POD Translation by *pod2pdf*

ajf@afco.demon.co.uk

Batch.pm

Table of Contents
Batch.pm

NAME

MARC::Batch - Perl module for handling files of MARC::Record objects

SYNOPSIS

MARC::Batch hides all the file handling of files of `MARC::Record`s. `MARC::Record` still does the file I/O, but `MARC::Batch` handles the multiple-file aspects.

```
use MARC::Batch;

my $batch = new MARC::Batch( 'USMARC', @files );
while ( my $marc = $batch->next ) {
    print $marc->subfield(245,"a"), "\n";
}
```

EXPORT

None. Everything is a class method.

METHODS

new($type, @files)

Create a `MARC::Batch` object that will process `@files`.
`$type` must be either "USMARC" or "MicroLIF". If you want to specify "MARC::File::USMARC" or "MARC::File::MicroLIF", that's OK, too. `new()` returns a new MARC::Batch object.
`@files` can be a list of filenames:

```
my $batch = MARC::Batch->new( 'USMARC', 'file1.marc', 'file2.marc' );
```

Your `@files` may also contain filehandles. So if you've got a large file that's gzipped you can open a pipe to ***gzip*** and pass it in:

```
my $fh = IO::File->new( 'gunzip -c marc.dat.gz |' );
my $batch = MARC::Batch->new( 'USMARC', $fh );
```

And you can mix and match if you really want to:

```
my $batch = MARC::Batch->new( 'USMARC', $fh, 'file1.marc' );
```

next()

Read the next record from that batch, and return it as a MARC::Record object. If the current file is at EOF, close it and open the next one. `next()` will return `undef` when there is no more data to be read from any batch files.
By default, `next()` also will return `undef` if an error is encountered while reading from the batch. If not checked for this can cause your iteration to terminate prematurely. To alter this behavior, see `strict_off()`. You can retrieve warning messages using the `warnings()` method.
Optionally you can pass in a filter function as a subroutine reference if you are only interested in particular fields from the record. This can boost performance.

strict_off()

If you would like `MARC::Batch` to continue after it has encountered what it believes to be bad MARC data then use this method to turn strict **OFF**. A call to `strict_off()` always returns true (1).
`strict_off()` can be handy when you don't care about the quality of your MARC data, and just want to plow through it. For safety, `MARC::Batch` strict is **ON** by default.

strict_on()

The opposite of `strict_off()`, and the default state. You shouldn't have to use this method unless you've previously used `strict_off()`, and want it back on again. When strict is **ON** calls to next() will return undef when an error is encountered while reading MARC data. strict_on() always returns true (1).

warnings()

Returns a list of warnings that have accumulated while processing a particular batch file. As a side effect the warning buffer will be cleared.

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

This method is also used internally to set warnings, so you probably don't want to be passing in anything as this will set warnings on your batch object.

`warnings()` will return the empty list when there are no warnings.

warnings_off()

Turns off the default behavior of printing warnings to STDERR. However, even with warnings off the messages can still be retrieved using the warnings() method if you wish to check for them.
`warnings_off()` always returns true (1).

warnings_on()

Turns on warnings so that diagnostic information is printed to STDERR. This is on by default so you shouldn't have to use it unless you've previously turned off warnings using warnings_off().
warnings_on() always returns true (1).

filename()

Returns the currently open filename or `undef` if there is not currently a file open on this batch object.

RELATED MODULES

MARC::Record, *MARC::Lint*

TODO

None yet. Send me your ideas and needs.

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, `< <andy@petdance.com>`