

---

**cle**

**The angr Project**

**Jun 04, 2026**



## CONTENTS:

<b>1 CLE</b>	<b>3</b>
<b>2 API Documentation</b>	<b>7</b>
<b>3 Indices and tables</b>	<b>101</b>
<b>Python Module Index</b>	<b>103</b>
<b>Index</b>	<b>105</b>



This documentation has two sections. The first shows basic usage of CLE, and the second shows the API reference.



CLE loads binaries and their associated libraries, resolves imports and provides an abstraction of process memory the same way as if it was loader by the OS's loader.

## 1.1 Project Links

Project repository: <https://github.com/angr/cle>

Documentation: <https://api.angr.io/projects/cle/en/latest/>

## 1.2 Installation

```
pip install cle
```

## 1.3 Usage example

```
>>> import cle
>>> ld = cle.Loader("/bin/ls")
>>> hex(ld.main_object.entry)
'0x4048d0'
>>> ld.shared_objects
{'ld-linux-x86-64.so.2': <ELF Object ld-2.21.so, maps [0x50000000:0x522312f]>,
 'libacl.so.1': <ELF Object libacl.so.1.1.0, maps [0x20000000:0x220829f]>,
 'libattr.so.1': <ELF Object libattr.so.1.1.0, maps [0x40000000:0x4204177]>,
 'libc.so.6': <ELF Object libc-2.21.so, maps [0x30000000:0x33a1a0f]>,
 'libcap.so.2': <ELF Object libcap.so.2.24, maps [0x10000000:0x1203c37]>}}
>>> ld.addr_belongs_to_object(0x50000000)
<ELF Object ld-2.21.so, maps [0x50000000:0x522312f]>
>>> libc_main_reloc = ld.main_object.imports['__libc_start_main']
>>> hex(libc_main_reloc.addr)      # Address of GOT entry for libc_start_main
'0x61c1c0'
>>> import pyvex
>>> some_text_data = ld.memory.load(ld.main_object.entry, 0x100)
>>> irsb = pyvex.lift(some_text_data, ld.main_object.entry, ld.main_object.arch)
>>> irsb.pp()
IRSB {
  t0:Itty_I32 t1:Itty_I32 t2:Itty_I32 t3:Itty_I64 t4:Itty_I64 t5:Itty_I64 t6:Itty_I32 t7:Itty_
```

(continues on next page)

```

↪I64 t8:Ity_I32 t9:Ity_I64 t10:Ity_I64 t11:Ity_I64 t12:Ity_I64 t13:Ity_I64 t14:Ity_I64

15 | ----- IMark(0x4048d0, 2, 0) -----
16 | t5 = 32Uto64(0x00000000)
17 | PUT(rbp) = t5
18 | t7 = GET:I64(rbp)
19 | t6 = 64to32(t7)
20 | t2 = t6
21 | t9 = GET:I64(rbp)
22 | t8 = 64to32(t9)
23 | t1 = t8
24 | t0 = Xor32(t2,t1)
25 | PUT(cc_op) = 0x000000000000000013
26 | t10 = 32Uto64(t0)
27 | PUT(cc_dep1) = t10
28 | PUT(cc_dep2) = 0x0000000000000000
29 | t11 = 32Uto64(t0)
30 | PUT(rbp) = t11
31 | PUT(rip) = 0x0000000000004048d2
32 | ----- IMark(0x4048d2, 3, 0) -----
33 | t12 = GET:I64(rdx)
34 | PUT(r9) = t12
35 | PUT(rip) = 0x0000000000004048d5
36 | ----- IMark(0x4048d5, 1, 0) -----
37 | t4 = GET:I64(rsp)
38 | t3 = LDle:I64(t4)
39 | t13 = Add64(t4,0x0000000000000008)
40 | PUT(rsp) = t13
41 | PUT(rsi) = t3
42 | PUT(rip) = 0x0000000000004048d6
43 | t14 = GET:I64(rip)
NEXT: PUT(rip) = t14; Ijk_Boring
}

```

## 1.4 Valid options

For a full listing and description of the options that can be provided to the loader and the methods it provides, please examine the docstrings in `cle/loader.py`. If anything is unclear or poorly documented (there is much) please complain through whatever channel you feel appropriate.

## 1.5 Loading Backends

CLE's loader is implemented in the `Loader` class. There are several backends that can be used to load a single file:

- ELF, as its name says, loads ELF binaries. ELF files loaded this way are statically parsed using `PyElfTools`.
- PE is a backend to load Microsoft's Portable Executable format, effectively Windows binaries. It uses the (optional) `pefile` module.
- Mach-O is a backend to load, you guessed it, Mach-O binaries. Support is limited for this backend.
- Blob is a backend to load unknown data. It requires that you specify the architecture it would be run on, in the form of a class from `ArchInfo`.

Which backend you use can be specified as an argument to Loader. If left unspecified, the loader will pick a reasonable default.

## 1.6 Finding shared libraries

- If the `auto_load_libs` option is set to `False`, the Loader will not automatically load libraries requested by loaded objects. Otherwise...
- The loader determines which shared objects are needed when loading binaries, and searches for them in the following order:
  - in the current working directory
  - in folders specified in the `ld_path` option
  - in the same folder as the main binary
  - in the system (in the corresponding library path for the architecture of the binary, e.g., `/usr/arm-linux-gnueabi/lib` for ARM, note that you need to install cross libraries for this, e.g., `libc6-powerpc-cross` on Debian - needs emdebian repos)
  - in the system, but with mismatched version numbers from what is specified as a dependency, if the `ignore_import_version_numbers` option is `True`
- If no binary is found with the correct architecture, the loader raises an exception if `except_missing_libs` option is `True`. Otherwise it simply leaves the dependencies unresolved.



## API DOCUMENTATION

### 2.1 Loading Interface

**class** `cle.Loader`

Bases: `object`

The loader loads all the objects and exports an abstraction of the memory of the process. What you see here is an address space with loaded and rebased binaries.

```
__init__(main_binary: str | BinaryIO | Path | Backend, auto_load_libs: bool = False,
         concrete_target=None, force_load_libs: Iterable[str | BinaryIO | Path] = (), skip_libs:
         Iterable[str] = (), main_opts: dict[str, Any] | None = None, lib_opts: dict[str, dict[str, Any]] |
         None = None, ld_path: Iterable[str | Path] = (), use_system_libs: bool = True,
         ignore_import_version_numbers: bool = True, case_insensitive: bool = False, rebase_granularity:
         int = 1048576, except_missing_libs: bool = False, aslr: bool = False, perform_relocations: bool
         = True, load_debug_info: bool = False, page_size: int = 1, preload_libs: Iterable[str | BinaryIO |
         Path] = (), arch: Arch | str | None = None, force_tls: str | None = None)
```

#### Parameters

- **main\_binary** (`str | BinaryIO | Path | Backend`) – The path to the main binary you’re loading, or a file-like object with the binary in it.
- **auto\_load\_libs** (`bool`) – Whether to automatically load shared libraries that loaded objects depend on.
- **load\_debug\_info** (`bool`) – Whether to automatically parse DWARF data and search for debug symbol files.
- **concrete\_target** – Whether to instantiate a concrete target for a concrete execution of the process. if this is the case we will need to instantiate a `SimConcreteEngine` that wraps the `ConcreteTarget` provided by the user.
- **force\_load\_libs** (`Iterable[str | BinaryIO | Path]`) – A list of libraries to load regardless of if they’re required by a loaded object.
- **skip\_libs** (`Iterable[str]`) – A list of libraries to never load, even if they’re required by a loaded object.
- **main\_opts** (`dict[str, Any] | None`) – A dictionary of options to be used loading the main binary.
- **lib\_opts** (`dict[str, dict[str, Any]] | None`) – A dictionary mapping library names to the dictionaries of options to be used when loading them.
- **ld\_path** (`Iterable[str | Path]`) – A list of paths in which we can search for shared libraries.

- **use\_system\_libs** (*bool*) – Whether or not to search the system load path for requested libraries. Default True.
- **ignore\_import\_version\_numbers** (*bool*) – Whether libraries with different version numbers in the filename will be considered equivalent, for example `libc.so.6` and `libc.so.0`
- **case\_insensitive** (*bool*) – If this is set to True, filesystem loads will be done case-insensitively regardless of the case-sensitivity of the underlying filesystem.
- **rebase\_granularity** (*int*) – The alignment to use for rebasing shared objects
- **except\_missing\_libs** (*bool*) – Throw an exception when a shared library can't be found.
- **aslr** (*bool*) – Load libraries in symbolic address space. Do not use this option.
- **page\_size** (*int*) – The granularity with which data is mapped into memory. Set to `0x1000` if you are working in an environment where data will always be memory mapped in a page-granular way.
- **preload\_libs** (*Iterable[str | BinaryIO | Path]*) – Similar to `force_load_libs` but will provide for symbol resolution, with precedence over any dependencies.
- **perform\_relocations** (*bool*)
- **arch** (*Arch | str | None*)
- **force\_tls** (*str | None*)

#### Variables

- **memory** (`cle.memory.Clemory`) – The loaded, rebased, and relocated memory of the program.
- **main\_object** – The object representing the main binary (i.e., the executable).
- **shared\_objects** – A dictionary mapping loaded library names to the objects representing them.
- **all\_objects** – A list containing representations of all the different objects loaded.
- **requested\_names** – A set containing the names of all the different shared libraries that were marked as a dependency by somebody.
- **initial\_load\_objects** – A list of all the objects that were loaded as a result of the initial load request.

When reference is made to a dictionary of options, it requires a dictionary with zero or more of the following keys:

- `backend` : “elf”, “pe”, “mach-o”, “blob” : which loader backend to use
- `arch` : The `archinfo.Arch` object to use for the binary
- `base_addr` : The address to rebase the object at
- `entry_point` : The entry point to use for the object

More keys are defined on a per-backend basis.

property `main_object`: *Backend*

property `original_main_object`: *Backend*

property `memory`: *Clemory*

**property memory\_ro\_view:** `CMemoryReadOnlyView` | `None`

**property tls:** `ThreadManager`

`close()`

**property max\_addr:** `int`

The maximum address loaded as part of any loaded object (i.e., the whole address space).

**property min\_addr:** `int`

The minimum address loaded as part of any loaded object (i.e., the whole address space).

**property initializers:** `list[int]`

Return a list of all the initializers that should be run before execution reaches the entry point, in the order they should be run.

**property finalizers:** `list[int]`

Return a list of all the finalizers that should be run before the program exits. I'm not sure what order they should be run in.

**property linux\_loader\_object:** `Backend` | `None`

If the linux dynamic loader is present in memory, return it

**property elfcore\_object:** `ELFCore` | `None`

If a corefile was loaded, this returns the actual core object instead of the main binary

**property extern\_object:** `ExternObject`

Return the extern object used to provide addresses to unresolved symbols and Angr internals.

Accessing this property will load this object into memory if it was not previously present.

proposed model for how multiple extern objects should work:

- 1) extern objects are a linked list. the one in `loader._extern_object` is the head of the list
- 2) **each round of explicit loads generates a new extern object if it has unresolved dependencies. this object**  
has exactly the size necessary to hold all its exports.
- 3) **All requests for size are passed down the chain until they reach an object which has the space to service**  
it or an object which has not yet been mapped. If all objects have been mapped and are full, a new extern object is mapped with a fixed size.

**property kernel\_object:** `KernelObject`

Return the object used to provide addresses to syscalls.

Accessing this property will load this object into memory if it was not previously present.

**property all\_elf\_objects:** `list[MetaELF]`

Return a list of every object that was loaded from an ELF file.

**property all\_pe\_objects:** `list[PE]`

Return a list of every object that was loaded from an ELF file.

**property missing\_dependencies:** `set[str]`

Return a set of every name that was requested as a shared object dependency but could not be loaded

**property auto\_load\_libs:** `bool`

**describe\_addr**(*addr*: *int*) → *str*

Returns a textual description of what's in memory at the provided address

**Return type**

*str*

**Parameters**

**addr** (*int*)

**find\_object**(*spec*: *Backend* | *str*, *extra\_objects*: *Iterable*[*Backend*] = ()) → *Backend* | *None*

If the given library specification has been loaded, return its object, otherwise return None.

**Return type**

*Backend* | *None*

**Parameters**

- **spec** (*Backend* | *str*)
- **extra\_objects** (*Iterable*[*Backend*])

**find\_object\_containing**(*addr*: *int*, *membership\_check*: *bool* = *True*) → *Backend* | *None*

Return the object that contains the given address, or None if the address is unmapped. This lookup method does not include outer objects (*obj.is\_outer* is *True*).

**Parameters**

- **addr** (*int*) – The address that should be contained in the object.
- **membership\_check** (*bool*) – Whether a membership check should be performed or not (*True* by default). This option can be set to *False* if you are certain that the target object does not have “holes”.

**Return type**

*Backend* | *None*

**Returns**

The object or None.

**find\_segment\_containing**(*addr*: *int*, *skip\_pseudo\_objects*: *bool* = *True*) → *Segment* | *None*

Find the section object that the address belongs to.

**Parameters**

- **addr** (*int*) – The address to test
- **skip\_pseudo\_objects** (*bool*) – Skip objects that CLE adds during loading.

**Returns**

The section that the address belongs to, or None if the address does not belong to any section, or if section information is not available.

**Return type**

*Segment* | *None*

**find\_section\_containing**(*addr*: *int*, *skip\_pseudo\_objects*=*True*) → *Section* | *None*

Find the section object that the address belongs to.

**Parameters**

- **addr** (*int*) – The address to test.
- **skip\_pseudo\_objects** (*bool*) – Skip objects that CLE adds during loading.

**Returns**

The section that the address belongs to, or None if the address does not belong to any section, or if section information is not available.

**Return type**

*Section* | None

**find\_loadable\_containing**(*addr: int, skip\_pseudo\_objects=True*) → *Region* | None

Find the section or segment object the address belongs to. Sections will only be used if the corresponding object does not have segments.

**Parameters**

- **addr** (*int*) – The address to test
- **skip\_pseudo\_objects** – Skip objects that CLE adds during loading.

**Return type**

*Region* | None

**Returns**

The section or segment that the address belongs to, or None if the address does not belong to any section or segment.

**find\_section\_next\_to**(*addr: int, skip\_pseudo\_objects=True*) → *Section* | None

Find the next section after the given address.

**Parameters**

- **addr** (*int*) – The address to test.
- **skip\_pseudo\_objects** (*bool*) – Skip objects that CLE adds during loading.

**Returns**

The next section that goes after the given address, or None if there is no section after the address, or if section information is not available.

**Return type**

*Section* | None

**find\_symbol**(*thing, fuzzy=False*) → *Symbol* | None

Search for the symbol with the given name or address.

**Parameters**

- **thing** – Either the name or address of a symbol to look up
- **fuzzy** – Set to True to return the first symbol before or at the given address

**Return type**

*Symbol* | None

**Returns**

A `cle.backends.Symbol` object if found, None otherwise.

**property symbols:** `Iterator[Symbol]`

**find\_all\_symbols**(*name: str, exclude\_imports=True, exclude\_externs=False, exclude\_forwards=True*) → `Iterable[Symbol]`

Iterate over all symbols present in the set of loaded binaries that have the given name

**Parameters**

- **name** (*str*) – The name to search for

- **exclude\_imports** – Whether to exclude import symbols. Default True.
- **exclude\_externs** – Whether to exclude symbols in the extern object. Default False.
- **exclude\_forwards** – Whether to exclude forward symbols. Default True.

**Return type**`Iterable[Symbol]`**find\_plt\_stub\_name**(*addr: int*) → `str | None`Return the name of the PLT stub starting at *addr*.**Return type**`str | None`**Parameters****addr** (*int*)**find\_relevant\_relocations**(*name: str*) → `Iterator[Relocation]`

Iterate through all the relocations referring to the symbol with the given name

**Return type**`Iterator[Relocation]`**Parameters****name** (*str*)**perform\_irelative\_relocs**(*resolver\_func*)Use this method to satisfy `IRelative` relocations in the binary that require execution of loaded code.Note that this does NOT handle `IFunc` symbols, which must be handled separately. (this could be changed, but at the moment it's desirable to support lazy `IFunc` resolution, since emulation is usually slow)**Parameters****resolver\_func** – A callback function that takes an address, runs the code at that address, and returns the return value from the emulated function.**dynamic\_load**(*spec*)Load a file into the address space. Note that the semantics of `auto_load_libs` and `except_missing_libs` apply at all times.**Parameters****spec** – The path to the file to load. May be an absolute path, a relative path, or a name to search in the load path.**Returns**A list of all the objects successfully loaded, which may be empty if this object was previously loaded. If the object specified in `spec` failed to load for any reason, including the file not being found, return `None`.**get\_loader\_symbolic\_constraints**()

Do not use this method.

**gen\_ro\_memview**() → `None`Generate a read-only view of the memory, and update `self._memory_ro_view` for faster data loading. Please call this method again for updating the read-only view, or `discard_ro_memview()` to discard any previously generated read-only views.**Return type**`None`

`discard_ro_memview()` → `None`

Discard any previously generated read-only views of the memory.

**Return type**

`None`

`fast_memory_load_pointer(addr: int, size: int | None = None)` → `int | None`

Perform a fast memory loading of a pointer.

**Parameters**

- **addr** (`int`) – Address to read from.
- **size** (`int | None`) – Size of the pointer. Default to machine-word size.

**Return type**

`int | None`

**Returns**

A pointer or `None` if the address does not exist.

## 2.2 Backend Interface

`class cle.backends.backend.FunctionHintSource`

Bases: `object`

Enums that describe the source of function hints.

`EH_FRAME = 0`

`EXTERNAL_EH_FRAME = 1`

`EXPORT_TABLE = 2`

`class cle.backends.backend.FunctionHint`

Bases: `object`

Describes a function hint.

**Variables**

- **addr** (`int`) – Address of the function.
- **size** (`int`) – Size of the function.
- **source** (`int`) – Source of this hint.
- **name** (`str | None`) – Optional symbol name, if known.

`__init__(addr, size, source, name=None)`

`addr`

`size`

`source`

`name`

**class** cle.backends.backend.ExceptionHandlingBases: `object`

Describes an exception handling.

Exception handlers are usually language-specific. In C++, it is usually implemented as `try {} catch {}` blocks.**Variables**

- **start\_addr** (*int*) – The beginning of the try block.
- **size** (*int*) – Size of the try block.
- **handler\_addr** (*Optional[int]*) – Address of the exception handler code.
- **type** – Type of the exception handler. Optional.
- **func\_addr** (*Optional[int]*) – Address of the function. Optional.

`__init__(start_addr, size, handler_addr=None, type_=None, func_addr=None)`**start\_addr****size****handler\_addr****type****func\_addr****class** cle.backends.backend.BackendBases: `object`

Main base class for CLE binary objects.

An alternate interface to this constructor exists as the static method `cle.loader.Loader.load_object()`**Variables**

- **binary** – The path to the file this object is loaded from
- **binary\_basename** – The basename of the filepath, or a short representation of the stream it was loaded from
- **is\_main\_bin** – Whether this binary is loaded as the main executable
- **segments** – A listing of all the loaded segments in this file
- **sections** – A listing of all the demarked sections in the file
- **sections\_map** – A dict mapping from section name to section
- **imports** (*dict[str, Relocation]*) – A mapping from symbol name to import relocation
- **resolved\_imports** – A list of all the import symbols that are successfully resolved
- **relocs** (*list[Relocation]*) – A list of all the relocations in this binary
- **irelatives** – A list of tuples representing all the irelative relocations that need to be performed. The first item in the tuple is the address of the resolver function, and the second item is the address of where to write the result. The destination address is an RVA.
- **jmprel** – A mapping from symbol name to the address of its jump slot relocation, i.e. its GOT entry.
- **arch** (*archinfo.arch.Arch*) – The architecture of this binary

- **os** (*str*) – The operating system this binary is meant to run under
- **mapped\_base** (*int*) – The base address of this object in virtual memory
- **deps** – A list of names of shared libraries this binary depends on
- **linking** – ‘dynamic’ or ‘static’
- **linked\_base** – The base address this object requests to be loaded at
- **pic** (*bool*) – Whether this object is position-independent
- **execstack** (*bool*) – Whether this executable has an executable stack
- **provides** (*str*) – The name of the shared library dependancy that this object resolves
- **symbols** (*list*) – A list of symbols provided by this object, sorted by address
- **has\_memory** – Whether this backend is backed by a Clemory or not. As it stands now, a backend should still define *min\_addr* and *max\_addr* even if *has\_memory* is False.

**is\_default** = False

**is\_outer** = False

**\_\_init\_\_** (*binary, binary\_stream, loader=None, is\_main\_bin=False, entry\_point=None, arch=None, base\_addr=None, force\_rebase=False, has\_memory=True, \*\*kwargs*)

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**load\_args**: `dict[str, Any]`

**unpacked\_name**: `str | None`

**imports**: `dict[str, Relocation]`

**relocs**: `list[Relocation]`

**child\_objects**: `list[Backend]`

**exception\_handlings**: `list[ExceptionHandler]`

**function\_hints**: `list[FunctionHint]`

**meta\_regions**: `list[MemRegion]`

**memory**: `Clemory`

**cached\_content**: `bytes | None`

**property md5**: `bytes | None`

MD5 digest of the binary, computed lazily on first access.

**property sha256**: `bytes | None`

SHA-256 digest of the binary, computed lazily on first access.

**property arch**: `Arch`

**property loader:** *Loader*

**close()** → None

**Return type**  
None

**set\_arch(*arch*)**

**set\_load\_args(\*\**kwargs*)** → None

**Return type**  
None

**property image\_base\_delta**

**property entry**

**property segments:** *Regions[Segment]*

**property sections:** *Regions[Section]*

**property symbols\_by\_addr**

**rebase(*new\_base*)**

Rebase backend's regions to the new base where they were mapped by the loader

**relocate()**

Apply all resolved relocations to memory.

The meaning of “resolved relocations” is somewhat subtle - there is a linking step which attempts to resolve each relocation, currently only present in the main internal loading function since the calculation of which objects should be available

**contains\_addr(*addr*)**

Is *addr* in one of the binary's segments/sections we have loaded? (i.e. is it mapped into memory ?)

**find\_loadable\_containing(*addr*)**

**find\_segment\_containing(*addr: int*)** → *Segment* | None

Returns the segment that contains *addr*, or None.

**Return type**  
*Segment* | None

**Parameters**  
**addr** (*int*)

**find\_section\_containing(*addr: int*)** → *Section* | None

Returns the section that contains *addr* or None.

**Return type**  
*Section* | None

**Parameters**  
**addr** (*int*)

**addr\_to\_offset(*addr: int*)** → *int* | None

**Return type**  
*int* | None

**Parameters****addr** (*int*)**offset\_to\_addr**(*offset: int*) → *int* | *None***Return type***int* | *None***Parameters****offset** (*int*)**property min\_addr:** *int*

This returns the lowest virtual address contained in any loaded segment of the binary.

**property max\_addr:** *int*

This returns the highest virtual address contained in any loaded segment of the binary.

**property initializers:** *list[int]*

Stub function. Should be overridden by backends that can provide initializer functions that ought to be run before execution reaches the entry point. Addresses should be rebased.

**property finalizers:** *list[int]*

Stub function. Like initializers, but with finalizers.

**property threads:** *list*

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This property should contain a list of names for these threads, which should be unique.

**thread\_registers**(*thread=None*) → *dict[str, Any]*If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This method should return the register file for a given thread (as named in `Backend.threads`) as a dict mapping register names (as seen in `archinfo`) to numbers. If the thread is not specified, it should return the context for a “default” thread. If there are no threads, it should return an empty dict.**Return type***dict[str, Any]***initial\_register\_values**()

Deprecated

**get\_symbol**(*name: str*) → *Symbol* | *None*Stub function. Implement to find the symbol with name *name*.**Return type***Symbol* | *None***Parameters****name** (*str*)**property available\_arches:** *list[str]*

Return the list of architecture names available in this binary.

For most backends this is a single-element list derived from the loaded architecture. Container backends (e.g. `Universal2`) override this to report every architecture present in the file.**static extract\_soname**(*path*) → *str* | *None*Extracts the shared object identifier from the path, or returns `None` if it cannot.**Return type***str* | *None*

**classmethod** `is_compatible(stream) → bool`

Determine quickly whether this backend can load an object from this stream

**Return type**

`bool`

**classmethod** `check_compatibility(spec, obj) → bool`

Performs a minimal static load of `spec` and returns whether it's compatible with `other_obj`

**Return type**

`bool`

**classmethod** `check_magic_compatibility(stream: BinaryIO) → bool`

Check if a stream of bytes contains the same magic number as the main object

**Return type**

`bool`

**Parameters**

**stream** (*BinaryIO*)

`cle.backends.backend.register_backend(name, cls)`

**class** `cle.backends.symbol.SymbolType`

Bases: `Enum`

ABI-agnostic symbol types

`TYPE_OTHER = 0`

`TYPE_NONE = 1`

`TYPE_FUNCTION = 2`

`TYPE_OBJECT = 3`

`TYPE_SECTION = 4`

`TYPE_TLS_OBJECT = 5`

`TYPE_FUNCTION_OR_OBJECT = 6`

**class** `cle.backends.symbol.SymbolSubType`

Bases: `Enum`

Abstract base class for ABI-specific symbol types

**to\_base\_type()** `→ SymbolType`

A subclass' ABI-specific mapping to `:SymbolType`:

**Return type**

*SymbolType*

**class** `cle.backends.symbol.Symbol`

Bases: `object`

Representation of a symbol from a binary file. Smart enough to rebase itself.

There should never be more than one `Symbol` instance representing a single symbol. To make sure of this, only use the `cle.backends.Backend.get_symbol()` to create new symbols.

**Variables**

- **owner** (*cle.backends.Backend*) – The object that contains this symbol
- **name** (*str*) – The name of this symbol
- **addr** (*int*) – The un-based address of this symbol, an RVA
- **size** (*int*) – The size of this symbol
- **\_type** (*SymbolType*) – The ABI-agnostic type of this symbol
- **resolved** (*bool*) – Whether this import symbol has been resolved to a real symbol
- **resolvedby** (*None* or *cle.backends.Symbol*) – The real symbol this import symbol has been resolve to
- **resolvewith** (*str*) – The name of the library we must use to resolve this symbol, or None if none is required.

**\_\_init\_\_** (*owner: Backend, name: str, relative\_addr: int, size: int, sym\_type: SymbolType*)

Not documenting this since if you try calling it, you're wrong.

#### Parameters

- **owner** (*Backend*)
- **name** (*str*)
- **relative\_addr** (*int*)
- **size** (*int*)
- **sym\_type** (*SymbolType*)

**owner:** *Backend*

**resolve**(*obj*)

**property type:** *SymbolType*

The ABI-agnostic SymbolType. Must be overridden by derived types.

**property subtype:** *SymbolSubType*

A subclass' ABI-specific types

**property rebased\_addr**

The address of this symbol in the global memory space

**property linked\_addr**

**property is\_function**

Whether this symbol is a function

**is\_static** = False

**is\_common** = False

**is\_import** = False

**is\_export** = False

**is\_local** = False

**is\_weak** = False

**is\_extern** = False

**is\_forward** = False

**resolve\_forwarder**()

If this symbol is a forwarding export, return the symbol the forwarding refers to, or None if it cannot be found

**property owner\_obj**

**class** cle.backends.regions.**Regions**

Bases: [Generic](#)

A container class acting as a list of regions (sections or segments). Additionally, it keeps an sorted list of all regions that are mapped into memory to allow fast lookups.

We assume none of the regions overlap with others.

**\_\_init\_\_**(*lst: list[R] | None = None*)

**Parameters**

**lst** (*list[R] | None*)

**property raw\_list**: [list\[R\]](#)

Get the internal list. Any change to it is not tracked, and therefore `_sorted_list` will not be updated. Therefore you probably does not want to modify the list.

**Returns**

The internal list container.

**Return type**

[list](#)

**property max\_addr**: [int | None](#)

Get the highest address of all regions.

**Returns**

The highest address of all regions, or None if there is no region available.

**Return type**

[int](#) or [None](#)

**append**(*region: R*)

Append a new Region instance into the list.

**Parameters**

**region** ([TypeVar](#)(R, bound= [Region](#))) – The region to append.

**remove**(*region: R*) → [None](#)

Remove an existing Region instance from the list.

**Parameters**

**region** ([TypeVar](#)(R, bound= [Region](#))) – The region to remove.

**Return type**

[None](#)

**find\_region\_containing**(*addr: int*) → [R | None](#)

Find the region that contains a specific address. Returns None if none of the regions covers the address.

**Parameters**

**addr** ([int](#)) – The address.

**Return type**`Optional[TypeVar(R, bound= Region)]`**Returns**

The region that covers the specific address, or None if no such region is found.

**find\_region\_next\_to**(*addr: int*) → R | None

Find the next region after the given address.

**Parameters**

**addr** (int) – The address to test.

**Return type**`Optional[TypeVar(R, bound= Region)]`**Returns**

The next region that goes after the given address, or None if there is no section after the address,

**class** cle.backends.region.Region

Bases: `object`

A region of memory that is mapped in the object's file.

**Variables**

- **offset** – The offset into the file the region starts.
- **vaddr** – The virtual address.
- **filesize** – The size of the region in the file.
- **memsize** – The size of the region when loaded into memory.

The prefix *v-* on a variable or parameter name indicates that it refers to the virtual, loaded memory space, while a corresponding variable without the *v-* refers to the flat zero-based memory of the file.

When used next to each other, *addr* and *offset* refer to virtual memory address and file offset, respectively.

`__init__(offset, vaddr, filesize, memsize)`**vaddr**: int**memsize**: int**filesize**: int**contains\_addr**(*addr*)

Does this region contain this virtual address?

**contains\_offset**(*offset*)

Does this region contain this offset into the file?

**addr\_to\_offset**(*addr*)

Convert a virtual memory address into a file offset

**offset\_to\_addr**(*offset*)

Convert a file offset into a virtual memory address

**property** **max\_addr**

The maximum virtual address of this region

**property min\_addr**

The minimum virtual address of this region

**property max\_offset**

The maximum file offset of this region

**min\_offset()**

The minimum file offset of this region

**property is\_readable: bool****property is\_writable: bool****property is\_executable: bool****class cle.backends.region.Segment**

Bases: *Region*

**class cle.backends.region.EmptySegment**

Bases: *Segment*

A segment with no static content, and permissions

**\_\_init\_\_**(*vaddr*, *memszie*, *is\_readable=True*, *is\_writable=True*, *is\_executable=False*)

**property is\_executable****property is\_writable****property is\_readable****property only\_contains\_uninitialized\_data**

Whether this section is initialized to zero after the executable is loaded.

**class cle.backends.region.Section**

Bases: *Region*

Simple representation of a loaded section.

**Variables**

**name** (*str*) – The name of the section

**\_\_init\_\_**(*name*, *offset*, *vaddr*, *size*)

**Parameters**

- **name** (*str*) – The name of the section
- **offset** (*int*) – The offset into the binary file this section begins
- **vaddr** (*int*) – The address in virtual memory this section begins
- **size** (*int*) – How large this section is

**property is\_readable**

Whether this section has read permissions

**property is\_writable**

Whether this section has write permissions

**property is\_executable**

Whether this section has execute permissions

**property only\_contains\_uninitialized\_data**

Whether this section is initialized to zero after the executable is loaded.

