Dynext: Running a C Compiler/Linker

Version 9.4.0.5

November 15, 2026

The `"dynext"` collection provides libraries for using a platform-specific C compiler and linker.

Contents

1 Compilation **3**

1.1 Compilation Parameters 3

1.2 Helper functions 5

1.3 Signature 5

1.4 Unit 6

2 Linking **7**

2.1 Linking Parameters 7

2.2 Helper Functions 9

2.3 Signature 9

2.4 Unit 9

3 Filenames **10**

3.1 Signature 11

3.2 Unit 11

1 Compilation

```
(require dynext/compile)
```
package: dynext-lib

```
(compile-extension quiet?
                   input-file
                   output-file
                   include-dirs) → any/c
  quiet? : any/c
  input-file : path-string?
  output-file : path-string?
  include-dirs : (listof path-string?)
```

Compiles the given input file (C source) to the given output file (a compiled-object file). The *quiet?* argument indicates whether command should be echoed to the current output port. The *include-dirs* argument is a list of directories to search for include files; the Racket installation's "include" directories are added automatically.

1.1 Compilation Parameters

```
(current-extension-compiler) → (or/c path-string? #f)
(current-extension-compiler compiler) → void?
  compiler : (or/c path-string? #f)
```

A parameter that determines the executable for the compiler.

The default is set by searching for an executable using the PATH environment variable, or using the CC or MZSCHEME_DYNEXT_COMPILER environment variable if either is defined (and the latter takes precedence). On Windows, the search looks for "cl.exe", then "gcc.exe", then "bcc32.exe" (Borland). On Unix, it looks for "gcc", then "cc". A #f value indicates that no compiler could be found.

```
(current-extension-compiler-flags)
 → (listof (or/c path-string?
                 (-> (or/c null? (listof string?)))))
(current-extension-compiler-flags flags) → void?
  flags : (listof (or/c path-string?
                        (-> (or/c null? (listof string?)))))
```

A parameter that determines strings passed to the compiler as flags. See also expand-for-compile-variant.

On Windows, the default is (list "/c" "/O2" "/MT" *3m-flag-thunk*) for "cl.exe", or (list "-c" "-O2" "-fPIC" *3m-flag-thunk*) for "gcc.exe" and "bcc32.exe",

where *3m-flag-thunk* returns (list "-DMZ_PRECISE_GC") for the 3m variant and null for the CGC variant. On Unix, the default is usually (list "-c" "-O2" "-fPIC" *3m-flag-thunk*). If the CFLAGS or MZSCHEME_DYNEXT_COMPILER_FLAGS environment variable is defined (the latter takes precedence), then its value is parsed as a list of strings that is appended before the defaults.

```
(current-make-compile-include-strings)
 → (-> path-string? (listof string?))
(current-make-compile-include-strings proc) → void?
  proc : (-> path-string? (listof string?))
```

A parameter the processes include-path inputs to the compiler; the parameter values takes an include directory path and returns a list of strings for the command line.

On Windows, the default converts "dir" to (list "/Idir") for "cl.exe", (list "-Idir") for "gcc.exe" and "bcc32.exe". On Unix, the default converts "dir" to (list "-Idir"). If the CFLAGS environment variable is defined, then its value is parsed as a list of flags that is appended before the defaults.

```
(current-make-compile-input-strings)
 → (-> (or/c string? path?) (listof string?))
(current-make-compile-input-strings proc) → void?
  proc : (-> (or/c string? path?) (listof string?))
```

A parameter that processes inputs to the compiler; the parameter's values takes an input file path and returns a list of strings for the command line. The default is list.

```
(current-make-compile-output-strings)
 → (-> (or/c string? path?) (listof string?))
(current-make-compile-output-strings proc) → void?
  proc : (-> (or/c string? path?) (listof string?))
```

A parameter that processes outputs specified for the compiler; the parameter's value takes an output file path and returns a list of strings for the command line.

On Windows, the default converts "file" to (list "/Fofile") for "cl.exe", or to (list "-o" "file") for "gcc.exe" and "bcc32.exe". On Unix, the default converts "file" to (list "-o" "file").

```
(current-extension-preprocess-flags)
 → (listof (or/c string? path? (-> (or/c string? path?))))
(current-extension-preprocess-flags flags) → void?
  flags : (listof (or/c string? path? (-> (or/c string? path?))))
```

A parameters that specifies flags to the compiler preprocessor, instead of to the compiler proper; use these flags for preprocessing instead of current-extension-compiler-flags.

