the offset array doesn't care about the RT. It just uses each record's length to determine the offset of the next one.

- If the RecLen is a few bytes too small, SP_RecordByOffset will not read the record, but the next one can be read because ReadRecord will skip the RT (and any preceding non-digit data) to get to the next RecLen.
- If the RecLen is too large, or more than a few bytes too small, the file is not usable after that point, since the next numeric read will not be the RecLen. There is room for improvement here by using the algorithm for client LLL.

Online Help

There was a separate help document on the Systems Planning website for each release of MV, so users would not be reading about features they don't have; the help version is set in MainForm_Create. For example, release 3.10 uses http://systemsplanning.com/marc/Mvhelp310.asp.

There was also an unnumbered MVHelp.asp for MV releases prior to 3.10; this showed the menu and help for 3.9, since we could not go back and provide numbered help for earlier versions. (Web help didn't start until 3.8, so this just means that 3.8 is using 3.9's help. MV 3.7 and earlier use Windows help installed with program.)

For MC, each client has a separate Help file on systemsplanning.com. Each client's Help file includes the MV help of the appropriate release, depending on the date the MC version was prepared.

Help files are ASP so they can integrate MV and MC help. Users can SaveAs from their browsers if they want a local copy.

However, for the open-source version, we combined the MV 3.12 help and the MC help for all clients into a single HTML file. Use the parts you require for your project.

Context-sensitive help

In addition to the Help button in the main menu, MV includes context-sensitive help from every on-screen form and dialog.

For opening context-sensitive help from buttons in dialogs, here are the options:

1. Do the same thing as the main button. This opens up a new browser window and positions the help using an anchor. User ends up with multiple browser windows. There seems to be no way to detect that a browser window is already displaying the Help so we can reuse that window.
2. Remove buttons and replace by messages. User can be told help is available, but would

have to close dialog, open help, then reopen dialog.

3. Replace http ref in help code with Javascript to window.open. This way we could ***possibly*** use the same two windows (the JS one and the one that is opened). The JS one could close itself, but what a kludge.

We chose the first option, even though it opens the help in a new browser window each time. At least the user is positioned to the relevant section of help.

UNIMARC Support

MV and MC support UNIMARC. Here is what was required to do this:

- Basic structure is the same: leader, directory, tags, indicators, subfield codes. MV displays the record already
- Tags have different meanings, but not an issue as MV does not display tag labels. However, some Leader bytes have different meanings (display issue and also Statistics).
- Different tags mean different choices for NavGrid. The user may make four sets of choices for the NavGrid, MARC21 bib and auth, and UNIMARC bib and auth.
- Character set is different. Added a new field in Symbols.cpp. Also, hex 88 and 89 are the start and end of non-filing strings, represented in documentation as {NSB} and {NSE}. MV displays 88 and 89 as { and }, and shows them and their included text in green.
- Definition of a control field is different. UNIMARC documentation says the 001 is "almost the only field not to have indicators". Unfortunately it does not say what the others might be, and gives no examples. The LC site also cites the 005. We are assuming therefore 001 thru 005.
- Probably differences in minimum length of a record and other basic validation (ignored for now)
- How does MV/MC recognize a UNIMARC file? There is no Leader byte for this, unfortunately. MV/MC assumes MARC21 records have an 008, and UNIMARC records do not. Opening a file did not scan the directory, so we now check the first record looking for a 008. SP_UseFile returns 1 for a MARC21 file, 0 for a UNIMARC file.

We have not yet tried to export from UNIMARC. Export warns if user tries to.

MARCXML

MV can read and display MARCXML records.

Records must use the schema found at www.loc.gov/standards/marcxml/ which has only the elements <record>, <controlfield>, <datafield>, and <subfield>. These can be prefixed by a namespace, such as <marc:datafield>, which complicates the syntax.

At present we make no attempt to validate the file or records, as we do for MARC.

We have not yet developed export from MARCXML. Export warns if user tries to.

Reading records

MV builds the record array using file offsets just as for MARC. When a record is read, the record is stored, stripping each line of leading spaces and tabs and trailing CRLF. Fields are obtained by scanning the record; the scan generates a MARC-like field with indicators, subfield delimiters, subfield codes, etc.

Because MV replicates a MARC field, SP_Subfield needed no modifications.

Character conversion

MV assumes all MARCXML files are UTF-8. Therefore conversion for display and searching does not reference the user's Settings (which are only for MARC21 files).

Displaying records

Records are displayed exactly like MARC21 records. They use the same grid settings in Settings.

Hexview for MARCXML is almost the same as for MARC21, but of course shows the XML file, including tabs, linefeeds, etc. Double-clicking to highlight an entire field is not possible, as there is no clue to where the element tags are without doing a record map.

Statistics

Statistics works for XML files.

MARCSQL

MARCSQL works for XML files.

Splash Screen

MV displays a splash screen to occupy space when user does a Send-To on a large file (but not for opening files from the menu). The SS disappears when the main window opens.

The splash is not appropriate for MC since it has the word "MARCView" hard-coded in its bitmap. Anyway, MC doesn't use Send-To, so it doesn't need it.

MARConvert

MC does various kinds of conversions:

1. imports records into MARC
2. exports records from MARC
3. converts character encodings of MARC records
4. converts from MARC21 to UNIMARC
5. creates blocked records
6. unblocks RLIN files

These are handled by the class Import (#1), Export (#2), and MarcToMarc (#3-5). #6 is not yet integrated into MC.

Conversion Policies

An A is an A. MC only converts the format and character encoding. For example, it does not restructure the data, such as to put the author into a 245$c. Exception: it does restructure from MARC21 to UNIMARC according to the format specifications.

Maker/Breaker. Although MC can convert to and from LC's MARC Maker/Breaker character encoding, it does not provide options to use the same text format, such as the = before the tag, the mandatory two spaces as the Label Separator, the option to use \ for a space, multi-line fields, etc. After all, if users have such records, they can convert them to MARC using Maker/Breaker, then to another text format using MARConvert.

Command-Line Functionality

A command-line mode is a standard feature of all MC deliveries. In some versions, the command line reads a "command file" to run multiple files in batch mode; other versions just take command-line parameters to run a single file so they can include MC in a script. The MCClient::MCBatchFiles variable records each client's preference.

Here's what happens if the program is invoked from the command line:

- For MV with no param, it simply opens. If there is a parameter after "marcview" on the command line, MV assumes the parameter is a MARC filename, and opens and displays the file. In fact, this is how the Windows "Send To" function works. (Send-

To functionality is disabled in MC, as command-line parameters mean other things to MC.)

- For MC for client GGG, MC works only in console mode, whether or not there are parameters. If parameters, it tries to convert; if none, it shows an error message.

- For all other versions of MC, if there is a parameter, MC opens in console mode and processes the parameters (either a single conversion or a batch, depending on the client). If no param, it opens in interactive mode. (Not sure if tested for all clients.)

Note that MC's command mode gets from the INI file any params not in the command line or command file.

Some MC clients insist that the console close automatically at the end of the run. For client MMM this meant that if SQL Server BCP failed, no diagnostic was visible. So our practice now is not to close the console if there is an error -- but we won't insist on it if the client has some other way to find errors.

Client-Specific Processing

By definition, MC is different for each client. It reads variables from the INI file, as well as settings in MCClient for settings the client shouldn't change.

We still need the #define MARCONVERT to build MC. Do not remove it, because without it a user with both MV and MC could run the wrong one.

New Primitives

Based on Paul Graham's ideas, we began to build into MC a "language" for creating MARC records -- needed by MtM when converting to UNIMARC, but could also be used by Import. Gray items are not needed because MakeTree takes care of them.

- Create new field and return pointer to field X (creates indicators slot automatically if not control field. Also updates index.
- Find any field and return its pointer.
- Read/Write indicators for a field
- Create/Read/Write a subfield for a field (inserts new sf in correct alphabetical order, allowing repeats) -- return pointer to the subfield
- Create/Read/Write segment for a field or subfield
- Copy structure of field to new instance to add a repeatable field -- update index
- Write a subfield or segment with an attachment (to override the one in TT).

