SCons

API Documentation

August 16, 2010

Contents

1 Package SCons

SCons

The main package for the SCons software construction utility. **Version:** 2.0.1

Date: 2010/08/16 23:02:40

1.1 Modules

1.2 Variables

Name	Description
__build__	**Value:** `'r5134'`
__buildsys__	**Value:** `'cooldog'`
__developer__	**Value:** `'bdeegan'`
__package__	**Value:** `'SCons'`
__revision__	**Value:** `'src/engine/SCons/__init__.py 5134 2010/08/16 23:02:40 bd...`

2 Module SCons.Action

SCons.Action

This encapsulates information about executing any sort of action that can build one or more target Nodes (typically files) from one or more source Nodes (also typically files) given a specific Environment.

The base class here is ActionBase. The base class supplies just a few OO utility methods and some generic methods for displaying information about an Action in response to the various commands that control printing.

A second-level base class is _ActionAction. This extends ActionBase by providing the methods that can be used to show and perform an action. True Action objects will subclass _ActionAction; Action factory class objects will subclass ActionBase.

The heavy lifting is handled by subclasses for the different types of actions we might execute:

> CommandAction CommandGeneratorAction FunctionAction ListAction

The subclasses supply the following public interface methods used by other modules:

__call__() THE public interface, "calling" an Action object executes the command or Python function. This also takes care of printing a pre-substitution command for debugging purposes.

get_contents() Fetches the "contents" of an Action for signature calculation plus the varlist. This is what gets MD5 checksummed to decide if a target needs to be rebuilt because its action changed.

genstring() Returns a string representation of the Action *without* command substitution, but allows a CommandGeneratorAction to generate the right action based on the specified target, source and env. This is used by the Signature subsystem (through the Executor) to obtain an (imprecise) representation of the Action operation for informative purposes.

Subclasses also supply the following methods for internal use within this module:

__str__() Returns a string approximation of the Action; no variable substitution is performed.

execute() The internal method that really, truly, actually handles the execution of a command or Python function. This is used so that the __call__() methods can take care of displaying any pre-substitution representations, and *then* execute an action without worrying about the specific Actions involved.

get_presig() Fetches the "contents" of a subclass for signature calculation. The varlist is added to this to produce the Action's contents.

strfunction() Returns a substituted string representation of the Action. This is used by the _ActionAction.show() command to display the command/function that will be executed to generate the target(s).

There is a related independent ActionCaller class that looks like a regular Action, and which serves as a wrapper for arbitrary functions that we want to let the user specify the arguments to now, but actually execute later (when an out-of-date check determines that it's needed to be executed, for example). Objects of this class are returned by an ActionFactory class that provides a __call__() method as a convenient way for wrapping up the functions.

2.1 Functions

rfile(*n*)

default_exitstatfunc(*s*)

remove_set_lineno_codes(*x*)

Action(*act*, **args*, ***kw*)

A factory for action objects.

get_default_ENV(*env*)

2.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Action.py 5134 2010/08/16 23:02:40 bdee...`
print_actions	**Value:** `1`
execute_actions	**Value:** `1`
print_actions_presub	**Value:** `0`
SET_LINENO	**Value:** `dis.SET_LINENO`
HAVE_ARGUMENT	**Value:** `dis.HAVE_ARGUMENT`
strip_quotes	**Value:** `re.compile(r'^[\'"](.*)[\'"]$')`
default_ENV	**Value:** `None`
__package__	**Value:** `'SCons'`

2.3 Class ActionBase

object
└ **SCons.Action.ActionBase**

Known Subclasses: SCons.Action._ActionAction, SCons.Action.CommandGeneratorAction, SCons.Action.ListAction

Base class for all types of action objects that can be held by other objects (Builders, Executors, etc.)
This provides the common methods for manipulating and combining those actions.