**class cle.backends.named\_region.NamedRegion**

Bases: *Backend*

A NamedRegion represents a region of memory that has a name, a location, but no static content.

This region also has permissions; with no memory, these obviously don't do anything on their own, but they help inform any other code that relies on CLE (e.g., angr)

This can be used as a placeholder for memory that should exist in CLE's view, but for which it does not need data, like RAM, MMIO, etc

**is\_default = False**

**\_\_init\_\_**(*name, start, end, is\_readable=True, is\_writable=True, is\_executable=False, \*\*kwargs*)

Create a NamedRegion.

**Parameters**

- **name** – The name of the region
- **start** – The start address of the region
- **end** – The end address (exclusive) of the region
- **is\_readable** – Whether the region is readable
- **is\_writable** – Whether the region is writable
- **is\_executable** – Whether the region is executable
- **kwargs**

**has\_memory = False**

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**property min\_addr**

This returns the lowest virtual address contained in any loaded segment of the binary.

**property max\_addr**

This returns the highest virtual address contained in any loaded segment of the binary.

**function\_name**(*addr*)

NamedRegions don't support function names.

**contains\_addr**(*addr*)

Is *addr* in one of the binary's segments/sections we have loaded? (i.e. is it mapped into memory ?)

**classmethod check\_compatibility**(*spec, obj*)

Performs a minimal static load of *spec* and returns whether it's compatible with other\_obj

**class cle.backends.externs.ExternSegment**

Bases: *Segment*

**\_\_init\_\_**(*map\_size*)

**addr\_to\_offset**(*addr*)

Convert a virtual memory address into a file offset

**offset\_to\_addr**(*offset*)

Convert a file offset into a virtual memory address

**contains\_offset**(*offset*)

Does this region contain this offset into the file?

**is\_readable** = True

**is\_writable** = True

**is\_executable** = True

**class** cle.backends.externs.TOCRelocation

Bases: *Relocation*

**property value**

**class** cle.backends.externs.ExternObject

Bases: *Backend*

**\_\_init\_\_**(*loader, map\_size=0, tls\_size=0*)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**rebase**(*new\_base*)

Rebase backend's regions to the new base where they were mapped by the loader

**make\_extern**(*name, size=0, alignment=None, thumb=False, sym\_type=SymbolType.TYPE\_FUNCTION, point\_to=None, libname=None*) → *Symbol*

**Return type**

*Symbol*

**get\_pseudo\_addr**(*name*) → int

**Return type**

int

**allocate**(*size=1, alignment=8, thumb=False, tls=False*) → int

**Return type**

int

**property max\_addr**

This returns the highest virtual address contained in any loaded segment of the binary.

**make\_import**(*name, sym\_type*)

**class** cle.backends.externs.KernelObject

Bases: *Backend*

`__init__(loader, map_size=32768)`

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**add\_name**(*name, addr*)

**property max\_addr**

This returns the highest virtual address contained in any loaded segment of the binary.

**class** cle.backends.externs.PointToPrecise

Bases: *PointTo*

**pointto\_precise** = None

**relocations**()

Maybe implement me: If you like, return a list of relocation objects to apply. To create new import symbols, use `self.owner.make_extern_import`.

**class** cle.backends.externs.simdata.SimData

Bases: *Symbol*

A SimData class is used to provide data when there is an unresolved data import symbol.

To use it, subclass this class and implement the below attributes and methods.

#### Variables

- **name** – The name of the symbol to provide
- **libname** – The name of the library from which the symbol originally comes (currently unused).
- **type** – The type of the symbol, usually `SymbolType.TYPE_OBJECT`.

Use the below *register* method to register SimData subclasses with CLE.

NOTE: `SimData.type` hides the `Symbol.type` instance property

**name:** `str` = NotImplemented

**type:** `SymbolType` = NotImplemented

**libname:** `str` = NotImplemented

**classmethod static\_size**(*owner*) → int

Implement me: return the size of the symbol in bytes before it gets constructed

#### Parameters

**owner** – The ExternObject owning the symbol-to-be. Useful to get at `owner.arch`.

#### Return type

int

**value**() → bytes

Implement me: the initial value of the bytes in memory for the symbol. Should return a bytestring of the same length as `static_size` returned. (owner is `self.owner` now)

**Return type**

bytes

**relocations()** → list[*Relocation*]

Maybe implement me: If you like, return a list of relocation objects to apply. To create new import symbols, use `self.owner.make_extern_import`.

**Return type**list[*Relocation*]

`cle.backends.externs.simdata.lookup(name: str, libname) → type[SimData] | None`

**Return type**type[*SimData*] | None**Parameters****name** (*str*)

`cle.backends.externs.simdata.register(simdata_cls: type[SimData])`

Register the given *SimData* class with CLE so it may be used during loading

**Parameters****simdata\_cls** (type[*SimData*])

**class** `cle.backends.externs.simdata.simdata.SimData`

Bases: *Symbol*

A *SimData* class is used to provide data when there is an unresolved data import symbol.

To use it, subclass this class and implement the below attributes and methods.

**Variables**

- **name** – The name of the symbol to provide
- **libname** – The name of the library from which the symbol originally comes (currently unused).
- **type** – The type of the symbol, usually `SymbolType.TYPE_OBJECT`.

Use the below *register* method to register *SimData* subclasses with CLE.

NOTE: `SimData.type` hides the `Symbol.type` instance property

**name:** `str = NotImplemented`

**type:** `SymbolType = NotImplemented`

**libname:** `str = NotImplemented`

**classmethod static\_size(owner) → int**

Implement me: return the size of the symbol in bytes before it gets constructed

**Parameters****owner** – The `ExternObject` owning the symbol-to-be. Useful to get at `owner.arch`.**Return type**

int

**value()** → bytes

Implement me: the initial value of the bytes in memory for the symbol. Should return a bytestring of the same length as `static_size` returned. (`owner` is `self.owner` now)

**Return type**

bytes

**relocations()** → list[*Relocation*]

Maybe implement me: If you like, return a list of relocation objects to apply. To create new import symbols, use `self.owner.make_extern_import`.

**Return type**list[*Relocation*]**cle.backends.externs.simdata.simdata.register**(*simdata\_cls*: type[*SimData*])

Register the given *SimData* class with CLE so it may be used during loading

**Parameters****simdata\_cls** (type[*SimData*])**cle.backends.externs.simdata.simdata.lookup**(*name*: str, *libname*) → type[*SimData*] | None**Return type**type[*SimData*] | None**Parameters****name** (str)**class cle.backends.externs.simdata.common.StaticData**Bases: *SimData*

A simple *SimData* utility class to use when you have a *SimData* which should provide just a static set of bytes. To use, implement the following:

**Variables**

- **name** – The name of the symbol to provide.
- **libname** – The name of the library from which the symbol originally comes (currently unused).
- **data** – The bytes to provide

**type:** *SymbolType* = 3**data:** bytes = **NotImplemented****classmethod static\_size**(*owner*)

Implement me: return the size of the symbol in bytes before it gets constructed

**Parameters****owner** – The *ExternObject* owning the symbol-to-be. Useful to get at `owner.arch`.**value()**

Implement me: the initial value of the bytes in memory for the symbol. Should return a bytestring of the same length as `static_size` returned. (`owner` is `self.owner` now)

**class cle.backends.externs.simdata.common.StaticWord**Bases: *SimData*

A simple *SimData* utility class to use when you have a *SimData* which should provide just a static integer. To use, implement the following:

**Variables**

- **name** – The name of the symbol to provide.

- **libname** – The name of the library from which the symbol originally comes (currently unused).
- **word** – The value to provide
- **wordsize** – (optional) The size of the value in bytes, default the CPU wordsize

**type:** `SymbolType = 3`

**word:** `int = NotImplemented`

**wordsize:** `int = None`

**classmethod static\_size**(*owner*)

Implement me: return the size of the symbol in bytes before it gets constructed

#### Parameters

**owner** – The ExternObject owning the symbol-to-be. Useful to get at `owner.arch`.

**value()**

Implement me: the initial value of the bytes in memory for the symbol. Should return a bytestring of the same length as `static_size` returned. (owner is `self.owner` now)

**class** `cle.backends.externs.simdata.common.PointTo`

Bases: `SimData`

A simple `SimData` utility class to use when you have a `SimData` which should provide just a pointer to some other symbol. To use, implement the following:

#### Variables

- **name** – The name of the symbol to provide.
- **libname** – The name of the library from which the symbol originally comes (currently unused).
- **pointto\_name** – The name of the symbol to point to
- **pointto\_type** – The type of the symbol to point to (usually `SymbolType.TYPE_FUNCTION` or `SymbolType.TYPE_OBJECT`)
- **addend** – (optional) an integer to be added to the symbol's address before storage

**pointto\_name:** `str = NotImplemented`

**pointto\_type:** `SymbolType = NotImplemented`

**type:** `SymbolType = 3`

**addend:** `int = 0`

**classmethod static\_size**(*owner*)

Implement me: return the size of the symbol in bytes before it gets constructed

#### Parameters

**owner** – The ExternObject owning the symbol-to-be. Useful to get at `owner.arch`.

**value()**

Implement me: the initial value of the bytes in memory for the symbol. Should return a bytestring of the same length as `static_size` returned. (owner is `self.owner` now)

**relocations()**

Maybe implement me: If you like, return a list of relocation objects to apply. To create new import symbols, use `self.owner.make_extern_import`.

**class cle.backends.externs.simdata.common.SimDataSimpleRelocation**

Bases: *Relocation*

A relocation used to implement PointTo. Pretty simple.

**\_\_init\_\_**(*owner, symbol, addr, addend, preresolved=False*)

**resolve\_symbol**(*solist, \*\*kwargs*)

**property value**

## 2.3 Backends

### 2.3.1 ELF Backend

**class cle.backends.ELF**

Bases: *MetaELF*

The main loader class for statically loading ELF executables. Uses the pyreadelf library where useful.

Useful backend options:

- **debug\_symbols**: Provides the path to a separate file which contains the binary's debug symbols
- **discard\_section\_headers**: Do not parse section headers. Use this if they are corrupted or malicious.
- **discard\_program\_headers**: **Do not parse program headers. Use this if the binary is for a platform whose ELF loader only looks at section headers, but whose toolchain generates program headers anyway.**

**is\_default = True**

**\_\_init\_\_**(\*args, *addend=None, debug\_symbols=None, discard\_section\_headers=False, discard\_program\_headers=False, \*\*kwargs*)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call `close`.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**addr\_to\_line**: `SortedDict[int, set[tuple[int, int]]]`

**variables**: `list[Variable] | None`

**compilation\_units**: `list[CompilationUnit] | None`

**functions\_debug\_info**: `dict[int, Subprogram]`

**extern\_size\_hints**: `dict[str, int]`

**close()**

**classmethod** `check_compatibility(spec, obj)`

Performs a minimal static load of `spec` and returns whether it's compatible with `other_obj`

**classmethod** `check_magic_compatibility(stream)`

Check if a stream of bytes contains the same magic number as the main object

**static** `is_compatible(stream)`

Determine quickly whether this backend can load an object from this stream

**static** `extract_arch(reader)`

**property initializers**

Stub function. Should be overridden by backends that can provide initializer functions that ought to be run before execution reaches the entry point. Addresses should be rebased.

**property finalizers**

Stub function. Like initializers, but with finalizers.

**property symbols\_by\_name**

**get\_symbol(symid, symbol\_table=None)**

Gets a Symbol object for the specified symbol.

**Parameters**

**symid** – Either an index into `.dynsym` or the name of a symbol.

**rebase(new\_base)**

Rebase backend's regions to the new base where they were mapped by the loader

**class** `cle.backends.elf.ELFCore`

Bases: `ELF`

Loader class for ELF core files.

One key pain point when analyzing a core dump generated on a remote machine is that the paths to binaries are absolute (and may not exist or be the same on your local machine).

Therefore, you can use the options `remote_file_mapping` to specify a dict mapping (easy if there are a small number of mappings) or `remote_file_mapper` to specify a function that accepts a remote file name and returns the local file name (useful if there are many mappings).

If you specify both `remote_file_mapping` and `remote_file_mapper`, `remote_file_mapping` is applied first, then the result is passed to `remote_file_mapper`.

**Parameters**

- **executable** – Optional path to the main binary of the core dump. If not supplied, `ELFCore` will attempt to figure it out automatically from the core dump.
- **remote\_file\_mapping** – Optional dict that maps specific file names in the core dump to other file names.
- **remote\_file\_mapper** – Optional function that is used to map every file name in the core dump to whatever is returned from this function.

**is\_default = True**

**\_\_init\_\_**(\*args, executable=None, remote\_file\_mapping=None, remote\_file\_mapper=None, \*\*kwargs)

**Parameters**

- **binary** – The path to the binary to load

- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**property threads**

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This property should contain a list of names for these threads, which should be unique.

**thread\_registers**(*thread=None*)

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This method should return the register file for a given thread (as named in `Backend.threads`) as a dict mapping register names (as seen in `archinfo`) to numbers. If the thread is not specified, it should return the context for a “default” thread. If there are no threads, it should return an empty dict.

**class** `cle.backends.elf.MetaELF`

Bases: *Backend*

A base class that implements functions used by all backends that can load an ELF.

**\_\_init\_\_**(\*args, \*\*kwargs)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**supported\_filetypes** = ['elf']

**property plt**

Maps names to addresses.

**property reverse\_plt**

Maps addresses to names.

**property is\_ppc64\_abiv1**

Returns whether the arch is PowerPC64 ABIv1.

**Returns**

True if PowerPC64 ABIv1, False otherwise.

**property is\_ppc64\_abiv2**

Returns whether the arch is PowerPC64 ABIv2.

**Returns**

True if PowerPC64 ABIv2, False otherwise.

**property ppc64\_initial\_rtoc**

Get initial rtoc value for PowerPC64 architecture.

**static extract\_soname**(*path*)

Extracts the shared object identifier from the path, or returns None if it cannot.

```
class cle.backends.elf.metaelf.Relro
```

```
    Bases: Enum
```

```
    NONE = 0
```

```
    PARTIAL = 1
```

```
    FULL = 2
```

```
class cle.backends.elf.symbol.ELFSymbol
```

```
    Bases: Symbol
```

```
    Represents a symbol for the ELF format.
```

#### Variables

- **binding** (*str*) – The binding of this symbol as an ELF enum string
- **section** – The section associated with this symbol, or None
- **\_subtype** – The ELFSymbolType of this symbol

```
__init__(owner, symb)
```

```
    Not documenting this since if you try calling it, you're wrong.
```

```
property subtype: ELFSymbolType
```

```
    A subclass' ABI-specific types
```

```
class cle.backends.elf.symbol_type.ELFSymbolType
```

```
    Bases: SymbolSubType
```

```
    ELF-specific symbol types
```

```
    STT_NOTYPE = (0, None)
```

```
    STT_OBJECT = (1, None)
```

```
    STT_FUNC = (2, None)
```

```
    STT_SECTION = (3, None)
```

```
    STT_FILE = (4, None)
```

```
    STT_COMMON = (5, None)
```

```
    STT_TLS = (6, None)
```

```
    STT_LOOS = (10, None)
```

```
    STT_HIOS = (12, None)
```

```
    STT_LOPROC = (13, None)
```

```
    STT_HIPROC = (15, None)
```

```
    STT_GNU_IFUNC = (10, 'gnu')
```

```
__init__(*args)
```

```
property elf_value
```

```
property os_proc
```

**property is\_custom\_os\_proc**

**to\_base\_type()**

A subclass' ABI-specific mapping to :SymbolType:

**\_\_new\_\_**(value)

This is just a nice way to allow for just specifying the *int* for default types: *ELFSymbolType(10)* rather than *ELFSymbolType((10,None))*.

Idea courtesy: <https://stackoverflow.com/q/24105268/1137728>.

We don't need to implement the *str* parsing like the SO link above since *Enum* already has built-in item access: *ELFSymbolType['STT\_FUNC']*.

**class cle.backends.elf.regions.ELFSegment**

Bases: *Segment*

Represents a segment for the ELF format.

**\_\_init\_\_**(readelf\_seg, relro=False)

**property is\_readable**

**property is\_writable**

**property is\_executable**

**property is\_relro**

**class cle.backends.elf.regions.ELFSection**

Bases: *Section*

**SHF\_WRITE** = 1

**SHF\_ALLOC** = 2

**SHF\_EXECINSTR** = 4

**SHF\_STRINGS** = 32

**SHT\_NULL** = 'SHT\_NULL'

**\_\_init\_\_**(readelf\_sec, remap\_offset=0)

#### Parameters

- **name** (*str*) – The name of the section
- **offset** (*int*) – The offset into the binary file this section begins
- **vaddr** (*int*) – The address in virtual memory this section begins
- **size** (*int*) – How large this section is

**property is\_readable**

Whether this section has read permissions

**property is\_active**

**property is\_writable**

Whether this section has write permissions

**property occupies\_memory**

**property is\_executable**

Whether this section has execute permissions

**property is\_strings**

**property only\_contains\_uninitialized\_data**

Whether this section is initialized to zero after the executable is loaded.

**class cle.backends.elf.variable.Variable**

Bases: `object`

Variable for DWARF from a `DW_TAG_variable` or `DW_TAG_formal_parameter`

**Variables**

- **name** (`str`) – The name of the variable
- **relative\_addr** (`int` | `None`) – The relative addr (base addr depends on the type)
- **lexical\_block** (`LexicalBlock` | `None`) – For a local variable, the lexical block where the variable is declared

`__init__`(*elf\_object*: `ELF`)

**Parameters**

**elf\_object** (`ELF`)

**static from\_die**(*die*: `DIE`, *expr\_parser*, *elf\_object*: `ELF`, *lexical\_block*: `LexicalBlock` | `None` = `None`, *namespace*: `list[str]` | `None` = `None`)

**Parameters**

- **die** (`DIE`)
- **elf\_object** (`ELF`)
- **lexical\_block** (`LexicalBlock` | `None`)
- **namespace** (`list[str]` | `None`)

**rebased\_addr\_from\_cfa**(*cfa*: `int`) → `int`

The address of this variable in the global memory.

**Parameters**

**cfa** (`int`) – The canonical frame address as described by the DWARF standard.

**Return type**

`int`

**property rebased\_addr**

**property addr**

Please use ‘relative\_addr’ or ‘rebased\_addr’ instead.

**property type**: `VariableType`

**property sort**: `str`

**class** cle.backends.elf.variable.**MemoryVariable**

Bases: *Variable*

This includes all variables that are not on the stack and not in a register. So all global variables, and also local static variables in C!

`__init__(elf_object: ELF, relative_addr)`

**Parameters**

`elf_object` (ELF)

**property** rebased\_addr

**property** sort: `str`

**class** cle.backends.elf.variable.**StackVariable**

Bases: *Variable*

Stack Variable from DWARF.

`__init__(elf_object: ELF, relative_addr)`

**Parameters**

`elf_object` (ELF)

`rebased_addr_from_cfa`(*cfa*: int)

The address of this variable in the global memory.

**Parameters**

`cfa` (int) – The canonical frame address as described by the DWARF standard.

**property** sort: `str`

**class** cle.backends.elf.variable.**RegisterVariable**

Bases: *Variable*

Register Variable from DWARF.

`__init__(elf_object: ELF, register_addr)`

**Parameters**

`elf_object` (ELF)

**property** sort: `str`

cle.backends.elf.variable\_type.**resolve\_reference\_addr**(*die*: DIE, *attr\_name*: str) → int

Resolves a reference attribute to the underlying DIE location :type die: DIE :param die: The DIE containing the reference attribute :type attr\_name: str :param attr\_name: The name of the attribute as a string

**Return type**

int

**Returns**

The address of the DIE referred to by the reference

**Parameters**

- `die` (DIE)
- `attr_name` (str)

---

**class** `cle.backends.elf.variable_type.VariableType`

Bases: `object`

Entry class for `DW_TAG_XXX_type`

**Parameters**

- **name** (`str`) – name of the type
- **byte\_size** (`int`) – amount of bytes the type take in memory
- **elf\_object** – elf object to reference to (useful for pointer,...)

**Variables**

- **name** – name of the type
- **byte\_size** – amount of bytes the type take in memory

`__init__`(*name: str, byte\_size: int, elf\_object*)

**Parameters**

- **name** (`str`)
- **byte\_size** (`int`)

**static** `read_from_die`(*die: DIE, elf\_object*)

entry method to read a `DW_TAG_XXX_type`

**Parameters**

**die** (`DIE`)

**static** `supported_die`(*die: DIE*) → `bool`

**Return type**

`bool`

**Parameters**

**die** (`DIE`)

**class** `cle.backends.elf.variable_type.PointerType`

Bases: `VariableType`

Entry class for `DW_TAG_pointer_type`. It is inherited from `VariableType`

**Parameters**

- **byte\_size** (`int`) – amount of bytes the type take in memory
- **elf\_object** – elf object to reference to (useful for pointer,...)
- **referenced\_offset** (`int`) – type of the referenced as offset in the `compilation_unit`

`__init__`(*byte\_size: int, elf\_object, referenced\_offset: int*)

**Parameters**

- **byte\_size** (`int`)
- **referenced\_offset** (`int`)

**classmethod** `read_from_die`(*die: DIE, elf\_object*)

read an entry of `DW_TAG_pointer_type`. return `None` when there is no `byte_size` or `type` attribute.

**Parameters**

**die** (`DIE`)

**property referenced\_type**

attribute to get the referenced type. Return None if the type is not loaded

**class** `cle.backends.elf.variable_type.BaseType`

Bases: *VariableType*

Entry class for DW\_TAG\_base\_type. It is inherited from VariableType

**classmethod** `read_from_die(die: DIE, elf_object)`

read an entry of DW\_TAG\_base\_type. return None when there is no byte\_size attribute.

**Parameters**

**die** (*DIE*)

**class** `cle.backends.elf.variable_type.StructType`

Bases: *VariableType*

Entry class for DW\_TAG\_structure\_type. It is inherited from VariableType

**Parameters**

- **name** (*str*) – name of the type
- **byte\_size** (*int*) – amount of bytes the type take in memory
- **elf\_object** – elf object to reference to (useful for pointer,...)

`__init__` (*name: str, byte\_size: int, elf\_object, members*)

**Parameters**

- **name** (*str*)
- **byte\_size** (*int*)

**classmethod** `read_from_die(die: DIE, elf_object)`

read an entry of DW\_TAG\_structure\_type. return None when there is no byte\_size attribute.

**Parameters**

**die** (*DIE*)

**class** `cle.backends.elf.variable_type.UnionType`

Bases: *StructType*

Entry class for DW\_TAG\_union\_type. Inherits from StructType to make it trivial.

**class** `cle.backends.elf.variable_type.StructMember`

Bases: *object*

Entry class for DW\_TAG\_member. This is not a type but a named member inside a struct. Use the property *type* to get its variable type.

**Parameters**

- **name** (*str*) – name of the member
- **addr\_offset** (*int*) – address offset of the member in the struct
- **elf\_object** – elf object to reference to (useful for pointer,...)
- **type\_offset** – type as offset in the compilation\_unit

**Variables**

**name** – name of the member

`__init__(name: str, addr_offset: int, type_offset, elf_object)`

**Parameters**

- **name** (*str*)
- **addr\_offset** (*int*)

**classmethod** `read_from_die(die: DIE, elf_object)`

read an entry of DW\_TAG\_member\_type. return None when there is no type attribute.

**Parameters**

**die** (*DIE*)

**property** `type`

attribute to get the type of the member. Return None if the type is not loaded

**class** `cle.backends.elf.variable_type.ArrayType`

Bases: *VariableType*

Entry class for DW\_TAG\_array\_type. It is inherited from VariableType

**Parameters**

- **byte\_size** – amount of bytes the type take in memory
- **elf\_object** – elf object to reference to (useful for pointer,...)
- **element\_offset** – type of the array elements as offset in the compilation\_unit

`__init__(byte_size, elf_object, element_offset)`

**classmethod** `read_from_die(die: DIE, elf_object)`

read an entry of DW\_TAG\_array\_type. return None when there is no type attribute.

**Parameters**

**die** (*DIE*)

**property** `element_type`

**class** `cle.backends.elf.variable_type.TypedefType`

Bases: *VariableType*

Entry class for DW\_TAG\_typedef. Inherits from VariableType.

**Parameters**

- **name** (*str*) – name of the new type
- **elf\_object** – elf object to reference to (useful for pointer,...)
- **type\_offset** – type as offset in the compilation\_unit

`__init__(name: str, byte_size, elf_object, type_offset)`

**Parameters**

**name** (*str*)

**classmethod** `read_from_die(die: DIE, elf_object)`

read an entry of DW\_TAG\_member\_type. return None when there is no type attribute.

**Parameters**

**die** (*DIE*)

**property type**

attribute to get the type of the member. Return None if the type is not loaded

**References**

- <http://www.hexblog.com/wp-content/uploads/2012/06/Recon-2012-Skochinsky-Compiler-Internals.pdf>
- <https://www.airs.com/blog/archives/460>
- <https://www.airs.com/blog/archives/464>

**class cle.backends.elf.lsdas.ExceptionTableHeader**

Bases: `object`

`__init__(lp_start, ttype_encoding, ttype_offset, call_site_encoding, call_site_table_len)`

`lp_start`

`ttype_encoding`

`ttype_offset`

`call_site_encoding`

`call_site_table_len`

**class cle.backends.elf.lsdas.CallSiteEntry**

Bases: `object`

`__init__(cs_start, cs_len, cs_lp, cs_action)`

`cs_start`

`cs_len`

`cs_lp`

`cs_action`

**class cle.backends.elf.lsdas.LSDAExceptionTable**

Bases: `object`

LSDA exception table parser.

TODO: Much of this class should be eventually moved to pyelftools.

`__init__(stream, bits, little_endian=True)`

`parse_lsdas(address, offset)`

**class cle.backends.elf.hashtables.ELFHashTable**

Bases: `object`

Functions to do lookup from a HASH section of an ELF file.

Information: [http://docs.oracle.com/cd/E23824\\_01/html/819-0690/chapter6-48031.html](http://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-48031.html)

`__init__(symtab, stream, offset, arch)`

**Parameters**

- `symtab` – The symbol table to perform lookups from (as a pyelftools SymbolTableSection).

- **stream** – A file-like object to read from the ELF’s memory.
- **offset** – The offset in the object where the table starts.
- **arch** – The ArchInfo object for the ELF file.

**get**(*k*)

Perform a lookup. Returns a pyelftools Symbol object, or None if there is no match.

**Parameters**

**k** – The string to look up.

**static elf\_hash**(*key*)

**class** cle.backends.elf.hashtable.**GNUHashTable**

Bases: `object`

Functions to do lookup from a GNU\_HASH section of an ELF file.

Information: [https://blogs.oracle.com/ali/entry/gnu\\_hash\\_elf\\_sections](https://blogs.oracle.com/ali/entry/gnu_hash_elf_sections)

**\_\_init\_\_**(*symtab, stream, offset, arch*)

**Parameters**

- **symtab** – The symbol table to perform lookups from (as a pyelftools SymbolTableSection).
- **stream** – A file-like object to read from the ELF’s memory.
- **offset** – The offset in the object where the table starts.
- **arch** – The ArchInfo object for the ELF file.

**get**(*k*)

Perform a lookup. Returns a pyelftools Symbol object, or None if there is no match.

**Parameters**

**k** – The string to look up

**static gnu\_hash**(*key*)

**class** cle.backends.elf.subprogram.**LexicalBlock**

Bases: `object`

A lexical block is a sequence of source statements, e.g. a while/for loop or an if statement or some bracketed block.

Corresponds to a DW\_TAG\_LexicalBlock in DWARF.

**Parameters**

- **super\_block** – The lexical block which contains this block
- **low\_pc** (*int* | *None*) – The relative start address of the block
- **high\_pc** (*int* | *None*) – The relative end address of the block

**Variables**

- **low\_pc** – The relative start address of the subprogram
- **high\_pc** – The relative end address of the subprogram
- **child\_blocks** (*list*[*LexicalBlock*]) – Lexical blocks inside this block (only direct childs)

```
__init__(low_pc: int | None, high_pc: int | None, ranges: list[tuple[int, int]] | None = None, source_file: str | None = None, source_line: int | None = None) → None
```

#### Parameters

- **low\_pc** (*int* | *None*)
- **high\_pc** (*int* | *None*)
- **ranges** (*list[tuple[int, int]]* | *None*)
- **source\_file** (*str* | *None*)
- **source\_line** (*int* | *None*)

#### Return type

*None*

```
rebase(delta: int)
```

#### Parameters

**delta** (*int*)

```
class cle.backends.elf.subprogram.Subprogram
```

Bases: *LexicalBlock*

DW\_TAG\_subprogram for DWARF. The behavior is mostly inherited from *LexicalBlock* to avoid redundancy.

#### Parameters

- **name** (*str* | *None*) – The name of the function/program
- **low\_pc** (*int* | *None*) – The relative start address of the subprogram
- **high\_pc** (*int* | *None*) – The relative end address of the subprogram

#### Variables

- **name** – The name of the function/program
- **local\_variables** (*list[Variable]*) – All local variables in a Subprogram (they may reside in several child blocks)

```
__init__(name: str | None, low_pc: int | None, high_pc: int | None, ranges: list[tuple[int, int]] | None = None, source_file: str | None = None, source_line: int | None = None) → None
```

#### Parameters

- **name** (*str* | *None*)
- **low\_pc** (*int* | *None*)
- **high\_pc** (*int* | *None*)
- **ranges** (*list[tuple[int, int]]* | *None*)
- **source\_file** (*str* | *None*)
- **source\_line** (*int* | *None*)

#### Return type

*None*

```
rebase(delta: int)
```

#### Parameters

**delta** (*int*)

```
class cle.backends.elf.compilation_unit.CompilationUnit
```

Bases: `object`

CompilationUnit for DWARF See <http://dwarfstd.org/doc/DWARF5.pdf> page 60

```
__init__(name, comp_dir, low_pc, high_pc, language, elf_object, ranges)
```

```
property min_addr
```

```
property max_addr
```

## 2.3.2 PE

```
class cle.backends.PE
```

Bases: `Backend`

Representation of a PE (i.e. Windows) binary.