Began adding these to yTrans to simplify access to the TLL and Index -- which were adequate to store the components of a MARC record, but were hard to access.

This means that there are two ways to add data to the nodes: The original InsertString was based on looking up the "sourcefield" and inserting the "sourcedata" into it. The new routines also permit inserting into a node directly (because we save the addresses of critical nodes that will need this).

MCClient

The MCClient class reads the INI file and sets the MC variables for each client. This replaced a lot of complex code in other modules that had to be updated for each new client.

MCBadFile indicates whether client wants bad records to be output to a separate file. MC maintains and displays a badcount in either case (see **Definitions**).

Character encodings are set in MCSourceCharset and MCTargetCharset. If these are missing, the INI file is checked. Likewise, we can use the presence of MCClient settings to suppress the display of the radios.

- For Export, INI held only the target charset, so we check for MCTargetCharset == NOTSPECIFIED
- Import -- to do
- MtM -- to do.

070924 Some time ago we added the MCBatchFiles variable, which can be either SINGLE or MULTIPLE. For clients like MMM and PPP that are writing files for db import or ODBC updating, we need to write a single set of files even if MULTIPLE; in other words, they Export a number of MARC files and then want to import them into a db in a single step -- or if they are using ODBC they still want a single set of text files. Therefore if MCExportSQL is TRUE, the output files are written where the report is written. This is true for both command and interactive modes if MCExportSQL==TRUE.

Interface

Browse button

The Browse button allows the user to browse for an input file and creates full paths/filenames for all other files except the Translation Table (TT). Until 070722 it also created a path/filename for the TT, but since most clients use the same TT all the time, we no longer do this. Instead we get the TT from the INI always.

Import

At present, only MARC21 records can be created.

How import works -- tagged-text files

Import proceeds as follows

1. A line is read from the source file. We use a char array t_line to read from the file, and copy that into a WString t_string for processing (Optima doesn't seem to have a way to read a file into a string, and we need the string to simplify processing). We enforce the immediate copying of t_line into t_string by using GetNextLine to read the data (but not the TT).

2. The "label separator" is looked for. If found, the line is broken into thisname and thisdata strings; otherwise only the thisname string is created.

3. If thisname holds the "record separator" or the eof mark, or we are at eof:

    - If there is a record in process, scan the fieldarray and put each element into the linked list of translation nodes, adding new ones if necessary for repeated segments, subfields, or fields.

    - Then scan the linked list of trans nodes and combine and output all the segments, subfields, and fields into a MARC record.

    - During the traversal, special routines may be called for any segment, subfield, or field; these routines do special routines that are specific to a single input file. Note that the special routines are done during output, not before the data is stored. Thus they are keyed to MARC content designators, not the fieldnames in the input file.

4. Otherwise if there is a thisdata:

    - If there is not a record in process, initialize for one

    - Insert this segment into the fieldarray.

It can be seen that the lines are thus processed in sequential order. However, the lines are output to the MARC record in the order they are listed in the Translation Table.

For historical reasons, Import uses a fieldarray to store the data and then move it into the linked list. MtM stores data into the linked list directly, which is simpler, and could be used for Import as well.

How import works -- delimited files

000602 Delimited files, such as comma-separated or tab-separated exports from databases, can now be imported. The following restrictions apply:

- Each record must have the same number of fields (for client RRR we permit CRLFs inside data)
- FieldSep must be a single character (because we use Optima's Parse method), and cannot be a CRLF
- QuoteChar must be " for now, make it a client option if necessary
- Records must be separated by a CRLF only
- First record must be the list of fieldnames, delimited the same way the data is
- Fieldnames must match in case those in Translation Table.
- Either the input file or the Translation Table or a combination of the two must contain the LDR and 008

MC knows it's dealing with a delimited file if the second line of the Translation Table is FieldSep rather than LabelSep and RecordSep. The FieldSep is the delimiter; the QuoteChar is the character used to surround data that may include the delimiter (usually this is "). Use of QuoteChars around data and fieldnames is optional if the data does not include the delimiter. Use T to indicate "tab" in the FieldSep.

Import Translation Table

The Translation Table (TT) has three columns:

1. Column 1 codes the MARC tag and subfield that is the destination of the data. The code consists of a three-digit MARC tag, optionally followed by a subfield code, optionally followed by and ampersand and segment number. At present segments must be in the table in ascending order. Tags can have segment numbers without subfields also. "LDR" is used for the leader.

   If the MARC tag is 000, the fieldname is ignored. This prevents the report filling up with unwanted warnings, since fieldnames in the input file that are not in the TT are noted in the report.

   If column 1 contains "DDDDDD", the current date will be generated in yymmdd form (for the 008/0-5). The DDDDDD must be in a segment by itself.

2. Column 2 codes the fieldname in the input file that is the source of the data. Alternatively, may contain a Constant in double quotes that is put into each record. Except that at present, we cannot insert a field that is just constants, as the first subfield must be a variable. 040310 This check was turned off for client DDD (see yTrans::MakeTree).

3. Column 3 contains Attachment strings into which the data is inserted. Attachments may contain "+++" anywhere in the string, indicating the position where the data is placed within the string. Attachments should be in double quotes: +++ is also within the quotes.

The lines in the Translation Table must be in the order the MARC fields are to be added to the records.

There must be an entry for each MARC element that is to be created, even if the program creates it. For example, the client may map fields to the 1xx tags, but MC has to move all but the first to 7xx tags; the 7xx tags need lines in the TT. (Typically we use in the second column pseudo-fieldnames like "MC_700a" that can't be mistaken for client fieldnames.)

Fixed-field issues

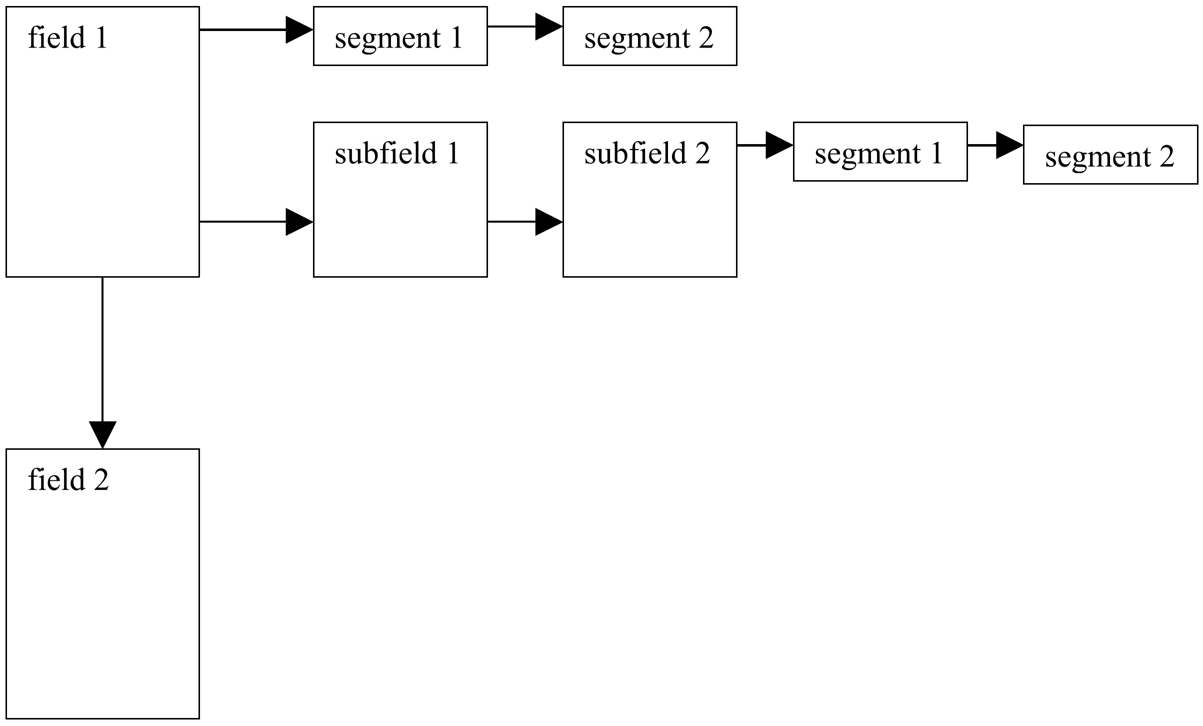
At present we don't have a solution in the TT syntax for the problem of a missing field in the input data, such as publication date. We want the field-data to be inserted into the 008/7-10 if present, but "uuuu" if the field is not present. We have to handle this in the program. It's probably wise to do this in the program anyway, so the length of the field-data can be checked.