2.3.1 Methods

__cmp__(*self*, *other*)

no_batch_key(*self*, *env*, *target*, *source*)

batch_key(*self*, *env*, *target*, *source*)

genstring(*self*, *target*, *source*, *env*)

get_contents(*self*, *target*, *source*, *env*)

__add__(*self*, *other*)

__radd__(*self*, *other*)

presub_lines(*self*, *env*)

get_varlist(*self*, *target*, *source*, *env*, *executor*=`None`)

get_targets(*self*, *env*, *executor*)

Returns the type of targets ($TARGETS, $CHANGED_TARGETS) used by this action.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

2.3.2 Properties

Name	Description
Inherited from object	
__class__	

2.4 Class CommandAction

object
SCons.Action.ActionBase
SCons.Action._ActionAction
SCons.Action.CommandAction

Known Subclasses: SCons.Action.LazyAction

Class for command-execution actions.

2.4.1 Methods

__init__(*self*, *cmd*, ***kw*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

process(*self*, *target*, *source*, *env*, *executor*=`None`)

strfunction(*self*, *target*, *source*, *env*, *executor*=`None`)

execute(*self*, *target*, *source*, *env*, *executor*=`None`)

Execute a command action.

This will handle lists of commands as well as individual commands, because construction variable substitution may turn a single "command" into a list. This means that this class can actually handle lists of commands, even though that's not how we use it externally.

get_presig(*self*, *target*, *source*, *env*, *executor*=`None`)

Return the signature contents of this action's command line.

This strips $(-$) and everything in between the string, since those parts don't affect signatures.

get_implicit_deps(*self*, *target*, *source*, *env*, *executor*=`None`)

Inherited from SCons.Action._ActionAction

__call__(), print_cmd_line()

Inherited from SCons.Action.ActionBase(Section 2.3)

__add__(), __cmp__(), __radd__(), batch_key(), genstring(), get_contents(), get_targets(), get_varlist(), no_batch_key(), presub_lines()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

2.4.2 Properties

Name	Description
Inherited from object	
__class__	

2.5 Class CommandGeneratorAction

object

SCons.Action.ActionBase

SCons.Action.CommandGeneratorAction

Known Subclasses: SCons.Action.LazyAction

Class for command-generator actions.

2.5.1 Methods

__init__(*self*, *generator*, *kw*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

---

batch_key(*self*, *env*, *target*, *source*)

Overrides: SCons.Action.ActionBase.batch_key

---

genstring(*self*, *target*, *source*, *env*, *executor*=`None`)

Overrides: SCons.Action.ActionBase.genstring

__call__(*self*, *target*, *source*, *env*, *exitstatfunc*=`<class 'SCons.Action._null'>`, *presub*=`<class 'SCons.Action._null'>`, *show*=`<class 'SCons.Action._null'>`, *execute*=`<class 'SCons.Action._null'>`, *chdir*=`<class 'SCons.Action._null'>`, *executor*=`None`)

---

get_presig(*self*, *target*, *source*, *env*, *executor*=`None`)

Return the signature contents of this action's command line.

This strips $(-$) and everything in between the string, since those parts don't affect signatures.

---

get_implicit_deps(*self*, *target*, *source*, *env*, *executor*=`None`)

---

get_varlist(*self*, *target*, *source*, *env*, *executor*=`None`)

Overrides: SCons.Action.ActionBase.get_varlist

---

get_targets(*self*, *env*, *executor*)

Returns the type of targets ($TARGETS, $CHANGED_TARGETS) used by this action. Overrides: SCons.Action.ActionBase.get_targets extit(inherited documentation)

Inherited from SCons.Action.ActionBase(Section 2.3)