Useful backend options:

- `debug_symbols`: Provides the path to a PDB file which contains the binary's debug symbols
- `debug_symbol_dirs`: List of directories to search for PDB files (searched before symbol servers)
- `debug_symbol_path_str`: A string indicating symbol search paths, which may be provided in the `_NT_SYMBOL_PATH` format.
- `download_debug_symbols`: Whether to attempt downloading debug symbols from symbol servers (if provided) or not. Default to False.
- `download_debug_symbol_confirm`: A callable that takes a URL string and returns True if downloading the debug symbol from the URL is allowed by the user, False otherwise.
- `download_debug_symbol_progress`: A callable that takes two integer arguments: bytes downloaded and total bytes. This callable is called periodically to report download progress.
- `search_microsoft_symserver`: Whether to include the Microsoft symbol server in symbol searches. Default to True. Requires `download_debug_symbols` to be True to have any effect.

```
is_default = True
```

```
__init__(*args, debug_symbols=None, debug_symbol_dirs=None, debug_symbol_path_str: str | None = None, download_debug_symbols: bool = False, download_debug_symbol_confirm: Callable[[str], bool] | None = None, download_debug_symbol_progress: Callable[[int, int | None], bool] | None = None, search_microsoft_symserver: bool = True, **kwargs)
```

### Parameters

- `binary` – The path to the binary to load
- `binary_stream` – The open stream to this binary. The reference to this will be held until you call close.
- `is_main_bin` – Whether this binary should be loaded as the main executable
- `debug_symbol_path_str` (`str` | `None`)
- `download_debug_symbols` (`bool`)
- `download_debug_symbol_confirm` (`Callable[[str], bool]` | `None`)

- **download\_debug\_symbol\_progress** (*Callable*[[*int*, *int* | *None*], *bool*] | *None*)
- **search\_microsoft\_symserver** (*bool*)

**classmethod** **is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**classmethod** **check\_magic\_compatibility**(*stream*)

Check if a stream of bytes contains the same magic number as the main object

**classmethod** **check\_compatibility**(*spec*, *obj*)

Performs a minimal static load of *spec* and returns whether it's compatible with *other\_obj*

**close**()

**get\_symbol**(*name*)

Look up the symbol with the given name. Symbols can be looked up by ordinal with the name "ordinal.  
%d" % num

**load\_symbols\_from\_pdb**(*pdb\_path*)

Load available symbols from PDB at *pdb\_path*

**class** `cle.backends.pe.regions.PESection`

Bases: *Section*

Represents a section for the PE format.

**\_\_init\_\_**(*pe\_section*, *remap\_offset=0*, *name: str* | *None = None*)

#### Parameters

- **name** (*str* | *None*) – The name of the section
- **offset** (*int*) – The offset into the binary file this section begins
- **vaddr** (*int*) – The address in virtual memory this section begins
- **size** (*int*) – How large this section is

**property** **is\_readable**

Whether this section has read permissions

**property** **is\_writable**

Whether this section has write permissions

**property** **is\_executable**

Whether this section has execute permissions

**property** **only\_contains\_uninitialized\_data**

Whether this section is initialized to zero after the executable is loaded.

**class** `cle.backends.pe.symbol.WinSymbol`

Bases: *Symbol*

Represents a symbol for the PE format.

**\_\_init\_\_**(*owner*, *name*, *addr*, *is\_import*, *is\_export*, *ordinal\_number*, *forwarder*,  
*symbol\_type=SymbolType.TYPE\_FUNCTION*)

Not documenting this since if you try calling it, you're wrong.

**resolve\_forwarder()**

If this symbol is a forwarding export, return the symbol the forwarding refers to, or None if it cannot be found

**2.3.3 Mach-O**

**class** cle.backends.macho.macho.MachO

Bases: *Backend*

**Mach-O binaries for CLE**

The Mach-O format is notably different from other formats. Specifically:

- Sections are always part of a segment, so *self.sections* will be empty.
- Symbols cannot be categorized like in ELF.
- Symbol resolution must be handled by the binary.
- Rebasing in dyld is implemented by adding a small slide to addresses inside the binary, instead of changing the base address of the binary. Consequently, the addresses are absolute rather than relative. CLE requires relative addresses, leading to numerous *AT.from\_lva().to\_rva()* calls in this backend.

**is\_default** = True

**MH\_MAGIC\_64** = 4277009103

**MH\_CIGAM\_64** = 3489328638

**MH\_MAGIC** = 4277009102

**MH\_CIGAM** = 3472551422

**\_\_init\_\_**(\*args, \*\*kwargs)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**ncmds**: int

**sizeofcmd**: int

**property stubs**

Maps names to addresses.

**property plt**

Maps names to addresses.

**property reverse\_stubs**

Maps addresses to names.

**property reverse\_plt**

Maps addresses to names.

**set\_arch**(arch)

**property min\_addr**

This returns the lowest virtual address contained in any loaded segment of the binary.

**classmethod check\_compatibility**(*spec, obj*)

Performs a minimal static load of *spec* and returns whether it's compatible with *other\_obj*

**static extract\_soname**(*path*)

Extracts the shared object identifier from the path, or returns None if it cannot.

**classmethod is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**is\_thumb\_interworking**(*address*)

Returns true if the given address is a THUMB interworking address

**decode\_thumb\_interworking**(*address*)

Decodes a thumb interworking address

**find\_segment\_by\_name**(*name*)**do\_binding**()**get\_string**(*start*)

Loads a string from the string table

**parse\_lc\_str**(*f, start, limit: int | None = None*)

Parses a lc\_str data structure

**Parameters**

**limit** (*int* | *None*)

S = ~S

**get\_symbol\_by\_address\_fuzzy**(*address*)

Locates a symbol by checking the given address against *sym.addr*, *sym.bind\_xrefs* and *sym.symbol\_stubs*

**get\_symbol**(*name, include\_stab=False, fuzzy=False*)

Returns all symbols matching name.

Note that especially when *include\_stab=True* there may be multiple symbols with the same name, therefore this method always returns an array.

**Parameters**

- **name** – the name of the symbol
- **include\_stab** – Include debugging symbols NOT RECOMMENDED
- **fuzzy** – Replace exact match with “contains”-style match

**get\_symbol\_by\_insertion\_order**(*idx: int*) → *AbstractMachOSymbol***Parameters**

**idx** (*int*) – idx when this symbol was inserted

**Return type**

*AbstractMachOSymbol*

**Returns**

**get\_segment\_by\_name**(*name*)

Searches for a MachOSegment with the given name and returns it :type name: :param name: Name of the sought segment :return: MachOSegment or None

**class** cle.backends.macho.macho.**SymbolList**

Bases: SortedKeyList

Special data structure that extends SortedKeyList to allow looking up a MachO library by name and ordinal quickly without having to iterate over the whole list

**\_\_init\_\_**(*\*\*kwargs*)

Initialize sorted-key list instance.

Optional *iterable* argument provides an initial iterable of values to initialize the sorted-key list.

Optional *key* argument defines a callable that, like the *key* argument to Python's *sorted* function, extracts a comparison key from each value. The default is the identity function.

Runtime complexity:  $O(n \cdot \log(n))$

```
>>> from operator import neg
>>> skl = SortedKeyList(key=neg)
>>> skl
SortedKeyList([], key=<built-in function neg>)
>>> skl = SortedKeyList([3, 1, 2], key=neg)
>>> skl
SortedKeyList([3, 2, 1], key=<built-in function neg>)
```

**Parameters**

- **iterable** – initial values (optional)
- **key** – function used to extract comparison key (optional)

**add**(*value*: AbstractMachOSymbol)

Add *value* to sorted-key list.

Runtime complexity:  $O(\log(n))$  – approximate.

```
>>> from operator import neg
>>> skl = SortedKeyList(key=neg)
>>> skl.add(3)
>>> skl.add(1)
>>> skl.add(2)
>>> skl
SortedKeyList([3, 2, 1], key=<built-in function neg>)
```

**Parameters**

**value** (*AbstractMachOSymbol*) – value to add to sorted-key list

**get\_by\_name\_and\_ordinal**(*name*: str, *ordinal*: int, *include\_stab*=False) → list[AbstractMachOSymbol]**Return type**

list[AbstractMachOSymbol]

**Parameters**

- **name** (*str*)

- `ordinal (int)`

**class** `cle.backends.macho.symbol.SymbolType`

Bases: `Enum`

ABI-agnostic symbol types

`TYPE_OTHER = 0`

`TYPE_NONE = 1`

`TYPE_FUNCTION = 2`

`TYPE_OBJECT = 3`

`TYPE_SECTION = 4`

`TYPE_TLS_OBJECT = 5`

`TYPE_FUNCTION_OR_OBJECT = 6`

**class** `cle.backends.macho.symbol.AbstractMachOSymbol`

Bases: `Symbol`

Base class for Mach-O symbols. Defines the minimum common properties all types of mach-o symbols must have

**owner:** `MachO`

**\_\_init\_\_** (*owner: Backend, name: str, relative\_addr: int, size: int, sym\_type: SymbolType*)

Not documenting this since if you try calling it, you're wrong.

#### Parameters

- **owner** (`Backend`)
- **name** (`str`)
- **relative\_addr** (`int`)
- **size** (`int`)
- **sym\_type** (`SymbolType`)

**property** `library_ordinal`

**property** `is_stab: bool`

**property** `library_name: str | None`

**property** `library_base_name: str | None`

**class** `cle.backends.macho.symbol.SymbolTableSymbol`

Bases: `AbstractMachOSymbol`

“Regular” symbol. Made to be (somewhat) compatible with `backends.Symbol`. A `SymbolTableSymbol` is an entry in the binary’s symbol table.

Note that ELF-specific fields from `backends.Symbol` are not used and semantics of the remaining fields differ in many cases. As a result most stock functionality from Angr and related libraries WILL NOT WORK PROPERLY on `MachOSymbol`.

Much of the code below is based on heuristics as official documentation is sparse, consider yourself warned!

The relevant struct with documentation is `nlist_64` defined in `mach-o/nlist.h`

`__init__` (*owner*: `MachO`, *syntab\_offset*, *n\_strx*, *n\_type*, *n\_sect*, *n\_desc*, *n\_value*)

Not documenting this since if you try calling it, you're wrong.

#### Parameters

**owner** (`MachO`)

**property library\_name**: `str` | `None`

**property segment\_name**

**property section\_name**

**property value**

**property referenced\_symbol\_index**

For indirect symbols `n_value` contains an index into the string table indicating the referenced symbol's name

**property is\_weak**

`bool(x) -> bool`

Returns True when the argument `x` is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

**property is\_function**

Whether this symbol is a function

**property is\_stab**: `bool`

**property is\_private\_external**: `bool`

**property is\_external**: `bool`

**property sym\_type**

**property is\_common**

`bool(x) -> bool`

Returns True when the argument `x` is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

**property common\_align**

**property reference\_type**

**property library\_ordinal**

**property is\_no\_dead\_strip**

**property is\_desc\_discarded**

**property is\_weak\_referenced**

**property is\_weak\_defined**

**property is\_reference\_to\_weak**

**property is\_thumb\_definition**

**property is\_symbol\_resolver**

**property is\_alt\_entry**

```
class cle.backends.macho.symbol.BindingSymbol
```

Bases: *AbstractMachOSymbol*

“Binding” symbol. Made to be (somewhat) compatible with backends.Symbol. A BindingSymbol is an imported symbol discovered during the binding process.

Note that ELF-specific fields from backends.Symbol are not used and semantics of the remaining fields differ in many cases. As a result most stock functionality from Angr and related libraries WILL NOT WORK PROPERLY on MachOSymbol.

Much of the code below is based on heuristics as official documentation is sparse, consider yourself warned!

```
__init__(owner, name, lib_ordinal)
```

Not documenting this since if you try calling it, you’re wrong.

```
property library_name: str | None
```

```
property is_function
```

Whether this symbol is a function

```
demangled_name()
```

```
property library_ordinal
```

```
class cle.backends.macho.symbol.DyldBoundSymbol
```

Bases: *AbstractMachOSymbol*

The new kind of symbol handling introduced with ios15

```
owner: MachO
```

```
__init__(owner, name, lib_ordinal)
```

Based on the constructor of BindingSymbol

```
property library_name: str | None
```

```
property is_function
```

Whether this symbol is a function

```
demangled_name()
```

```
property library_ordinal
```

```
class cle.backends.macho.section.MachOSection
```

Bases: *Section*

Mach-O Section, only defined within the context of a Mach-O Segment.

- offset is the offset into the file the region starts
- vaddr (or just addr) is the virtual address
- filesize (or just size) is the size of the region in the file
- memsize (or vsize) is the size of the region when loaded into memory
- segname is the corresponding segment’s name without padding
- sectname is the section’s name without padding
- align is the sections alignment as a power of 2

- `reloff` is the file offset to the section's relocation entries
- `nreloc` is the number of relocation entries for this section
- `flags` is a bit vector containing per-section flags
- `r1` and `r2` are values for the `reserved1` and `reserved2` fields respectively

`__init__`(*offset, vaddr, size, vsize, segname, sectname, align, reloff, nreloc, flags, r1, r2, parent\_segment: MachOSegment | None = None*)

#### Parameters

- **name** (*str*) – The name of the section
- **offset** (*int*) – The offset into the binary file this section begins
- **vaddr** (*int*) – The address in virtual memory this section begins
- **size** (*int*) – How large this section is
- **parent\_segment** (*MachOSegment | None*)

**property full\_name:** *str*

**property type**

**property attributes**

**property is\_readable**

Always true, because sections should always be readable :return:

**property is\_writable**

Returns the permission of the parent segment, because MachO sections simply inherit that :return:

**property is\_executable**

Returns the permission of the parent segment, because MachO sections simply inherit that :return:

**property only\_contains\_uninitialized\_data**

I actually don't know if this is true, but it seems like a saner assumption than true :return:

**class** `cle.backends.macho.segment.MachOSegment`

Bases: *Segment*

Mach-O Segment

- `offset` is the offset into the file the region starts
- `vaddr` (or just `addr`) is the virtual address
- `filesize` (or just `size`) is the size of the region in the file
- `memsize` (or just `vsize`) is the size of the region when loaded into memory
- `segname` is the segment's name without padding
- `nsect` is the number of sections contained in this segment
- `sections` is an array of `MachOSections`
- `flags` is a bit vector containing per-segment flags
- `initprot` and `maxprot` are initial and maximum permissions respectively

`__init__`(*offset, vaddr, size, vsize, segname, nsect, sections, flags, initprot, maxprot*)

**get\_section\_by\_name**(*name*)

Searches for a section by name within this segment :type name: :param name: Name of the section :return: MachOSection or None

**property is\_readable**

**property is\_writable**

**property is\_executable**

`cle.backends.macho.binding.chh(x)`

`cle.backends.macho.binding.read_uleb(blob: bytes, offset: int) → tuple[int, int]`

Reads a number encoded as uleb128

**Return type**

`tuple[int, int]`

**Parameters**

- **blob** (*bytes*)
- **offset** (*int*)

`cle.backends.macho.binding.read_sleb(blob, offset)`

Reads a number encoded as sleb128

**class** `cle.backends.macho.binding.BindingState`

Bases: `object`

State object

`__init__(is_64)`

`add_address_ov(address, addend)`

this is a very ugly klugde. It is needed because dyld relies on overflow semantics and represents several negative offsets through BIG ulebs

`check_address_bounds()`

**class** `cle.backends.macho.binding.BindingHelper`

Bases: `object`

Factors out binding logic from MachO. Intended to work in close conjunction with MachO not for standalone use

`__init__(binary)`

**binary:** `MachO`

`do_normal_bind(blob: bytes)`

Performs non-lazy, non-weak bindings :type blob: `bytes` :param blob: Blob containing binding opcodes

**Parameters**

**blob** (*bytes*)

`do_lazy_bind(blob)`

Performs lazy binding

**do\_rebases**(*blob: bytes*)

Handles the rebase blob Implementation based closely on ImageLoaderMachOCompressed::rebase from dyld <https://github.com/apple-opensource/dyld/blob/e3f88907bebb8421f50f0943595f6874de70ebe0/src/ImageLoaderMachOCompressed.cpp#L463>

**Parameters**

**blob** (*bytes*)

**Returns**

**static read\_uleb**(*blob, offset*) → *tuple[int, int]*

little helper to read ulebs, that also returns the new index :type blob: :param blob: :type offset: :param offset:

**Return type**

*tuple[int, int]*

**Returns**

**rebase\_at**(*address: int, ty: RebaseType*)

**Parameters**

- **address** (*int*)
- **ty** (*RebaseType*)

`cle.backends.macho.binding.n_opcode_done`(*s: BindingState, \_b: MachO, \_i: int, \_blob: bytes*) → *BindingState*

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **\_b** (*MachO*)
- **\_i** (*int*)
- **\_blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_set_dylib_ordinal_imm`(*s: BindingState, \_b: MachO, i: int, \_blob: bytes*) → *BindingState*

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **\_b** (*MachO*)
- **i** (*int*)
- **\_blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_set_dylib_ordinal_uleb`(*s: BindingState, \_b: MachO, \_i: int, blob: bytes*) → *BindingState*

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **\_b** (*MachO*)
- **\_i** (*int*)
- **blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_set_dylib_special_imm(s: BindingState, _b: MachO, i: int, _blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **\_b** (*MachO*)
- **i** (*int*)
- **\_blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_set_trailing_flags_imm(s: BindingState, _b: MachO, i: int, blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **\_b** (*MachO*)
- **i** (*int*)
- **blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_set_type_imm(s: BindingState, _b: MachO, i: int, _blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **\_b** (*MachO*)
- **i** (*int*)
- **\_blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_set_addend_sleb(s: BindingState, _b: MachO, _i: int, blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)

- **\_b** (*Mach0*)
- **\_i** (*int*)
- **blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_set_segment_and_offset_uleb(s: BindingState, b: MachO, i: int, blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **b** (*Mach0*)
- **i** (*int*)
- **blob** (*bytes*)

`cle.backends.macho.binding.l_opcode_set_segment_and_offset_uleb(s: BindingState, b: MachO, i: int, blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **b** (*Mach0*)
- **i** (*int*)
- **blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_add_addr_uleb(s: BindingState, _b: MachO, _i: int, blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **\_b** (*Mach0*)
- **\_i** (*int*)
- **blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_do_bind(s: BindingState, b: MachO, _i: int, _blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **b** (*Mach0*)
- **\_i** (*int*)

- **\_blob** (*bytes*)

`cle.backends.macho.binding.l_opcode_do_bind(s: BindingState, b: MachO, _i: int, _blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **b** (*MachO*)
- **\_i** (*int*)
- **\_blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_do_bind_add_addr_uleb(s: BindingState, b: MachO, _i: int, blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **b** (*MachO*)
- **\_i** (*int*)
- **blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_do_bind_add_addr_imm_scaled(s: BindingState, b: MachO, i: int, _blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **b** (*MachO*)
- **i** (*int*)
- **\_blob** (*bytes*)

`cle.backends.macho.binding.n_opcode_do_bind_uleb_times_skipping_uleb(s: BindingState, b: MachO, _i: int, blob: bytes) → BindingState`

**Return type**

*BindingState*

**Parameters**

- **s** (*BindingState*)
- **b** (*MachO*)
- **\_i** (*int*)
- **blob** (*bytes*)

**class** cle.backends.macho.binding.MachOSymbolRelocation

Bases: *Relocation*

Generic Relocation for MachO. It handles relocations that point to symbols

**symbol:** *AbstractMachOSymbol*

**\_\_init\_\_**(owner: MachO, symbol: AbstractMachOSymbol, relative\_addr: int, data)

#### Parameters

- **owner** (MachO)
- **symbol** (AbstractMachOSymbol)
- **relative\_addr** (int)

**resolve\_symbol**(solist, thumb=False, extern\_object=None, \*\*kwargs)

**property** dest\_addr

**property** value

**class** cle.backends.macho.binding.MachOPointerRelocation

Bases: *Relocation*

A relocation for a pointer without any associated symbol These are either generated while handling the rebase blob, or while parsing chained fixups

**\_\_init\_\_**(owner: MachO, relative\_addr: int, data)

#### Parameters

- **owner** (MachO)
- **relative\_addr** (int) – the relative addr where this relocation is located
- **data** – the rebase offset relative to the linked base

**property** value

**resolve\_symbol**(solist, thumb=False, extern\_object=None, \*\*kwargs)

This relocation has no associated symbol, so we don't need to resolve it. :type solist: :param solist: :type thumb: :param thumb: :type extern\_object: :param extern\_object: :type kwargs: :param kwargs: :return:

cle.backends.macho.binding.default\_binding\_handler(state: BindingState, binary: MachO)

Binds location to the symbol with the given name and library ordinal

#### Parameters

- **state** (BindingState)
- **binary** (MachO)

**class** cle.backends.macho.structs.HelperStruct

Bases: *Structure*

Subclass of ctypes.Structure that adds a helpful repr method for debugging

**class** cle.backends.macho.structs.DyldImportFormats

Bases: *IntEnum*

<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L249-L254>

```

DYLD_CHAINED_IMPORT = 1
DYLD_CHAINED_IMPORT_ADDEND = 2
DYLD_CHAINED_IMPORT_ADDEND64 = 3
__new__(value)

```

```
class cle.backends.macho.structs.DyldChainedPtrFormats
```

```
Bases: IntEnum
```

```
https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L89-L104
```

```

DYLD_CHAINED_PTR_ARM64E = 1
DYLD_CHAINED_PTR_64 = 2
DYLD_CHAINED_PTR_32 = 3
DYLD_CHAINED_PTR_32_CACHE = 4
DYLD_CHAINED_PTR_32_FIRMWARE = 5
DYLD_CHAINED_PTR_64_OFFSET = 6
DYLD_CHAINED_PTR_ARM64E_KERNEL = 7
DYLD_CHAINED_PTR_64_KERNEL_CACHE = 8
DYLD_CHAINED_PTR_ARM64E_USERLAND = 9
DYLD_CHAINED_PTR_ARM64E_FIRMWARE = 10
DYLD_CHAINED_PTR_X86_64_KERNEL_CACHE = 11
DYLD_CHAINED_PTR_ARM64E_USERLAND24 = 12
__new__(value)

```

```
class cle.backends.macho.structs.dyld_chained_ptr_arm64e_auth_rebase
```

```
Bases: HelperStruct
```

```
https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L128-L138
```

```
addrDiv
```

```
Structure/Union member
```

```
auth
```

```
Structure/Union member
```

```
bind
```

```
Structure/Union member
```

```
diversity
```

```
Structure/Union member
```

```
key
```

```
Structure/Union member
```

```
next
```

```
Structure/Union member
```

**target**

Structure/Union member

**class** cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bindBases: *HelperStruct*<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L140-L151>**addrDiv**

Structure/Union member

**auth**

Structure/Union member

**bind**

Structure/Union member

**diversity**

Structure/Union member

**key**

Structure/Union member

**next**

Structure/Union member

**ordinal**

Structure/Union member

**zero**

Structure/Union member

**class** cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_rebaseBases: *HelperStruct*<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L107-L115>**auth**

Structure/Union member

**bind**

Structure/Union member

**high8**

Structure/Union member

**next**

Structure/Union member

**target**

Structure/Union member

**class** cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bindBases: *HelperStruct*<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L117-L126>**addend**

Structure/Union member

**auth**

Structure/Union member

**bind**

Structure/Union member

**next**

Structure/Union member

**ordinal**

Structure/Union member

**zero**

Structure/Union member

**class** cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind24Bases: *HelperStruct*<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L164-L173>**class** cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind24Bases: *HelperStruct*<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L175-L186>**addrDiv**

Structure/Union member

**auth**

Structure/Union member

**bind**

Structure/Union member

**diversity**

Structure/Union member

**key**

Structure/Union member

**next**

Structure/Union member

**ordinal**

Structure/Union member

**zero**

Structure/Union member

**class** cle.backends.macho.structs.Arm64eBases: *Union*named after the Union *Arm64e* from dyld MachOLoaded.h <https://github.com/apple-opensource/dyld/blob/852.2/dyld3/MachOLoaded.h#L89-L103>**authRebase:** *dyld\_chained\_ptr\_arm64e\_auth\_rebase*

Structure/Union member

**authBind:** *dyld\_chained\_ptr\_arm64e\_auth\_bind*

Structure/Union member

**rebase:** *dyld\_chained\_ptr\_arm64e\_rebase*

Structure/Union member

**bind:** *dyld\_chained\_ptr\_arm64e\_bind*

Structure/Union member

**bind24:** *dyld\_chained\_ptr\_arm64e\_bind24*

Structure/Union member

**authBind24:** *dyld\_chained\_ptr\_arm64e\_auth\_bind24*

Structure/Union member

**property sign\_extended\_addend**

**property unpack\_target**

**static check\_valid\_pointer\_format**(*pointer\_format: DyldChainedPtrFormats*) → bool

helper to check if a pointer format is relevant for this :type *pointer\_format: DyldChainedPtrFormats*

:param *pointer\_format:*

**Return type**

bool

**Returns**

**Parameters**

**pointer\_format** (*DyldChainedPtrFormats*)

**class** cle.backends.macho.structs.dyld\_chained\_ptr\_64\_rebase

Bases: *HelperStruct*

<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L153-L161>

**target:** **int**

Structure/Union member

**high8:** **int**

Structure/Union member

**next:** **int**

Structure/Union member

**bind:** **int**

Structure/Union member

**property unpackedTarget**

**class** cle.backends.macho.structs.dyld\_chained\_ptr\_64\_bind

Bases: *HelperStruct*

<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L189-L197>

**ordinal:** **int**

Structure/Union member

**addend:** **int**

Structure/Union member

**next:** **int**

Structure/Union member

**bind:** `int`

Structure/Union member

**class** `cle.backends.macho.structs.Generic64`

Bases: `Union`

named after the Union `Generic64` from `dyld MachOLoaded.h` <https://github.com/apple-opensource/dyld/blob/852.2/dyld3/MachOLoaded.h#L105-L111>

**rebase:** `dyld_chained_ptr_64_rebase`

Structure/Union member

**bind:** `dyld_chained_ptr_64_bind`

Structure/Union member

**static** `check_valid_pointer_format(pointer_format: DyldChainedPtrFormats) → bool`

**Return type**

`bool`

**Parameters**

`pointer_format` (`DyldChainedPtrFormats`)

**class** `cle.backends.macho.structs.ChainedFixupPointerOnDisk`

Bases: `Union`

the `ChainedFixupPointerOnDisk` union from `dyld MachOLoaded.h` <https://github.com/apple-opensource/dyld/blob/852.2/dyld3/MachOLoaded.h#L87-L141>

**generic64:** `Generic64`

Structure/Union member

**arm64e:** `Arm64e`

Structure/Union member

**isBind**(`pointer_format: DyldChainedPtrFormats`) → `tuple[int, int] | None`

Port of `ChainedFixupPointerOnDisk::isBind(uint16_t pointerFormat, uint32_t& bindOrdinal, int64_t& addend)` <https://github.com/apple-opensource/dyld/blob/852.2/dyld3/MachOLoaded.cpp#L1098-L1147> Returns `None` if not a bind (so *if* `struct.isBind()` works),

**Return type**

`tuple[int, int] | None`

**Returns**

**Parameters**

`pointer_format` (`DyldChainedPtrFormats`)

**isRebase**(`pointer_format: DyldChainedPtrFormats, preferredLoadAddress: int`) → `int | None`

port of `ChainedFixupPointerOnDisk::isRebase(uint16_t pointerFormat, uint64_t preferredLoadAddress, uint64_t& targetRuntimeOffset)` <https://github.com/apple-opensource/dyld/blob/852.2/dyld3/MachOLoaded.cpp#L1046-L1096> :type `pointer_format: DyldChainedPtrFormats` :param `pointer_format: :type preferredLoadAddress: int` :param `preferredLoadAddress: I think that's just the requested base address`

**Return type**

`int | None`

**Returns**

**Parameters**

- **pointer\_format** (*DyldChainedPtrFormats*)
- **preferredLoadAddress** (*int*)

**class** cle.backends.macho.structs.**DyldImportStruct**

Bases: *HelperStruct*

Meta Struct for the different kind of import structs and the fields they are all guaranteed to have

**lib\_ordinal:** *int*

**weak\_import:** *bool*

**name\_offset:** *int*

**static** **get\_struct**(*pointer: DyldImportFormats*) → *type[DyldImportStruct]*

**Return type**

*type[DyldImportStruct]*

**Parameters**

**pointer** (*DyldImportFormats*)

**class** cle.backends.macho.structs.**dyld\_chained\_import**

Bases: *DyldImportStruct*

Struct for symbol format DYLD\_CHAINED\_IMPORT

**lib\_ordinal:** *int*

Structure/Union member

**name\_offset:** *int*

Structure/Union member

**weak\_import:** *bool*

Structure/Union member

**class** cle.backends.macho.structs.**dyld\_chained\_import\_addend**

Bases: *DyldImportStruct*

<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L264-L271>

**addend:** *int*

Structure/Union member

**lib\_ordinal:** *int*

Structure/Union member

**name\_offset:** *int*

Structure/Union member

**weak\_import:** *bool*

Structure/Union member

**class** cle.backends.macho.structs.**dyld\_chained\_import\_addend64**

Bases: *DyldImportStruct*

<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L273-L281>

**addend:** *int*

Structure/Union member

**lib\_ordinal:** `int`  
Structure/Union member

**name\_offset:** `int`  
Structure/Union member

**reserved**  
Structure/Union member

**weak\_import:** `bool`  
Structure/Union member

**class** `cle.backends.macho.structs.dylib_chained_fixups_header`

Bases: *HelperStruct*

<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L36-L46>

**fixups\_version:** `int`  
Structure/Union member

**starts\_offset:** `int`  
Structure/Union member

**imports\_offset:** `int`  
Structure/Union member

**symbols\_offset:** `int`  
Structure/Union member

**imports\_count:** `int`  
Structure/Union member

**imports\_format:** *DyldImportFormats*  
Structure/Union member

**symbols\_format:** `int`  
Structure/Union member

**class** `cle.backends.macho.structs.dylib_chained_starts_in_image`

Bases: *Structure*

<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L48-L54>

**seg\_count:** `int`  
Structure/Union member

**seg\_info\_offset:** `Array`  
Structure/Union member

**class** `cle.backends.macho.structs.dylib_chained_starts_in_segment`

Bases: *HelperStruct*

<https://github.com/apple-opensource/dyld/blob/852.2/include/mach-o/fixup-chains.h#L56-L72>

**page\_size:** `int`  
Structure/Union member

**segment\_offset:** `int`  
Structure/Union member

```

max_valid_pointer: int
    Structure/Union member

page_count: int
    Structure/Union member

page_start: int
    Structure/Union member

property pointer_format: DyldChainedPtrFormats

```

### 2.3.4 Binary Ninja

```
class cle.backends.binja.BinjaSymbol
```

Bases: *Symbol*

```
BINJA_FUNC_SYM_TYPES = []
```

```
BINJA_DATA_SYM_TYPES = []
```

```
BINJA_IMPORT_TYPES = []
```

```
__init__(owner, sym)
```

Not documenting this since if you try calling it, you're wrong.

```
class cle.backends.binja.BinjaReloc
```

Bases: *Relocation*

**property value**

```
class cle.backends.binja.BinjaBin
```

Bases: *Backend*

Get information from binaries using Binary Ninja. Basing this on idabin.py, but will try to be more complete. TODO: add more features as Binary Ninja's feature set improves

```
is_default = True
```

```

BINJA_ARCH_MAP = {'aarch64': <Arch AARCH64 (LE)>, 'armv7': <Arch ARMEL (LE)>,
                  'armv7eb': <Arch ARMEL (BE)>, 'mips32': <Arch MIPS32 (BE)>, 'mipsel32': <Arch
MIPS32 (LE)>, 'ppc': <Arch PPC32 (BE)>, 'ppc_le': <Arch PPC32 (LE)>, 'thumb2':
<Arch ARMEL (LE)>, 'thumb2eb': <Arch ARMEL (BE)>, 'x86': <Arch X86 (LE)>,
                  'x86_64': <Arch AMD64 (LE)>}

```

```
__init__(binary, *args, **kwargs)
```

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

```
static is_compatible(stream)
```

Determine quickly whether this backend can load an object from this stream

**in\_which\_segment**(*addr*)

Return the segment name at address *addr*.

**get\_symbol\_addr**(*sym*)

Get the address of the symbol *sym* from IDA.

**Returns**

An address.

**function\_name**(*addr*)

Return the function name at address *addr*.