Import translation nodes

The Translation Table is read and its information is stored in a tree -- called the Translation Linked List or TLL -- whose nodes store the parameters and data used to build each part of the record. This structure is defined and managed by the Trans class.

Each field-node can have a chain of zero or more segment-nodes to store the various parts of the field value.

Each field-node can also have a chain of zero or more subfield-nodes to store its subfields. Each subfield node can have a chain of zero or more segment-nodes.



The Trans nodes are set up during initiation based on the Translation Table. Nodes and branches may be copied if the record contains repeated fields and subfields. New nodes and branches thus created are reset after each record but retained for the run to save having to recreate them for later records.

Trans Index

Since any label can be repeated, and nodes are not deleted until the end of the run (the end of each file in batch mode), the tree can grow enormously. In order to avoid scanning the whole tree for the next unused instance of a label when we need to insert a node (which would be complicated), MC keeps a linked-list index as well (class TIndex). Its nodes point to the first and current Trans Node for each label; this means fields, subfields, segments, and constants are all indexed. (The current instance is the last instance to be used, or the first instance if none have been used.)

Furthermore, each tree node includes a pointer to the first index node for that node's label; we can then use the index node to find the tree node's current instance.

If the node needed to insert data is already used, we make a copy OF THE ENTIRE FIELD STRUCTURE. This means the default is to repeat a field, not a subfield.

Note the tree is rebuilt for each file, if in batch mode. Although this may slow things down somewhat, using the same tree for multiple files might cause it to get too large.

More on Constants and Attachments

Constants and Attachments are somewhat similar. The difference is that constants (in column 2 of the Translation Table) are put into each record, whereas Attachments (in column 3) is put in only if the fieldname in column 2 is present in the particular record.

All constants and Attachments MUST be enclosed in double quotes. Otherwise the constant or attachment is read as a field label. Attachments use +++ within the quotation marks to indicate where the field data should go in the string.

Constants and Attachments must give the user the choice between adding a string to the data within the current subfield, or creating a new subfield.

- For Constants, to create a new subfield, add the subfield code to the tag in column 1; it is illegal to add $ and subfield code to the Constant itself. To add the Constant to an existing subfield, use Segments.
- Attachments may optionally contain a $ and subfield code.

CONTROL FIELDS					
Column 1	**Segment**	**Constant or Attachment**	**Attachment has $ and sfc***	**Result**	**Example (" -- " represents the tab)**
Tag only	0	Constant		Add tag with constant value	005 -- "19991231000000.0"
Tag only	0	Attachment	No	Combine addtext with data	LDR -- STATUS -- "LLLLL+++z 22BBBBBn 4500"
Tag/segmen t	not 0	Constant		Add text to data in field as specified by segment number	LDR&1 --"LLLLL"
Tag/segmen t	not 0	Attachment	No	Add text to data in segment as specified by +++	008&2 -- DATE -- "s+++ enk c 00010 eng d"
NON-CONTROL FIELDS					
Column 1	**Segment**	**Constant or Attachment**	**Attachment has $ and sfc***	**Result**	**Example (" -- " represents the tab)**
Tag/sfc	0	Constant		New subfield added in the position this line occupies relative to others of the same tag	100x -- "Constant"
Tag/sfc	0	Attachment	Yes	New subfield added where specified by +++	100a -- AUTHOR -- "$xAttachment+++"
Tag/sfc	0	Attachment	No	Add text to data in subfield as specified by +++	100a -- AUTHOR -- "Attachment+ ++"
Tag/sfc/ segment	not 0	Constant*		Add text to data in subfield as specified by segment number	151c&10 -- ",scheme=TGN)"
Tag/sfc/ segment	not 0	Attachment	Yes	Add text to data in segment as specified by +++	100a&3 -- FORENAME -- "+++ $eeditor"
Tag/sfc/ segment	not 0	Attachment	No	Add text to data in segment as specified by +++	551a&2 -- PARENT -- "++ +,scheme=TGN)"

Constant problems (pun intended)

* Important note: A constant in the first segment of a subfield is not allowed, because since constant segments are not indexed, when a chain is copied for a repeatable subfield with a constant segment 1, there is no index entry, so the repeated subfield is not imported. Possible solutions to this problem:

1. Make a subfield an object that can manage its segments. Index entries for all segments of the subfield would point to the subfield object. -- This would be a major rewrite.

2. When chain is copied, start the copy with the first var seg, and use the constant seg 1 for all. -- Problem is that seg 2 would not know how to find the constant in seg 1. Even worse if the subfield is repeated more than twice.

3. Do not allow constants in segment 1 of subfields, except for control fields, as these are always present (? LDR and 008 are, how about others?) and never repeated. This is not ideal, because it doesn't allow for leading ISBD

punctuation to be output regardless of the segments present for that subfield -- but this would happen rarely, we consider. -- This is a restriction on user's flexibility, but the only viable solution seen at present. 040310 This also prevents having a field consisting only of a constant, such as an 040 cataloging source.

000827 Discovered that this problem applies when the first subfield in a field is a constant. For example, if 650- is "10" and 650a is an Attachment. The second 650 field is not imported. This seems much less likely to occur, but still need to warn the user. 040310 Actually this restriction prevents having any constant indicators.

040310 Turned this check off for client DDD, which has no repeating fields/subfields. Recheck whether necessary and how to get around it the next time we have an import client that would have repeating fields/subfields.

New idea for this: in Export, if the constant has no +++, then it is to be used only if there is no data. Could this be applied to this problem as well?

050312 What's wrong with just indexing constant subs and segs? Believe we did this because if the previous record had two instances of a field, and this record has only one, the constants in the first seg/sub of the second instance would cause it to be output for this record without any variable data. (Whereas if the first seg/sub were variable, it would be empty and prevent the instance from being output.) Maybe if we have a check during output, so if the first seg/sub is constant, we check the rest of the field/sub to see if there is any variable data.

050312 Going to try allowing a branch that starts with a constant node to be copied. This means we have to check such a branch on output to see if it has a variable node before outputting. This is to avoid the problem that if the previous record had two instances of a field, and this record has only one, the constants in the first seg/sub of the second instance would cause it to be output for this record without any variable data. Later -- this works! We are able to create multiple 650s, each with inds. We never had to check whether the nodes were used or not, maybe because the constant first element happens to be inds; need to verify with some other sub/seg constant in first position.

Indicators

Indicators may be added conditionally or unconditionally, in any of these ways: as data values, as Constants, or as Attachments (see Help for syntax and examples).

The hyphen is used as a subfield code for indicators.

Repeating fields or subfields

The TT is ambiguous about whether to repeat the field or the subfield. If the TT line is

```
      650a         SUBJ
```

does it mean multiple SUBJs are put into repeating $a subfields in one 650, or does it mean repeating 650 fields with a 650$a for each SUBJ? Since RFs are more common, they are the default. We have considered syntax for repeating subfields, such as adding a + or ! as in

```
      650a!        SUBJ
```

but this has not been implemented.

For client EEE, we have something like this:

```
      650-         "04"
      650a         SUBJ
```

This means for each repeating field we have to generate the indicators as well. So the entire set of subfields for the 650 has to be copied. Since the first one is a constant, it might cause a problem as discussed above.

008 generation

Bytes 00-17 and 35-39 are the same for all forms of material; bytes 18-34 vary by form. We have not so far encountered data that would go into bytes 18-34.

Client RRR had an 008 they wanted us to use, even though some bytes are non-standard.

For client EEE we use a generic 008, with only bytes 00-05 generated.

Export

The export function does not currently handle UNIMARC or MARCXML. It does not convert any encodings except MARC-8 (but if no conversion is required then it should work with any encoding). It does not check leader byte to determine encoding but assumes MARC-8.