__add__(), __cmp__(), __radd__(), get_contents(), no_batch_key(), presub_lines()

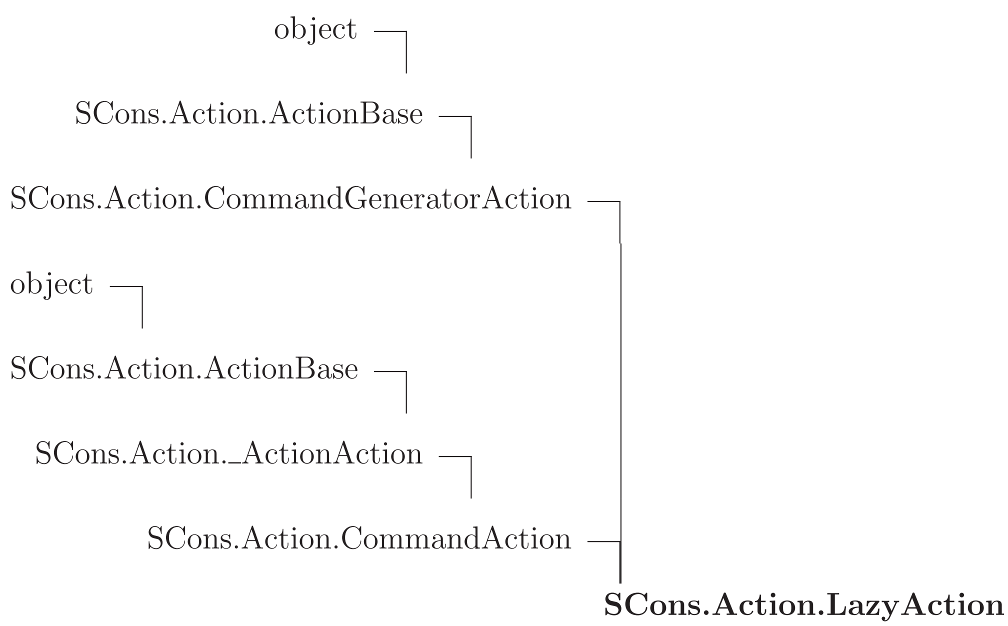
Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

2.5.2 Properties

Name	Description
Inherited from object	
__class__	

2.6 Class LazyAction



2.6.1 Methods

__init__(*self*, *var*, *kw*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

get_parent_class(*self*, *env*)

__call__(*self*, *target*, *source*, *env*, **args*, ***kw*)

Overrides: SCons.Action._ActionAction.__call__

get_presig(*self*, *target*, *source*, *env*)

Return the signature contents of this action's command line.

This strips $(-$) and everything in between the string, since those parts don't affect signatures. Overrides: SCons.Action.CommandAction.get_presig extit(inherited documentation)

get_varlist(*self*, *target*, *source*, *env*, *executor*=`None`)

Overrides: SCons.Action.ActionBase.get_varlist

Inherited from SCons.Action.CommandGeneratorAction(Section 2.5)

__str__(), batch_key(), genstring(), get_implicit_deps(), get_targets()

Inherited from SCons.Action.CommandAction(Section 2.4)

execute(), process(), strfunction()

Inherited from SCons.Action._ActionAction

print_cmd_line()

Inherited from SCons.Action.ActionBase(Section 2.3)

__add__(), __cmp__(), __radd__(), get_contents(), no_batch_key(), presub_lines()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

2.6.2 Properties

Name	Description
Inherited from object	
__class__	

2.7 Class FunctionAction

object
SCons.Action.ActionBase
SCons.Action._ActionAction
SCons.Action.FunctionAction

Class for Python function actions.

2.7.1 Methods

__init__(*self*, *execfunction*, *kw*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

function_name(*self*)

strfunction(*self*, *target*, *source*, *env*, *executor*=`None`)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

execute(*self*, *target*, *source*, *env*, *executor*=`None`)

get_presig(*self*, *target*, *source*, *env*)

Return the signature contents of this callable action.

get_implicit_deps(*self*, *target*, *source*, *env*)

Inherited from SCons.Action._ActionAction

__call__(), print_cmd_line()

Inherited from SCons.Action.ActionBase(Section 2.3)

__add__(), __cmp__(), __radd__(), batch_key(), genstring(), get_contents(), get_targets(), get_varlist(), no_batch_key(), presub_lines()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

2.7.2 Properties

Name	Description
Inherited from object	
__class__	

2.8 Class ListAction

object
SCons.Action.ActionBase
SCons.Action.ListAction

Class for lists of other actions.

2.8.1 Methods

__init__(*self*, *actionlist*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

genstring(*self*, *target*, *source*, *env*)

Overrides: SCons.Action.ActionBase.genstring

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

presub_lines(*self*, *env*)

Overrides: SCons.Action.ActionBase.presub_lines

get_presig(*self*, *target*, *source*, *env*)

Return the signature contents of this action list.

Simple concatenation of the signatures of the elements.