The defaults are similar to `current-extension-compiler-flags`, but with `"/E"` (Windows `"cl.exe"`) or `"-E"` and without non-`"-D"` flags.

```
(compile-variant) → (or/c 'normal 'cgc '3m)
(compile-variant variant-symbol) → void?
  variant-symbol : (or/c 'normal 'cgc '3m)
```

A parameter that indicates the target for compilation, where `'normal` is an alias for the result of `(system-type 'gc)`

1.2 Helper functions

```
(use-standard-compiler name) → any
  name : (apply or/c (get-standard-compilers))
```

Sets the parameters described in §1.1 “Compilation Parameters” for a particular known compiler. The acceptable names are platforms-specific:

- Unix: `'cc` or `'gcc`
- Windows: `'gcc`, `'msvc`, or `'borland`
- MacOS: `'cw`

```
(get-standard-compilers) → (listof symbol?)
```

Returns a list of standard compiler names for the current platform. See `use-standard-compiler`.

```
(expand-for-compile-variant l) → any
  l : (listof (or/c path-string? (-> (listof string?))))
```

Takes a list of paths and thunks and returns a list of strings. Each thunk in the input list is applied to get a list of strings that is inlined in the corresponding position in the output list. This expansion enables occasional parametrization of flag lists, etc., depending on the current compile variant.

1.3 Signature

`(require dynext/compile-sig)` package: `dynext-lib`

```
dynext:compile^ : signature
```

Includes everything exported by the `dynext/compile` module.

1.4 Unit

```
(require dynext/compile-unit)
```
package: `dynext-lib`

```
dynext:compile@ : unit?
```

Imports nothing, exports `dynext:compile^`.

2 Linking

```
(require dynext/link)
```
package: dynext-lib

```
(link-extension quiet?
                input-files
                output-file) → any
  quiet? : any/c
  input-files : (listof path-string?)
  output-file : path-string?
```

Links object files to create an extension (normally of a form that can be loaded with `load-extension`).

The *quiet?* argument indicates whether command should be echoed to the current output port. The *input-files* argument is list of compiled object filenames, and *output-file* is the destination extension filename.

2.1 Linking Parameters

```
(current-extension-linker) → (or/c path-string? #f)
(current-extension-linker linker) → void?
  linker : (or/c path-string? #f)
```

A parameter that determines the executable used as a linker.

The default is set by searching for an executable using the `PATH` environment variable, or by using the `LD` or `MZSCHEME_DYNEXT_LINKER` environment variable if it is defined (and the latter takes precedence). On Windows, it looks for `"cl.exe"`, then `"ld.exe"` (gcc), then `"ilink32.exe"` (Borland). On Cygwin, Solaris, FreeBSD 2.x, or HP/UX, it looks for `"ld"`. On other Unix variants, it looks for `"cc"`. `#f` indicates that no linker could be found.

```
(current-extension-linker-flags)
 → (listof (or/c path-string? (-> (listof string?))))
(current-extension-linker-flags flags) → void?
  flags : (listof (or/c path-string? (-> (listof string?))))
```

A parameter that determines flags provided to the linker. See also `expand-for-link-variant`.

On Windows, default is `(list "/LD")` for `"cl.exe"`, `(list "--dll")` for `"ld.exe"`, and `(list "/Tpd" "/c")` for `"ilink32.exe"`. On Unix, the default varies greatly among platforms. If the `LDFLAGS` or `MZSCHEME_DYNEXT_LINKER_FLAGS` (the latter takes precedence) environment variable is defined, then its value is parsed as a list of strings that is appended before the defaults.

```
(current-make-link-input-strings)
 → (-> path-string? (listof string?))
(current-make-link-input-strings proc) → void?
  proc : (-> path-string? (listof string?))
```

A parameter that processes linker input arguments; the parameter value takes an input file path and returns a list of strings for the command line. The default is `list`.

```
(current-make-link-output-strings)
 → (-> path-string? (listof string?))
(current-make-link-output-strings proc) → void?
  proc : (-> path-string? (listof string?))
```

A parameter that processes linker output arguments; the parameter value takes an output file path and returns a list of strings for the command line.

On Windows, the default converts `"file"` to `(list "/Fefile")` for `"cl.exe"`, something like `(list "-e" "_dll_entry@12" "-o" "file")` for `"ld.exe"`, and something complex for `"ilink32.exe"`. On Unix, the default converts `"file"` to `(list "-o" "file")`.

```
(current-standard-link-libraries)
 → (listof (or/c path-string? (-> (listof string?))))
(current-standard-link-libraries libs) → void?
  libs : (listof (or/c path-string? (-> (listof string?))))
```

A parameter that determines libraries supplied to the linker, in addition to other inputs. See also `expand-for-link-variant`.

