## NAME

ExtUtils::MM_Any - Platform-agnostic MM methods

## SYNOPSIS

```
FOR INTERNAL USE ONLY!

package ExtUtils::MM_SomeOS;

# Temporarily, you have to subclass both.  Put MM_Any first.
require ExtUtils::MM_Any;
require ExtUtils::MM_Unix;
@ISA = qw(ExtUtils::MM_Any ExtUtils::Unix);
```

## DESCRIPTION

**FOR INTERNAL USE ONLY!**

ExtUtils::MM_Any is a superclass for the ExtUtils::MM_* set of modules. It contains methods which are either inherently cross-platform or are written in a cross-platform manner.

Subclass off of ExtUtils::MM_Any *and* ExtUtils::MM_Unix. This is a temporary solution.

**THIS MAY BE TEMPORARY!**

## Inherently Cross-Platform Methods

These are methods which are by their nature cross-platform and should always be cross-platform.

installvars

```
my @installvars = $mm->installvars;
```

A list of all the INSTALL* variables without the INSTALL prefix. Useful for iteration or building related variable sets.

os_flavor_is

```
$mm->os_flavor_is($this_flavor);
$mm->os_flavor_is(@one_of_these_flavors);
```

Checks to see if the current operating system is one of the given flavors.

This is useful for code like:

```
if( $mm->os_flavor_is('Unix') ) {
    $out = `foo 2>&1`;
}
else {
    $out = `foo`;
}
```

## File::Spec wrappers

ExtUtils::MM_Any is a subclass of File::Spec. The methods noted here override File::Spec.

catfile

File::Spec <= 0.83 has a bug where the file part of catfile is not canonicalized. This override fixes that bug.

## Thought To Be Cross-Platform Methods

These are methods which are thought to be cross-platform by virtue of having been written in a way to avoid incompatibilities. They may require partial overrides.

**split_command**

```
    my @cmds = $MM->split_command($cmd, @args);
```

Most OS have a maximum command length they can execute at once. Large modules can easily generate commands well past that limit. Its necessary to split long commands up into a series of shorter commands.

split_command() will return a series of @cmds each processing part of the args. Collectively they will process all the arguments. Each individual line in @cmds will not be longer than the $self->max_exec_len being careful to take into account macro expansion.

$cmd should include any switches and repeated initial arguments.

If no @args are given, no @cmds will be returned.

Pairs of arguments will always be preserved in a single command, this is a heuristic for things like pm_to_blib and pod2man which work on pairs of arguments. This makes things like this safe:

```
    $self->split_command($cmd, %pod2man);
```

**echo**

```
    my @commands = $MM->echo($text);
    my @commands = $MM->echo($text, $file);
    my @commands = $MM->echo($text, $file, $appending);
```

Generates a set of @commands which print the $text to a $file.

If $file is not given, output goes to STDOUT.

If $appending is true the $file will be appended to rather than overwritten.

init_VERSION

```
    $mm->init_VERSION
```

Initialize macros representing versions of MakeMaker and other tools

MAKEMAKER: path to the MakeMaker module.

MM_VERSION: ExtUtils::MakeMaker Version

MM_REVISION: ExtUtils::MakeMaker version control revision (for backwards compat)

VERSION: version of your module

VERSION_MACRO: which macro represents the version (usually 'VERSION')

VERSION_SYM: like version but safe for use as an RCS revision number

DEFINE_VERSION: -D line to set the module version when compiling

XS_VERSION: version in your .xs file. Defaults to $(VERSION)

XS_VERSION_MACRO: which macro represents the XS version.

XS_DEFINE_VERSION: -D line to set the xs version when compiling.

Called by init_main.

wraplist

Takes an array of items and turns them into a well-formatted list of arguments. In most cases this is simply something like:

```
    FOO \
        BAR \
        BAZ
```

manifypods

Defines targets and routines to translate the pods into manpages and put them into the INST_* directories.

manifypods_target

```
  my $manifypods_target = $self->manifypods_target;
```

Generates the manifypods target. This target generates man pages from all POD files in MAN1PODS and MAN3PODS.

makemakerdflt_target

```
  my $make_frag = $mm->makemakerdflt_target
```

Returns a make fragment with the makemakerdeflt_target specified. This target is the first target in the Makefile, is the default target and simply points off to 'all' just in case any make variant gets confused or something gets snuck in before the real 'all' target.

special_targets

```
  my $make_frag = $mm->special_targets
```

Returns a make fragment containing any targets which have special meaning to make. For example, .SUFFIXES and .PHONY.

POD2MAN_macro

```
  my $pod2man_macro = $self->POD2MAN_macro
```

Returns a definition for the POD2MAN macro. This is a program which emulates the pod2man utility. You can add more switches to the command by simply appending them on the macro.

Typical usage:

```
    $(POD2MAN) --section=3 --perm_rw=$(PERM_RW) podfile1 man_page1
...
```

test_via_harness

```
  my $command = $mm->test_via_harness($perl, $tests);
```

Returns a $command line which runs the given set of $tests with Test::Harness and the given $perl.

Used on the t/*.t files.

test_via_script

```
  my $command = $mm->test_via_script($perl, $script);
```

Returns a $command line which just runs a single test without Test::Harness. No checks are done on the results, they're just printed.

Used for test.pl, since they don't always follow Test::Harness formatting.

libscan

```
  my $wanted = $self->libscan($path);
```

Takes a path to a file or dir and returns an empty string if we don't want to include this file in the library. Otherwise it returns the the $path unchanged.

Mainly used to exclude RCS, CVS, and SCCS directories from installation.

tool_autosplit

Defines a simple perl call that runs autosplit. May be deprecated by pm_to_blib soon.

all_target

Generate the default target 'all'.

metafile_target

```
    my $target = $mm->metafile_target;
```

Generate the metafile target.