__call__(*self*, *target*, *source*, *env*, *exitstatfunc*=`<class 'SCons.Action._null'>`, *presub*=`<class 'SCons.Action._null'>`, *show*=`<class 'SCons.Action._null'>`, *execute*=`<class 'SCons.Action._null'>`, *chdir*=`<class 'SCons.Action._null'>`, *executor*=`None`)

get_implicit_deps(*self*, *target*, *source*, *env*)

get_varlist(*self*, *target*, *source*, *env*, *executor*=`None`)

Overrides: SCons.Action.ActionBase.get_varlist

Inherited from SCons.Action.ActionBase(Section 2.3)

__add__(), __cmp__(), __radd__(), batch_key(), get_contents(), get_targets(), no_batch_key()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

2.8.2 Properties

Name	Description
Inherited from object	
__class__	

2.9 Class ActionCaller

object ─┐
SCons.Action.ActionCaller

A class for delaying calling an Action function with specific (positional and keyword) arguments until the Action is actually executed.

This class looks to the rest of the world like a normal Action object, but what it's really doing is hanging on to the arguments until we have a target, source and env to use for the expansion.

2.9.1 Methods

__init__(*self*, *parent*, *args*, *kw*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

get_contents(*self*, *target*, *source*, *env*)

subst(*self*, *s*, *target*, *source*, *env*)

subst_args(*self*, *target*, *source*, *env*)

subst_kw(*self*, *target*, *source*, *env*)

__call__(*self*, *target*, *source*, *env*, *executor*=`None`)

strfunction(*self*, *target*, *source*, *env*)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

2.9.2 Properties

Name	Description
Inherited from object	
__class__	

2.10 Class ActionFactory

object ─┐
SCons.Action.ActionFactory

A factory class that will wrap up an arbitrary function as an SCons-executable Action object.

The real heavy lifting here is done by the ActionCaller class. We just collect the (positional and keyword) arguments that we're called with and give them to the ActionCaller object we create, so it can hang onto them until it needs them.

2.10.1 Methods

__init__(*self*, *actfunc*, *strfunc*, *convert*=`<function <lambda> at 0x132d578>`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, **args*, ***kw*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

2.10.2 Properties

Name	Description
Inherited from object	
__class__	

3 Module SCons.Builder

```
SCons.Builder

Builder object subsystem.

A Builder object is a callable that encapsulates information about how
to execute actions to create a target Node (file) from source Nodes
(files), and how to create those dependencies for tracking.

The main entry point here is the Builder() factory method.  This provides
a procedural interface that creates the right underlying Builder object
based on the keyword arguments supplied and the types of the arguments.

The goal is for this external interface to be simple enough that the
vast majority of users can create new Builders as necessary to support
building new types of files in their configurations, without having to
dive any deeper into this subsystem.

The base class here is BuilderBase.  This is a concrete base class which
does, in fact, represent the Builder objects that we (or users) create.

There is also a proxy that looks like a Builder:

    CompositeBuilder

        This proxies for a Builder with an action that is actually a
        dictionary that knows how to map file suffixes to a specific
        action.  This is so that we can invoke different actions
        (compilers, compile options) for different flavors of source
        files.

Builders and their proxies have the following public interface methods
used by other modules:

    __call__()
        THE public interface.  Calling a Builder object (with the
        use of internal helper methods) sets up the target and source
        dependencies, appropriate mapping to a specific action, and the
        environment manipulation necessary for overridden construction
        variable.  This also takes care of warning about possible mistakes
        in keyword arguments.

    add_emitter()
        Adds an emitter for a specific file suffix, used by some Tool
```

```
        modules to specify that (for example) a yacc invocation on a .y
        can create a .h *and* a .c file.

    add_action()
        Adds an action for a specific file suffix, heavily used by
        Tool modules to add their specific action(s) for turning
        a source file into an object file to the global static
        and shared object file Builders.

There are the following methods for internal use within this module:

    _execute()
        The internal method that handles the heavily lifting when a
        Builder is called.  This is used so that the __call__() methods
        can set up warning about possible mistakes in keyword-argument
        overrides, and *then* execute all of the steps necessary so that
        the warnings only occur once.

    get_name()
        Returns the Builder's name within a specific Environment,
        primarily used to try to return helpful information in error
        messages.

    adjust_suffix()
    get_prefix()
    get_suffix()
    get_src_suffix()
    set_src_suffix()
        Miscellaneous stuff for handling the prefix and suffix
        manipulation we use in turning source file names into target
        file names.
```

3.1 Functions

match_splitext(*path*, *suffixes*=`[]`)

Builder(***kw*)

A factory for builder objects.

is_a_Builder(*obj*)

"Returns True iff the specified obj is one of our Builder classes.