2003 Modifications

For client JJJ, the following changes were made:

- Implemented the badfile for invalid records.
- Need to store a repeat separator for each node. This is entered by program now, but could be added to the TT if needed later.
- Need to store a subfield separator for each node. This will be the one in the TT unless overridden by the program

- In order to allow spaces as well as tabs to delimit columns in the TT, values with spaces must be in double quotes.
- We scan each MARC record, looking up each field in the linked list. (We used to scan the linked list and then look in the records just for its nodes, but this did not allow MC to check for MARC fields it didn't know about, missing mandatory fields, and to document in the TT fields not to be converted.
- We store the output in a string in the linked list, then output them all at the end of each record. This is so we can output fields in TT order, and also so we can manipulate and combine fields. (We already do this for Import.) Before outputting, make sure all mandatories are there.

How export works

Export reads each MARC record and searches its Translation nodes for a match on the tag. If found, it then checks the node to see if the whole field is to be exported, only certain bytes, or only certain subfields. If not the whole field, the relevant parts are obtained. If more than one subfield is exported on a line, the subfield delimiter is replaced by the user's specified SubfieldSep.

Since the entire MARC record is scanned (not just the parts needed for export), we need a way to avoid warnings; see below about the Translation Table for how this is done. We also need a way for the user to be able to scan a file and create a skeleton TT; this is the purpose of the Build button (see below).

The results of the scan of a record are saved in the Translation node. At the end of each record, the Translation nodes are output in their node order; this is the way that the client can determine the output order. (Obviously if there are multiple output files as for an RDBMS, all fields for each file should be together in the TT.)

Delimited files

Since delimited files must have the same number of fields in each record, something must be exported for each Translation node. (The number is kept as small as possible as we do not build nodes for parts of the MARC record the user does not want to export -- as indicated by no value in the second column of the Translation Table. A MARC field with no corresponding line in the TT will cause a warning, so use the MARC tag in col 1 and a hyphen in col 2 if the MARC tag is to be ignored.)

If MARC file contains repeated fields to be exported, the delimited file will end up with different number of fields per record. Users are warned about this in the online help, but a better solution is needed.

Multiple output files

Note: hard-coded limit of 20 files in class ExportTT

Delimited output intended to be imported into an RDBMS may need to be in several files. The files need to be related with foreign keys (FKs). In addition, some of the files may be many-to-one in relation to the first file (which is assumed to be the master). This was first done for client KKK, later generalized for MMM.

The relationship is expressed in the TT by just repeating the FK. For example, KKK's objtitles table has its first line as "001 objtitles/bibid", in other words outputting the 001 as the FK (it is also output in the primary file). I don't think the FK has to be the first field, but it seems like a wise convention. The FK is called the **Pivot Field**.

When there are multiple output files, that means there is a one-to-many relationship for some field or group of fields. For example, KKK wants a separate line in the objtitles file for each title of an object in the objects file. In this case, titles are 245, 246, and 240, and we call these the **Detail Fields** for the objtitles file. (Of course, not every MARC record will have more than one title so the one-to-many could be one-to-one for some records.) This is how repeating MARC tags are handled. For MMM order recs the Detail File has only one Detail Field, but generally MARC uses repeating groups of fields such as 24x, 65x, etc.

Here's how to do it:

- Code Detail Fields in the TT by preceding them with * in the first column.
- Fill the "output" string array with the names of the output files
- Each node has a data element indicating which string and output file that node writes to
- Each node has a data element indicating whether the field is a Detail Field. We use a meaningless sequence here just so we can keep track of which one we are working on, avoiding the clumsy lookups in KKK's WriteDetailFiles. We are going to try a single sequence for ALL output files for simplicity.

Here are the assumptions for use of Detail Fields:

1. There is exactly 1 row in the main table per MARC record.
2. All other tables will have 0 or more row per MARC record.
3. Each table has exactly 1 set of Detail Fields that trigger it, but only one Detail Field per file (for example a Detail File can't have both authors and titles since both are repeatable).
4. The TT lists all columns for a target table together (contiguously)
5. Detail Tables can include constants as we did for KKK.

Unfortunately, problems arose with client PPP. Assumption 3 doesn't work because PPP combines multiple fields into a single output row, but each field is its own column, meaning its own line in the TT.

We are planning to bypass the TT for PPP for the Detail Tables, to avoid copying nodes and also because of time constraints.

Multiple output files for MMM

The approach for client KKK is rather a kludge. First there is a fixed-size array (3) of output files. Second, the first output file is written in the normal loop but the others are written in a call to WriteAuxFiles. Instead, this was generalized for MMM (although one of their tables, VAR2_FIELDS, does not conform to this):

- Use yExport::GetComplexFilenames to find out the number of output files needed (one per target-data-model table, of course). Maybe the output files can also have meaningful names based on the tables in the target.
- Use the qualified names in the nodes to determine which output file to write to. 070829 Client may not determine output filenames, just the extension.
- Use a single routine for all writing, both primary and aux files, and whether the client uses one or multiple output files.
- Use the modern TT format using .-qualified names in the target column.

Translation nodes

Export uses a different node structure to hold its Translation Table from that used for Import. It probably could use the more complex one of Import, but it doesn't now. They are different because Export was coded first, and exporting from MARC is inherently simpler than importing into MARC.

Export Translation Table

The first line is ignored. The next few lines determine whether the output will be tagged-text or delimited:

	Tagged	**Delimited**
Line 2	LabelSep	FieldSep
Line 3	RecordSep	SubfieldSep
Line 4	SubfieldSep	[1st line of translations]

The lines describing the translation for each field have three or four columns, depending on the client:

- Column 1: Source. This holds the MARC tag and subfield that is the source of the data. The code consists of a three-digit MARC tag, optionally followed by one or more subfield codes (or bytes for fixed fields); do not use a dollar sign or double dagger between the tag and subfield code. A subfield code of '-' (hyphen) means "indicators".

  If column 1 contains only a hyphen:
  - if there is a Constant in column 3, the Constant is output to the field in column 2.
  - if there is no Constant in column 3, the line is ignored.

- Column 2: Destination. The field label in the output file.

If column 2 contains only a hyphen, the line is ignored. The user can record a value in column 1 for documentation purposes; this is the way to avoid warnings in the report about MARC fields encountered for which there is no TT entry.

- Column 3 (for all clients other than MMM and PPP): a Constant. See **Conditions and constants** section below.
- Column 3 (for clients MMM and PPP): the Target Datatype. Allowable types are "text" and "num" -- used to determine whether to enclose values in single-quotes in SQL statements.
- Column 4 (for clients MMM and PPP): a Constant.

Exported data is output in the order the lines are found in the Translation Table. This makes it very hard to determine which fields in the MARC record have no lines in the TT. Read on...

Build button

The Build button creates a skeleton Translation Table for a file to be exported, with every field and subfield represented. This helps the user avoid the almost impossible task of making sure all fields and subfields are considered. The user can edit this skeleton, adding values to column 2 as desired. Since MARC tags found in the record but not in TT generate a warning, putting a hyphen in column 2 for a MARC tag will suppress reporting on tags the user says they don't want.

Constants

Column 3 of the TT can record a Constant to be output to the field specified in column 2.

Constants can contain +++ to indicate where the data should be placed within the constant. If there is no +++, then the constant is to be used only if there is no data.

A constant can include the string "YYYYMMDD". In this case the current date is calculated and inserted into the constant. This can be combined with +++.

Writing to a database

Controlled by MCClient::MCExportSQL, MCExportODBCLoad, and MCExportDSN.

If MCClient::MCExportSQL is TRUE:
- The export is intended to be loaded into a DBMS. Only SQL Server has been used so far.
- We name the output files for the SS tables, so these are not user options.
- TT must have a line for each column to generate correct output.
- There is no bad file, since there would have to be one for each table; instead, in case of a bad record, the tuple for the main table is written to the report.
- Since the text files will be loaded using BCP, the FieldSep must be T (tab), which BCP

prefers.

