Record.pm

Table of Contents
Record.pm

NAME

MARC::Record - Perl extension for handling MARC records

VERSION 1.32

```
    $Id: Record.pm,v 1.69 2003/11/06 15:30:26 edsummers Exp $
```

DESCRIPTION

Module for handling MARC records as objects. The file-handling stuff is in MARC::File::*.

EXPORT

None.

ERROR HANDLING

Any errors generated are stored in `$MARC::Record::ERROR`. Warnings are kept with the record and accessible in the `warnings()` method.

CONSTRUCTORS

new()

Base constructor for the class. It just returns a completely empty record. To get real data, you‘ll need to populate it with fields, or use one of the MARC::File::* modules to read from a file.

new_from_usmarc($marcblob [, \&filter_func($tagno,$tagdata)])

This is a wrapper around `MARC::File::USMARC::decode()` for compatibility with older versions of MARC::Record.

The `wanted_func()` is optional. See *MARC::File::USMARC*::decode for details.

COMMON FIELD RETRIEVAL METHODS

Following are a number of convenience methods for commonly-retrieved data fields. Please note that they each return strings, not MARC::Field objects. They return empty strings if the appropriate field or subfield is not found. This is as opposed to the field()/subfield() methods which return `undef` if something‘s not found. My assumption is that these methods are used for quick & dirty reports and you don‘t want to mess around with noting if something is undef.

Also note that no punctuation cleanup is done. If the 245a is "Programming Perl / ", then that‘s what you‘ll get back, rather than "Programming Perl".

title()

Returns the title from the 245 tag.

title_proper()

Returns the title proper from the 245 tag, subfields a, n and p.

author()

Returns the author from the 100, 110 or 111 tag.

edition()

Returns the edition from the 250 tag, subfield a.

publication_date()

Returns the publication date from the 260 tag, subfield c.

FIELD & SUBFIELD ACCESS METHODS

fields()

Returns a list of all the fields in the record. The list contains a MARC::Field object for each field in the record.

field(tagspec(s))

Returns a list of tags that match the field specifier, or in scalar context, just the first matching tag.

The field specifier can be a simple number (i.e. "245"), or use the "." notation of wildcarding (i.e. subject tags are "6..").

subfield(tag,subfield)

Shortcut method for getting just a subfield for a tag. These are equivalent:

```
my $title = $marc->field('245')->subfield("a");
my $title = $marc->subfield('245',"a");
```

If either the field or subfield can't be found, `undef` is returned.

append_fields(@fields)

Appends the field specified by `$field` to the end of the record. `@fields` need to be MARC::Field objects.

```
    my $field = MARC::Field->new('590','','','a' => 'My local note.');
    $record->append_fields($field);
```

Returns the number of fields appended.

insert_fields_before($before_field,@new_fields)

Inserts the field specified by `$new_field` before the field `$before_field`. Returns the number of fields inserted, or undef on failures. Both `$before_field` and all `@new_fields` need to be MARC::Field objects. If they are not an exception will be thrown.

```
    my $before_field = $record->field('260');
    my $new_field = MARC::Field->new('250','','','a' => '2nd ed.');
    $record->insert_fields_before($before_field,$new_field);
```

insert_fields_after($after_field,@new_fields)

Identical to `insert_fields_before()`, but fields are added after `$after_field`. Remeber, $after_field and any new fields must be valid MARC::Field objects or else an exception will be thrown.

insert_fields_ordered(@new_fields)

Will insert fields in strictly numerical order. So a 008 will be filed after a 001 field. See insert_grouped_field() for an additional ordering.

insert_grouped_field(field)

Will insert the specified MARC::Field object into the record in grouped order and return true (1) on success, and false (undef) on failure.

```
    my $field = MARC::Field->new( '510', 'Indexed by Google.' );
    $record->insert_grouped_field( $field );
```

For example, if a '650' field is inserted with insert_grouped_field() it will be inserted at the end of the 6XX group of tags. After discussion most people wanted the ability to add a new field to the end of the hundred group where it belonged. The reason is that according to the MARC format, fields within a record are supposed to be grouped by block (hundred groups). This means that fields may not necessarily be in tag order.

delete_field($field)

Deletes a field from the record.
The field must have been retrieved from the record using the `field()` method. For example, to delete a 526 tag if it exists:

```
    my $tag526 = $marc->field( "526" );
    if ( $tag526 ) {
        $marc->delete_field( $tag526 );
    }
```

`delete_field()` returns the number of fields that were deleted. This shouldn't be 0 unless you didn't get the tag properly.

as_usmarc()

This is a wrapper around `MARC::File::USMARC::encode()` for compatibility with older versions of MARC::Record.

as_formatted()

Returns a pretty string for printing in a MARC dump.

leader()

Returns the leader for the record. Sets the leader if *text* is defined. No error checking is done on the validity of the leader.

set_leader_lengths($reclen,$baseaddr)

Internal function for updating the leader‘s length and base address.

clone()