**property min\_addr**

this is probably not “right”

**Type**

Get the min address of the binary. (note

**property max\_addr**

Get the max address of the binary.

**property entry**

**get\_strings**()

Extract strings from binary (Binary Ninja).

**Returns**

An array of strings.

**set\_got\_entry**(*name*, *newaddr*)

Resolve import *name* with address *newaddr*. That is, update the GOT entry for *name* with *newaddr*.

**close**()

Release the BinaryView we created in `__init__`:return: None

### 2.3.5 Blob

**class** cle.backends.blob.Blob

Bases: *Backend*

Representation of a binary blob, i.e. an executable in an unknown file format.

**is\_default** = True

**\_\_init\_\_**(\*args, offset=None, segments=None, \*\*kwargs)

**Parameters**

- **arch** – (required) an `archinfo.Arch` for the binary blob.
- **offset** – Skip this many bytes from the beginning of the file.
- **segments** – List of tuples describing how to map data into memory. Tuples are of (file\_offset, mem\_addr, size).

You can’t specify both `offset` and `segments`.

**classmethod** **is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**property min\_addr**

This returns the lowest virtual address contained in any loaded segment of the binary.

**property max\_addr**

This returns the highest virtual address contained in any loaded segment of the binary.

**function\_name(addr)**

Blobs don't support function names.

**contains\_addr(addr)**

Is *addr* in one of the binary's segments/sections we have loaded? (i.e. is it mapped into memory ?)

**in\_which\_segment(addr)**

Blobs don't support segments.

**classmethod check\_compatibility(spec, obj)**

Performs a minimal static load of *spec* and returns whether it's compatible with *other\_obj*

## 2.3.6 COFF

Basic MS COFF object loader based on <https://docs.microsoft.com/en-us/windows/win32/debug/pe-format>

**class cle.backends.coff.IMAGE\_FILE\_MACHINE**

Bases: `IntEnum`

Machine Types

**I386 = 332**

**AMD64 = 34404**

**\_\_new\_\_(value)**

**class cle.backends.coff.CoffFileHeader**

Bases: `Structure`

COFF File Header

**Characteristics**

Structure/Union member

**Machine**

Structure/Union member

**NumberOfSections**

Structure/Union member

**NumberOfSymbols**

Structure/Union member

**PointerToSymbolTable**

Structure/Union member

**SizeOfOptionalHeader**

Structure/Union member

**TimeStamp**

Structure/Union member

---

```
class cle.backends.coff.IMAGE_SCN
    Bases: IntFlag
    Section Flags (Characteristics field)
    MEM_EXECUTE = 536870912
    MEM_READ = 1073741824
    MEM_WRITE = 2147483648
    CNT_UNINITIALIZED_DATA = 128
    __new__(value)

class cle.backends.coff.CoffSectionTableEntry
    Bases: Structure
    COFF Section Header
    Characteristics
        Structure/Union member
    Name
        Structure/Union member
    NumberOfLinenumbers
        Structure/Union member
    NumberOfRelocations
        Structure/Union member
    PointerToLinenumbers
        Structure/Union member
    PointerToRawData
        Structure/Union member
    PointerToRelocations
        Structure/Union member
    SizeOfRawData
        Structure/Union member
    VirtualAddress
        Structure/Union member
    VirtualSize
        Structure/Union member

class cle.backends.coff.IMAGE_SYM_CLASS
    Bases: IntEnum
    Symbol Storage Class
    EXTERNAL = 2
    STATIC = 3
    LABEL = 6
```

**FUNCTION = 101**

**\_\_new\_\_(value)**

**class** cle.backends.coff.CoffSymbolTableEntry

Bases: [Structure](#)

COFF Symbol Table Entry

**Name**

Structure/Union member

**NumberOfAuxSymbols**

Structure/Union member

**SectionNumber**

Structure/Union member

**StorageClass**

Structure/Union member

**Type**

Structure/Union member

**Value**

Structure/Union member

**class** cle.backends.coff.IMAGE\_REL\_I386

Bases: [IntEnum](#)

i386 Relocation Types

**DIR32 = 6**

**DIR32NB = 7**

**REL32 = 20**

**SECTION = 10**

**SECREL = 11**

**\_\_new\_\_(value)**

**class** cle.backends.coff.IMAGE\_REL\_AMD64

Bases: [IntEnum](#)

AMD64 Relocation Types

**ADDR64 = 1**

**ADDR32NB = 3**

**REL32 = 4**

**SECTION = 10**

**SECREL = 11**

**\_\_new\_\_(value)**

---

```

class cle.backends.coff.CoffRelocationTableEntry
    Bases: Structure
    COFF Relocations
    SymbolTableIndex
        Structure/Union member
    Type
        Structure/Union member
    VirtualAddress
        Structure/Union member
class cle.backends.coff.CoffParser
    Bases: object
    Parses COFF object files.
    header: CoffFileHeader
    sections: list[CoffSectionTableEntry]
    relocations: list[list[CoffRelocationTableEntry]]
    symbols: list[CoffSymbolTableEntry]
    idx_to_symbol_name: dict[int, str]
    symbol_name_to_idx: dict[str, int]
    __init__(data: bytes)
        Parameters
            data (bytes)
    data: bytes
    get_symbol_name(symbol_idx: int, true_name: bool = False) → str
        Return type
            str
        Parameters
            • symbol_idx (int)
            • true_name (bool)
    get_section_name(section_idx: int) → str
        Return type
            str
        Parameters
            section_idx (int)
class cle.backends.coff.CoffSection
    Bases: Section
    Section of the COFF object.

```

`__init__` (*name: str, file\_offset: int, file\_size: int, virtual\_addr: int, virtual\_size: int, coff\_sec: CoffSectionTableEntry*)

#### Parameters

- **name** (*str*) – The name of the section
- **offset** (*int*) – The offset into the binary file this section begins
- **vaddr** (*int*) – The address in virtual memory this section begins
- **size** (*int*) – How large this section is
- **file\_offset** (*int*)
- **file\_size** (*int*)
- **virtual\_addr** (*int*)
- **virtual\_size** (*int*)
- **coff\_sec** (*CoffSectionTableEntry*)

#### property **is\_readable**

Whether this section has read permissions

#### property **is\_writable**

Whether this section has write permissions

#### property **is\_executable**

Whether this section has execute permissions

#### property **only\_contains\_uninitialized\_data**

Whether this section is initialized to zero after the executable is loaded.

#### class `cle.backends.coff.CoffRelocation`

Bases: *Relocation*

Relocation for a COFF object.

**PACK\_FORMAT** = '<i'

#### **relocate()**

Applies this relocation. Will make changes to the memory object of the object it came from.

This implementation is a generic version that can be overridden in subclasses.

#### class `cle.backends.coff.CoffRelocationREL32`

Bases: *CoffRelocation*

Relocation for IMAGE\_REL\_\*\_REL32

#### property **value**

#### class `cle.backends.coff.CoffRelocationDIR32`

Bases: *CoffRelocation*

Relocation for IMAGE\_REL\_\*\_DIR32

#### property **value**

---

```
class cle.backends.coff.CoffRelocationDIR32NB
```

```
    Bases: CoffRelocation
```

```
    Relocation for IMAGE_REL_*_DIR32
```

```
    property value
```

```
class cle.backends.coff.CoffRelocationADDR32NB
```

```
    Bases: CoffRelocation
```

```
    Relocation for IMAGE_REL_AMD64_ADDR32NB
```

```
    PACK_FORMAT = '<I'
```

```
    property value: int
```

```
class cle.backends.coff.CoffRelocationADDR64
```

```
    Bases: CoffRelocation
```

```
    Relocation for IMAGE_REL_AMD64_ADDR64
```

```
    PACK_FORMAT = '<Q'
```

```
    property value
```

```
class cle.backends.coff.CoffRelocationSECTION
```

```
    Bases: CoffRelocation
```

```
    Relocation for IMAGE_REL_*_SECTION
```

```
    PACK_FORMAT = '<H'
```

```
    property value
```

```
class cle.backends.coff.CoffRelocationSECREL
```

```
    Bases: CoffRelocation
```

```
    Relocation for IMAGE_REL_*_SECREL
```

```
    PACK_FORMAT = '<I'
```

```
    property value
```

```
class cle.backends.coff.Coff
```

```
    Bases: Backend
```

```
    COFF object loader.
```

```
    is_default = True
```

```
    __init__(*args, **kwargs)
```

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

```
    classmethod is_compatible(stream)
```

```
        Determine quickly whether this backend can load an object from this stream
```

**get\_symbol**(*name*: *str*, *produce\_extern\_symbols*: *bool* = *False*) → *Symbol* | *None*

Stub function. Implement to find the symbol with name *name*.

**Return type**

*Symbol* | *None*

**Parameters**

- **name** (*str*)
- **produce\_extern\_symbols** (*bool*)

## 2.3.7 CGC

**class** cle.backends.cgc.CGC

Bases: *ELF*

Backend to support the CGC elf format used by the Cyber Grand Challenge competition.

See : [https://github.com/CyberGrandChallenge/libcgcef/blob/master/cgc\\_executable\\_format.md](https://github.com/CyberGrandChallenge/libcgcef/blob/master/cgc_executable_format.md)

**is\_default** = **True**

**\_\_init\_\_**(*binary*, *binary\_stream*, *\*args*, *\*\*kwargs*)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**supported\_filetypes** = ['cgc']

**class** cle.backends.cgc.BackedCGC

Bases: *CGC*

This is a backend for CGC executables that allows user provide a memory backer and a register backer as the initial state of the running binary.

**is\_default** = **True**

**\_\_init\_\_**(*\*args*, *memory\_backer*=*None*, *register\_backer*=*None*, *writes\_backer*=*None*, *permissions\_map*=*None*, *current\_allocation\_base*=*None*, *\*\*kwargs*)

**Parameters**

- **path** – File path to CGC executable.
- **memory\_backer** – A dict of memory content, with beginning address of each segment as key and actual memory content as data.
- **register\_backer** – A dict of all register contents. EIP will be used as the entry point of this executable.
- **permissions\_map** – A dict of memory region to permission flags
- **current\_allocation\_base** – An integer representing the current address of the top of the CGC heap.

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**property threads**

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This property should contain a list of names for these threads, which should be unique.

**thread\_registers**(*thread=None*)

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This method should return the register file for a given thread (as named in `Backend.threads`) as a dict mapping register names (as seen in `archinfo`) to numbers. If the thread is not specified, it should return the context for a “default” thread. If there are no threads, it should return an empty dict.

**class** `cle.backends.cgc.cgc.CGC`

Bases: *ELF*

Backend to support the CGC elf format used by the Cyber Grand Challenge competition.

See : [https://github.com/CyberGrandChallenge/libcgccef/blob/master/cgc\\_executable\\_format.md](https://github.com/CyberGrandChallenge/libcgccef/blob/master/cgc_executable_format.md)

**is\_default = True**

**\_\_init\_\_**(*binary, binary\_stream, \*args, \*\*kwargs*)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**supported\_filetypes = ['cgc']**

**class** `cle.backends.cgc.backedcgc.FakeSegment`

Bases: *Segment*

**\_\_init\_\_**(*start, size*)

**class** `cle.backends.cgc.backedcgc.BackedCGC`

Bases: *CGC*

This is a backend for CGC executables that allows user provide a memory backer and a register backer as the initial state of the running binary.

**is\_default = True**

**\_\_init\_\_**(*\*args, memory\_backer=None, register\_backer=None, writes\_backer=None, permissions\_map=None, current\_allocation\_base=None, \*\*kwargs*)

**Parameters**

- **path** – File path to CGC executable.
- **memory\_backer** – A dict of memory content, with beginning address of each segment as key and actual memory content as data.
- **register\_backer** – A dict of all register contents. EIP will be used as the entry point of this executable.

- **permissions\_map** – A dict of memory region to permission flags
- **current\_allocation\_base** – An integer representing the current address of the top of the CGC heap.

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**property threads**

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This property should contain a list of names for these threads, which should be unique.

**thread\_registers**(*thread=None*)

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This method should return the register file for a given thread (as named in `Backend.threads`) as a dict mapping register names (as seen in `archinfo`) to numbers. If the thread is not specified, it should return the context for a “default” thread. If there are no threads, it should return an empty dict.

### 2.3.8 IHex

**class** `cle.backends.ihex.Hex`

Bases: *Backend*

A loader for Intel Hex Objects See [https://en.wikipedia.org/wiki/Intel\\_HEX](https://en.wikipedia.org/wiki/Intel_HEX)

**is\_default** = `True`

**static parse\_record**(*line*)

**static coalesce\_regions**(*regions*)

**\_\_init\_\_**(\*args, *ignore\_missing\_arch: bool = False*, *extra\_regions: list[dict[str, int | bool]] = None*, \*\*kwargs)

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call `close`.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable
- **ignore\_missing\_arch** (*bool*)
- **extra\_regions** (*list[dict[str, int | bool]]*)

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

### 2.3.9 Java

Constant values for lifecycle of Apk.

**class** `cle.backends.java.apk.Apk`

Bases: *Soot*

Backend for lifting Apk’s to Soot.

**is\_default** = `True`

`__init__(apk_path, binary_stream, entry_point=None, entry_point_params=(), android_sdk=None, supported_jni_archs=None, jni_libs=None, jni_libs_ld_path=None, **options)`

#### Parameters

- **apk\_path** – Path to APK.
- **android\_sdk** – Path to Android SDK folder (e.g. “/home/angr/android/platforms”)

The following parameters are optional

#### Parameters

- **entry\_point** – Fully qualified name of method that should be used as the entry point.
- **supported\_jni\_archs** – List of supported JNI architectures (ABIs) in descending order of preference.
- **jni\_libs** – Name(s) of JNI libs to load (if any). If not specified, we try to extract JNI libs from the APK.
- **jni\_libs\_ld\_path** – Path(s) where to find libs defined by param `jni_libs`. Note: Directory of the APK is added by default.

`get_callbacks(class_name: str, callback_names: list[str]) → list[None]`

Get callback methods from the name of callback methods.

#### Parameters

- **class\_name** (`str`) – Name of the class.
- **callback\_names** (`list[str]`) – Name list of the callbacks.

#### Returns

The method object which is callback.

#### Return type

`list[None]`

`static is_compatible(stream)`

Determine quickly whether this backend can load an object from this stream

`class cle.backends.java.jar.Jar`

Bases: `Soot`

Backend for lifting JARs to Soot.

`is_default = True`

`__init__(jar_path, binary_stream, entry_point=None, entry_point_params=('java.lang.String[]'), jni_libs=None, jni_libs_ld_path=None, **kwargs)`

#### Parameters

**jar\_path** – Path to JAR.

The following parameters are optional

#### Parameters

- **entry\_point** – Fully qualified name of method that should be used as the entry point. If not specified, we try to parse it from the manifest.
- **additional\_jars** – Additional JARs.
- **additional\_jar\_roots** – Additional JAR roots.

- **jni\_libs** – Name(s) of JNI libs to load (if any).
- **jni\_libs\_ld\_path** – Path(s) where to find libs defined by param `jni_libs`. Note: Directory of the JAR is added by default.

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**get\_manifest**(*binary\_path=None*)

Load the MANIFEST.MF file

**Returns**

A dict of meta info

**Return type**

dict

**class** `cle.backends.java.soot.Soot`

Bases: *Backend*

The basis backend for lifting and loading bytecode from JARs and APKs to Soot IR.

Note that `self.min_addr` will be 0 and `self.max_addr` will be 1. Hopefully no other object will be mapped at address 0.

**\_\_init\_\_**(*\*args, entry\_point=None, entry\_point\_params=(), input\_format=None, additional\_jars=None, additional\_jar\_roots=None, jni\_libs\_ld\_path=None, jni\_libs=None, android\_sdk=None, \*\*kwargs*)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call `close`.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**property** `max_addr`

This returns the highest virtual address contained in any loaded segment of the binary.

**property** `entry`

**property** `classes`

**get\_soot\_class**(*cls\_name, none\_if\_missing=False*)

Get Soot class object.

**Parameters**

**cls\_name** (*str*) – Name of the class.

**Returns**

The class object.

**Return type**

`pysoot.soot.SootClass`

**get\_soot\_method**(*thing, class\_name=None, params=(), none\_if\_missing=False*)

Get Soot method object.

**Parameters**

- **thing** – Descriptor or the method, or name of the method.

- **class\_name** (*str*) – Name of the class. If not specified, class name can be parsed from `method_name`.

**Returns**

Soot method that satisfy the criteria.

**property main\_methods**

Find all Main methods in this binary.

**Returns**

All main methods in each class.

**Return type**

iterator

**static is\_zip\_archive**(*stream*)

### 2.3.10 Minidump

**exception** `cle.backends.minidump.MinidumpMissingStreamError`

Bases: `Exception`

**\_\_init\_\_**(*stream, message=None*)

**class** `cle.backends.minidump.Minidump`

Bases: `Backend`

**is\_default** = `True`

**\_\_init\_\_**(\**args*, \*\**kwargs*)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call `close`.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**close**()

**static is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**property threads**

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This property should contain a list of names for these threads, which should be unique.

**thread\_registers**(*thread=None*)

If this backend represents a dump of a running program, it may contain one or more thread contexts, i.e. register files. This method should return the register file for a given thread (as named in `Backend.threads`) as a dict mapping register names (as seen in `archinfo`) to numbers. If the thread is not specified, it should return the context for a “default” thread. If there are no threads, it should return an empty dict.

**get\_thread\_registers\_by\_id**(*thread\_id*)

### 2.3.11 Static Archive

**class** cle.backends.static\_archive.**StaticArchive**

Bases: *Backend*

**classmethod** **is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**is\_default** = **True**

**is\_outer** = **True**

**\_\_init\_\_**(\*args, \*\*kwargs)

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

### 2.3.12 UEFI Firmware

**exception** cle.backends.uefi\_firmware.**UefiDriverLoadError**

Bases: *Exception*

This error is raised (and caught internally) if the data contained in the UEFI entity tree doesn't make sense.

**class** cle.backends.uefi\_firmware.**UefiFirmware**

Bases: *Backend*

A UEFI firmware blob loader. Support is provided by the uefi\_firmware package.

**is\_default** = **True**

**classmethod** **is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**\_\_init\_\_**(\*args, \*\*kwargs) → *None*

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

#### Return type

*None*

**class** cle.backends.uefi\_firmware.**UefiModulePending**

Bases: *object*

A worklist entry for the UEFI firmware loader.

**name**: *str* | *None* = *None*

**pe\_image**: *bytes* | *None* = *None*

**te\_image:** `bytes | None = None`

**build**(parent: `UefiFirmware`, guid: `UUID`) → `UefiModuleMixin`

**Return type**

`UefiModuleMixin`

**Parameters**

- **parent** (`UefiFirmware`)
- **guid** (`UUID`)

**\_\_init\_\_**(name: `str | None = None`, pe\_image: `bytes | None = None`, te\_image: `bytes | None = None`) → `None`

**Parameters**

- **name** (`str | None`)
- **pe\_image** (`bytes | None`)
- **te\_image** (`bytes | None`)

**Return type**

`None`

**class** `cle.backends.uefi_firmware.UefiModuleMixin`

Bases: `Backend`

A mixin to make other kinds of backends load as UEFI modules.

**\_\_init\_\_**(\*args, guid: `UUID`, name: `str | None`, \*\*kwargs)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable
- **guid** (`UUID`)
- **name** (`str | None`)

**class** `cle.backends.uefi_firmware.UefiPE`

Bases: `UefiModuleMixin`, `PE`

A PE file contained in a UEFI image.

**class** `cle.backends.uefi_firmware.UefiTE`

Bases: `UefiModuleMixin`, `TE`

A TE file contained in a UEFI image.

**class** `cle.backends.te.HeaderType`

Bases: `tuple`

`HeaderType`(signature, machine, number\_of\_sections, subsystem, stripped\_size, address\_of\_entry\_point, base\_of\_code, image\_base, data\_directory\_0\_virtual\_address, data\_directory\_0\_size, data\_directory\_1\_virtual\_address, data\_directory\_1\_size)

```
static __new__(_cls, signature, machine, number_of_sections, subsystem, stripped_size,  
address_of_entry_point, base_of_code, image_base, data_directory_0_virtual_address,  
data_directory_0_size, data_directory_1_virtual_address, data_directory_1_size)
```

Create new instance of HeaderType(signature, machine, number\_of\_sections, subsystem, stripped\_size, address\_of\_entry\_point, base\_of\_code, image\_base, data\_directory\_0\_virtual\_address, data\_directory\_0\_size, data\_directory\_1\_virtual\_address, data\_directory\_1\_size)

**address\_of\_entry\_point**

Alias for field number 5

**base\_of\_code**

Alias for field number 6

**data\_directory\_0\_size**

Alias for field number 9

**data\_directory\_0\_virtual\_address**

Alias for field number 8

**data\_directory\_1\_size**

Alias for field number 11

**data\_directory\_1\_virtual\_address**

Alias for field number 10

**image\_base**

Alias for field number 7

**machine**

Alias for field number 1

**number\_of\_sections**

Alias for field number 2

**signature**

Alias for field number 0

**stripped\_size**

Alias for field number 4

**subsystem**

Alias for field number 3

**class** cle.backends.te.**SectionHeaderType**

Bases: `tuple`

SectionHeaderType(section\_name, physical\_address\_virtual\_size, virtual\_address, size\_of\_raw\_data, pointer\_to\_raw\_data, pointer\_to\_relocations, pointer\_to\_line\_numbers, number\_of\_relocations, number\_of\_line\_numbers, characteristics)

```
static __new__(_cls, section_name, physical_address_virtual_size, virtual_address, size_of_raw_data,  
pointer_to_raw_data, pointer_to_relocations, pointer_to_line_numbers,  
number_of_relocations, number_of_line_numbers, characteristics)
```

Create new instance of SectionHeaderType(section\_name, physical\_address\_virtual\_size, virtual\_address, size\_of\_raw\_data, pointer\_to\_raw\_data, pointer\_to\_relocations, pointer\_to\_line\_numbers, number\_of\_relocations, number\_of\_line\_numbers, characteristics)

**characteristics**

Alias for field number 9

**number\_of\_line\_numbers**

Alias for field number 8

**number\_of\_relocations**

Alias for field number 7

**physical\_address\_virtual\_size**

Alias for field number 1

**pointer\_to\_line\_numbers**

Alias for field number 6

**pointer\_to\_raw\_data**

Alias for field number 4

**pointer\_to\_relocations**

Alias for field number 5

**section\_name**

Alias for field number 0

**size\_of\_raw\_data**

Alias for field number 3

**virtual\_address**

Alias for field number 2

**class** cle.backends.te.**TE**

Bases: *Backend*

A “Terse Executable” format image, commonly used as part of UEFI firmware drivers.

**is\_default** = True

**classmethod** **is\_compatible**(*stream*)

Determine quickly whether this backend can load an object from this stream

**\_\_init\_\_**(\*args, \*\*kwargs)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**2.3.13 Xbox Executable****class** cle.backends.xbe.**XBESection**

Bases: *Section*

**\_\_init\_\_**(*name, file\_offset, file\_size, virtual\_addr, virtual\_size, xbe\_sec*)

**Parameters**

- **name** (*str*) – The name of the section

- **offset** (*int*) – The offset into the binary file this section begins
- **vaddr** (*int*) – The address in virtual memory this section begins
- **size** (*int*) – How large this section is

**property is\_readable**

Whether this section has read permissions

**property is\_writable**

Whether this section has write permissions

**property is\_executable**

Whether this section has execute permissions

**property only\_contains\_uninitialized\_data**

We load every section in, they're all initialized

**class cle.backends.xbe.XBE**

Bases: *Backend*

The main loader class for statically loading XBE executables.

**is\_default = True**

**\_\_init\_\_** (*\*args, \*\*kwargs*)

**Parameters**

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

**close**()

**static is\_compatible** (*stream*)

Determine quickly whether this backend can load an object from this stream

**property min\_addr**

This returns the lowest virtual address contained in any loaded segment of the binary.

**property max\_addr**

This returns the highest virtual address contained in any loaded segment of the binary.

**classmethod check\_compatibility** (*spec, obj*)

Performs a minimal static load of *spec* and returns whether it's compatible with *other\_obj*

## 2.4 Relocations

CLE's loader implements program relocations. If you would like to add support for more relocations, you can do so by subclassing the `Relocation` class and overriding any relevant methods or properties. Then, add or uncomment the appropriate line in the `relocations_table` dict at the bottom of the file. Look at the existing versions for details.

**class cle.backends.relocation.Relocation**

Bases: `object`

A representation of a relocation in a binary file. Smart enough to relocate itself.

## Variables

- **owner** – The binary this relocation was originally found in, as a cle object
- **symbol** – The Symbol object this relocation refers to
- **relative\_addr** – The address in owner this relocation would like to write to
- **resolvedby** (*Symbol* | *None*) – If the symbol this relocation refers to is an import symbol and that import has been resolved, this attribute holds the symbol from a different binary that was used to resolve the import.
- **resolved** (*bool*) – Whether the application of this relocation was successful

**\_\_init\_\_** (*owner*: *Backend*, *symbol*: *Symbol* | *None*, *relative\_addr*: *int*)

## Parameters

- **owner** (*Backend*)
- **symbol** (*Symbol* | *None*)
- **relative\_addr** (*int*)

**resolvedby**: *Symbol* | *None*

**resolved**: *bool*

**AUTO\_HANDLE\_NONE** = *False*

**resolve\_symbol** (*solist*: *list*[*Any*], *thumb*=*False*, *extern\_object*=*None*, *\*\*kwargs*)

## Parameters

**solist** (*list*[*Any*])

**resolve** (*obj*, *extern\_object*=*None*)

**property rebased\_addr**

The address in the global memory space this relocation would like to write to

**property linked\_addr**

**property dest\_addr**

**property value**: *int*

**relocate**()

Applies this relocation. Will make changes to the memory object of the object it came from.

This implementation is a generic version that can be overridden in subclasses.

**property owner\_obj**

`cle.backends.elf.relocation.get_relocation(arch, r_type)`

`class cle.backends.elf.relocation.elfreloc.ELFReloc`

Bases: *Relocation*

**\_\_init\_\_** (*owner*, *symbol*, *relative\_addr*, *addend*=*None*)

**property addend**

**property value**

```
class cle.backends.elf.relocation.generic.GenericTLSOffsetReloc
```

```
    Bases: ELFReloc
```

```
    property value
```

```
    resolve_symbol(solist, **kwargs)
```

```
class cle.backends.elf.relocation.generic.GenericTLSOffsetReloc
```

```
    Bases: ELFReloc
```

```
    AUTO_HANDLE_NONE = True
```

```
    relocate()
```

```
        Applies this relocation. Will make changes to the memory object of the object it came from.
```

```
        This implementation is a generic version that can be overridden in subclasses.
```

```
class cle.backends.elf.relocation.generic.GenericTLSDescriptorReloc
```

```
    Bases: ELFReloc
```

```
    RESOLVER_ADDR: int = NotImplemented
```

```
    AUTO_HANDLE_NONE = True
```

```
    relocate()
```

```
        Applies this relocation. Will make changes to the memory object of the object it came from.
```

```
        This implementation is a generic version that can be overridden in subclasses.
```

```
class cle.backends.elf.relocation.generic.GenericTLSModIdReloc
```

```
    Bases: ELFReloc
```

```
    AUTO_HANDLE_NONE = True
```

```
    relocate()
```

```
        Applies this relocation. Will make changes to the memory object of the object it came from.
```

```
        This implementation is a generic version that can be overridden in subclasses.
```

```
class cle.backends.elf.relocation.generic.GenericIRelativeReloc
```

```
    Bases: ELFReloc
```

```
    AUTO_HANDLE_NONE = True
```

```
    relocate()
```

```
        Applies this relocation. Will make changes to the memory object of the object it came from.
```

```
        This implementation is a generic version that can be overridden in subclasses.
```

```
class cle.backends.elf.relocation.generic.GenericAbsoluteAddendReloc
```

```
    Bases: ELFReloc
```

```
    property value
```

```
class cle.backends.elf.relocation.generic.GenericPCRelativeAddendReloc
```

```
    Bases: ELFReloc
```

```
    property value
```

```
class cle.backends.elf.relocation.generic.GenericJumpslotReloc
```

```
    Bases: ELFReloc
```

property value

```
class cle.backends.elf.relocation.generic.GenericRelativeReloc
```

Bases: *ELFReloc*

```
AUTO_HANDLE_NONE = True
```

property value

```
class cle.backends.elf.relocation.generic.GenericAbsoluteReloc
```

Bases: *ELFReloc*

property value

```
class cle.backends.elf.relocation.generic.GenericCopyReloc
```

Bases: *ELFReloc*

```
resolve_symbol(solist, **kwargs)
```

```
relocate()
```

Applies this relocation. Will make changes to the memory object of the object it came from.

This implementation is a generic version that can be overridden in subclasses.

```
class cle.backends.elf.relocation.generic.MipsGlobalReloc
```

Bases: *GenericAbsoluteReloc*

```
class cle.backends.elf.relocation.generic.MipsLocalReloc
```

Bases: *ELFReloc*

```
AUTO_HANDLE_NONE = True
```

```
resolve_symbol(solist, **kwargs)
```

```
relocate()
```

Applies this relocation. Will make changes to the memory object of the object it came from.

This implementation is a generic version that can be overridden in subclasses.

```
class cle.backends.elf.relocation.generic.RelocTruncate32Mixin
```

Bases: *object*

A mix-in class for relocations that cover a 32-bit field regardless of the architecture's address word length.

```
check_zero_extend = False
```

```
check_sign_extend = False
```

```
relocate()
```

```
class cle.backends.elf.relocation.generic.RelocGOTMixin
```

Bases: *object*

A mix-in class which will cause the symbol to be resolved to a pointer to the symbol instead of the symbol

```
resolve(symbol, extern_object=None)
```

Relocation types for PowerPC 32-bit architecture.

Reference: [http://refspecs.linux-foundation.org/elf/elfspec\\_ppc.pdf](http://refspecs.linux-foundation.org/elf/elfspec_ppc.pdf) page 4-18

Only relocations 1-37 are described in the document. The rest are from the GNU binutils source code. See `include/elf/ppc.h` in the binutils source code. Relocation types for PPC64.

Reference: <http://refspecs.linuxfoundation.org/ELF/ppc64/PPC-elf64abi-1.9.pdf> pages 57-59 Relocation types for i386.

Reference: <https://github.com/hjl-tools/x86-psABI/wiki/intel386-psABI-1.1.pdf> page 36 Relocations for amd64/x86\_64

Reference: <https://gitlab.com/x86-psABIs/x86-64-ABI/-/jobs/artifacts/master/raw/x86-64-ABI/abi.pdf?job=build> page 73 Relocation types for MIPS 32-bit.

Reference: <https://refspecs.linuxfoundation.org/elf/mipsabi.pdf> page 4-19

The main document is old and does not contain all the relocation types. I could not find a more recent document, so I had to rely on the source code of GNU binutils for all relocations that are not in the main document. See `include/elf/mips.h` in the binutils source code. Relocation types for ARM.

Reference: <https://github.com/ARM-software/abi-aa/blob/main/aaelf32/aaelf32.rst#relocation-codes> Relocations for AARCH64

Reference: <https://github.com/ARM-software/abi-aa/blob/main/aaelf64/aaelf64.rst#relocation> Relocation types for the S390X architecture.