Loading with ODBC

If MCExportODBCLoad is true and there is an MCExportDSN, then each record is loaded using an "insert into" statement as it is converted. MCExportDSN is set by the client's adding an INI param DSN in [Export]. The reason for having both these controls is that for some clients the DSN may be used for other purposes than loading, such as to empty the tables before each run. (The program checks to make sure that TheTransaction exists as well.) Likewise some clients may want to turn ODBCLoad on and off, which is done by providing or not providing a DSN. MC adjusts the format of the statements to satisfy both ODBC and BCP, such as using commas or tabs for field separators, adjusting quotes, etc.

We set MC to commit only at the end, since MC cannot start a run in the middle of a file.

Opening TheTransaction in the yExport constructor didn't work, moved it to ConvertMARCToText even though this will be repeated for each file.

Bulk loading

For client MMM, MC loads the output files directly into SQL Server (SS) by spawning BCP. This means that the ODBC option (turned on by having a DSN line in the INI) is no longer available (otherwise everything would be loaded twice).

The client should set their db as follows for max performance:

- Logging (SS only) -- Simple is OK; or if they use Full, switch to Bulk-logged recovery model just before the load and reset after.
- Drop indexes (rebuild them later)
- Disable constraints
- Disable triggers
- Empty tables if possible

For committing in batches, see SS file for "Managing Batches for Bulk Import" -- but as with ODBC loading. However, specifying the number of rows is supposed to help.

We call BCP at the end of the run (if we called at the end of each MARC file we would have to have complex code to avoid the BCP err file from being overwritten).

Summary of parameters

MCExportSQL	on	on	on	on	off
MCExportODBCLoad	on	on	off	off	n/a
MCExportDSN	yes	no	yes	no	n/a
Result	Text files suitable for loading	Text files suitable for loading			Neither (some other kind of export)

	Also ODBC load	No ODBC load	

MarcToMarc Conversion

This class handles (1) conversions of MARC records to other character sets; (2) conversion of MARC21 to UNIMARC; (3) creation of blocked files.

For MARC21-to-UNIMARC, a new subroutine was added to the MtM class called ConvertToUNIMARC, plus several related subs. This subroutine uses the same node-structure as Import's, as we need to be able to handle repeatable fields and subfields. There is also a TOUNIMARC conversion target for ConvertAnsel.

MtM includes a routine for handling batch runs (BatchConvert) as well as interactive. Both of these call other routines for the actual char conversion or blocked-file conversion.

Unlike Export and Import in interactive mode, MtM doesn't offer the option of limiting the number of records to convert.

Also we removed one step from what was done for Import. There we stored data into a fieldarray, then scanned that into the linked list of nodes, then built the MARC rec from that. We do without the fieldarray in MtM and write directly to the nodes; Import could do the same.

ConvertToUNIMARC is controlled by a TT, but not one the client can see. It is an "include" file to MtM. We started with a table developed by OCLC. There is MUCH in this table that has not been addressed as it was not needed for client FFF: basically anything besides the most common fields for monographs in English are not analyzed/mapped. Lines not tested are commented out so that the program will generate a warning if the fields occur in an input file.

Added feature to yTrans::InsertString so if a node is not found, it looks for a 000-node for the tag as a whole. This saves us from having to add a 000-node for each sf for fields we don't want to convert. (A 000-node is one in which the target of the conversion is "000", meaning do not convert.) It is rather strange that if a 000-node is found, its address is returned, but there you are.

When a duplicate subfield is found, yTrans::InsertString always creates a new field (not subfield). We need a way to handle this in the map, but meanwhile we kludged it for FFF so that InsertSubfield is used for 110b's and 710b's.

MV/MC Character Conversion

Goals

The original design goals for character conversion were these:

1. Convert the minimum length possible, rather than to reduce calls or simplify program structure. Therefore conversion is done to subfields, rather than to fields or records.
2. The Record and Field global variables (but not Subfield) should always remain in the original character set. Callers converting the encoding of a record or field should make a copy. Therefore, SP_Record and SP_Field have no parameter to request conversion; callers who need to convert a whole field or record should do so in a separate call to a conversion routine, using a copy of the field or record.

Subroutines

Character conversion is done by four subroutines: SP_ConvertAnsel (that is, MARC-8), SP_ConvertLatin, SP_ConvertUnimarc, and SP_ConvertUTF8. These receive a parameter to specify the target encoding.

SP_ConvertLatin has not been fully tested, since there has not been a demand for it. Therefore in debug mode it calls the older routines SP_LatinToAnsel and SP_LatinToLatin and compares their output to its own.

Conversion Methods

There are three methods of encoding conversion in MC:

1. Method 1 ("Symbols") is to look up symbols in Symbols.cpp (simply a structure) and find the desired equivalent. This method converts among MARC-8 (called "ansel" in the code), Latin-1, Plain, UNIMARC, and UTF-8. This is the older method, replaced by Method 2 when more complete translations between MARC-8 and UTF-8 were required for MC. (This Method is sufficient for MV as it can display only Latin-1 symbols.)

2. Method 2 ("Codetables") is to look up symbols in these three text files: Marc2utf.txt, Utf2marc.txt, and Decomp.txt. These were derived from the Library of Congress' Codetables.xml and the Unicode Foundation's database. These files contain all 16,000+ MARC-8 symbols and their UTF-8 equivalents, plus decomposition information.

3. Conversion to and from LC's MakrBrkr format is hard-coded. (This has not been used in 10 years, so should be tested before relying on it.)

So far, the Codetables method has been applied only to MtM conversions.

Warning: Whether or not to use the Codetables method is hard-coded for each client in SP_UseFile. This is not ideal, of course. In addition, Import (if it ever needs the Codetables) will need another way to indicate that, since Import does not call SP_Usefile.

Method 1: Symbols.cpp

Although Symbols.cpp includes codes to convert between MARC-8 and UTF-8, it is impossible to have all the combinations of Ansel diacritics and characters, so it includes only those with Latin equivalents, plus some extras for common normalized UTF-8 symbols. In other words, if records are in UTF-8, use Symbols.cpp only for display and searching; use Method 2 for conversion.

Note that Symbols does not always contain exact translations but the closest match. For example, n-acute in UTF-8 is converted to a plain n in Latin-1, since there is no n-acute in Latin-1. There is no warning about this as it is by design.

InitializeSymbols must be called to load the symbols in Symbols.cpp. For MV, Export, and MarcToMarc, this is done in SP_Usefile, which these three call. For Import we must do it explicitly.

Method 2: Codetables.xml

This method is based on the LC Codetables.xml. It was last downloaded and checked for changes on Oct. 26, 2007. The LC file is undated, but the latest change found in it is dated March 2005.

The Codetables method required a lot more sophistication than MC had before:

- Converting non-latins to MARC-8 required use of escape codes
- Converting from non-latin UTF-8 precomposed characters
- Converting to non-latin MARC-8 precomposed chars
- MARC-8 records require an 066$c if they use alternate code tables using Technique 2 (see MARC specs for char sets, p. 18).

MC does not handle cases in which a single Unicode can convert to multiple MARC-8s (there are variants for each language, unlike in Unicode).

0512 The Codetables are loaded by InitializeCodetables, which is called only if the MC client needs them; the subroutine loads the three text tables into memory. See below for details.

Implementation considerations

The only relevant fields in Codetables.xml are the MARC-8 and UTF-8 codes, the "Iscombining" indicator, and an indication of which MARC-8 charset the code belongs to.

We did not want the client to have to set up a DSN, which ruled out a database approach. Speed was also a consideration; we wanted the conversion tables in memory.

The first approach was to incorporate the table into the source code as an array of char, but this proved impossible. Optima ran out of memory when we had 15,000 lines of string values assigned to the array. Can't think why, as it is a perfectly legit approach, and why would it take so much memory?

The current approach is to produce three text tables Marc2utf.txt, Utf2marc.txt, and Decomp.txt from Codetables.xml and load them into memory during startup. There is one text table for MARC-8-to-UTF-8, and another for UTF-8-to-MARC-8. This allows the program to load both, and perform binary searches on the structure in memory regardless of which direction the translation is going. These tables must be delivered with the program for MC applications.