The `clone()` method makes a copy of an existing MARC record and returns the new version. Note that you cannot just say:

```
my $newmarc = $oldmarc;
```

This just makes a copy of the reference, not a new object. You must use the `clone()` method like so:

```
my $newmarc = $oldmarc->clone;
```

You can also specify field specs to filter down only a certain subset of fields. For instance, if you only wanted the title and ISBN tags from a record, you could do this:

```
my $small_marc = $marc->clone( 245, ’020’ );
```

The order of the fields is preserved as it was in the original record.

warnings()

Returns the warnings (as a list) that were created when the record was read. These are things like "Invalid indicators converted to blanks".

```
my @warnings = $record->warnings();
```

The warnings are items that you might be interested in, or might not. It depends on how stringently you‘re checking data. If you‘re doing some grunt data analysis, you probably don‘t care.
A side effect of calling warnings() is that the warning buffer will be cleared.

add_fields()

`add_fields()` is now deprecated, and users are encouraged to use `append_fields()`, `insert_fields_after()`, and `insert_fields_before()` since they do what you want probably. It is still here though, for backwards compatability.
`add_fields()` adds MARC::Field objects to the end of the list. Returns the number of fields added, or `undef` if there was an error.
There are three ways of calling `add_fields()` to add data to the record.

1 Create a MARC::Field object and add it

```
my $author = MARC::Field->new(
              100, "1", " ", a => "Arnosky, Jim."
              );
$marc->add_fields( $author );
```

2 Add the data fields directly, and let `add_fields()` take care of the objectifying.

```
$marc->add_fields(
      245, "1", "0",
              a => "Raccoons and ripe corn /",
              c => "Jim Arnosky.",
              );
```

3 Same as #2 above, but pass multiple fields of data in anonymous lists

```
$marc->add_fields(
      [ 250, " ", " ", a => "1st ed." ],
      [ 650, "1", " ", a => "Raccoons." ],
      );
```

DESIGN NOTES

A brief discussion of why MARC::Record is done the way it is:

- It‘s built for quick prototyping

  One of the areas Perl excels is in allowing the programmer to create easy solutions quickly. MARC::Record is designed along those same lines. You want a program to dump all the 6XX tags in a file? MARC::Record is your friend.

- It‘s built for extensibility

  Currently, I‘m using MARC::Record for analyzing bibliographic data, but who knows what might happen in the future? MARC::Record needs to be just as adept at authority data, too.
- It‘s designed around accessor methods

  I use method calls everywhere, and I expect calling programs to do the same, rather than accessing internal data directly. If you access an object‘s hash fields on your own, future releases may break your code.
- It‘s not built for speed

  One of the tradeoffs in using accessor methods is some overhead in the method calls. Is this slow? I don‘t know, I haven‘t measured. I would suggest that if you‘re a cycle junkie that you use Benchmark.pm to check to see where your bottlenecks are, and then decide if MARC::Record is for you.

RELATED MODULES

MARC::Record, *MARC::Lint*

SEE ALSO

- perl4lib (*http://www.rice.edu/perl4lib/*)

  A mailing list devoted to the use of Perl in libraries.
- Library Of Congress MARC pages (*http://www.loc.gov/marc/*)

  The definitive source for all things MARC.
- *Understanding MARC Bibliographic* (*http://lcweb.loc.gov/marc/umb/*)

  Online version of the free booklet. An excellent overview of the MARC format. Essential.
- Tag Of The Month (*http://www.tagofthemonth.com/*)

  Follett Software Company‘s (*http://www.fsc.follett.com/*) monthly discussion of various MARC tags.

TODO

- Incorporate MARC.pm in the distribution.

  Combine MARC.pm and MARC::* into one distribution.
- Podify MARC.pm
- Allow regexes across the entire tag

  Imagine something like this:

  ```
    my @sears_headings = $marc->tag_grep( /Sears/ );
  ```

  (from Mike O‘Regan)
- Insert a field in an arbitrary place in the record
- Allow deleting a field

  ```
    for my $field ( $record->field( "856" ) ) {
          $record->delete_field( $field ) unless useful($field);
          } # for
  ```

  (from Anne Highsmith hismith@tamu.edu)
- Modifying an existing field

BUGS, WISHES AND CORRESPONDENCE

Please feel free to email me at andy@petdance.com. I‘m glad to help as best I can, and I‘m always interested in bugs, suggestions and patches.

The MARC::Record development team uses the RT bug tracking system at *http://rt.cpan.org*. If your email is about a bug or suggestion, please report it through the RT system. This is a huge help for the team, and you‘ll be notified of progress as things get fixed or updated. If you prefer not to use the website, you can send your bug to bug-MARC-Record@rt.cpan.org.

IDEAS

Ideas are things that have been considered, but nobody‘s actually asked for.

- Create multiple output formats.

  These could be ASCII, XML, or MarcMaker.

LICENSE

This code may be distributed under the same terms as Perl itself.

Please note that these modules are not products of or supported by the employers of the various contributors to the code.

AUTHOR

Andy Lester, <marc@petdance.com>