Reference: [https://github.com/IBM/s390x-abi/releases/download/v1.6.1/lzsabi\\_s390x.pdf](https://github.com/IBM/s390x-abi/releases/download/v1.6.1/lzsabi_s390x.pdf) pages 51-52

`cle.backends.pe.relocation.get_relocation(arch, r_type)`

**class** `cle.backends.pe.relocation.pereloc.PEReloc`

Bases: *Relocation*

**AUTO\_HANDLE\_NONE** = **True**

**\_\_init\_\_**(owner, symbol, addr, resolvewith=None)

**resolve\_symbol**(solist, bypass\_compatibility=False, extern\_object=None, \*\*kwargs)

**relocate**()

Applies this relocation. Will make changes to the memory object of the object it came from.

This implementation is a generic version that can be overridden in subclasses.

**property value**

**property is\_base\_reloc**

These relocations are ignored by the linker if the executable is loaded at its preferred base address. There is no associated symbol with base relocations.

**property is\_import**

**class** `cle.backends.pe.relocation.generic.DllImport`

Bases: *PEReloc*

There's nothing special to be done for DLL imports but this class provides a unique name to the relocation type.

**class** `cle.backends.pe.relocation.generic.IMAGE_REL_BASED_ABSOLUTE`

Bases: *PEReloc*

**relocate**()

Applies this relocation. Will make changes to the memory object of the object it came from.

This implementation is a generic version that can be overridden in subclasses.

---

```
class cle.backends.pe.relocation.generic.IMAGE_REL_BASED_HIGHADJ
```

```
Bases: PEReloc
```

```
__init__(owner, addr, next_rva)
```

```
property value
```

In all the other cases, we can ignore the relocation difference part of the calculation because we simply use `to_mvna()` to get our rebased address. In this case, however, we have to adjust the un-rebased address first.

```
class cle.backends.pe.relocation.generic.IMAGE_REL_BASED_HIGHLOW
```

```
Bases: PEReloc
```

```
property value
```

```
class cle.backends.pe.relocation.generic.IMAGE_REL_BASED_DIR64
```

```
Bases: PEReloc
```

```
property value
```

```
class cle.backends.pe.relocation.generic.IMAGE_REL_BASED_HIGH
```

```
Bases: PEReloc
```

```
property value
```

```
class cle.backends.pe.relocation.generic.IMAGE_REL_BASED_LOW
```

```
Bases: PEReloc
```

```
property value
```

## 2.5 Thread-local storage

```
class cle.backends.tls.ThreadManager
```

```
Bases: object
```

This class tracks what data is thread-local and can generate thread initialization images

Most of the heavy lifting will be handled in a subclass

```
__init__(loader, arch, max_modules=256)
```

```
register_object(obj)
```

```
static initialization_image(obj) → bytes | None
```

**Return type**

```
bytes | None
```

```
new_thread(insert=True)
```

```
class cle.backends.tls.InternalTLSRelocation
```

```
Bases: Relocation
```

```
AUTO_HANDLE_NONE = True
```

```
__init__(val, offset, owner)
```

```
property value
```

```
class cle.backends.tls.TLSObject
```

Bases: *Backend*

```
__init__(loader, arch)
```

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

```
class cle.backends.tls.ELFThreadManager
```

Bases: *ThreadManager*

```
__init__(*args, **kwargs)
```

```
register_object(obj)
```

```
class cle.backends.tls.ELFCoreThreadManager
```

Bases: *ThreadManager*

```
__init__(loader, arch, **kwargs)
```

```
new_thread(insert=False)
```

```
register_object(obj)
```

```
class cle.backends.tls.PEThreadManager
```

Bases: *ThreadManager*

```
register_object(obj)
```

```
class cle.backends.tls.MinidumpThreadManager
```

Bases: *ThreadManager*

```
__init__(loader, arch, **kwargs)
```

```
new_thread(insert=False)
```

```
register_object(obj)
```

```
class cle.backends.tls.tls_object.ThreadManager
```

Bases: *object*

This class tracks what data is thread-local and can generate thread initialization images

Most of the heavy lifting will be handled in a subclass

```
__init__(loader, arch, max_modules=256)
```

```
register_object(obj)
```

```
static initialization_image(obj) → bytes | None
```

#### Return type

bytes | None

```
new_thread(insert=True)
```

```
class cle.backends.tls.tls_object.InternalTLSRelocation
```

Bases: *Relocation*

```
AUTO_HANDLE_NONE = True
```

```
__init__(val, offset, owner)
```

property value

```
class cle.backends.tls.tls_object.TLSObject
```

Bases: *Backend*

```
__init__(loader, arch)
```

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable

This module is used when parsing the Thread Local Storage of an ELF binary. It heavily uses the TLSArchInfo named-tuple from archinfo.

ELF TLS is implemented based on the following documents:

- <https://www.uclibc.org/docs/tls.pdf>
- <https://www.uclibc.org/docs/tls-ppc.txt>
- <https://www.uclibc.org/docs/tls-ppc64.txt>
- <https://www.linux-mips.org/wiki/NPTL>

```
cle.backends.tls.elf_tls.roundup(val, to=16)
```

```
class cle.backends.tls.elf_tls.ELFThreadManager
```

Bases: *ThreadManager*

```
__init__(*args, **kwargs)
```

```
register_object(obj)
```

```
class cle.backends.tls.elf_tls.ELFTLSObject
```

Bases: *TLSObject*

```
__init__(thread_manager: ELFThreadManager)
```

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call close.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable
- **thread\_manager** (*ELFThreadManager*)

property **thread\_pointer**

The thread pointer. This is a technical term that refers to a specific location in the TLS segment.

**property user\_thread\_pointer**

The thread pointer that is exported to the user

**property max\_addr**

This returns the highest virtual address contained in any loaded segment of the binary.

**get\_addr**(*module\_id*, *offset*)

basically `__tls_get_addr`.

**class** `cle.backends.tls.elf_tls.ELFTLSObjectV1`

Bases: *ELFTLSObject*

**class** `cle.backends.tls.elf_tls.ELFTLSObjectV2`

Bases: *ELFTLSObject*

**class** `cle.backends.tls.pe_tls.PEThreadManager`

Bases: *ThreadManager*

**register\_object**(*obj*)**class** `cle.backends.tls.pe_tls.PETLSObject`

Bases: *TLSObject*

This class is used when parsing the Thread Local Storage of a PE binary. It represents both the TLS array and the TLS data area for a specific thread.

In memory the PETLSObj is laid out as follows:

```
+-----+-----+
| TLS array           | TLS data area           |
+-----+-----+
```

A more detailed description of the TLS array and TLS data areas is given below.

**TLS array**

The TLS array is an array of addresses that points into the TLS data area. In memory it is laid out as follows:

```
+-----+-----+-----+-----+
| address | address | ... | address |
+-----+-----+-----+-----+
| index = 0 | index = 1 |     | index = n |
+-----+-----+-----+-----+
```

The size of each address is architecture independent (e.g. on X86 it is 4 bytes). The number of addresses in the TLS array is equal to the number of modules that contain TLS data. At load time (i.e. in the `finalize` method), each module is assigned an index into the TLS array. The address of this module's TLS data area is then stored at this location in the array.

**TLS data area**

The TLS data area directly follows the TLS array and contains the actual TLS data for each module. In memory it is laid out as follows:

```
+-----+-----+-----+-----+
| TLS data | zero fill | TLS data | zero fill | ... |
+-----+-----+-----+-----+
|         module a         |         module b         | ... |
+-----+-----+-----+-----+
```

The size of each module's TLS data area is variable and can be found in the module's `tls_data_size` property. The same applies to the zero fill. At load time (i.e. in the `finalize` method), the initial TLS data values are copied into the TLS data area. Because a TLS index is also assigned to each module, we can access a module's TLS data area using this index into the TLS array to get the start address of the TLS data.

`__init__` (*thread\_manager*: `PThreadManager`)

#### Parameters

- **binary** – The path to the binary to load
- **binary\_stream** – The open stream to this binary. The reference to this will be held until you call `close`.
- **is\_main\_bin** – Whether this binary should be loaded as the main executable
- **thread\_manager** (`PThreadManager`)

`get_tls_data_addr` (*tls\_idx*)

Get the start address of a module's TLS data area via the module's TLS index.

From the PE/COFF spec:

The code uses the TLS index and the TLS array location (multiplying the index by the word size and using it as an offset into the array) to get the address of the TLS data area for the given program and module.

property `max_addr`

This returns the highest virtual address contained in any loaded segment of the binary.

property `thread_pointer`

property `user_thread_pointer`

`class cle.backends.tls.elfcore_tls.ELFCoreThreadManager`

Bases: `ThreadManager`

`__init__` (*loader*, *arch*, *\*\*kwargs*)

`new_thread` (*insert=False*)

`register_object` (*obj*)

`class cle.backends.tls.elfcore_tls.ELFCoreThread`

Bases: `object`

`__init__` (*loader*, *arch*: `Arch`, *threadinfo*)

#### Parameters

**arch** (`Arch`)

property `dtv`

`get_addr` (*module\_id*, *offset*)

basically `__tls_get_addr`.

`class cle.backends.tls.minidump_tls.MinidumpThreadManager`

Bases: `ThreadManager`

`__init__` (*loader*, *arch*, *\*\*kwargs*)

`new_thread(insert=False)`

`register_object(obj)`

**class** `cle.backends.tls.minidump_tls.MinidumpThread`

Bases: `object`

`__init__(loader, arch: Arch, registers)`

**Parameters**

**arch** (*Arch*)

`get_tls_data_addr(tls_idx)`

## 2.6 Errors

**exception** `cle.errors.CLError`

Bases: `Exception`

Base class for errors raised by CLE.

**exception** `cle.errors.CLEUnknownFormatError`

Bases: `CLError`

Error raised when CLE encounters an unknown executable file format.

**exception** `cle.errors.CLEFileNotFoundError`

Bases: `CLError`

Error raised when a file does not exist.

**exception** `cle.errors.CLEInvalidBinaryError`

Bases: `CLError`

Error raised when an executable file is invalid or corrupted.

**exception** `cle.errors.CLEOperationError`

Bases: `CLError`

Error raised when a problem is encountered in the process of loading an executable.

**exception** `cle.errors.CLECompatibilityError`

Bases: `CLError`

Error raised when loading an executable that is not currently supported by CLE.

**exception** `cle.errors.CLEMemoryError`

Bases: `CLError`

Error raised when performing memory operations on unmapped addresses

## 2.7 Misc. Utilities

`cle.gdb.convert_info_sharedlibrary(fname)`

Convert a dump from gdb's `info sharedlibrary` command to a set of options that can be passed to CLE to replicate the address space from the gdb session

**Parameters**

**fname** – The name of a file containing the dump

**Returns**

A dict appropriate to be passed as **\*\*kwargs** for `anгр.Project` or `cle.Loader`

`cle.gdb.convert_info_proc_maps(fname)`

Convert a dump from gdb's `info proc maps` command to a set of options that can be passed to CLE to replicate the address space from the gdb session

**Parameters**

**fname** – The name of a file containing the dump

**Returns**

A dict appropriate to be passed as **\*\*kwargs** for `anгр.Project` or `cle.Loader`

**class** `cle.memory.ClemetryBase`

Bases: `object`

The base class of all Clemetry classes.

`__init__(arch)`

`load(addr, n)`

`store(addr, data)`

`backers(addr=0)`

`find(data, search_min=None, search_max=None) → Iterator[int]`

**Return type**

`Iterator[int]`

`unpack(addr: int, fmt: str) → tuple[Any, ...]`

Use the `struct` module to unpack the data at address `addr` with the format `fmt`.

**Return type**

`tuple[Any, ...]`

**Parameters**

- **addr** (`int`)
- **fmt** (`str`)

`unpack_word(addr: int, size: int | None = None, signed: bool = False, endness: Endness | None = None) → int`

Use the `struct` module to unpack a single integer from the address `addr`.

You may override any of the attributes of the word being extracted:

**Parameters**

- **size** (`int | None`) – The size in bytes to pack/unpack. Defaults to `wordsize` (e.g. 4 bytes on a 32 bit architecture)
- **signed** (`bool`) – Whether the data should be extracted signed/unsigned. Default unsigned
- **endness** (`Endness | None`) – The endian to use in packing/unpacking. Defaults to memory endness
- **addr** (`int`)

**Return type**

`int`

**load\_null\_terminated\_bytes**(*addr*: int, *max\_size*: int = 4096) → bytes

Load a null-terminated string from memory at address *addr* with a maximum size of *max\_size*. Useful

**Return type**

bytes

**Parameters**

- **addr** (int)
- **max\_size** (int)

**pack**(*addr*: int, *fmt*: str, \**data*)

Use the `struct` module to pack *data* into memory at address *addr* with the format *fmt*.

**Parameters**

- **addr** (int)
- **fmt** (str)

**pack\_word**(*addr*: int, *data*: int, *size*: int | None = None, *signed*: bool = False, *endness*: Endness | None = None)

Use the `struct` module to pack a single integer *data* into memory at the address *addr*.

You may override any of the attributes of the word being packed:

**Parameters**

- **size** (int | None) – The size in bytes to pack/unpack. Defaults to `wordsize` (e.g. 4 bytes on a 32 bit architecture)
- **signed** (bool) – Whether the data should be extracted signed/unsigned. Default unsigned
- **endness** (Endness | None) – The endian to use in packing/unpacking. Defaults to memory endness
- **addr** (int)
- **data** (int)

**read**(*nbytes*: int)

The stream-like function that reads up to a number of bytes starting from the current position and updates the current position. Use with `seek()`.

Up to *nbytes* bytes will be read, halting at the beginning of the first unmapped region encountered.

**Parameters**

**nbytes** (int)

**seek**(*value*: int)

The stream-like function that sets the “file’s” current position. Use with `read()`.

**Parameters**

**value** (int) – The position to seek to.

**tell**() → int

**Return type**

int

**close**()

**class** cle.memory.ClemoryBases: *ClemoryBase*

An object representing a memory space.

Accesses can be made with [index] notation.

**\_\_init\_\_**(*arch: Arch, root: bool = False*)**Parameters**

- **arch** (*Arch*)
- **root** (*bool*)

**consecutive:** *bool***min\_addr:** *int***max\_addr:** *int***add\_backer**(*start: int, data: bytes | bytearray | memoryview | list[int] | Clemory | mmap, overwrite: bool = False*)

Adds a backer to the memory.

**Parameters**

- **start** (*int*) – The address where the backer should be loaded.
- **data** (*bytes | bytearray | memoryview | list[int] | Clemory | mmap*) – The backer itself. Can be either a bytestring or another *Clemory*.
- **overwrite** (*bool*) – If True and the range overlaps any existing backer, the existing backer will be split up and the overlapping part will be replaced with the new backer.

**split\_backer**(*addr: int*)Ensures that *addr* is the start of a backer, if it is backed.**Parameters****addr** (*int*)**remove\_backer**(*start*)**backers**(*addr=0*) → *Iterator[tuple[int, bytearray | memoryview | mmap | list[int]]]*Iterate through each backer for this *clemory* and all its children, yielding tuples of (*start\_addr*, *backer*) where each backer is a *bytearray*.**Parameters****addr** – An optional starting address - all backers before and not including this address will be skipped.**Return type***Iterator[tuple[int, bytearray | memoryview | mmap | list[int]]]***load**(*addr, n*)Read up to *n* bytes at address *addr* in memory and return a bytes object.Reading will stop at the beginning of the first unallocated region found, or when *n* bytes have been read.**store**(*addr, data*)Write bytes from *data* at address *addr*.Note: If the store runs off the end of a backer and into unbacked space, this function will update the backer but also raise *KeyError*.

**find**(*data*, *search\_min=None*, *search\_max=None*) → `Iterator[int]`

Find all occurrences of a bytestring in memory.

**Parameters**

- **data** (*bytes*) – The bytestring to search for
- **search\_min** (*int*) – Optional: The first address to include as valid
- **search\_max** (*int*) – Optional: The last address to include as valid

**Return `Iterator[int]`**

Iterates over addresses at which the bytestring occurs

**Return type**

`Iterator[int]`

**class** `cle.memory.ClemoryView`

Bases: `ClemoryBase`

A Clemory which presents a subset of another Clemory as an address space.

**\_\_init\_\_**(*backer*, *start*, *end*, *offset=0*)

**Parameters**

- **backer** – The parent clemory to use
- **start** – The address in the parent to start at
- **end** – The address in the parent to end at (exclusive)
- **offset** – Where the address space should start in this Clemory. Default 0.

**backers**(*addr=0*)

**load**(*addr*, *n*)

**store**(*addr*, *data*)

**find**(*data*, *search\_min=None*, *search\_max=None*) → `Iterator[int]`

**Return type**

`Iterator[int]`

**class** `cle.memory.ClemoryTranslator`

Bases: `ClemoryBase`

Uses a function to translate between address spaces when accessing a child clemory. Intended to be used only as a stream object.

**\_\_init\_\_**(*backer*: `ClemoryBase`, *func*)

**Parameters**

**backer** (`ClemoryBase`)

**load**(*addr*, *n*)

**store**(*addr*, *data*)

**backers**(*addr=0*)

**find**(*data*, *search\_min=None*, *search\_max=None*) → Iterator[int]

**Return type**

Iterator[int]

**class** cle.memory.UninitializedClemory

Bases: *Clemory*

A special kind of Clemory that acts as a placeholder for uninitialized and invalid memory. This is needed for the PAGEZERO segment for MachO binaries, which is 4GB worth of memory. This does `_not_` handle data being written to it, this is only for uninitialized memory that is technically occupied but should never be accessed.

**\_\_init\_\_**(*arch*, *size*)

**max\_addr**: int

**add\_backer**(*start*, *data*, *overwrite=False*)

Adds a backer to the memory.

**Parameters**

- **start** – The address where the backer should be loaded.
- **data** – The backer itself. Can be either a bytestring or another *Clemory*.
- **overwrite** – If True and the range overlaps any existing backer, the existing backer will be split up and the overlapping part will be replaced with the new backer.

**split\_backer**(*addr*)

Ensures that *addr* is the start of a backer, if it is backed.

**remove\_backer**(*start*)

**backers**(*addr=0*)

Technically this object has no real backer. We could create a fake backer on demand, but that would be a waste of memory, and code like the function `prolog_discovery` for MachO binaries would search 4GB worth of nullbytes for a prolog, which is a waste of time. Instead we just return an empty byte array, which seems to pass the test cases. :type *addr*: :param *addr*: :return:

**load**(*addr*, *n*)

Read up to *n* bytes at address *addr* in memory and return a bytes object.

Reading will stop at the beginning of the first unallocated region found, or when *n* bytes have been read.

**store**(*addr*, *data*)

Write bytes from *data* at address *addr*.

Note: If the store runs off the end of a backer and into unbacked space, this function will update the backer but also raise `KeyError`.

**find**(*data*, *search\_min=None*, *search\_max=None*) → Iterator[int]

The memory has no value, so matter what is searched for, it won't be found.

**Return type**

Iterator[int]

**consecutive**: bool

**min\_addr**: int

**class** cle.patched\_stream.PatchedStream

Bases: `object`

An object that wraps a readable stream, performing passthroughs on seek and read operations, except to make it seem like the data has actually been patched by the given patches.

`__init__(stream, patches)`

**Parameters**

- **stream** – The stream to patch
- **patches** – A list of tuples of (addr, patch data)

`read(*args, **kwargs)`

`seek(*args, **kwargs)`

`tell()`

`close()`

**class** cle.address\_translator.AddressTranslator

Bases: `object`

`__init__(rva, owner)`

**Parameters**

- **rva** (`int`) – virtual address relative to owner’s object image base
- **owner** (`cle.Backend`) – The object owner address relates to

**classmethod** `from_lva(lva, owner)`

Loads address translator with LVA

**classmethod** `from_mva(mva, owner)`

Loads address translator with MVA

**classmethod** `from_rva(rva, owner)`

Loads address translator with RVA

**classmethod** `from_raw(raw, owner)`

Loads address translator with RAW address

**classmethod** `from_linked_va(lva, owner)`

Loads address translator with LVA

**classmethod** `from_va(mva, owner)`

Loads address translator with MVA

**classmethod** `from_mapped_va(mva, owner)`

Loads address translator with MVA

**classmethod** `from_relative_va(rva, owner)`

Loads address translator with RVA

`to_lva()`

VA -> LVA :rtype: int

**to\_mva()**

RVA -> MVA :rtype: int

**to\_rva()**

RVA -> RVA :rtype: int

**to\_raw()**

RVA -> RAW :rtype: int

**to\_linked\_va()**

VA -> LVA :rtype: int

**to\_va()**

RVA -> MVA :rtype: int

**to\_mapped\_va()**

RVA -> MVA :rtype: int

**to\_relative\_va()**

RVA -> RVA :rtype: int

**cle.address\_translator.AT**

alias of *AddressTranslator*

**cle.utils.ALIGN\_DOWN**(*base, size*)

**cle.utils.ALIGN\_UP**(*base, size*)

**cle.utils.get\_mmaped\_data**(*stream, offset, length, page\_size*)

**cle.utils.stream\_or\_path**(*obj, perms='rb'*)

**cle.utils.key\_bisect\_floor\_key**(*lst, key, lo=0, hi=None, keyfunc=<function <lambda>>*)

**cle.utils.key\_bisect\_find**(*lst, item, lo=0, hi=None, keyfunc=<function <lambda>>*)

**cle.utils.key\_bisect\_insort\_left**(*lst, item, lo=0, hi=None, keyfunc=<function <lambda>>*)

**cle.utils.key\_bisect\_insort\_right**(*lst, item, lo=0, hi=None, keyfunc=<function <lambda>>*)

**cle.utils.get\_text\_offset**(*path*)

Offset of text section in the binary.

**cle.utils.extract\_null\_terminated\_bytestr**(*data: bytes | bytearray, offset: int = 0, sentinel\_value: bytes = b'\x00'*) → bytes

Return an exclusive null-terminated sequence of bytes at *offset* in *data*.

#### Return type

bytes

#### Parameters

- **data** (*bytes | bytearray*)
- **offset** (*int*)
- **sentinel\_value** (*bytes*)



## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### C

- cle.address\_translator, 98
- cle.backends.backend, 13
- cle.backends.binja, 64
- cle.backends.blob, 65
- cle.backends.cgc, 72
- cle.backends.cgc.backedcgc, 73
- cle.backends.cgc.cgc, 73
- cle.backends.coff, 66
- cle.backends.elf.compilation\_unit, 41
- cle.backends.elf.hashtable, 39
- cle.backends.elf.lsd, 39
- cle.backends.elf.relocation, 83
- cle.backends.elf.relocation.amd64, 86
- cle.backends.elf.relocation.arm, 86
- cle.backends.elf.relocation.arm64, 86
- cle.backends.elf.relocation.elfreloc, 83
- cle.backends.elf.relocation.generic, 83
- cle.backends.elf.relocation.i386, 86
- cle.backends.elf.relocation.mips, 86
- cle.backends.elf.relocation.ppc, 85
- cle.backends.elf.relocation.ppc64, 86
- cle.backends.elf.relocation.s390x, 86
- cle.backends.elf.subprogram, 40
- cle.backends.elf.variable, 34
- cle.backends.elf.variable\_type, 35
- cle.backends.externs, 23
- cle.backends.externs.simdata, 25
- cle.backends.externs.simdata.common, 27
- cle.backends.externs.simdata.simdata, 26
- cle.backends.ihex, 74
- cle.backends.java, 76
- cle.backends.java.android\_lifecycle, 74
- cle.backends.java.apk, 74
- cle.backends.java.jar, 75
- cle.backends.java.soot, 76
- cle.backends.macho.binding, 51
- cle.backends.macho.section, 49
- cle.backends.macho.segment, 50
- cle.backends.macho.structs, 56
- cle.backends.macho.symbol, 47
- cle.backends.minidump, 77
- cle.backends.named\_region, 23
- cle.backends.pe.regions, 43
- cle.backends.pe.relocation, 86
- cle.backends.pe.relocation.arm, 87
- cle.backends.pe.relocation.generic, 86
- cle.backends.pe.relocation.mips, 87
- cle.backends.pe.relocation.pereloc, 86
- cle.backends.pe.relocation.riscv, 87
- cle.backends.pe.symbol, 43
- cle.backends.region, 21
- cle.backends.regions, 20
- cle.backends.relocation, 82
- cle.backends.static\_archive, 78
- cle.backends.symbol, 18
- cle.backends.te, 79
- cle.backends.tls, 87
- cle.backends.tls.elf\_tls, 89
- cle.backends.tls.elfcore\_tls, 91
- cle.backends.tls.minidump\_tls, 91
- cle.backends.tls.pe\_tls, 90
- cle.backends.tls.tls\_object, 88
- cle.backends.uefi\_firmware, 78
- cle.backends.xbe, 81
- cle.errors, 92
- cle.gdb, 92
- cle.memory, 93
- cle.patched\_stream, 97
- cle.utils, 99



## Symbols

- `__init__()` (*cle.Loader* method), 7
- `__init__()` (*cle.address\_translator.AddressTranslator* method), 98
- `__init__()` (*cle.backends.ELF* method), 29
- `__init__()` (*cle.backends.PE* method), 42
- `__init__()` (*cle.backends.backend.Backend* method), 15
- `__init__()` (*cle.backends.backend.ExceptionHandling* method), 14
- `__init__()` (*cle.backends.backend.FunctionHint* method), 13
- `__init__()` (*cle.backends.binja.BinjaBin* method), 64
- `__init__()` (*cle.backends.binja.BinjaSymbol* method), 64
- `__init__()` (*cle.backends.blob.Blob* method), 65
- `__init__()` (*cle.backends.cgc.BackedCGC* method), 72
- `__init__()` (*cle.backends.cgc.CGC* method), 72
- `__init__()` (*cle.backends.cgc.backedcgc.BackedCGC* method), 73
- `__init__()` (*cle.backends.cgc.backedcgc.FakeSegment* method), 73
- `__init__()` (*cle.backends.cgc.cgc.CGC* method), 73
- `__init__()` (*cle.backends.coff.Coff* method), 71
- `__init__()` (*cle.backends.coff.CoffParser* method), 69
- `__init__()` (*cle.backends.coff.CoffSection* method), 69
- `__init__()` (*cle.backends.elf.ELFCore* method), 30
- `__init__()` (*cle.backends.elf.MetaELF* method), 31
- `__init__()` (*cle.backends.elf.compilation\_unit.CompilationUnit* method), 42
- `__init__()` (*cle.backends.elf.hashtable.ELFHashTable* method), 39
- `__init__()` (*cle.backends.elf.hashtable.GNUHashTable* method), 40
- `__init__()` (*cle.backends.elf.ldda.CallSiteEntry* method), 39
- `__init__()` (*cle.backends.elf.ldda.ExceptionTableHeader* method), 39
- `__init__()` (*cle.backends.elf.ldda.LSDAExceptionTable* method), 39
- `__init__()` (*cle.backends.elf.regions.ELFSection* method), 33
- `__init__()` (*cle.backends.elf.regions.ELFSegment* method), 33
- `__init__()` (*cle.backends.elf.relocation.elfreloc.ELFReloc* method), 83
- `__init__()` (*cle.backends.elf.subprogram.LexicalBlock* method), 40
- `__init__()` (*cle.backends.elf.subprogram.Subprogram* method), 41
- `__init__()` (*cle.backends.elf.symbol.ELFSymbol* method), 32
- `__init__()` (*cle.backends.elf.symbol\_type.ELFSymbolType* method), 32
- `__init__()` (*cle.backends.elf.variable.MemoryVariable* method), 35
- `__init__()` (*cle.backends.elf.variable.RegisterVariable* method), 35
- `__init__()` (*cle.backends.elf.variable.StackVariable* method), 35
- `__init__()` (*cle.backends.elf.variable.Variable* method), 34
- `__init__()` (*cle.backends.elf.variable\_type.ArrayType* method), 38
- `__init__()` (*cle.backends.elf.variable\_type.PointerType* method), 36
- `__init__()` (*cle.backends.elf.variable\_type.StructMember* method), 37
- `__init__()` (*cle.backends.elf.variable\_type.StructType* method), 37
- `__init__()` (*cle.backends.elf.variable\_type.TypedefType* method), 38
- `__init__()` (*cle.backends.elf.variable\_type.VariableType* method), 36
- `__init__()` (*cle.backends.externs.ExternObject* method), 24
- `__init__()` (*cle.backends.externs.ExternSegment* method), 23
- `__init__()` (*cle.backends.externs.KernelObject* method), 24
- `__init__()` (*cle.backends.externs.simdata.common.SimDataSimpleReloc* method), 29
- `__init__()` (*cle.backends.ihex.Hex* method), 74
- `__init__()` (*cle.backends.java.apk.Apk* method), 74
- `__init__()` (*cle.backends.java.jar.Jar* method), 75

- `__init__` () (*cle.backends.java.soot.Soot* method), 76  
`__init__` () (*cle.backends.macho.binding.BindingHelper* method), 51  
`__init__` () (*cle.backends.macho.binding.BindingState* method), 51  
`__init__` () (*cle.backends.macho.binding.MachOPointerRelocation* method), 56  
`__init__` () (*cle.backends.macho.binding.MachOSymbolRelocation* method), 56  
`__init__` () (*cle.backends.macho.macho.MachO* method), 44  
`__init__` () (*cle.backends.macho.macho.SymbolList* method), 46  
`__init__` () (*cle.backends.macho.section.MachOSection* method), 50  
`__init__` () (*cle.backends.macho.segment.MachOSegment* method), 50  
`__init__` () (*cle.backends.macho.symbol.AbstractMachOSymbol* method), 47  
`__init__` () (*cle.backends.macho.symbol.BindingSymbol* method), 49  
`__init__` () (*cle.backends.macho.symbol.DyldBoundSymbol* method), 49  
`__init__` () (*cle.backends.macho.symbol.SymbolTableSymbol* method), 47  
`__init__` () (*cle.backends.minidump.Minidump* method), 77  
`__init__` () (*cle.backends.minidump.MinidumpMissingStream* method), 77  
`__init__` () (*cle.backends.named\_region.NamedRegion* method), 23  
`__init__` () (*cle.backends.pe.regions.PESection* method), 43  
`__init__` () (*cle.backends.pe.relocation.generic.IMAGE\_RELOCATION\_GENERIC* method), 87  
`__init__` () (*cle.backends.pe.relocation.pereloc.PEReloc* method), 86  
`__init__` () (*cle.backends.pe.symbol.WinSymbol* method), 43  
`__init__` () (*cle.backends.region.EmptySegment* method), 22  
`__init__` () (*cle.backends.region.Region* method), 21  
`__init__` () (*cle.backends.region.Section* method), 22  
`__init__` () (*cle.backends.regions.Regions* method), 20  
`__init__` () (*cle.backends.relocation.Relocation* method), 83  
`__init__` () (*cle.backends.static\_archive.StaticArchive* method), 78  
`__init__` () (*cle.backends.symbol.Symbol* method), 19  
`__init__` () (*cle.backends.te.TE* method), 81  
`__init__` () (*cle.backends.tls.ELFCoreThreadManager* method), 88  
`__init__` () (*cle.backends.tls.ELFThreadManager* method), 88  
`__init__` () (*cle.backends.tls.InternalTLSRelocation* method), 87  
`__init__` () (*cle.backends.tls.MinidumpThreadManager* method), 88  
`__init__` () (*cle.backends.tls.TLSObject* method), 88  
`__init__` () (*cle.backends.tls.ThreadManager* method), 87  
`__init__` () (*cle.backends.tls.elf\_tls.ELFTLSObject* method), 89  
`__init__` () (*cle.backends.tls.elf\_tls.ELFThreadManager* method), 89  
`__init__` () (*cle.backends.tls.elfcore\_tls.ELFCoreThread* method), 91  
`__init__` () (*cle.backends.tls.elfcore\_tls.ELFCoreThreadManager* method), 91  
`__init__` () (*cle.backends.tls.minidump\_tls.MinidumpThread* method), 92  
`__init__` () (*cle.backends.tls.minidump\_tls.MinidumpThreadManager* method), 91  
`__init__` () (*cle.backends.tls.pe\_tls.PETLSObject* method), 91  
`__init__` () (*cle.backends.tls.tls\_object.InternalTLSRelocation* method), 89  
`__init__` () (*cle.backends.tls.tls\_object.TLSObject* method), 89  
`__init__` () (*cle.backends.tls.tls\_object.ThreadManager* method), 88  
`__init__` () (*cle.backends.uefi\_firmware.UefiFirmware* method), 78  
`__init__` () (*cle.backends.uefi\_firmware.UefiModuleMixin* method), 79  
`__init__` () (*cle.backends.uefi\_firmware.UefiModulePending* method), 79  
`__init__` () (*cle.backends.xbe.XBE* method), 82  
`__init__` () (*cle.backends.xbe.XBESection* method), 81  
`__init__` () (*cle.memory.Clemory* method), 95  
`__init__` () (*cle.memory.ClemoryBase* method), 93  
`__init__` () (*cle.memory.ClemoryTranslator* method), 96  
`__init__` () (*cle.memory.ClemoryView* method), 96  
`__init__` () (*cle.memory.UninitializedClemory* method), 97  
`__init__` () (*cle.patched\_stream.PatchedStream* method), 98  
`__new__` () (*cle.backends.coff.IMAGE\_FILE\_MACHINE* method), 66  
`__new__` () (*cle.backends.coff.IMAGE\_REL\_AMD64* method), 68  
`__new__` () (*cle.backends.coff.IMAGE\_REL\_I386* method), 68  
`__new__` () (*cle.backends.coff.IMAGE\_SCN* method), 67  
`__new__` () (*cle.backends.coff.IMAGE\_SYM\_CLASS* method), 68  
`__new__` () (*cle.backends.elf.symbol\_type.ELFSymbolType*