The only columns needed in the text files are Table, Marc, Utf8, Iscombining, and Charset. This is 16 bytes per record, or 262,304 for 16,394 records; this is doubled because there are two tables (one for each direction). Quite reasonable to gain the speed of an in-memory database. (If we were to do this again we would use a SQLite database, which requires no DSN.)

Alt and Altutf8 columns -- In the table, only a few CJK symbols allow 3013 (Geta) as an alternative; however, MC uses Geta for any CJK symbol for which there is no other symbol when converting to UTF-8. (One client wanted the double-underscore instead.) The only other symbols with Alts are the double ligature and double tilde, which are handled in the code.

Implementation

Warning: Since a C/C++ array must be dimensioned when it is declared, if symbols are added to Codetables, change the dimensions in Symbols.hpp.

Decomp.txt is based on Unicode 4.10. Note that Decomp.txt does not attempt to replicate all decomposition information in the Unicode Foundation's Decompmap. Only Unicodes with MARC-8 equivalents are present.

When translating from MARC-8 to UTF-8, MC does not try to find precomposed characters for non-latins.

Use of Symbols.cpp for MARC-8 and UTF-8

MC does not check Symbols.cpp for MARC-to-UTF if client uses Codetables

When converting UTF8-to-MARC8, if a character is not in the Codetables, MC looks in Symbols.cpp as well. This is because there are UTFs that are not in the LC Codetables, such as 02BE, the older conversion of Alif (now 02BC). MC needs to be aware of this to convert older files. During conversion, a warning is generated about this obsolete usage.

It is assumed that all symbols found in Symbols.cpp are NOT combining, because in fact only a few common diacritics in Symbols ARE combining, and these would be found in the Codetables.

MARC-8 TO UNIMARC

Currently this is a 1:1 conversion. It does not handle escape sequences to other character sets.

There are two characters in the ASCII set not handled by MC. These are recent additions to MARC-8, for which UNIMARC equivalents are unknown:

- 8d, zero-width joiner
- 8e, zero-width non-joiner

In addition, we are assuming that 7e, non-spacing tilde, is the same in UNIMARC, since it is an ASCII character.

MARC-8 TO UTF-8

MC does not check Symbols.cpp for MARC-to-UTF if client uses Codetables

For several clients MC converts MARC-8 to UTF-8. There are two ways we handle this, depending on whether the client expects to have non-roman characters.

Symbols

If the client has romans only MC does not use Codetables but Symbols.cpp. This means there are potentially thousands of combinations, since a letter can be modified by more than one diacritic.

MC can handle multiple modifiers for a letter. To keep the size of Symbols.cpp down, all but the last one are converted to standalone Unicodes (not combining forms). So the output might be a UTF-8 modified letter followed by a few standalone modifiers.

MC normalizes common letter/modifier combinations, including many of those not in Latin-1, but the less common ones are output as separate letters and standalone modifiers.

Even for modifiers for which there are combinations in Symbols, MC checks the modifier as a standalone if it is not found in its combination. This is because there may be combinations not in Symbols.cpp.

This approach does not handle the ANSEL escape code to another charset. Add if needed.

Codetables

To handle the entire MARC-8 character set of 16,000+ symbols use the Codetables approach. See above.

UTF-8 TO LATIN

This routine converts only NORMALIZED Unicodes. We know of no algorithm or table for converting a letter followed by a combining character to the single Unicode. (In fact, the Unicode Consortium says that the list of Unicodes only represents the most common combinations anyway; others are possible in a non-normalized form.)

Since this is a conversion to Latin, the exotic and multiple-diacritic sequences would not be represented anyway.

UTF-8 TO MARC-8

This has the same two approaches as MARC-8 to UTF-8 (Symbols and Codetables).

Symbols

For conversion to Ansel, unnormalized UTFs do not have to be normalized but just output in reverse order. The common normalized UTFs are in Symbols.

For conversion to Plain, just output the letter and ignore the modifiers.

For conversion to Latin, MC has to either normalize or look up, but there are few of these. But MC does not need to handle unnormalized UTF-to-Latin; in fact it's only for display, which is less critical.

For the UTF-8 scanning, here are the ranges of interest and status (at present just for western European languages):

Range	Purpose	Coded	Tested
0021 - 007E	ascii characters	y	y
00A1 - 00FF	symbols and some accented letters		
0100 - 017F	accented letters		
0180 - 0217	accented letters		
0250 - 02A8	accented letters		
02B0 - 02E9	spacing modifier letters, treat like diacritics		
0300 - 0361	combining diacritics	y	y
0370 - 03F5	Greek	n	n
0400 - 04F9	Cyrillic	n	n

In other words, "modifiers" in the algorithm above means characters in the range 02B0 - 036F. "ASCIIs" means <= 007F. All others are UTF-8 characters.

Error Handling

MARCView

At first MV used to warn the user about conversion problems. However, this seems rather pointless as there is nothing the user can do about it. It now just silently ignores the problematic characters.

MARConvert

If a character cannot be converted, the client has these choices that can be programmed into MC:

Handling the character:

1. leave it
2. convert to nearest equivalent
3. convert to some neutral character like a space
4. convert to some flag character
5. remove it
6. for Unicode to MARC, convert it to a numerical representation like &#x0f;

Handling the record:

1. output it
2. put it in bad file

Counting:

1. increment "bad" count (always done if record written to bad file, but an option if written to good file)
2. do not count as a bad record (if written to the good file)

Handling the file:

1. continue
2. halt

Handling the batch:

1. continue with next file
2. halt

Reporting:

1. no warning
2. warning

MC/MV Development

Problems

If MV/MC will start but will not appear on screen (but there is an icon in the Windows toolbar), the INI probably has weird window-sizes. To see it, maximize the app from the toolbar. Believe this is now fixed.

During development of handling MARCXML for MV 3.9, often the navgrid would jump to the second record immediately after a file was opened. We could not find the code responsible. It went away when we added the Recent Files feature.

Release Procedure

Users should install MV and MC to different directories, so that each can be separately updated without worrying about the DLL version.

Installer

NSIS will uninstall only the files named in the script, so there is no problem leaving client's data files in place.

NSIS does not require uninstalling before installing a new version. The install simply overwrites -- silently at present, maybe there is an option to ask the user -- but would we want to?

How to Prepare Release

Comment #define MARCONVERT for MARCView

Uncomment #define MARCONVERT for MARConvert

Update product, release, version, and state names

Prepare and include translation table, command_file.txt, and codetables

Prepare installation script and generate the installation file

Client-Specific Notes

How to Add a New Client

- Add to enum in yMCClient
- setup all yMCClient params
- in yMCClient.Initialize, read the ClientCode and set defaults
- If client has a default charset (rather than selecting one), add the default in two places (interactive and command modes) -- now done with yMCClient params for MtM and Export, not yet done for Import
- Customize the code as necessary
- Test both interactive and command
- Update the installation "NSI" file.

What follows are notes on what was done for each client. The codes "AAA", "BBB" refer to specific clients to protect their anonymity. The same codes are used in the program code.

AAA

Purpose

To convert MARC records both ways between MARC and UTF-8, including all code tables and EACC (CJK). Uses Codetables. Detailed specs in folder. Command Mode handles multiple files. Uses bad-file for bad records.

Inserts a double underscore into the UTF-8 records if a MARC-8 character cannot be converted.

Had to add a new set of radios in the dialog box to specify direction to convert, since leader bytes not considered reliable.

BBB

Purpose

To convert MARC record in UTF-8 to MARC records in MARC-8.

Both interactive and command mode. Command mode uses a batch file to process multiple input files. A second param on command line indicates whether to suppress visibility of console.

Releases

1.1 Better message if character cannot be converted.

CCC

Purpose

To convert MARC records both ways between MARC and UTF-8, including all code tables and EACC (CJK). Uses Codetables. Command Mode handles multiple files. Uses bad-file for bad records.

Inserts a Geta (U-3013) into the UTF-8 records if a MARC-8 character cannot be converted.

Uses the two separate characters for double ligatures and double tildes, rather than the newer infix characters.

Releases

1.1 Records with unmatchable chars no longer put into badfile.

DDD

Purpose

To convert Amico records to MARC. In the dialog, the Translation Table defaults to that of previous run, instead of having a name constructed from the input file.