Writes the file META.yml, YAML encoded meta-data about the module. The format follows Module::Build's as closely as possible. Additionally, we include:

```
    version_from
    installdirs
```

metafile_addtomanifest_target

```
  my $target = $mm->metafile_addtomanifest_target
```

Adds the META.yml file to the MANIFEST.

## Abstract methods

Methods which cannot be made cross-platform and each subclass will have to do their own implementation.

oneliner

```
  my $oneliner = $MM->oneliner($perl_code);
  my $oneliner = $MM->oneliner($perl_code, \@switches);
```

This will generate a perl one-liner safe for the particular platform you're on based on the given $perl_code and @switches (a -e is assumed) suitable for using in a make target. It will use the proper shell quoting and escapes.

$(PERLRUN) will be used as perl.

Any newlines in $perl_code will be escaped. Leading and trailing newlines will be stripped. Makes this idiom much easier:

```
    my $code = $MM->oneliner(<<'CODE', [...switches...]);
some code here
another line here
CODE
```

Usage might be something like:

```
    # an echo emulation
    $oneliner = $MM->oneliner('print "Foo\n"');
    $make = '$oneliner > somefile';
```

All dollar signs must be doubled in the $perl_code if you expect them to be interpreted normally, otherwise it will be considered a make macro. Also remember to quote make macros else it might be used as a bareword. For example:

```
    # Assign the value of the $(VERSION_FROM) make macro to $vf.
    $oneliner = $MM->oneliner('$$vf = "$(VERSION_FROM)"');
```

Its currently very simple and may be expanded sometime in the figure to include more flexible code and switches.

**quote_literal**

```
    my $safe_text = $MM->quote_literal($text);
```

This will quote $text so it is interpreted literally in the shell.

For example, on Unix this would escape any single-quotes in $text and put single-quotes around the whole thing.

**escape_newlines**

```
    my $escaped_text = $MM->escape_newlines($text);
```

Shell escapes newlines in $text.

max_exec_len

```
    my $max_exec_len = $MM->max_exec_len;
```

Calculates the maximum command size the OS can exec. Effectively, this is the max size of a shell command line.

**init_others**

```
    $MM->init_others();
```

Initializes the macro definitions used by tools_other() and places them in the $MM object.

If there is no description, its the same as the parameter to WriteMakefile() documented in ExtUtils::MakeMaker.

Defines at least these macros.

```
  Macro             Description

  NOOP              Do nothing
  NOECHO            Tell make not to display the command itself

  MAKEFILE
  FIRST_MAKEFILE
  MAKEFILE_OLD
  MAKE_APERL_FILE   File used by MAKE_APERL

  SHELL             Program used to run
                    shell commands

  ECHO              Print text adding a newline on the end
  RM_F              Remove a file
  RM_RF             Remove a directory
  TOUCH             Update a file’s timestamp
  TEST_F            Test for a file’s existence
  CP                Copy a file
  MV                Move a file
  CHMOD             Change permissions on a
                    file

  UMASK_NULL        Nullify umask
  DEV_NULL          Supress all command output
```

init_DIRFILESEP

```
  $MM->init_DIRFILESEP;
  my $dirfilesep = $MM->{DIRFILESEP};
```

Initializes the DIRFILESEP macro which is the seperator between the directory and filename in a filepath. ie. / on Unix, \ on Win32 and nothing on VMS.

For example:

```
    # instead of $(INST_ARCHAUTODIR)/extralibs.ld
    $(INST_ARCHAUTODIR)$(DIRFILESEP)extralibs.ld
```

Something of a hack but it prevents a lot of code duplication between MM_* variants.

Do not use this as a seperator between directories. Some operating systems use different seperators between subdirectories as between directories and filenames (for example: VOLUME:[dir1.dir2]file on VMS).

init_linker

```
    $mm->init_linker;
```

Initialize macros which have to do with linking.

PERL_ARCHIVE: path to libperl.a equivalent to be linked to dynamic extensions.

PERL_ARCHIVE_AFTER: path to a library which should be put on the linker command line *after* the external libraries to be linked to dynamic extensions. This may be needed if the linker is one-pass, and Perl includes some overrides for C RTL functions, such as malloc().

EXPORT_LIST: name of a file that is passed to linker to define symbols to be exported.

Some OSes do not need these in which case leave it blank.

init_platform

```
    $mm->init_platform
```

Initialize any macros which are for platform specific use only.

A typical one is the version number of your OS specific mocule. (ie. MM_Unix_VERSION or MM_VMS_VERSION).

platform_constants

```
    my $make_frag = $mm->platform_constants
```

Returns a make fragment defining all the macros initialized in init_platform() rather than put them in constants().

os_flavor

```
    my @os_flavor = $mm->os_flavor;
```

@os_flavor is the style of operating system this is, usually corresponding to the MM_*.pm file we're using.

The first element of @os_flavor is the major family (ie. Unix, Windows, VMS, OS/2, MacOS, etc...) and the rest are sub families.

Some examples:

```
    Cygwin98       ('Unix',  'Cygwin', 'Cygwin9x')
    Windows NT     ('Win32', 'WinNT')
    Win98          ('Win32', 'Win9x')
    Linux          ('Unix',  'Linux')
    MacOS Classic  ('MacOS', 'MacOS Classic')
```

```
MacOS X         ('Unix',  'Darwin', 'MacOS', 'MacOS X')
OS/2            ('OS/2')
```

This is used to write code for styles of operating system. See os_flavor_is() for use.

## AUTHOR

Michael G Schwern <schwern@pobox.com> and the denizens of makemaker@perl.org with code from ExtUtils::MM_Unix and ExtUtils::MM_Win32.