- method), 33
- \_\_new\_\_() (cle.backends.macho.structs.DyldChainedPtrFormats method), 57
- \_\_new\_\_() (cle.backends.macho.structs.DyldImportFormats method), 57
- \_\_new\_\_() (cle.backends.te.HeaderType static method), 79
- \_\_new\_\_() (cle.backends.te.SectionHeaderType static method), 80
- ## A
- AbstractMachOSymbol (class in cle.backends.macho.symbol), 47
- add() (cle.backends.macho.macho.SymbolList method), 46
- add\_address\_ov() (cle.backends.macho.binding.Bindings method), 51
- add\_backer() (cle.memory.Clemetry method), 95
- add\_backer() (cle.memory.UninitializedClemetry method), 97
- add\_name() (cle.backends.externs.KernelObject method), 25
- addend (cle.backends.elf.relocation.elfreloc.ELFReloc property), 83
- addend (cle.backends.externs.simdata.common.PointTo attribute), 28
- addend (cle.backends.macho.structs.dyld\_chained\_import\_auth\_arm64e attribute), 62
- addend (cle.backends.macho.structs.dyld\_chained\_import\_auth\_arm64e attribute), 62
- addend (cle.backends.macho.structs.dyld\_chained\_ptr\_64\_auth\_arm64e attribute), 60
- addend (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth attribute), 58
- addr (cle.backends.backend.FunctionHint attribute), 13
- addr (cle.backends.elf.variable.Variable property), 34
- ADDR32NB (cle.backends.coff.IMAGE\_REL\_AMD64 attribute), 68
- ADDR64 (cle.backends.coff.IMAGE\_REL\_AMD64 attribute), 68
- addr\_to\_line (cle.backends.ELF attribute), 29
- addr\_to\_offset() (cle.backends.backend.Backend method), 16
- addr\_to\_offset() (cle.backends.externs.ExternSegment method), 23
- addr\_to\_offset() (cle.backends.region.Region method), 21
- addrDiv (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth attribute), 58
- addrDiv (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth attribute), 59
- addrDiv (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth attribute), 57
- address\_of\_entry\_point (cle.backends.te.HeaderType attribute), 80
- AddressTranslator (class in cle.address\_translator), 98
- ALIGN\_DOWN() (in module cle.utils), 99
- ALIGN\_UP() (in module cle.utils), 99
- all\_elf\_objects (cle.Loader property), 9
- all\_pe\_objects (cle.Loader property), 9
- allocate() (cle.backends.externs.ExternObject method), 24
- AMD64 (cle.backends.coff.IMAGE\_FILE\_MACHINE attribute), 66
- Apk (class in cle.backends.java.apk), 74
- append() (cle.backends.regions.Regions method), 20
- arch (cle.backends.backend.Backend property), 15
- Arm64e (class in cle.backends.macho.structs), 59
- arm64e (cle.backends.macho.structs.ChainedFixupPointerOnDisk attribute), 61
- ArrayType (class in cle.backends.elf.variable\_type), 38
- AT (in module cle.address\_translator), 99
- attributes (cle.backends.macho.section.MachOSection property), 50
- auth (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind attribute), 58
- auth (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind24 attribute), 59
- auth (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_rebase attribute), 57
- auth (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind attribute), 58
- auth (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_rebase attribute), 58
- authBind (cle.backends.macho.structs.Arm64e attribute), 59
- authBind24 (cle.backends.macho.structs.Arm64e attribute), 60
- authRebase (cle.backends.macho.structs.Arm64e attribute), 59
- AUTO\_HANDLE\_NONE (cle.backends.elf.relocation.generic.GenericIRelativeK attribute), 84
- AUTO\_HANDLE\_NONE (cle.backends.elf.relocation.generic.GenericRelativeR attribute), 85
- AUTO\_HANDLE\_NONE (cle.backends.elf.relocation.generic.GenericTLSDescr attribute), 84
- AUTO\_HANDLE\_NONE (cle.backends.elf.relocation.generic.GenericTLSModI attribute), 84
- AUTO\_HANDLE\_NONE (cle.backends.elf.relocation.generic.GenericTLSOffse attribute), 84
- AUTO\_HANDLE\_NONE (cle.backends.elf.relocation.generic.MipsLocalReloc attribute), 85
- AUTO\_HANDLE\_NONE (cle.backends.pe.relocation.pereloc.PEReloc attribute), 86
- AUTO\_HANDLE\_NONE (cle.backends.relocation.Relocation attribute), 83

- AUTO\_HANDLE\_NONE (*cle.backends.tls.InternalTLSRelocation* attribute), 87
- AUTO\_HANDLE\_NONE (*cle.backends.tls.tls\_object.InternalTLSRelocation* attribute), 89
- auto\_load\_libs (*cle.Loader* property), 9
- available\_arches (*cle.backends.backend.Backend* property), 17
- ## B
- BackedCGC (*class in cle.backends.cgc*), 72
- BackedCGC (*class in cle.backends.cgc.backedcgc*), 73
- Backend (*class in cle.backends.backend*), 14
- backers() (*cle.memory.Clemory* method), 95
- backers() (*cle.memory.ClemoryBase* method), 93
- backers() (*cle.memory.ClemoryTranslator* method), 96
- backers() (*cle.memory.ClemoryView* method), 96
- backers() (*cle.memory.UninitializedClemory* method), 97
- base\_of\_code (*cle.backends.te.HeaderType* attribute), 80
- BaseType (*class in cle.backends.elf.variable\_type*), 37
- binary (*cle.backends.macho.binding.BindingHelper* attribute), 51
- bind (*cle.backends.macho.structs.Arm64e* attribute), 60
- bind (*cle.backends.macho.structs.dyld\_chained\_ptr\_64\_bind* attribute), 60
- bind (*cle.backends.macho.structs.dyld\_chained\_ptr\_64\_rebase* attribute), 60
- bind (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind* attribute), 58
- bind (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind24* attribute), 59
- bind (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind4* attribute), 57
- bind (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind* attribute), 59
- bind (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_rebase* attribute), 58
- bind (*cle.backends.macho.structs.Generic64* attribute), 61
- bind24 (*cle.backends.macho.structs.Arm64e* attribute), 60
- BindingHelper (*class in cle.backends.macho.binding*), 51
- BindingState (*class in cle.backends.macho.binding*), 51
- BindingSymbol (*class in cle.backends.macho.symbol*), 49
- BINJA\_ARCH\_MAP (*cle.backends.binja.BinjaBin* attribute), 64
- BINJA\_DATA\_SYM\_TYPES (*cle.backends.binja.BinjaSymbol* attribute), 64
- BINJA\_FUNC\_SYM\_TYPES (*cle.backends.binja.BinjaSymbol* attribute), 64
- BINJA\_IMPORT\_TYPES (*cle.backends.binja.BinjaSymbol* attribute), 64
- BinjaBin (*class in cle.backends.binja*), 64
- BinjaReloc (*class in cle.backends.binja*), 64
- BinjaSymbol (*class in cle.backends.binja*), 64
- Blob (*class in cle.backends.blob*), 65
- build() (*cle.backends.uefi\_firmware.UefiModulePending* method), 79
- ## C
- cached\_content (*cle.backends.backend.Backend* attribute), 15
- call\_site\_encoding (*cle.backends.elf.lsda.ExceptionTableHeader* attribute), 39
- call\_site\_table\_len (*cle.backends.elf.lsda.ExceptionTableHeader* attribute), 39
- CallSiteEntry (*class in cle.backends.elf.lsda*), 39
- CGC (*class in cle.backends.cgc*), 72
- CGC (*class in cle.backends.cgc.cgc*), 73
- ChainedFixupPointerOnDisk (*class in cle.backends.macho.structs*), 61
- Characteristics (*cle.backends.coff.CoffFileHeader* attribute), 66
- Characteristics (*cle.backends.coff.CoffSectionTableEntry* attribute), 67
- characteristics (*cle.backends.te.SectionHeaderType* attribute), 80
- check\_address\_bounds() (*cle.backends.macho.binding.BindingState* method), 51
- check\_compatibility() (*cle.backends.backend.Backend* class method), 18
- check\_compatibility() (*cle.backends.blob.Blob* class method), 29
- check\_compatibility() (*cle.backends.ELF* class method), 29
- check\_compatibility() (*cle.backends.macho.macho.MachO* class method), 45
- check\_compatibility() (*cle.backends.named\_region.NamedRegion* class method), 23
- check\_compatibility() (*cle.backends.PE* class method), 43
- check\_compatibility() (*cle.backends.xbe.XBE* class method), 82
- check\_magic\_compatibility() (*cle.backends.backend.Backend* class method), 18
- check\_magic\_compatibility() (*cle.backends.ELF* class method), 30

`check_magic_compatibility()` (*cle.backends.PE class method*), 43  
`check_sign_extend()` (*cle.backends.elf.relocation.generic.RelocationMips attribute*), 85  
`check_valid_pointer_format()` (*cle.backends.macho.structs.Arm64e static method*), 60  
`check_valid_pointer_format()` (*cle.backends.macho.structs.Generic64 static method*), 61  
`check_zero_extend()` (*cle.backends.elf.relocation.generic.RelocationMips attribute*), 85  
`chh()` (*in module cle.backends.macho.binding*), 51  
`child_objects` (*cle.backends.backend.Backend attribute*), 15  
`classes` (*cle.backends.java.soot.Soot property*), 76  
`cle.address_translator` module, 98  
`cle.backends.backend` module, 13  
`cle.backends.binja` module, 64  
`cle.backends.blob` module, 65  
`cle.backends.cgc` module, 72  
`cle.backends.cgc.backedcgc` module, 73  
`cle.backends.cgc.cgc` module, 73  
`cle.backends.coff` module, 66  
`cle.backends.elf.compilation_unit` module, 41  
`cle.backends.elf.hashtable` module, 39  
`cle.backends.elf.lsd` module, 39  
`cle.backends.elf.relocation` module, 83  
`cle.backends.elf.relocation.amd64` module, 86  
`cle.backends.elf.relocation.arm` module, 86  
`cle.backends.elf.relocation.arm64` module, 86  
`cle.backends.elf.relocation.elfreloc` module, 83  
`cle.backends.elf.relocation.generic` module, 83  
`cle.backends.elf.relocation.i386` module, 86  
`cle.backends.elf.relocation.mips` module, 86  
`cle.backends.elf.relocation.ppc` module, 85  
`cle.backends.elf.relocation.ppc64` module, 86  
`cle.backends.elf.relocation.s390x` module, 86  
`cle.backends.elf.subprogram` module, 40  
`cle.backends.elf.variable` module, 34  
`cle.backends.elf.variable_type` module, 35  
`cle.backends.externs` module, 23  
`cle.backends.externs.simdata` module, 25  
`cle.backends.externs.simdata.common` module, 27  
`cle.backends.externs.simdata.simdata` module, 26  
`cle.backends.ihex` module, 74  
`cle.backends.java` module, 76  
`cle.backends.java.android_lifecycle` module, 74  
`cle.backends.java.apk` module, 74  
`cle.backends.java.jar` module, 75  
`cle.backends.java.soot` module, 76  
`cle.backends.macho.binding` module, 51  
`cle.backends.macho.section` module, 49  
`cle.backends.macho.segment` module, 50  
`cle.backends.macho.structs` module, 56  
`cle.backends.macho.symbol` module, 47  
`cle.backends.minidump` module, 77  
`cle.backends.named_region` module, 23  
`cle.backends.pe.regions` module, 43  
`cle.backends.pe.relocation` module, 86  
`cle.backends.pe.relocation.arm` module, 87  
`cle.backends.pe.relocation.generic` module, 86

- cle.backends.pe.relocation.mips
  - module, 87
- cle.backends.pe.relocation.pereloc
  - module, 86
- cle.backends.pe.relocation.riscv
  - module, 87
- cle.backends.pe.symbol
  - module, 43
- cle.backends.region
  - module, 21
- cle.backends.regions
  - module, 20
- cle.backends.relocation
  - module, 82
- cle.backends.static\_archive
  - module, 78
- cle.backends.symbol
  - module, 18
- cle.backends.te
  - module, 79
- cle.backends.tls
  - module, 87
- cle.backends.tls.elf\_tls
  - module, 89
- cle.backends.tls.elfcore\_tls
  - module, 91
- cle.backends.tls.minidump\_tls
  - module, 91
- cle.backends.tls.pe\_tls
  - module, 90
- cle.backends.tls.tls\_object
  - module, 88
- cle.backends.uefi\_firmware
  - module, 78
- cle.backends.xbe
  - module, 81
- cle.errors
  - module, 92
- cle.gdb
  - module, 92
- cle.memory
  - module, 93
- cle.patched\_stream
  - module, 97
- cle.utils
  - module, 99
- CLECompatibilityError, 92
- CLError, 92
- CLEFileNotFoundError, 92
- CLEInvalidBinaryError, 92
- CLEMemoryError, 92
- Clemory (class in cle.memory), 94
- ClemoryBase (class in cle.memory), 93
- ClemoryTranslator (class in cle.memory), 96
- ClemoryView (class in cle.memory), 96
- CLEOperationError, 92
- CLEUnknownFormatError, 92
- close() (cle.backends.backend.Backend method), 16
- close() (cle.backends.binja.BinjaBin method), 65
- close() (cle.backends.ELF method), 29
- close() (cle.backends.minidump.Minidump method), 77
- close() (cle.backends.PE method), 43
- close() (cle.backends.xbe.XBE method), 82
- close() (cle.Loader method), 9
- close() (cle.memory.ClemoryBase method), 94
- close() (cle.patched\_stream.PatchedStream method), 98
- CNT\_UNINITIALIZED\_DATA
  - (cle.backends.coff.IMAGE\_SCN attribute), 67
- coalesce\_regions() (cle.backends.ihex.Hex static method), 74
- Coff (class in cle.backends.coff), 71
- CoffFileHeader (class in cle.backends.coff), 66
- CoffParser (class in cle.backends.coff), 69
- CoffRelocation (class in cle.backends.coff), 70
- CoffRelocationADDR32NB (class in cle.backends.coff), 71
- CoffRelocationADDR64 (class in cle.backends.coff), 71
- CoffRelocationDIR32 (class in cle.backends.coff), 70
- CoffRelocationDIR32NB (class in cle.backends.coff), 70
- CoffRelocationREL32 (class in cle.backends.coff), 70
- CoffRelocationSECREL (class in cle.backends.coff), 71
- CoffRelocationSECTION (class in cle.backends.coff), 71
- CoffRelocationTableEntry (class in cle.backends.coff), 68
- CoffSection (class in cle.backends.coff), 69
- CoffSectionTableEntry (class in cle.backends.coff), 67
- CoffSymbolTableEntry (class in cle.backends.coff), 68
- common\_align (cle.backends.macho.symbol.SymbolTableSymbol property), 48
- compilation\_units (cle.backends.ELF attribute), 29
- CompilationUnit (class in cle.backends.elf.compilation\_unit), 41
- consecutive (cle.memory.Clemory attribute), 95
- consecutive (cle.memory.UninitializedClemory attribute), 97
- contains\_addr() (cle.backends.backend.Backend method), 16
- contains\_addr() (cle.backends.blob.Blob method), 66
- contains\_addr() (cle.backends.named\_region.NamedRegion method), 23
- contains\_addr() (cle.backends.region.Region method), 21
- contains\_offset() (cle.backends.externs.ExternSegment

- method), 24
- contains\_offset() (cle.backends.region.Region method), 21
- convert\_info\_proc\_maps() (in module cle.gdb), 93
- convert\_info\_sharedlibrary() (in module cle.gdb), 92
- cs\_action (cle.backends.elf.ltda.CallSiteEntry attribute), 39
- cs\_len (cle.backends.elf.ltda.CallSiteEntry attribute), 39
- cs\_lp (cle.backends.elf.ltda.CallSiteEntry attribute), 39
- cs\_start (cle.backends.elf.ltda.CallSiteEntry attribute), 39
- ## D
- data (cle.backends.coff.CoffParser attribute), 69
- data (cle.backends.externs.simdata.common.StaticData attribute), 27
- data\_directory\_0\_size (cle.backends.te.HeaderType attribute), 80
- data\_directory\_0\_virtual\_address (cle.backends.te.HeaderType attribute), 80
- data\_directory\_1\_size (cle.backends.te.HeaderType attribute), 80
- data\_directory\_1\_virtual\_address (cle.backends.te.HeaderType attribute), 80
- decode\_thumb\_interworking() (cle.backends.macho.macho.MachO method), 45
- default\_binding\_handler() (in module cle.backends.macho.binding), 56
- demangled\_name() (cle.backends.macho.symbol.BindingSymbol method), 49
- demangled\_name() (cle.backends.macho.symbol.DyldBoundSymbol method), 49
- describe\_addr() (cle.Loader method), 9
- dest\_addr (cle.backends.macho.binding.MachOSymbolRelocation property), 56
- dest\_addr (cle.backends.relocation.Relocation property), 83
- DIR32 (cle.backends.coff.IMAGE\_REL\_I386 attribute), 68
- DIR32NB (cle.backends.coff.IMAGE\_REL\_I386 attribute), 68
- discard\_ro\_memview() (cle.Loader method), 12
- diversity (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind attribute), 58
- diversity (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind attribute), 59
- diversity (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind attribute), 57
- DllImport (class in cle.backends.pe.relocation.generic), 86
- do\_binding() (cle.backends.macho.macho.MachO method), 45
- do\_lazy\_bind() (cle.backends.macho.binding.BindingHelper method), 51
- do\_normal\_bind() (cle.backends.macho.binding.BindingHelper method), 51
- do\_rebases() (cle.backends.macho.binding.BindingHelper method), 51
- dtv (cle.backends.tls.elfcore\_tls.ELFCoreThread property), 91
- dyld\_chained\_fixups\_header (class in cle.backends.macho.structs), 63
- dyld\_chained\_import (class in cle.backends.macho.structs), 62
- DYLD\_CHAINED\_IMPORT (cle.backends.macho.structs.DyldImportFormats attribute), 56
- dyld\_chained\_import\_addend (class in cle.backends.macho.structs), 62
- DYLD\_CHAINED\_IMPORT\_ADDEND (cle.backends.macho.structs.DyldImportFormats attribute), 57
- dyld\_chained\_import\_addend64 (class in cle.backends.macho.structs), 62
- DYLD\_CHAINED\_IMPORT\_ADDEND64 (cle.backends.macho.structs.DyldImportFormats attribute), 57
- DYLD\_CHAINED\_PTR\_32 (cle.backends.macho.structs.DyldChainedPtrFormats attribute), 57
- DYLD\_CHAINED\_PTR\_32\_CACHE (cle.backends.macho.structs.DyldChainedPtrFormats attribute), 57
- DYLD\_CHAINED\_PTR\_32\_FIRMWARE (cle.backends.macho.structs.DyldChainedPtrFormats attribute), 57
- DYLD\_CHAINED\_PTR\_64 (cle.backends.macho.structs.DyldChainedPtrFormats attribute), 57
- dyld\_chained\_ptr\_64\_bind (class in cle.backends.macho.structs), 60
- DYLD\_CHAINED\_PTR\_64\_KERNEL\_CACHE (cle.backends.macho.structs.DyldChainedPtrFormats attribute), 57
- DYLD\_CHAINED\_PTR\_64\_OFFSET (cle.backends.macho.structs.DyldChainedPtrFormats attribute), 57
- dyld\_chained\_ptr\_64\_rebase (class in cle.backends.macho.structs), 60
- DYLD\_CHAINED\_PTR\_ARM64E (cle.backends.macho.structs.DyldChainedPtrFormats attribute), 57
- dyld\_chained\_ptr\_arm64e\_auth\_bind (class in cle.backends.macho.structs), 58

- dyld\_chained\_ptr\_arm64e\_auth\_bind24 (class in *cle.backends.macho.structs*), 59
- dyld\_chained\_ptr\_arm64e\_auth\_rebase (class in *cle.backends.macho.structs*), 57
- dyld\_chained\_ptr\_arm64e\_bind (class in *cle.backends.macho.structs*), 58
- dyld\_chained\_ptr\_arm64e\_bind24 (class in *cle.backends.macho.structs*), 59
- DYLD\_CHAINED\_PTR\_ARM64E\_FIRMWARE (class in *cle.backends.macho.structs.DyldChainedPtrFormats* attribute), 57
- DYLD\_CHAINED\_PTR\_ARM64E\_KERNEL (class in *cle.backends.macho.structs.DyldChainedPtrFormats* attribute), 57
- dyld\_chained\_ptr\_arm64e\_rebase (class in *cle.backends.macho.structs*), 58
- DYLD\_CHAINED\_PTR\_ARM64E\_USERLAND (class in *cle.backends.macho.structs.DyldChainedPtrFormats* attribute), 57
- DYLD\_CHAINED\_PTR\_ARM64E\_USERLAND24 (class in *cle.backends.macho.structs.DyldChainedPtrFormats* attribute), 57
- DYLD\_CHAINED\_PTR\_X86\_64\_KERNEL\_CACHE (class in *cle.backends.macho.structs.DyldChainedPtrFormats* attribute), 57
- dyld\_chained\_starts\_in\_image (class in *cle.backends.macho.structs*), 63
- dyld\_chained\_starts\_in\_segment (class in *cle.backends.macho.structs*), 63
- DyldBoundSymbol (class in *cle.backends.macho.symbol*), 49
- DyldChainedPtrFormats (class in *cle.backends.macho.structs*), 57
- DyldImportFormats (class in *cle.backends.macho.structs*), 56
- DyldImportStruct (class in *cle.backends.macho.structs*), 62
- dynamic\_load() (*cle.Loader* method), 12
- ## E
- EH\_FRAME (*cle.backends.backend.FunctionHintSource* attribute), 13
- element\_type (*cle.backends.elf.variable\_type.ArrayType* property), 38
- ELF (class in *cle.backends*), 29
- elf\_hash() (*cle.backends.elf.hashtable.ELFHashTable* static method), 40
- elf\_value (*cle.backends.elf.symbol\_type.ELFSymbolType* property), 32
- ELFCore (class in *cle.backends.elf*), 30
- elfcore\_object (*cle.Loader* property), 9
- ELFCoreThread (class in *cle.backends.tls.elfcore\_tls*), 91
- ELFCoreThreadManager (class in *cle.backends.tls*), 88
- ELFCoreThreadManager (class in *cle.backends.tls.elfcore\_tls*), 91
- ELFHashTable (class in *cle.backends.elf.hashtable*), 39
- ELFReloc (class in *cle.backends.elf.relocation.elfreloc*), 83
- ELFSection (class in *cle.backends.elf.regions*), 33
- ELFSegment (class in *cle.backends.elf.regions*), 33
- ELFSymbol (class in *cle.backends.elf.symbol*), 32
- ELFSymbolType (class in *cle.backends.elf.symbol\_type*), 32
- ELFThreadManager (class in *cle.backends.tls*), 88
- ELFThreadManager (class in *cle.backends.tls.elf\_tls*), 89
- ELFTLSObject (class in *cle.backends.tls.elf\_tls*), 89
- ELFTLSObjectV1 (class in *cle.backends.tls.elf\_tls*), 90
- ELFTLSObjectV2 (class in *cle.backends.tls.elf\_tls*), 90
- EmptySegment (class in *cle.backends.region*), 22
- entry (*cle.backends.backend.Backend* property), 16
- entry (*cle.backends.binja.BinjaBin* property), 65
- entry (*cle.backends.java.soot.Soot* property), 76
- exception\_handlings (*cle.backends.backend.Backend* attribute), 15
- ExceptionHandler (class in *cle.backends.backend*), 13
- ExceptionHandlerHeader (class in *cle.backends.elf.lsda*), 39
- EXPORT\_TABLE (*cle.backends.backend.FunctionHintSource* attribute), 13
- extern\_object (*cle.Loader* property), 9
- extern\_size\_hints (*cle.backends.ELF* attribute), 29
- EXTERNAL (*cle.backends.coff.IMAGE\_SYM\_CLASS* attribute), 67
- EXTERNAL\_EH\_FRAME (*cle.backends.backend.FunctionHintSource* attribute), 13
- ExternObject (class in *cle.backends.externs*), 24
- ExternSegment (class in *cle.backends.externs*), 23
- extract\_arch() (*cle.backends.ELF* static method), 30
- extract\_null\_terminated\_bytestr() (in module *cle.utils*), 99
- extract\_soname() (*cle.backends.backend.Backend* static method), 17
- extract\_soname() (*cle.backends.elf.MetaELF* static method), 31
- extract\_soname() (*cle.backends.macho.macho.MachO* static method), 45
- ## F
- FakeSegment (class in *cle.backends.cgc.backedcgc*), 73
- fast\_memory\_load\_pointer() (*cle.Loader* method), 13
- filesize (*cle.backends.region.Region* attribute), 21
- finalizers (*cle.backends.backend.Backend* property), 17
- finalizers (*cle.backends.ELF* property), 30
- finalizers (*cle.Loader* property), 9

- find() (*cle.memory.Clemetry* method), 96  
 find() (*cle.memory.ClemetryBase* method), 93  
 find() (*cle.memory.ClemetryTranslator* method), 96  
 find() (*cle.memory.ClemetryView* method), 96  
 find() (*cle.memory.UninitializedClemetry* method), 97  
 find\_all\_symbols() (*cle.Loader* method), 11  
 find\_loadable\_containing() (*cle.backends.backend.Backend* method), 16  
 find\_loadable\_containing() (*cle.Loader* method), 11  
 find\_object() (*cle.Loader* method), 10  
 find\_object\_containing() (*cle.Loader* method), 10  
 find\_plt\_stub\_name() (*cle.Loader* method), 12  
 find\_region\_containing() (*cle.backends.regions.Regions* method), 20  
 find\_region\_next\_to() (*cle.backends.regions.Regions* method), 21  
 find\_relevant\_relocations() (*cle.Loader* method), 12  
 find\_section\_containing() (*cle.backends.backend.Backend* method), 16  
 find\_section\_containing() (*cle.Loader* method), 10  
 find\_section\_next\_to() (*cle.Loader* method), 11  
 find\_segment\_by\_name() (*cle.backends.macho.macho.MachO* method), 45  
 find\_segment\_containing() (*cle.backends.backend.Backend* method), 16  
 find\_segment\_containing() (*cle.Loader* method), 10  
 find\_symbol() (*cle.Loader* method), 11  
 fixups\_version (*cle.backends.macho.structs.dyld\_chained\_fixups\_header* attribute), 63  
 from\_die() (*cle.backends.elf.variable.Variable* static method), 34  
 from\_linked\_va() (*cle.address\_translator.AddressTranslator* class method), 98  
 from\_lva() (*cle.address\_translator.AddressTranslator* class method), 98  
 from\_mapped\_va() (*cle.address\_translator.AddressTranslator* class method), 98  
 from\_mvva() (*cle.address\_translator.AddressTranslator* class method), 98  
 from\_raw() (*cle.address\_translator.AddressTranslator* class method), 98  
 from\_relative\_va() (*cle.address\_translator.AddressTranslator* class method), 98  
 from\_rva() (*cle.address\_translator.AddressTranslator* class method), 98  
 from\_va() (*cle.address\_translator.AddressTranslator* class method), 98  
 FULL (*cle.backends.elf.metaelf.Relro* attribute), 32  
 full\_name (*cle.backends.macho.section.MachOSection* property), 50  
 func\_addr (*cle.backends.backend.ExceptionHandling* attribute), 14  
 FUNCTION (*cle.backends.coff.IMAGE\_SYM\_CLASS* attribute), 67  
 function\_hints (*cle.backends.backend.Backend* attribute), 15  
 function\_name() (*cle.backends.binja.BinjaBin* method), 65  
 function\_name() (*cle.backends.blob.Blob* method), 66  
 function\_name() (*cle.backends.named\_region.NamedRegion* method), 23  
 FunctionHint (class in *cle.backends.backend*), 13  
 FunctionHintSource (class in *cle.backends.backend*), 13  
 functions\_debug\_info (*cle.backends.ELF* attribute), 29
- ## G
- gen\_ro\_memview() (*cle.Loader* method), 12  
 Generic64 (class in *cle.backends.macho.structs*), 61  
 generic64 (*cle.backends.macho.structs.ChainedFixupPointerOnDisk* attribute), 61  
 GenericAbsoluteAddendReloc (class in *cle.backends.elf.relocation.generic*), 84  
 GenericAbsoluteReloc (class in *cle.backends.elf.relocation.generic*), 85  
 GenericCopyReloc (class in *cle.backends.elf.relocation.generic*), 85  
 GenericIRelativeReloc (class in *cle.backends.elf.relocation.generic*), 84  
 GenericJumpslotReloc (class in *cle.backends.elf.relocation.generic*), 84  
 GenericPCRelativeAddendReloc (class in *cle.backends.elf.relocation.generic*), 84  
 GenericRelativeReloc (class in *cle.backends.elf.relocation.generic*), 85  
 GenericTLSDescriptorReloc (class in *cle.backends.elf.relocation.generic*), 84  
 GenericTLSOffsetReloc (class in *cle.backends.elf.relocation.generic*), 83  
 GenericTLSModIdReloc (class in *cle.backends.elf.relocation.generic*), 84  
 GenericTLSOffsetReloc (class in *cle.backends.elf.relocation.generic*), 84  
 get() (*cle.backends.elf.hashtable.ELFHashTable* method), 40  
 get() (*cle.backends.elf.hashtable.GNUHashTable* method), 40  
 get\_addr() (*cle.backends.tls.elf\_tls.ELFTLSObject* method), 90  
 get\_addr() (*cle.backends.tls.elfcore\_tls.ELFCoreThread* method), 91