Records have two 007 fields intentionally, both constants.

EEE

Purpose

To convert BASIS TechLib records to MARC. Command Mode uses the "command file" in order to process multiple files.

The input files can be "catalog data", which is converted to the MARC bibliographic format, or "copy data", which is converted to the MARC holdings formats. Each copy in the copy data is in a separate record; therefore we create a separate holdings record for each copy. As far as we can tell, all catalog records are for the "basic bibliographic unit" rather than for supplements or indexes (see www.loc.gov/marc/holdings/echdgenr.html).

Since there are separate input and output files for catalog data and holdings data, we use two separate Translation Tables. This required another textbox in the dialog. In the dialog, the TT defaults to that of previous run, instead of having a name constructed from the input file. The client must be careful to select whether the run is for catalog or copy data, or garbage results.

See the help file for notes on custom processing.

Releases

1.0 Was just for catalog data.

1.1 Added copy data.

1.2 Corrected identification of 010 as a control field (it's not).

Catalog data

Input data fieldnames are followed by an **increment** (our term) for repeating fields; for example, AU(1) is the first author, AU(2) is the second.

AU is personal author, CORP is corporate author. AUTH appears in both cases, repeating the AU or CORP, so is likely a tracing field (we are ignoring it). DAUTH is probably "display author", ignoring. AU_SX appears to be a cutter, ignoring.

TITLES appear to be tracings. None found in small sample that were not also present as TI, ALTTI, or SERIES.

Every record seems to have a REV_DT even if it's the same as ADD_DT, so we are ignoring ADD_DT.

Ignoring DOC, as it seems to have little relevance to format. MTYPE is the material type.

We set the Leader/06-Type of record from the file the record arrives in. So Books_cat.txt is assume to have language material, Videos_cat.txt is assumed to have videos. However, the file contents aren't always consistent; for example, Videos_cat.txt also contains sound recordings. Thus we have to examine the 007 created and possibly adjust the Leader/06.

The 007 has only the first (and sometime second) positions filled; the rest is "no attempt to code". We were not contracted to do the extensive parsing that filling the other positions would require. Our goal is only to generate a legal MARC record.

Copy data

The client uses copy records not only for copy info, but also for analytic cataloging. For example, a series cataloged in the catalog file may have its pieces cataloged in the copy file. This means we must find a home for certain bib data not normally needed in the holdings records.

- NOTES usually holds the item title, so we are putting it in 876z (item information public note). Often it holds location info and should be in the 852z, but we can't discriminate.
- YEAR is the year of the piece, presumably. We are putting this into the 876z as well.

For the same reason we cannot set the Leader/06 to "v" for sets, multi-volume works, kits, and collections. (If this turns out to be critical I suppose we could look in the corresponding catalog file and try to find out -- but what a mess!)

Implementation notes

All these were done except where indicated. Blue indicates action taken based on client's replies.

what's the diff betw AU and AUTH? they are always the same in the sample -- one is tracings, which we are ignoring. same with TI and TITLES

authors -- how to know which 1xx field to use -- using first encountered

1xx inds -- no way to know, just use most common. same for 7xx and 6xx

SUBJs dn look like LCSH, what scheme? (for 2d ind) -- using "4" (not specified)

diff betw repeating the 653 (keywords) and repeating the 653a -- either is allowed, we are repeating the field

M700 actually contains a corp author (sb 710) -- ignore data problems

what are M110_EXP, M110_EXP, M600_EXP? they are always the same as the corresponding field (AU, CORP, SUBJ) -- ignoring

ALTTI not specific enough to assign to a tag, just guessing

do they want claim info? if so, 590 note? -- ignoring

- this includes ISSUE_ENUM_TYPE, NISS, prediction info, VOL_ENUM_TYPE -- ignoring

- it appears to be in Mono records as well -- why? -- ignoring

ignoring SEC_CD

ask if 007 wanted -- if so, add 007 for non-language material

- may ht go into holdings rec also

command line dnh copyrun stuff added yet (from INI) -- why not a sub for all this, as it is duplicated (rejected)

want circ data in holdings records? -- no

need addl cmd line param for "copy run", will require specifying overwrite -- no

M780 -- we don't know which relationship to code in ind2, so we take the first one (there is no default)

- likewise we dnk whether they want to display the note (ind1), but we can change if desired
- same for M785
- for M787, ind1 question only

ignoring ISSUE_ENUM_TYPE, VOL_ENUM_TYPE, and ENUM_TYPE3 (error?) in Journals_cat.txt

asked about MEDIUM in Journals_cat. client didn't know where to put it, but said it means they have a paper copy on the shelf. In that case i think i'll rely on the CALL to tell us that and ignore MEDIUM -- ie, it's more of a holdings datafield. need to verify on larger sample

008 is totally fill chars except Date entered on file. We could generate the pub date if needed -- but it's legal as it stands. we could generate the Place of pub code for a price, but probably not too well

make sure only one 1xx -- using first one encountered in record. change inds when moving from 1xx to 7xx

need to find # of nonfiling chars (english only). M130 field, 245, maybe others

recheck that M505 OK, adding CONTENTS both mapping to 505a -- CONTENTS getting wrong indicators. it is illegal in MC to have two fields mapping to the same MARC element.

- data in M505 actually belongs in 504, what should I do? -- map to some other note and NOTE THIS IN HELP FILE so we don't look stupid
- now she says there is also an M504 field, so trying to get her to correct this rec

what is M049 -- ignore it. client says same as item number in copy file, but who needs that in the bib rec?

what is purpose of M008 -- client says to ignore this

some of the stuff in Realia_cat.txt is actually kits -- ignoring this for now. too hard to identify kits

some of the stuff in Videos_cat.txt is actually sound recordings -- we adjust the 007 and leader/06 for this

unique ids in holdings records

ask her to look at techlib_copy.mci at flds not used -- because they look like circ fields

- CALL just repeats data in bib records

FFF

Purpose

To convert MARC21 to UNIMARC (monographs only). This conversion does not handle rarely used MARC21 fields and subfields.

This can occur on several levels:

- Meeting the format requirements (moderately hard, as it is hard to find out how to convert certain data)
- Converting character representation and other codes (easy)
- Ensuring all data is in the right place (difficult to impossible, as some decisions cannot be programmed). Examples are breaking personal names into surname and forenames, and separating 500 notes into various categories.

As it turns out, if we were to handle all possible formats and fields, it would have taken years. So we basically did those in the sample file, plus a few extras.

The conversion skips records for serials. Non-language material like sheet music seems to work, but cartographic and other non-language material will get a lot of errors for unmapped fields.

Codes for countries, national bibliographic services, relators, and sources must be converted from MARC21 codes to UNIMARC codes. These mappings are stored in text files.

UNI 100/00-07 requires a 4-digit year. We are assuming the client is selling current materials, so we make the century 20 if year is 10 or less, 19 otherwise. If they want NO century, just ask. Or they can have the date the record was converted by MC.

M21 008 dates -- cannot convert 0's to blanks (to indicate unknown decade or year), since the 0's may be meaningful.

Client can add to maps as codes change. No order is required, but alpha is easier to maintain. Note that 102 country of publication is not required, so this can be dropped.

Maps must be in the directory where the program is. We can change this if they wish.

Illegal MARC-8 chars are dropped. Escaped chars and certain other MARC-8 chars are not handled yet -- warnings are given. See "MV/MC Character Conversion -- MARC-8 to UNIMARC".

Haven't dealt with punctuation.

Problems

Gets slower as it runs. Not sure if because so many errors in report, or something screwy with the nodes

Releases

1.1 Corrections to 606/607 2d ind, 200 subtitles vs parallel titles, 702 forenames.

1.2 Console window closes at end of run.

GGG

Purpose

To convert MARC record in UTF-8 to MARC records in MARC-8. Command-line only. Command parameters must include output and report filenames.

Works only in console mode, whether or not there are parameters. If parameters, it tries to convert; if none, it shows an error message.

This conversion was originally programmed before the MarcToMarc module was developed. It has been since moved to MtM, but has not been tested.

HHH

Purpose

To convert UTF-8 to MARC-8. Uses Codetables.