For most platforms, the default is

```
(list (build-path (collection-path "mzscheme" "lib")
                  (system-library-subpath)
                  (mzdyn-thunk)))
```

where *mzdyn-thunk* produces `(list "mzdyn.o")` for the `'cgc` variant and `(list "mzdyn3m.o")` for the `'3m` variant. See also `current-use-mzdyn`).

```
(current-use-mzdyn) → boolean?
(current-use-mzdyn use-mzdyn?) → void?
  use-mzdyn? : boolean?
```

A parameter that determines whether the default standard link libraries include the `"mzdyn"` library which allows the resulting file to be loaded via `load-extension`. Defaults to `#t`.

```
(link-variant) → (or/c 'normal 'cgc '3m)
(link-variant variant-symbol) → void?
  variant-symbol : (or/c 'normal 'cgc '3m)
```

A parameter that indicates the target for linking, where `'normal` is an alias for the result of `(system-type 'gc)`.

2.2 Helper Functions

```
(use-standard-linker name) → void?
  name : (or/c 'cc 'gcc 'msvc 'borland 'cw)
```

Sets the parameters described in §2.1 “Linking Parameters” for a particular known linker.

```
(expand-for-link-variant l) → any
  l : (listof (or/c path?
                    string?
                    (-> (listof string?))))
```

The same as `expand-for-compile-variant`.

2.3 Signature

```
(require dynext/link-sig)
```
package: `dynext-lib`

```
dynext:link^ : signature
```

Includes everything exported by the `dynext/link` module.

2.4 Unit

```
(require dynext/link-unit)
```
package: `dynext-lib`

```
dynext:link@ : unit?
```

Imports nothing, exports `dynext:link^`.

3 Filenames

```
(require dynext/file)
```
package: base

```
(append-zo-suffix s) → path?
  s : (or/c string? path?)
```

Appends the `".zo"` file suffix to *s*, returning a path. The existing suffix, if any, is preserved and converted as with `path-add-suffix`.

```
(append-object-suffix s) → path?
  s : path-string?
```

Appends the platform-standard compiled object file suffix to *s*, returning a path.

```
(append-c-suffix s) → path?
  s : path-string?
```

Appends the platform-standard C source-file suffix to *s*, returning a path.

```
(append-constant-pool-suffix s) → path?
  s : (or/c string? path?)
```

Appends the constant-pool file suffix `".kp"` to *s*, returning a path.

```
(append-extension-suffix s) → path?
  s : (or/c string? path?)
```

Appends the platform-standard dynamic-extension file suffix to *s*, returning a path.

```
(extract-base-filename/ss s [program]) → (or/c path? #f)
  s : path-string?
  program : any/c = #f
```

Strips the Racket file suffix from *s* and returns a stripped path. The recognized suffixes are the ones reported by `(get-module-suffixes #:group 'libs)` when `extract-base-filename/ss` is first called.

Unlike the other functions below, when *program* is not `#f`, then any suffix (including no suffix) is allowed. If *s* is not a Racket file and *program* is `#f`, `#f` is returned.

```
(extract-base-filename/c s [program]) → (or/c path? #f)
  s : path-string?
  program : any/c = #f
```

Strips the Racket file suffix from *s* and returns a stripped path. If *s* is not a Racket file name and *program* is a symbol, and error is signaled. If *s* is not a Racket file and *program* is `#f`, `#f` is returned.

```
(extract-base-filename/kp s [program]) → (or/c path? #f)
  s : path-string?
  program : any/c = #f
```

Same as `extract-base-filename/c`, but for constant-pool files.

```
(extract-base-filename/o s [program]) → (or/c path? #f)
  s : path-string?
  program : any/c = #f
```

Same as `extract-base-filename/c`, but for compiled-object files.

```
(extract-base-filename/ext s [program]) → (or/c path? #f)
  s : path-string?
  program : any/c = #f
```

Same as `extract-base-filename/c`, but for extension files.

3.1 Signature

`(require dynext/file-sig)` package: `dynext-lib`

```
dynext:file^ : signature
```

Includes everything exported by the `dynext/file` module.

3.2 Unit

`(require dynext/file-unit)` package: `dynext-lib`

```
dynext:file@ : unit?
```

Imports nothing, exports `dynext:file^`.