- `get_by_name_and_ordinal()` (*cle.backends.macho.macho.SymbolList* method), 46  
`get_callbacks()` (*cle.backends.java.apk.Apk* method), 75  
`get_loader_symbolic_constraints()` (*cle.Loader* method), 12  
`get_manifest()` (*cle.backends.java.jar.Jar* method), 76  
`get_mmaped_data()` (in module *cle.utils*), 99  
`get_pseudo_addr()` (*cle.backends.externs.ExternObject* method), 24  
`get_relocation()` (in module *cle.backends.elf.relocation*), 83  
`get_relocation()` (in module *cle.backends.pe.relocation*), 86  
`get_section_by_name()` (*cle.backends.macho.segment.MachOSegment* method), 50  
`get_section_name()` (*cle.backends.coff.CoffParser* method), 69  
`get_segment_by_name()` (*cle.backends.macho.macho.MachO* method), 45  
`get_soot_class()` (*cle.backends.java.soot.Soot* method), 76  
`get_soot_method()` (*cle.backends.java.soot.Soot* method), 76  
`get_string()` (*cle.backends.macho.macho.MachO* method), 45  
`get_strings()` (*cle.backends.binja.BinjaBin* method), 65  
`get_struct()` (*cle.backends.macho.structs.DyldImportStruct* static method), 62  
`get_symbol()` (*cle.backends.backend.Backend* method), 17  
`get_symbol()` (*cle.backends.coff.Coff* method), 71  
`get_symbol()` (*cle.backends.ELF* method), 30  
`get_symbol()` (*cle.backends.macho.macho.MachO* method), 45  
`get_symbol()` (*cle.backends.PE* method), 43  
`get_symbol_addr()` (*cle.backends.binja.BinjaBin* method), 65  
`get_symbol_by_address_fuzzy()` (*cle.backends.macho.macho.MachO* method), 45  
`get_symbol_by_insertion_order()` (*cle.backends.macho.macho.MachO* method), 45  
`get_symbol_name()` (*cle.backends.coff.CoffParser* method), 69  
`get_text_offset()` (in module *cle.utils*), 99  
`get_thread_registers_by_id()` (*cle.backends.minidump.Minidump* method), 77  
`get_tls_data_addr()` (*cle.backends.tls.minidump\_tls.MinidumpThread* method), 92  
`get_tls_data_addr()` (*cle.backends.tls.pe\_tls.PETLSObject* method), 91  
`gnu_hash()` (*cle.backends.elf.hashtable.GNUHashTable* static method), 40  
GNUHashTable (class in *cle.backends.elf.hashtable*), 40
- ## H
- `handler_addr` (*cle.backends.backend.ExceptionHandling* attribute), 14  
`has_memory` (*cle.backends.named\_region.NamedRegion* attribute), 23  
`header` (*cle.backends.coff.CoffParser* attribute), 69  
HeaderType (class in *cle.backends.te*), 79  
HelperStruct (class in *cle.backends.macho.structs*), 56  
Hex (class in *cle.backends.ihex*), 74  
`high8` (*cle.backends.macho.structs.dyld\_chained\_ptr\_64\_rebase* attribute), 60  
`high8` (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_rebase* attribute), 58
- ## I
- I386 (*cle.backends.coff.IMAGE\_FILE\_MACHINE* attribute), 66  
`idx_to_symbol_name` (*cle.backends.coff.CoffParser* attribute), 69  
`image_base` (*cle.backends.te.HeaderType* attribute), 80  
`image_base_delta` (*cle.backends.backend.Backend* property), 16  
IMAGE\_FILE\_MACHINE (class in *cle.backends.coff*), 66  
IMAGE\_REL\_AMD64 (class in *cle.backends.coff*), 68  
IMAGE\_REL\_BASED\_ABSOLUTE (class in *cle.backends.pe.relocation.generic*), 86  
IMAGE\_REL\_BASED\_DIR64 (class in *cle.backends.pe.relocation.generic*), 87  
IMAGE\_REL\_BASED\_HIGH (class in *cle.backends.pe.relocation.generic*), 87  
IMAGE\_REL\_BASED\_HIGHADJ (class in *cle.backends.pe.relocation.generic*), 86  
IMAGE\_REL\_BASED\_HIGHLOW (class in *cle.backends.pe.relocation.generic*), 87  
IMAGE\_REL\_BASED\_LOW (class in *cle.backends.pe.relocation.generic*), 87  
IMAGE\_REL\_I386 (class in *cle.backends.coff*), 68  
IMAGE\_SCN (class in *cle.backends.coff*), 66  
IMAGE\_SYM\_CLASS (class in *cle.backends.coff*), 67  
`imports` (*cle.backends.backend.Backend* attribute), 15  
`imports_count` (*cle.backends.macho.structs.dyld\_chained\_fixups\_header* attribute), 63  
`imports_format` (*cle.backends.macho.structs.dyld\_chained\_fixups\_header* attribute), 63

*imports\_offset* (*cle.backends.macho.structs.dyld\_chained\_symbols* attribute), 63  
*in\_which\_segment*() (*cle.backends.binja.BinjaBin* method), 64  
*in\_which\_segment*() (*cle.backends.blob.Blob* method), 66  
*initial\_register\_values*() (*cle.backends.backend.Backend* method), 17  
*initialization\_image*() (*cle.backends.tls.ThreadManager* static method), 87  
*initialization\_image*() (*cle.backends.tls.tls\_object.ThreadManager* static method), 88  
*initializers* (*cle.backends.backend.Backend* property), 17  
*initializers* (*cle.backends.ELF* property), 30  
*initializers* (*cle.Loader* property), 9  
*InternalTLSRelocation* (class in *cle.backends.tls*), 87  
*InternalTLSRelocation* (class in *cle.backends.tls.tls\_object*), 88  
*is\_active* (*cle.backends.elf.regions.ELFSection* property), 33  
*is\_alt\_entry* (*cle.backends.macho.symbol.SymbolTableSymbol* property), 48  
*is\_base\_reloc* (*cle.backends.pe.relocation.perloc.PEReloc* property), 86  
*is\_common* (*cle.backends.macho.symbol.SymbolTableSymbol* property), 48  
*is\_common* (*cle.backends.symbol.Symbol* attribute), 19  
*is\_compatible*() (*cle.backends.backend.Backend* class method), 17  
*is\_compatible*() (*cle.backends.binja.BinjaBin* static method), 64  
*is\_compatible*() (*cle.backends.blob.Blob* class method), 65  
*is\_compatible*() (*cle.backends.cgc.BackedCGC* static method), 72  
*is\_compatible*() (*cle.backends.cgc.backedcgc.BackedCGC* static method), 74  
*is\_compatible*() (*cle.backends.cgc.CGC* static method), 72  
*is\_compatible*() (*cle.backends.cgc.cgc.CGC* static method), 73  
*is\_compatible*() (*cle.backends.coff.Coff* class method), 71  
*is\_compatible*() (*cle.backends.ELF* static method), 30  
*is\_compatible*() (*cle.backends.elf.ELFCore* static method), 31  
*is\_compatible*() (*cle.backends.ihex.Hex* static method), 74  
*is\_compatible*() (*cle.backends.java.apk.Apk* static method), 75  
*is\_compatible*() (*cle.backends.java.jar.Jar* static method), 76  
*is\_compatible*() (*cle.backends.macho.macho.MachO* class method), 45  
*is\_compatible*() (*cle.backends.minidump.Minidump* static method), 77  
*is\_compatible*() (*cle.backends.named\_region.NamedRegion* static method), 23  
*is\_compatible*() (*cle.backends.PE* class method), 43  
*is\_compatible*() (*cle.backends.static\_archive.StaticArchive* class method), 78  
*is\_compatible*() (*cle.backends.te.TE* class method), 81  
*is\_compatible*() (*cle.backends.uefi\_firmware.UefiFirmware* class method), 78  
*is\_compatible*() (*cle.backends.xbe.XBE* static method), 82  
*is\_custom\_os\_proc* (*cle.backends.elf.symbol\_type.ELFSymbolType* property), 32  
*is\_default* (*cle.backends.backend.Backend* attribute), 15  
*is\_default* (*cle.backends.binja.BinjaBin* attribute), 64  
*is\_default* (*cle.backends.blob.Blob* attribute), 65  
*is\_default* (*cle.backends.cgc.BackedCGC* attribute), 72  
*is\_default* (*cle.backends.cgc.backedcgc.BackedCGC* attribute), 73  
*is\_default* (*cle.backends.cgc.CGC* attribute), 72  
*is\_default* (*cle.backends.cgc.cgc.CGC* attribute), 73  
*is\_default* (*cle.backends.coff.Coff* attribute), 71  
*is\_default* (*cle.backends.ELF* attribute), 29  
*is\_default* (*cle.backends.elf.ELFCore* attribute), 30  
*is\_default* (*cle.backends.ihex.Hex* attribute), 74  
*is\_default* (*cle.backends.java.apk.Apk* attribute), 74  
*is\_default* (*cle.backends.java.jar.Jar* attribute), 75  
*is\_default* (*cle.backends.macho.macho.MachO* attribute), 44  
*is\_default* (*cle.backends.minidump.Minidump* attribute), 77  
*is\_default* (*cle.backends.named\_region.NamedRegion* attribute), 23  
*is\_default* (*cle.backends.PE* attribute), 42  
*is\_default* (*cle.backends.static\_archive.StaticArchive* attribute), 78  
*is\_default* (*cle.backends.te.TE* attribute), 81  
*is\_default* (*cle.backends.uefi\_firmware.UefiFirmware* attribute), 78  
*is\_default* (*cle.backends.xbe.XBE* attribute), 82  
*is\_desc\_discarded* (*cle.backends.macho.symbol.SymbolTableSymbol* property), 48  
*is\_executable* (*cle.backends.coff.CoffSection* property), 70  
*is\_executable* (*cle.backends.elf.regions.ELFSection* property), 34  
*is\_executable* (*cle.backends.elf.regions.ELFSegment*

- property), 33
- `is_executable` (*cle.backends.externs.ExternSegment attribute*), 24
- `is_executable` (*cle.backends.macho.section.MachOSection property*), 50
- `is_executable` (*cle.backends.macho.segment.MachOSegment property*), 51
- `is_executable` (*cle.backends.pe.regions.PESection property*), 43
- `is_executable` (*cle.backends.region.EmptySegment property*), 22
- `is_executable` (*cle.backends.region.Region property*), 22
- `is_executable` (*cle.backends.region.Section property*), 22
- `is_executable` (*cle.backends.xbe.XBESection property*), 82
- `is_export` (*cle.backends.symbol.Symbol attribute*), 19
- `is_extern` (*cle.backends.symbol.Symbol attribute*), 19
- `is_external` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_forward` (*cle.backends.symbol.Symbol attribute*), 19
- `is_function` (*cle.backends.macho.symbol.BindingSymbol property*), 49
- `is_function` (*cle.backends.macho.symbol.DyldBoundSymbol property*), 49
- `is_function` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_function` (*cle.backends.symbol.Symbol property*), 19
- `is_import` (*cle.backends.pe.relocation.pereloc.PEReloc property*), 86
- `is_import` (*cle.backends.symbol.Symbol attribute*), 19
- `is_local` (*cle.backends.symbol.Symbol attribute*), 19
- `is_no_dead_strip` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_outer` (*cle.backends.backend.Backend attribute*), 15
- `is_outer` (*cle.backends.static\_archive.StaticArchive attribute*), 78
- `is_ppc64_abiv1` (*cle.backends.elf.MetaELF property*), 31
- `is_ppc64_abiv2` (*cle.backends.elf.MetaELF property*), 31
- `is_private_external` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_readable` (*cle.backends.coff.CoffSection property*), 70
- `is_readable` (*cle.backends.elf.regions.ELFSection property*), 33
- `is_readable` (*cle.backends.elf.regions.ELFSegment property*), 33
- `is_readable` (*cle.backends.externs.ExternSegment attribute*), 24
- `is_readable` (*cle.backends.macho.section.MachOSection property*), 50
- `is_readable` (*cle.backends.macho.segment.MachOSegment property*), 51
- `is_readable` (*cle.backends.pe.regions.PESection property*), 43
- `is_readable` (*cle.backends.region.EmptySegment property*), 22
- `is_readable` (*cle.backends.region.Region property*), 22
- `is_readable` (*cle.backends.region.Section property*), 22
- `is_readable` (*cle.backends.xbe.XBESection property*), 82
- `is_reference_to_weak` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_relro` (*cle.backends.elf.regions.ELFSegment property*), 33
- `is_stab` (*cle.backends.macho.symbol.AbstractMachOSymbol property*), 47
- `is_stab` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_static` (*cle.backends.symbol.Symbol attribute*), 19
- `is_strings` (*cle.backends.elf.regions.ELFSection property*), 34
- `is_symbol_resolver` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_thumb_definition` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_thumb_interworking()` (*cle.backends.macho.macho.MachO method*), 45
- `is_weak` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_weak` (*cle.backends.symbol.Symbol attribute*), 19
- `is_weak_defined` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_weak_referenced` (*cle.backends.macho.symbol.SymbolTableSymbol property*), 48
- `is_writable` (*cle.backends.coff.CoffSection property*), 70
- `is_writable` (*cle.backends.elf.regions.ELFSection property*), 33
- `is_writable` (*cle.backends.elf.regions.ELFSegment property*), 33
- `is_writable` (*cle.backends.externs.ExternSegment attribute*), 24
- `is_writable` (*cle.backends.macho.section.MachOSection property*), 50
- `is_writable` (*cle.backends.macho.segment.MachOSegment property*), 51
- `is_writable` (*cle.backends.pe.regions.PESection property*), 43
- `is_writable` (*cle.backends.region.EmptySegment prop-*

- erty), 22
- is\_writable (cle.backends.region.Region property), 22
- is\_writable (cle.backends.region.Section property), 22
- is\_writable (cle.backends.xbe.XBESection property), 82
- is\_zip\_archive() (cle.backends.java.soot.Soot static method), 77
- isBind() (cle.backends.macho.structs.ChainedFixupPointerOnDisk property), 47  
method), 61
- isRebase() (cle.backends.macho.structs.ChainedFixupPointerOnDisk property), 49  
method), 61
- ## J
- Jar (class in cle.backends.java.jar), 75
- ## K
- kernel\_object (cle.Loader property), 9
- KernelObject (class in cle.backends.externs), 24
- key (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_attr attribute), 58
- key (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_attr attribute), 59
- key (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_attr attribute), 57
- key\_bisect\_find() (in module cle.utils), 99
- key\_bisect\_floor\_key() (in module cle.utils), 99
- key\_bisect\_insort\_left() (in module cle.utils), 99
- key\_bisect\_insort\_right() (in module cle.utils), 99
- ## L
- l\_opcode\_do\_bind() (in module cle.backends.macho.binding), 55
- l\_opcode\_set\_segment\_and\_offset\_uleb() (in module cle.backends.macho.binding), 54
- LABEL (cle.backends.coff.IMAGE\_SYM\_CLASS attribute), 67
- LexicalBlock (class in cle.backends.elf.subprogram), 40
- lib\_ordinal (cle.backends.macho.structs.dyld\_chained\_import attribute), 62
- lib\_ordinal (cle.backends.macho.structs.dyld\_chained\_import attribute), 62
- lib\_ordinal (cle.backends.macho.structs.dyld\_chained\_import attribute), 62
- lib\_ordinal (cle.backends.macho.structs.DyldImportStruct attribute), 62
- libname (cle.backends.externs.simdata.SimData attribute), 25
- libname (cle.backends.externs.simdata.simdata.SimData attribute), 26
- library\_base\_name (cle.backends.macho.symbol.AbstractMachOSymbol property), 47
- library\_name (cle.backends.macho.symbol.AbstractMachOSymbol property), 47
- library\_name (cle.backends.macho.symbol.BindingSymbol property), 49
- library\_name (cle.backends.macho.symbol.DyldBoundSymbol property), 49
- library\_name (cle.backends.macho.symbol.SymbolTableSymbol property), 48
- library\_ordinal (cle.backends.macho.symbol.AbstractMachOSymbol property), 47
- library\_ordinal (cle.backends.macho.symbol.BindingSymbol property), 49
- library\_ordinal (cle.backends.macho.symbol.DyldBoundSymbol property), 49
- library\_ordinal (cle.backends.macho.symbol.SymbolTableSymbol property), 48
- linked\_addr (cle.backends.relocation.Relocation property), 83
- linked\_addr (cle.backends.symbol.Symbol property), 19
- linux\_loader\_object (cle.Loader property), 9
- load() (cle.memory.Clemory method), 95
- load() (cle.memory.ClemoryBase method), 93
- load() (cle.memory.ClemoryTranslator method), 96
- load() (cle.memory.ClemoryView method), 96
- load() (cle.memory.UninitializedClemory method), 97
- load\_args (cle.backends.backend.Backend attribute), 15
- load\_null\_terminated\_bytes() (cle.memory.ClemoryBase method), 93
- load\_symbols\_from\_pdb() (cle.backends.PE method), 43
- Loader (class in cle), 7
- loader (cle.backends.backend.Backend property), 15
- lookup() (in module cle.backends.externs.simdata), 26
- lookup() (in module cle.backends.externs.simdata.simdata), 27
- lp\_start (cle.backends.elf.lsda.ExceptionTableHeader attribute), 39
- LSDAExceptionTable (class in cle.backends.elf.lsda), 39
- ## M
- Machine (cle.backends.coff.CoffFileHeader attribute), 66
- machine (cle.backends.te.HeaderType attribute), 80
- MachO (class in cle.backends.macho.macho), 44
- MachOPointerRelocation (class in cle.backends.macho.binding), 56
- MachOSection (class in cle.backends.macho.section), 49
- MachOSegment (class in cle.backends.macho.segment), 50
- MachOSymbolRelocation (class in cle.backends.macho.binding), 56
- main\_methods (cle.backends.java.soot.Soot property), 77
- main\_object (cle.Loader property), 8

- [make\\_extern\(\)](#) (*cle.backends.externs.ExternObject* method), 24  
[make\\_import\(\)](#) (*cle.backends.externs.ExternObject* method), 24  
[max\\_addr](#) (*cle.backends.backend.Backend* property), 17  
[max\\_addr](#) (*cle.backends.binja.BinjaBin* property), 65  
[max\\_addr](#) (*cle.backends.blob.Blob* property), 66  
[max\\_addr](#) (*cle.backends.elf.compilation\_unit.CompilationUnit* property), 42  
[max\\_addr](#) (*cle.backends.externs.ExternObject* property), 24  
[max\\_addr](#) (*cle.backends.externs.KernelObject* property), 25  
[max\\_addr](#) (*cle.backends.java.soot.Soot* property), 76  
[max\\_addr](#) (*cle.backends.named\_region.NamedRegion* property), 23  
[max\\_addr](#) (*cle.backends.region.Region* property), 21  
[max\\_addr](#) (*cle.backends.regions.Regions* property), 20  
[max\\_addr](#) (*cle.backends.tls.elf\_tls.ELFTLSObject* property), 90  
[max\\_addr](#) (*cle.backends.tls.pe\_tls.PETLSObject* property), 91  
[max\\_addr](#) (*cle.backends.xbe.XBE* property), 82  
[max\\_addr](#) (*cle.Loader* property), 9  
[max\\_addr](#) (*cle.memory.Clemory* attribute), 95  
[max\\_addr](#) (*cle.memory.UninitializedClemory* attribute), 97  
[max\\_offset](#) (*cle.backends.region.Region* property), 22  
[max\\_valid\\_pointer](#) (*cle.backends.macho.structs.dyld\_chained\_structs* attribute), 63  
[md5](#) (*cle.backends.backend.Backend* property), 15  
[MEM\\_EXECUTE](#) (*cle.backends.coff.IMAGE\_SCN* attribute), 67  
[MEM\\_READ](#) (*cle.backends.coff.IMAGE\_SCN* attribute), 67  
[MEM\\_WRITE](#) (*cle.backends.coff.IMAGE\_SCN* attribute), 67  
[memory](#) (*cle.backends.backend.Backend* attribute), 15  
[memory](#) (*cle.Loader* property), 8  
[memory\\_ro\\_view](#) (*cle.Loader* property), 8  
[MemoryVariable](#) (class in *cle.backends.elf.variable*), 34  
[memsize](#) (*cle.backends.region.Region* attribute), 21  
[meta\\_regions](#) (*cle.backends.backend.Backend* attribute), 15  
[MetaELF](#) (class in *cle.backends.elf*), 31  
[MH\\_CIGAM](#) (*cle.backends.macho.macho.MachO* attribute), 44  
[MH\\_CIGAM\\_64](#) (*cle.backends.macho.macho.MachO* attribute), 44  
[MH\\_MAGIC](#) (*cle.backends.macho.macho.MachO* attribute), 44  
[MH\\_MAGIC\\_64](#) (*cle.backends.macho.macho.MachO* attribute), 44  
[min\\_addr](#) (*cle.backends.backend.Backend* property), 17  
[min\\_addr](#) (*cle.backends.binja.BinjaBin* property), 65  
[min\\_addr](#) (*cle.backends.blob.Blob* property), 65  
[min\\_addr](#) (*cle.backends.elf.compilation\_unit.CompilationUnit* property), 42  
[min\\_addr](#) (*cle.backends.macho.macho.MachO* property), 44  
[min\\_addr](#) (*cle.backends.named\_region.NamedRegion* property), 23  
[min\\_addr](#) (*cle.backends.region.Region* property), 21  
[min\\_addr](#) (*cle.backends.xbe.XBE* property), 82  
[min\\_addr](#) (*cle.Loader* property), 9  
[min\\_addr](#) (*cle.memory.Clemory* attribute), 95  
[min\\_addr](#) (*cle.memory.UninitializedClemory* attribute), 97  
[min\\_offset\(\)](#) (*cle.backends.region.Region* method), 22  
[Minidump](#) (class in *cle.backends.minidump*), 77  
[MinidumpMissingStreamError](#), 77  
[MinidumpThread](#) (class in *cle.backends.tls.minidump\_tls*), 92  
[MinidumpThreadManager](#) (class in *cle.backends.tls*), 88  
[MinidumpThreadManager](#) (class in *cle.backends.tls.minidump\_tls*), 91  
[MipsGlobalReloc](#) (class in *cle.backends.elf.relocation.generic*), 85  
[MipsLocalReloc](#) (class in *cle.backends.elf.relocation.generic*), 85  
[missing\\_dependencies](#) (*cle.Loader* property), 9  
[module](#)  
     [cle.address\\_translator](#), 98  
     [cle.backends.backend](#), 13  
     [cle.backends.binja](#), 64  
     [cle.backends.blob](#), 65  
     [cle.backends.cgc](#), 72  
     [cle.backends.cgc.backedcgc](#), 73  
     [cle.backends.cgc.cgc](#), 73  
     [cle.backends.coff](#), 66  
     [cle.backends.elf.compilation\\_unit](#), 41  
     [cle.backends.elf.hashtable](#), 39  
     [cle.backends.elf.ldda](#), 39  
     [cle.backends.elf.relocation](#), 83  
     [cle.backends.elf.relocation.amd64](#), 86  
     [cle.backends.elf.relocation.arm](#), 86  
     [cle.backends.elf.relocation.arm64](#), 86  
     [cle.backends.elf.relocation.elfreloc](#), 83  
     [cle.backends.elf.relocation.generic](#), 83  
     [cle.backends.elf.relocation.i386](#), 86  
     [cle.backends.elf.relocation.mips](#), 86  
     [cle.backends.elf.relocation.ppc](#), 85  
     [cle.backends.elf.relocation.ppc64](#), 86  
     [cle.backends.elf.relocation.s390x](#), 86  
     [cle.backends.elf.subprogram](#), 40  
     [cle.backends.elf.variable](#), 34  
     [cle.backends.elf.variable\\_type](#), 35  
     [cle.backends.externs](#), 23  
     [cle.backends.externs.simdata](#), 25

cle.backends.externs.simdata.common, 27  
 cle.backends.externs.simdata.simdata, 26  
 cle.backends.ihex, 74  
 cle.backends.java, 76  
 cle.backends.java.android\_lifecycle, 74  
 cle.backends.java.apk, 74  
 cle.backends.java.jar, 75  
 cle.backends.java.soot, 76  
 cle.backends.macho.binding, 51  
 cle.backends.macho.section, 49  
 cle.backends.macho.segment, 50  
 cle.backends.macho.structs, 56  
 cle.backends.macho.symbol, 47  
 cle.backends.minidump, 77  
 cle.backends.named\_region, 23  
 cle.backends.pe.regions, 43  
 cle.backends.pe.relocation, 86  
 cle.backends.pe.relocation.arm, 87  
 cle.backends.pe.relocation.generic, 86  
 cle.backends.pe.relocation.mips, 87  
 cle.backends.pe.relocation.pereloc, 86  
 cle.backends.pe.relocation.riscv, 87  
 cle.backends.pe.symbol, 43  
 cle.backends.region, 21  
 cle.backends.regions, 20  
 cle.backends.relocation, 82  
 cle.backends.static\_archive, 78  
 cle.backends.symbol, 18  
 cle.backends.te, 79  
 cle.backends.tls, 87  
 cle.backends.tls.elf\_tls, 89  
 cle.backends.tls.elfcore\_tls, 91  
 cle.backends.tls.minidump\_tls, 91  
 cle.backends.tls.pe\_tls, 90  
 cle.backends.tls.tls\_object, 88  
 cle.backends.uefi\_firmware, 78  
 cle.backends.xbe, 81  
 cle.errors, 92  
 cle.gdb, 92  
 cle.memory, 93  
 cle.patched\_stream, 97  
 cle.utils, 99

## N

n\_opcode\_add\_addr\_uleb() (in module *cle.backends.macho.binding*), 54  
 n\_opcode\_do\_bind() (in module *cle.backends.macho.binding*), 54  
 n\_opcode\_do\_bind\_add\_addr\_imm\_scaled() (in module *cle.backends.macho.binding*), 55  
 n\_opcode\_do\_bind\_add\_addr\_uleb() (in module *cle.backends.macho.binding*), 55  
 n\_opcode\_do\_bind\_uleb\_times\_skipping\_uleb() (in module *cle.backends.macho.binding*), 55  
 n\_opcode\_done() (in module *cle.backends.macho.binding*), 52  
 n\_opcode\_set\_addend\_sleb() (in module *cle.backends.macho.binding*), 53  
 n\_opcode\_set\_dylib\_ordinal\_imm() (in module *cle.backends.macho.binding*), 52  
 n\_opcode\_set\_dylib\_ordinal\_uleb() (in module *cle.backends.macho.binding*), 52  
 n\_opcode\_set\_dylib\_special\_imm() (in module *cle.backends.macho.binding*), 53  
 n\_opcode\_set\_segment\_and\_offset\_uleb() (in module *cle.backends.macho.binding*), 54  
 n\_opcode\_set\_trailing\_flags\_imm() (in module *cle.backends.macho.binding*), 53  
 n\_opcode\_set\_type\_imm() (in module *cle.backends.macho.binding*), 53  
 name (*cle.backends.backend.FunctionHint* attribute), 13  
 Name (*cle.backends.coff.CoffSectionTableEntry* attribute), 67  
 Name (*cle.backends.coff.CoffSymbolTableEntry* attribute), 68  
 name (*cle.backends.externs.simdata.SimData* attribute), 25  
 name (*cle.backends.externs.simdata.simdata.SimData* attribute), 26  
 name (*cle.backends.uefi\_firmware.UefiModulePending* attribute), 78  
 name\_offset (*cle.backends.macho.structs.dyld\_chained\_import* attribute), 62  
 name\_offset (*cle.backends.macho.structs.dyld\_chained\_import\_addend* attribute), 62  
 name\_offset (*cle.backends.macho.structs.dyld\_chained\_import\_addend64* attribute), 63  
 name\_offset (*cle.backends.macho.structs.DyldImportStruct* attribute), 62  
 NamedRegion (class in *cle.backends.named\_region*), 23  
 ncnds (*cle.backends.macho.macho.MachO* attribute), 44  
 new\_thread() (*cle.backends.tls.elfcore\_tls.ELFCoreThreadManager* method), 91  
 new\_thread() (*cle.backends.tls.ELFCoreThreadManager* method), 88  
 new\_thread() (*cle.backends.tls.minidump\_tls.MinidumpThreadManager* method), 91  
 new\_thread() (*cle.backends.tls.MinidumpThreadManager* method), 88  
 new\_thread() (*cle.backends.tls.ThreadManager* method), 87  
 new\_thread() (*cle.backends.tls.tls\_object.ThreadManager* method), 88  
 next (*cle.backends.macho.structs.dyld\_chained\_ptr\_64\_bind* attribute), 60  
 next (*cle.backends.macho.structs.dyld\_chained\_ptr\_64\_rebase* attribute), 60  
 next (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind*