JJJ

Purpose

To convert MARC records to Heritage Canada's Bibliographic Conservation Information Network (BCIN) format. Much special processing.

KKK

Purpose

To output MARC records to three tab-delimited files for import into The Museum System.

Releases

1.2 Added command mode.

1.3 Only first 035z in a record is output.

LLL

Purpose

To convert a file of MARC 21 records into a blocked file as specified at http://www.loc.gov/marc/specifications/specexchtape1.html#seg

No character conversion, but checks for illegal characters.

Releases

1.1 Console closes automatically after run.

1.2 Added check for illegal chars.

1.3 Added output of last record if file is truncated.

1.4 Skip record(s) if Record Terminator not where Record Length expects it

MMM

Purpose

To export files of 1 million or so MARC21 records to delimited files for import into SQL Server. The output format is very client-specific.

Releases

2.0 Added automatic load into SQL Server using BCP.

2.1 Empty SS tables before the run

2.2 Console closes automatically at end of run

2.3 Console closes automatically at end of run only if no errors.

2.4 Added indexing of the SQL Server database after loading. Also drop subfield delimiters and codes from VAR_FIELDS2.

NNN

Purpose

To convert MARC21 records to tagged-text. No non-ASCII characters.

Since the output tags are identical to the MARC tags (except for LDR) but preceded by a #, it seems silly to require a Translation Table: it would just contain all the possible MARC tags with the same in the next column. In other words, for this client we are not using a TT.

Also doesn't need linked-list of nodes, since we can just directly output with minor modifications.

The 001 tag is output as #NEW instead of #001.

PPP

Purpose

To convert MARC21 records from HW Wilson into SQL Server. Each MARC record results in one MAIN record and 0 or more records for five other tables. Each table has its own integer primary key as well as the MAIN record's. Also convert MARC-8 to UTF-8.

Here is the error-handling desired:

- convert illegal chars to Geta -- these are not considered bad records
- bad records are written to the report
- increment badcount for bad recs
- continue processing a file after bad rec
- halt batch after a file with a bad rec
- report errors

Use of SubfieldSep and RepeatSep quite tricky, but OK if they are both the same as they are now ("; "). If either gets changed, retest 072, 110, and multiple x/y/z in 650/651.

Some character-conversion errors are shown twice in the report, once when the field is used, and again when the entire record is stored.

QQQ

Purpose

To convert MARC records in UTF-8 to MARC records in MARC-8. This does not use the Codetables as the UTF-8 characters used are the most common ones, which are already in Symbols.cpp.

RRR

Purpose

To import a tab-delimited file into MARC. The file contains the MARC content designators so no Translation Table is required. Instead we pre-read the input file to build the linked list of translation nodes.

SSS

Purpose

To convert Ovid Medline records to MARC.

The file has fieldnames and data on alternating lines, so it is neither tagged nor delimited.

Experimental Feature

MARCSQL

MARCSQL is an experimental feature to provide a SQL-like syntax for advanced searching. It has never been released. It currently works only on the currently open file. (Later it might be extended to (a) open the file; (b) work on multiple files, such as finding bib and holdings data for the same item.)

The search results fill the NavGrid, from which the results can be navigated and viewed.

SQL uses Find under the covers. It repeats the Find for each query term.

Since MARCXML records are read into a MARC structure, they can be searched as well; the user should specify the usual MARC tags and sfcs, not XML tags and attributes.

To turn this feature on, change TheRelease to "4.0" in MainForm. It will cause a "SQL" button to appear, as well as additional menu items.

Query syntax

Definitions:

- Clause: a main part of a SQL query, such as the select clause or the where clause.
- Term: a single search specification, such as `110='new york city'`

Since we retrieve full records rather than data fields, the select clause is not required.

Since currently a query would work only on the current file, the from clause is not required.

Since it is almost impossible to enter a MARC/ISBD field exactly, and there is little reason to, it seems that "=" should really mean "like", so that

```
where 100='Jones'
```

should mean

```
where 100 like '%Jones%'
```

When the user really does want an exact match, the "=" should be replaced by "==".

As with Find, it seems most useful to default to case-insensitivity. Double-quotes instead of single-quotes are currently used if case-sensitivity is wanted.

Thus the default search doesn't require the user to know SQL, which we can't expect anyway. They will just enter something like:

```
1xx=jones and 245$a=chemistry and 260$a='new york'
```

and get results. A more sophisticated user could enter

```
1xx="Jones" and 245$a='chemistry and physics' and 260$a=='new
york'
```

Note that if "and" is part of a search term, the term must be in quotes (single or double). In the above case we are looking for the phrase "chemistry and physics"; to specify that the 245a must contain "chemistry" and "physics", the query would be

```
1xx="Jones" and 245$a=chemistry and 245$a=physics and 260$a=='new
york'
```

(except that we are currently limiting searches to three terms).

"Not" searches are done in the SQL way:

```
1xx="Jones" and 260$a<>'new york'
```

rather than using "not" as an operator. The "not" operator is probably more familiar to librarians, but a query like this just looks too weird to be considered:

```
not 260$a='new york'
```

The new end-user functionalities we are adding with this SQL capability (beyond what Find offers) are these:

- phrase searching
- boolean search (and/or/not)

A search like 035='' finds all records with 035 fields.

Searches are currently limited to three terms (see **Multiple-term searches** below).

Other features

MV keeps a list of recent queries. Selecting one puts it into the textbox to be edited or reused.

Searches are limited to 10,000 hits because the NavigationGrid becomes slower and less valuable the larger it grows.

The user can interrupt a long-running search.

Implementation

MARCSQL is implemented by using the existing Find functionality, used as many times as there are search terms and merging the results; that is, we scan the file only once and apply all Finds to each record in turn. Naturally all SQL Finds must start with the first record.

The Goto, Find, and Again buttons are turned off when the user is viewing a result set, because these work on the entire file, not the result set. It's too confusing for the user, and too hard to move the cursor to the right place in the NavGrid. The SQL Reset menu item restores the full file to the NavGrid and turns these buttons back on (as of course reopening the file does too). Alternatively the user can perform another SQL query.

Multiple-term searches

If the query contains multiple terms, the file is scanned only once. As each record is read, it is compared it to all the search terms; if it passes, its number is saved, otherwise not. The end result is a single list of record numbers.

This approach avoids any necessity for query optimization, such as trying to find the smallest set first. No matter what the search, we have to read the entire file sequentially, so it matters little in which order we look for the terms in the records. (This is because there are no indexes, making this different from other SQL situations.) It will probably be very little slower, if any, than the existing Find. However, it would be smart to find a way to short-circuit checking for 2d and 3d terms if the first term for the record rejects it (not easy, because AND, OR, and NOT each call for different logic).

Nesting

Since we allow only three terms, the user's only meaningful options are to nest either the first two or the last two. MV's policy is that nesting the first two yields the same results as no nesting.

Nesting the last two is handled by rearranging the terms so that the first is last, thereby allowing the search to be processed in the normal sequential order.

Future ideas

More than three terms.

Order by.

Known Problems

MARC-to-MARC

Nodes being wasted

In FFF's UNIMARC conversion, each 040$a goes to a $b in a separate 801 field. However, because the target is 801$b, the nodetype to be copied in yTrans::InsertString is SUB rather than FIELD. So any new instances of 801 created for a record are ignored by the next record. So at the end of a large file we end up with tons of unused 801's. This may happen with other fields as well.

Does not affect results, but reduces performance as we have a huge tree to look thru for each insertion.

Believe solved by adding InsertField and modifying InsertSubfield.

Still a problem with unimarc 327's -- sf nodes nb reused

Also unimarc 210's -- seems to create a new one whenever a 2d one is needed.

Installation

If installed to a network drive, each computer needs to have an INI file. Add this to MC Help file (not relevant to MV unless they have a site license).

MARCSQL

If more than 10,000 hits, the file is subsetted anyway, but the Navigation Grid is missing. It should "reset" if >10,000 hits. Also the Reset menu item is inactive, so you have to reopen the file.

After a successful SQL and then a too-many-hits as in previous paragraph, Reset seems to do nothing