- attribute), 58
- next (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind attribute), 59
- next (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind attribute), 57
- next (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind attribute), 59
- next (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_rebase attribute), 60
- NONE (cle.backends.elf.metaelf.Relro attribute), 32
- number\_of\_line\_numbers (cle.backends.te.SectionHeaderType attribute), 81
- number\_of\_relocations (cle.backends.te.SectionHeaderType attribute), 81
- number\_of\_sections (cle.backends.te.HeaderType attribute), 80
- NumberOfAuxSymbols (cle.backends.coff.CoffSymbolTableEntry attribute), 68
- NumberOfLinenumbers (cle.backends.coff.CoffSectionTableEntry attribute), 67
- NumberOfRelocations (cle.backends.coff.CoffSectionTableEntry attribute), 67
- NumberOfSections (cle.backends.coff.CoffFileHeader attribute), 66
- NumberOfSymbols (cle.backends.coff.CoffFileHeader attribute), 66
- O**
- occupies\_memory (cle.backends.elf.regions.ELFSection property), 33
- offset\_to\_addr() (cle.backends.backend.Backend method), 17
- offset\_to\_addr() (cle.backends.externs.ExternSegment method), 23
- offset\_to\_addr() (cle.backends.region.Region method), 21
- only\_contains\_uninitialized\_data (cle.backends.coff.CoffSection property), 70
- only\_contains\_uninitialized\_data (cle.backends.elf.regions.ELFSection property), 34
- only\_contains\_uninitialized\_data (cle.backends.macho.section.MachOSection property), 50
- only\_contains\_uninitialized\_data (cle.backends.pe.regions.PESection property), 43
- only\_contains\_uninitialized\_data (cle.backends.region.EmptySegment property), 24
- only\_contains\_uninitialized\_data (cle.backends.region.Section property), 22
- only\_contains\_uninitialized\_data (cle.backends.xbe.XBESection property), 82
- ordinal (cle.backends.macho.structs.dyld\_chained\_ptr\_64\_bind attribute), 60
- ordinal (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind attribute), 58
- ordinal (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind attribute), 59
- ordinal (cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind attribute), 59
- original\_main\_object (cle.Loader property), 8
- os\_proc (cle.backends.elf.symbol\_type.ELFSymbolType property), 32
- owner (cle.backends.macho.symbol.AbstractMachOSymbol attribute), 47
- owner (cle.backends.macho.symbol.DyldBoundSymbol attribute), 49
- owner (cle.backends.symbol.Symbol attribute), 19
- owner\_obj (cle.backends.relocation.Relocation property), 83
- owner\_obj (cle.backends.symbol.Symbol property), 20
- P**
- pack() (cle.memory.CMemoryBase method), 94
- PACK\_FORMAT (cle.backends.coff.CoffRelocation attribute), 70
- PACK\_FORMAT (cle.backends.coff.CoffRelocationADDR32NB attribute), 71
- PACK\_FORMAT (cle.backends.coff.CoffRelocationADDR64 attribute), 71
- PACK\_FORMAT (cle.backends.coff.CoffRelocationSECRET attribute), 71
- PACK\_FORMAT (cle.backends.coff.CoffRelocationSECTION attribute), 71
- pack\_word() (cle.memory.CMemoryBase method), 94
- page\_count (cle.backends.macho.structs.dyld\_chained\_starts\_in\_segment attribute), 64
- page\_size (cle.backends.macho.structs.dyld\_chained\_starts\_in\_segment attribute), 63
- page\_start (cle.backends.macho.structs.dyld\_chained\_starts\_in\_segment attribute), 64
- parse\_lc\_str() (cle.backends.macho.macho.MachO method), 45
- parse\_lsda() (cle.backends.elf.lsda.LSDAExceptionTable method), 39
- parse\_record() (cle.backends.ihex.Hex static method), 74
- PARTIAL (cle.backends.elf.metaelf.Relro attribute), 32
- PatchedStream (class in cle.patched\_stream), 97
- PE (class in cle.backends), 42

- pe\_image (*cle.backends.uefi\_firmware.UefiModulePending* attribute), 78  
 PEReloc (*class in cle.backends.pe.relocation.pereloc*), 86  
 perform\_irelative\_relocs() (*cle.Loader* method), 12  
 PESection (*class in cle.backends.pe.regions*), 43  
 PETHreadManager (*class in cle.backends.tls*), 88  
 PETHreadManager (*class in cle.backends.tls.pe\_tls*), 90  
 PETLSObject (*class in cle.backends.tls.pe\_tls*), 90  
 physical\_address\_virtual\_size  
     (*cle.backends.te.SectionHeaderType* attribute), 81  
 plt (*cle.backends.elf.MetaELF* property), 31  
 plt (*cle.backends.macho.macho.MachO* property), 44  
 pointer\_format (*cle.backends.macho.structs.dyld\_chained\_structs.lib\_dyld* property), 64  
 pointer\_to\_line\_numbers  
     (*cle.backends.te.SectionHeaderType* attribute), 81  
 pointer\_to\_raw\_data  
     (*cle.backends.te.SectionHeaderType* attribute), 81  
 pointer\_to\_relocations  
     (*cle.backends.te.SectionHeaderType* attribute), 81  
 PointerToLinenumbers  
     (*cle.backends.coff.CoffSectionTableEntry* attribute), 67  
 PointerToRawData (*cle.backends.coff.CoffSectionTableEntry* attribute), 67  
 PointerToRelocations  
     (*cle.backends.coff.CoffSectionTableEntry* attribute), 67  
 PointerToSymbolTable  
     (*cle.backends.coff.CoffFileHeader* attribute), 66  
 PointerType (*class in cle.backends.elf.variable\_type*), 36  
 PointTo (*class in cle.backends.externs.simdata.common*), 28  
 pointto\_name (*cle.backends.externs.simdata.common.PointerTo* attribute), 28  
 pointto\_precise (*cle.backends.externs.PointerToPrecise* attribute), 25  
 pointto\_type (*cle.backends.externs.simdata.common.PointerTo* attribute), 28  
 PointToPrecise (*class in cle.backends.externs*), 25  
 ppc64\_initial\_rtoc (*cle.backends.elf.MetaELF* property), 31
- R**  
 raw\_list (*cle.backends.regions.Regions* property), 20  
 read() (*cle.memory.ClemoryBase* method), 94  
 read() (*cle.patched\_stream.PatchedStream* method), 98  
 read\_from\_die() (*cle.backends.elf.variable\_type.ArrayType* class method), 38  
 read\_from\_die() (*cle.backends.elf.variable\_type.BaseType* class method), 37  
 read\_from\_die() (*cle.backends.elf.variable\_type.PointerType* class method), 36  
 read\_from\_die() (*cle.backends.elf.variable\_type.StructMember* class method), 38  
 read\_from\_die() (*cle.backends.elf.variable\_type.StructType* class method), 37  
 read\_from\_die() (*cle.backends.elf.variable\_type.TypedefType* class method), 38  
 read\_from\_die() (*cle.backends.elf.variable\_type.VariableType* static method), 36  
 read\_uleb() (*cle.backends.macho.binding.BindingHelper* static method), 52  
 read\_uleb() (*in module cle.backends.macho.binding*), 51  
 rebase (*cle.backends.macho.structs.Arm64e* attribute), 59  
 rebase (*cle.backends.macho.structs.Generic64* attribute), 61  
 rebase() (*cle.backends.backend.Backend* method), 16  
 rebase() (*cle.backends.ELF* method), 30  
 rebase() (*cle.backends.elf.subprogram.LexicalBlock* method), 41  
 rebase() (*cle.backends.elf.subprogram.Subprogram* method), 41  
 rebase() (*cle.backends.externs.ExternObject* method), 24  
 rebase\_at() (*cle.backends.macho.binding.BindingHelper* method), 52  
 rebased\_addr (*cle.backends.elf.variable.MemoryVariable* property), 35  
 rebased\_addr (*cle.backends.elf.variable.Variable* property), 34  
 rebased\_addr (*cle.backends.relocation.Relocation* property), 83  
 rebased\_addr (*cle.backends.symbol.Symbol* property), 19  
 rebased\_addr\_from\_cfa() (*cle.backends.elf.variable.StackVariable* method), 35  
 rebased\_addr\_from\_cfa() (*cle.backends.elf.variable.Variable* method), 34  
 reference\_type (*cle.backends.macho.symbol.SymbolTableSymbol* property), 48  
 referenced\_symbol\_index  
     (*cle.backends.macho.symbol.SymbolTableSymbol* property), 48  
 referenced\_type (*cle.backends.elf.variable\_type.PointerType* property), 36

Region (class in *cle.backends.region*), 21  
 Regions (class in *cle.backends.regions*), 20  
 register() (in module *cle.backends.externs.simdata*), 26  
 register() (in module *cle.backends.externs.simdata.simdata*), 27  
 register\_backend() (in module *cle.backends.backend*), 18  
 register\_object() (*cle.backends.tls.elf\_tls.ELFThreadManager* method), 89  
 register\_object() (*cle.backends.tls.elfcore\_tls.ELFCoreThreadManager* method), 91  
 register\_object() (*cle.backends.tls.ELFCoreThreadManager* method), 88  
 register\_object() (*cle.backends.tls.ELFThreadManager* method), 88  
 register\_object() (*cle.backends.tls.minidump\_tls.MinidumpThreadManager* method), 92  
 register\_object() (*cle.backends.tls.MinidumpThreadManager* method), 88  
 register\_object() (*cle.backends.tls.pe\_tls.PEThreadManager* method), 90  
 register\_object() (*cle.backends.tls.PEThreadManager* method), 88  
 register\_object() (*cle.backends.tls.ThreadManager* method), 87  
 register\_object() (*cle.backends.tls.tls\_object.ThreadManager* method), 88  
 RegisterVariable (class in *cle.backends.elf.variable*), 35  
 REL32 (*cle.backends.coff.IMAGE\_REL\_AMD64* attribute), 68  
 REL32 (*cle.backends.coff.IMAGE\_REL\_I386* attribute), 68  
 relocate() (*cle.backends.backend.Backend* method), 16  
 relocate() (*cle.backends.coff.CoffRelocation* method), 70  
 relocate() (*cle.backends.elf.relocation.generic.GenericCopyReloc* method), 85  
 relocate() (*cle.backends.elf.relocation.generic.GenericIRelativeReloc* method), 84  
 relocate() (*cle.backends.elf.relocation.generic.GenericTLSDescriptorReloc* method), 84  
 relocate() (*cle.backends.elf.relocation.generic.GenericTLSModIdReloc* method), 84  
 relocate() (*cle.backends.elf.relocation.generic.GenericTLSOffsetReloc* method), 29  
 relocate() (*cle.backends.elf.relocation.generic.GenericTLSOffsetReloc* method), 84  
 relocate() (*cle.backends.elf.relocation.generic.MipsLocalReloc* method), 56  
 relocate() (*cle.backends.elf.relocation.generic.MipsLocalReloc* method), 85  
 relocate() (*cle.backends.elf.relocation.generic.RelocTruncate32Mixin* method), 56  
 relocate() (*cle.backends.elf.relocation.generic.RelocTruncate32Mixin* method), 85  
 relocate() (*cle.backends.pe.relocation.generic.IMAGE\_REL\_BASED\_ABSOLUTE* method), 86  
 relocate() (*cle.backends.pe.relocation.pereloc.PEReloc* method), 83  
 relocate() (method), 86  
 relocate() (*cle.backends.relocation.Relocation* method), 83  
 Relocation (class in *cle.backends.relocation*), 82  
 relocations (*cle.backends.coff.CoffParser* attribute), 69  
 relocations() (*cle.backends.externs.PointToPrecisionRelocations* method), 25  
 relocations() (*cle.backends.externs.simdata.common.PointToPrecisionRelocations* method), 28  
 relocations() (*cle.backends.externs.simdata.SimDataRelocations* method), 26  
 relocations() (*cle.backends.externs.simdata.simdata.SimDataRelocations* method), 27  
 RelocGOTMixin (class in *cle.backends.elf.relocation.generic*), 85  
 RelocTruncate32Mixin (class in *cle.backends.elf.relocation.generic*), 85  
 Relro (class in *cle.backends.elf.metaelf*), 31  
 remove() (*cle.backends.regions.Regions* method), 20  
 remove\_backer() (*cle.memory.Clemory* method), 95  
 remove\_backer() (*cle.memory.UninitializedClemory* method), 97  
 reserved (*cle.backends.macho.structs.dyld\_chained\_import\_addend64* attribute), 63  
 resolve() (*cle.backends.elf.relocation.generic.RelocGOTMixin* method), 85  
 resolve() (*cle.backends.relocation.Relocation* method), 83  
 resolve() (*cle.backends.symbol.Symbol* method), 19  
 resolve\_forwarder() (*cle.backends.pe.symbol.WinSymbol* method), 43  
 resolve\_forwarder() (*cle.backends.symbol.Symbol* method), 20  
 resolve\_reference\_addr() (in module *cle.backends.elf.variable\_type*), 35  
 resolve\_symbol() (*cle.backends.elf.relocation.generic.GenericCopyReloc* method), 85  
 resolve\_symbol() (*cle.backends.elf.relocation.generic.GenericTLSDescriptorReloc* method), 84  
 resolve\_symbol() (*cle.backends.elf.relocation.generic.MipsLocalReloc* method), 85  
 resolve\_symbol() (*cle.backends.externs.simdata.common.SimDataSimpleReloc* method), 29  
 resolve\_symbol() (*cle.backends.macho.binding.MachOPointerRelocation* method), 85  
 resolve\_symbol() (*cle.backends.macho.binding.MachOSymbolRelocation* method), 85  
 resolve\_symbol() (*cle.backends.pe.relocation.pereloc.PEReloc* method), 83  
 resolve\_symbol() (*cle.backends.relocation.Relocation* method), 83

- resolved (*cle.backends.relocation.Relocation* attribute), 83
- resolvedby (*cle.backends.relocation.Relocation* attribute), 83
- RESOLVER\_ADDR (*cle.backends.elf.relocation.generic.GeneralizedRelocation* attribute), 84
- reverse\_plt (*cle.backends.elf.MetaELF* property), 31
- reverse\_plt (*cle.backends.macho.macho.MachO* property), 44
- reverse\_stubs (*cle.backends.macho.macho.MachO* property), 44
- roundup() (in module *cle.backends.tls.elf\_tls*), 89
- ## S
- S (*cle.backends.macho.macho.MachO* attribute), 45
- SECRET (*cle.backends.coff.IMAGE\_REL\_AMD64* attribute), 68
- SECRET (*cle.backends.coff.IMAGE\_REL\_I386* attribute), 68
- Section (class in *cle.backends.region*), 22
- SECTION (*cle.backends.coff.IMAGE\_REL\_AMD64* attribute), 68
- SECTION (*cle.backends.coff.IMAGE\_REL\_I386* attribute), 68
- section\_name (*cle.backends.macho.symbol.SymbolTableSymbol* property), 48
- section\_name (*cle.backends.te.SectionHeaderType* attribute), 81
- SectionHeaderType (class in *cle.backends.te*), 80
- SectionNumber (*cle.backends.coff.CoffSymbolTableEntry* attribute), 68
- sections (*cle.backends.backend.Backend* property), 16
- sections (*cle.backends.coff.CoffParser* attribute), 69
- seek() (*cle.memory.ClemoryBase* method), 94
- seek() (*cle.patched\_stream.PatchedStream* method), 98
- seg\_count (*cle.backends.macho.structs.dyld\_chained\_starts\_in\_image* attribute), 63
- seg\_info\_offset (*cle.backends.macho.structs.dyld\_chained\_starts\_in\_image* attribute), 63
- Segment (class in *cle.backends.region*), 22
- segment\_name (*cle.backends.macho.symbol.SymbolTableSymbol* property), 48
- segment\_offset (*cle.backends.macho.structs.dyld\_chained\_starts\_in\_image* attribute), 63
- segments (*cle.backends.backend.Backend* property), 16
- set\_arch() (*cle.backends.backend.Backend* method), 16
- set\_arch() (*cle.backends.macho.macho.MachO* method), 44
- set\_got\_entry() (*cle.backends.binja.BinjaBin* method), 65
- set\_load\_args() (*cle.backends.backend.Backend* method), 16
- sha256 (*cle.backends.backend.Backend* property), 15
- SHF\_ALLOC (*cle.backends.elf.regions.ELFSection* attribute), 33
- SHF\_EXECINSTR (*cle.backends.elf.regions.ELFSection* attribute), 33
- SHF\_STRINGS (*cle.backends.elf.regions.ELFSection* attribute), 33
- SHF\_WRITE (*cle.backends.elf.regions.ELFSection* attribute), 33
- SHT\_NULL (*cle.backends.elf.regions.ELFSection* attribute), 33
- sign\_extended\_addend (*cle.backends.macho.structs.Arm64e* property), 60
- signature (*cle.backends.te.HeaderType* attribute), 80
- SimData (class in *cle.backends.externs.simdata*), 25
- SimData (class in *cle.backends.externs.simdata.simdata*), 26
- SimDataSimpleRelocation (class in *cle.backends.externs.simdata.common*), 29
- size (*cle.backends.backend.ExceptionHandling* attribute), 14
- size (*cle.backends.backend.FunctionHint* attribute), 13
- size\_of\_raw\_data (*cle.backends.te.SectionHeaderType* attribute), 81
- size\_of\_cmds (*cle.backends.macho.macho.MachO* attribute), 44
- SizeOfOptionalHeader (*cle.backends.coff.CoffFileHeader* attribute), 66
- SizeOfRawData (*cle.backends.coff.CoffSectionTableEntry* attribute), 67
- Soot (class in *cle.backends.java.soot*), 76
- sort (*cle.backends.elf.variable.MemoryVariable* property), 35
- sort (*cle.backends.elf.variable.RegisterVariable* property), 35
- sort (*cle.backends.elf.variable.StackVariable* property), 35
- sort (*cle.backends.elf.variable.Variable* property), 34
- source (*cle.backends.backend.FunctionHint* attribute), 13
- split\_backer() (*cle.memory.Clemory* method), 95
- split\_backer() (*cle.memory.UninitializedClemory* method), 97
- StackVariable (class in *cle.backends.elf.variable*), 35
- start\_addr (*cle.backends.backend.ExceptionHandling* attribute), 14
- starts\_offset (*cle.backends.macho.structs.dyld\_chained\_fixups\_header* attribute), 63
- STATIC (*cle.backends.coff.IMAGE\_SYM\_CLASS* attribute), 67
- static\_size() (*cle.backends.externs.simdata.common.PointTo* class method), 28
- static\_size() (*cle.backends.externs.simdata.common.StaticData* class method), 28

- `class method`), 27
  - `static_size()` (*cle.backends.externs.simdata.common.StaticWord* *class method*), 28
  - `static_size()` (*cle.backends.externs.simdata.SimData* *class method*), 25
  - `static_size()` (*cle.backends.externs.simdata.simdata.SimData* *class method*), 26
  - `StaticArchive` (*class in cle.backends.static\_archive*), 78
  - `StaticData` (*class in cle.backends.externs.simdata.common*), 27
  - `StaticWord` (*class in cle.backends.externs.simdata.common*), 27
  - `StorageClass` (*cle.backends.coff.CoffSymbolTableEntry* *attribute*), 68
  - `store()` (*cle.memory.Clemory* *method*), 95
  - `store()` (*cle.memory.ClemoryBase* *method*), 93
  - `store()` (*cle.memory.ClemoryTranslator* *method*), 96
  - `store()` (*cle.memory.ClemoryView* *method*), 96
  - `store()` (*cle.memory.UninitializedClemory* *method*), 97
  - `stream_or_path()` (*in module cle.utils*), 99
  - `stripped_size` (*cle.backends.te.HeaderType* *attribute*), 80
  - `StructMember` (*class in cle.backends.elf.variable\_type*), 37
  - `StructType` (*class in cle.backends.elf.variable\_type*), 37
  - `STT_COMMON` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_FILE` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_FUNC` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_GNU_IFUNC` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_HIOS` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_HIPROC` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_LOOS` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_LOPROC` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_NOTYPE` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_OBJECT` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_SECTION` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `STT_TLS` (*cle.backends.elf.symbol\_type.ELFSymbolType* *attribute*), 32
  - `stubs` (*cle.backends.macho.macho.MachO* *property*), 44
  - `Subprogram` (*class in cle.backends.elf.subprogram*), 41
  - `subsystem` (*cle.backends.te.HeaderType* *attribute*), 80
  - `subtype` (*cle.backends.elf.symbol.ELFSymbol* *property*), 32
  - `supported_die()` (*cle.backends.elf.variable\_type.VariableType* *static method*), 36
  - `supported_filetypes` (*cle.backends.cgc.CGC* *attribute*), 72
  - `supported_filetypes` (*cle.backends.cgc.cgc.CGC* *attribute*), 73
  - `supported_filetypes` (*cle.backends.elf.MetaELF* *attribute*), 31
  - `sym_type` (*cle.backends.macho.symbol.SymbolTableSymbol* *property*), 48
  - `Symbol` (*class in cle.backends.symbol*), 18
  - `symbol` (*cle.backends.macho.binding.MachOSymbolRelocation* *attribute*), 56
  - `symbol_name_to_idx` (*cle.backends.coff.CoffParser* *attribute*), 69
  - `SymbolList` (*class in cle.backends.macho.macho*), 46
  - `symbols` (*cle.backends.coff.CoffParser* *attribute*), 69
  - `symbols` (*cle.Loader* *property*), 11
  - `symbols_by_addr` (*cle.backends.backend.Backend* *property*), 16
  - `symbols_by_name` (*cle.backends.ELF* *property*), 30
  - `symbols_format` (*cle.backends.macho.structs.dyld\_chained\_fixups\_header* *attribute*), 63
  - `symbols_offset` (*cle.backends.macho.structs.dyld\_chained\_fixups\_header* *attribute*), 63
  - `SymbolSubType` (*class in cle.backends.symbol*), 18
  - `SymbolTableIndex` (*cle.backends.coff.CoffRelocationTableEntry* *attribute*), 69
  - `SymbolTableSymbol` (*class in cle.backends.macho.symbol*), 47
  - `SymbolType` (*class in cle.backends.macho.symbol*), 47
  - `SymbolType` (*class in cle.backends.symbol*), 18
- ## T
- `target` (*cle.backends.macho.structs.dyld\_chained\_ptr\_64\_rebase* *attribute*), 60
  - `target` (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_rebase* *attribute*), 57
  - `target` (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_rebase* *attribute*), 58
  - `TE` (*class in cle.backends.te*), 81
  - `te_image` (*cle.backends.uefi\_firmware.UefiModulePending* *attribute*), 78
  - `tell()` (*cle.memory.ClemoryBase* *method*), 94
  - `tell()` (*cle.patched\_stream.PatchedStream* *method*), 98
  - `thread_pointer` (*cle.backends.tls.elf\_tls.ELFTLSObject* *property*), 89
  - `thread_pointer` (*cle.backends.tls.pe\_tls.PETLSObject* *property*), 91
  - `thread_registers()` (*cle.backends.backend.Backend* *method*), 17

`thread_registers()` (*cle.backends.cgc.BackedCGC* type (*cle.backends.elf.variable\_type.StructMember* method), 73 property), 38  
`thread_registers()` (*cle.backends.cgc.backedcgc.BackedCGC* type (*cle.backends.elf.variable\_type.TypedefType* method), 74 property), 38  
`thread_registers()` (*cle.backends.elf.ELFCore* type (*cle.backends.externs.simdata.common.PointTo* attribute), 31 attribute), 28  
`thread_registers()` (*cle.backends.minidump.Minidump* type (*cle.backends.externs.simdata.common.StaticData* method), 77 attribute), 27  
`ThreadManager` (class in *cle.backends.tls*), 87 type (*cle.backends.externs.simdata.common.StaticWord* attribute), 28  
`ThreadManager` (class in *cle.backends.tls.tls\_object*), 88 type (*cle.backends.externs.simdata.SimData* attribute), 25  
`threads` (*cle.backends.backend.Backend* property), 17 type (*cle.backends.externs.simdata.simdata.SimData* attribute), 26  
`threads` (*cle.backends.cgc.BackedCGC* property), 73 type (*cle.backends.macho.section.MachOSection* property), 50  
`threads` (*cle.backends.cgc.backedcgc.BackedCGC* property), 74 type (*cle.backends.symbol.Symbol* property), 19  
`threads` (*cle.backends.elf.ELFCore* property), 31 `TYPE_FUNCTION` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`threads` (*cle.backends.minidump.Minidump* property), 77 `TYPE_FUNCTION` (*cle.backends.symbol.SymbolType* attribute), 18  
`TimeStamp` (*cle.backends.coff.CoffFileHeader* attribute), 66 `TYPE_FUNCTION_OR_OBJECT` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`tls` (*cle.Loader* property), 9 `TYPE_FUNCTION_OR_OBJECT` (*cle.backends.symbol.SymbolType* attribute), 18  
`TLSObject` (class in *cle.backends.tls*), 87 `TYPE_FUNCTION_OR_OBJECT` (*cle.backends.symbol.SymbolType* attribute), 18  
`TLSObject` (class in *cle.backends.tls.tls\_object*), 89 `TYPE_FUNCTION_OR_OBJECT` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`to_base_type()` (*cle.backends.elf.symbol\_type.ELFSymbolType* method), 33 `TYPE_FUNCTION_OR_OBJECT` (*cle.backends.symbol.SymbolType* attribute), 18  
`to_base_type()` (*cle.backends.symbol.SymbolSubType* method), 18 `TYPE_NONE` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`to_linked_va()` (*cle.address\_translator.AddressTranslator* method), 99 `TYPE_NONE` (*cle.backends.symbol.SymbolType* attribute), 18  
`to_lva()` (*cle.address\_translator.AddressTranslator* method), 98 `TYPE_OBJECT` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`to_mapped_va()` (*cle.address\_translator.AddressTranslator* method), 99 `TYPE_OBJECT` (*cle.backends.symbol.SymbolType* attribute), 18  
`to_mva()` (*cle.address\_translator.AddressTranslator* method), 98 `TYPE_OTHER` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`to_raw()` (*cle.address\_translator.AddressTranslator* method), 99 `TYPE_OTHER` (*cle.backends.symbol.SymbolType* attribute), 18  
`to_relative_va()` (*cle.address\_translator.AddressTranslator* method), 99 `TYPE_OTHER` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`to_rva()` (*cle.address\_translator.AddressTranslator* method), 99 `TYPE_OTHER` (*cle.backends.symbol.SymbolType* attribute), 18  
`to_va()` (*cle.address\_translator.AddressTranslator* method), 99 `TYPE_SECTION` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`TOCRelocation` (class in *cle.backends.externs*), 24 `TYPE_SECTION` (*cle.backends.symbol.SymbolType* attribute), 18  
`ttype_encoding` (*cle.backends.elf.lsd.ExceptionTableHeader* attribute), 39 `TYPE_TLS_OBJECT` (*cle.backends.macho.symbol.SymbolType* attribute), 47  
`ttype_offset` (*cle.backends.elf.lsd.ExceptionTableHeader* attribute), 39 `TYPE_TLS_OBJECT` (*cle.backends.symbol.SymbolType* attribute), 18  
`type` (*cle.backends.backend.ExceptionHandling* attribute), 14 `TypedefType` (class in *cle.backends.elf.variable\_type*), 38  
`Type` (*cle.backends.coff.CoffRelocationTableEntry* attribute), 69 **U**  
`Type` (*cle.backends.coff.CoffSymbolTableEntry* attribute), 68 `UefiDriverLoadError`, 78  
`type` (*cle.backends.elf.variable.Variable* property), 34 `UefiFirmware` (class in *cle.backends.uefi\_firmware*), 78

- UefiModuleMixin (class in *cle.backends.uefi\_firmware*), 79
- UefiModulePending (class in *cle.backends.uefi\_firmware*), 78
- UefiPE (class in *cle.backends.uefi\_firmware*), 79
- UefiTE (class in *cle.backends.uefi\_firmware*), 79
- UninitializedClemory (class in *cle.memory*), 97
- UnionType (class in *cle.backends.elf.variable\_type*), 37
- unpack() (*cle.memory.ClemoryBase* method), 93
- unpack\_target (*cle.backends.macho.structs.Arm64e* property), 60
- unpack\_word() (*cle.memory.ClemoryBase* method), 93
- unpacked\_name (*cle.backends.backend.Backend* attribute), 15
- unpackedTarget (*cle.backends.macho.structs.dyld\_chained\_import* property), 60
- user\_thread\_pointer (*cle.backends.tls.elf\_tls.ELFTLSObject* property), 89
- user\_thread\_pointer (*cle.backends.tls.pe\_tls.PETLSObject* property), 91
- ## V
- vaddr (*cle.backends.region.Region* attribute), 21
- value (*cle.backends.binja.BinjaReloc* property), 64
- value (*cle.backends.coff.CoffRelocationADDR32NB* property), 71
- value (*cle.backends.coff.CoffRelocationADDR64* property), 71
- value (*cle.backends.coff.CoffRelocationDIR32* property), 70
- value (*cle.backends.coff.CoffRelocationDIR32NB* property), 71
- value (*cle.backends.coff.CoffRelocationREL32* property), 70
- value (*cle.backends.coff.CoffRelocationSECREL* property), 71
- value (*cle.backends.coff.CoffRelocationSECTION* property), 71
- Value (*cle.backends.coff.CoffSymbolTableEntry* attribute), 68
- value (*cle.backends.elf.relocation.elfreloc.ELFR reloc* property), 83
- value (*cle.backends.elf.relocation.generic.GenericAbsoluteAddendReloc* property), 84
- value (*cle.backends.elf.relocation.generic.GenericAbsoluteReloc* property), 85
- value (*cle.backends.elf.relocation.generic.GenericJumpslotReloc* property), 84
- value (*cle.backends.elf.relocation.generic.GenericPCRelativeAddendReloc* property), 84
- value (*cle.backends.elf.relocation.generic.GenericRelativeReloc* property), 85
- value (*cle.backends.elf.relocation.generic.GenericTLSOffsetReloc* property), 84
- value (*cle.backends.externs.simdata.common.SimDataSimpleRelocation* property), 29
- value (*cle.backends.externs.TOCRelocation* property), 24
- value (*cle.backends.macho.binding.MachOPointerRelocation* property), 56
- value (*cle.backends.macho.binding.MachOSymbolRelocation* property), 56
- value (*cle.backends.macho.symbol.SymbolTableSymbol* property), 48
- value (*cle.backends.pe.relocation.generic.IMAGE\_REL\_BASED\_DIR64* property), 87
- value (*cle.backends.pe.relocation.generic.IMAGE\_REL\_BASED\_HIGHADJ* property), 87
- value (*cle.backends.pe.relocation.generic.IMAGE\_REL\_BASED\_HIGHLINK* property), 87
- value (*cle.backends.pe.relocation.generic.IMAGE\_REL\_BASED\_HIGHLINK* property), 87
- value (*cle.backends.pe.relocation.generic.IMAGE\_REL\_BASED\_LOW* property), 87
- value (*cle.backends.pe.relocation.pereloc.PEReloc* property), 86
- value (*cle.backends.relocation.Relocation* property), 83
- value (*cle.backends.tls.InternalTLSRelocation* property), 87
- value (*cle.backends.tls.tls\_object.InternalTLSRelocation* property), 89
- value() (*cle.backends.externs.simdata.common.PointTo* method), 28
- value() (*cle.backends.externs.simdata.common.StaticData* method), 27
- value() (*cle.backends.externs.simdata.common.StaticWord* method), 28
- value() (*cle.backends.externs.simdata.SimData* method), 25
- value() (*cle.backends.externs.simdata.simdata.SimData* method), 26
- Variable (class in *cle.backends.elf.variable*), 34
- variables (*cle.backends.ELF* attribute), 29
- VariableType (class in *cle.backends.elf.variable\_type*), 35
- virtual\_address (*cle.backends.te.SectionHeaderType* attribute), 81
- VirtualAddress (*cle.backends.coff.CoffRelocationTableEntry* attribute), 69
- VirtualAddress (*cle.backends.coff.CoffSectionTableEntry* attribute), 67
- VirtualSize (*cle.backends.coff.CoffSectionTableEntry* attribute), 67
- ## W
- weak\_import (*cle.backends.macho.structs.dyld\_chained\_import*

---

*attribute*), 62  
`weak_import` (*cle.backends.macho.structs.dyld\_chained\_import\_addend*  
*attribute*), 62  
`weak_import` (*cle.backends.macho.structs.dyld\_chained\_import\_addend64*  
*attribute*), 63  
`weak_import` (*cle.backends.macho.structs.DyldImportStruct*  
*attribute*), 62  
`WinSymbol` (*class in cle.backends.pe.symbol*), 43  
`word` (*cle.backends.externs.simdata.common.StaticWord*  
*attribute*), 28  
`wordsize` (*cle.backends.externs.simdata.common.StaticWord*  
*attribute*), 28

## X

`XBE` (*class in cle.backends.xbe*), 82  
`XBESection` (*class in cle.backends.xbe*), 81

## Z

`zero` (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind*  
*attribute*), 58  
`zero` (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_auth\_bind24*  
*attribute*), 59  
`zero` (*cle.backends.macho.structs.dyld\_chained\_ptr\_arm64e\_bind*  
*attribute*), 59