The test is complicated a bit by the fact that CompositeBuilder is a proxy, not a subclass of BuilderBase.

3.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Builder.py 5134 2010/08/16 23:02:40 bde...`
misleading_keywords	**Value:** `{'sources': 'source', 'targets': 'target'}`
__package__	**Value:** `'SCons'`

3.3 Class DictCmdGenerator

UserDict.UserDict
SCons.Util.OrderedDict
SCons.Util.Selector
SCons.Builder.DictCmdGenerator

This is a callable class that can be used as a command generator function. It holds on to a dictionary mapping file suffixes to Actions. It uses that dictionary to return the proper action based on the file suffix of the source file.

3.3.1 Methods

__init__(*self*, *dict*=`None`, *source_ext_match*=`1`)

Overrides: UserDict.UserDict.__init__

src_suffixes(*self*)

add_action(*self*, *suffix*, *action*)

Add a suffix-action pair to the mapping.

__call__(*self*, *target*, *source*, *env*, *for_signature*)

Overrides: SCons.Util.Selector.__call__

Inherited from SCons.Util.OrderedDict(Section 36.10)

__delitem__(), __setitem__(), clear(), copy(), items(), keys(), popitem(), setdefault(), update(), values()

Inherited from UserDict.UserDict

__cmp__(), __contains__(), __getitem__(), __len__(), __repr__(), fromkeys(), get(), has_key(), iteritems(), iterkeys(), itervalues(), pop()

3.4 Class CallableSelector

UserDict.UserDict
SCons.Util.OrderedDict
SCons.Util.Selector
SCons.Builder.CallableSelector

A callable dictionary that will, in turn, call the value it finds if it can.

3.4.1 Methods

__call__(*self*, *env*, *source*)

Overrides: SCons.Util.Selector.__call__

Inherited from SCons.Util.OrderedDict(Section 36.10)

__delitem__(), __init__(), __setitem__(), clear(), copy(), items(), keys(), popitem(), setdefault(), update(), values()

Inherited from UserDict.UserDict

__cmp__(), __contains__(), __getitem__(), __len__(), __repr__(), fromkeys(), get(), has_key(), iteritems(), iterkeys(), itervalues(), pop()

3.5 Class DictEmitter

UserDict.UserDict
SCons.Util.OrderedDict
SCons.Util.Selector
SCons.Builder.DictEmitter

A callable dictionary that maps file suffixes to emitters. When called, it finds the right emitter in its dictionary for the suffix of the first source file, and calls that emitter to get the right lists of targets and sources to return. If there's no emitter for the suffix in its dictionary, the original target and source are returned.

3.5.1 Methods

__call__(*self*, *target*, *source*, *env*)
Overrides: SCons.Util.Selector.__call__

Inherited from SCons.Util.OrderedDict(Section 36.10)

__delitem__(), __init__(), __setitem__(), clear(), copy(), items(), keys(), popitem(), setdefault(), update(), values()

Inherited from UserDict.UserDict

__cmp__(), __contains__(), __getitem__(), __len__(), __repr__(), fromkeys(), get(), has_key(), iteritems(), iterkeys(), itervalues(), pop()

3.6 Class ListEmitter

```
object ─┐
        │
    _abcoll.Sized ─┐
                   │
object ─┐          │
        │          │
  _abcoll.Iterable ┤
                   │
object ─┐          │
        │          │
 _abcoll.Container ┘
      _abcoll.Sequence ─┐
                        │
   _abcoll.MutableSequence ─┐
                            │
              UserList.UserList ─┐
                                 │
                    SCons.Builder.ListEmitter
```

A callable list of emitters that calls each in sequence, returning the result.

3.6.1 Methods

__call__(*self*, *target*, *source*, *env*)

Inherited from UserList.UserList

__add__(), __cmp__(), __contains__(), __delitem__(), __delslice__(), __eq__(), __ge__(), __getitem__(), __getslice__(), __gt__(), __iadd__(), __imul__(), __init__(), __le__(), __len__(), __lt__(), __mul__(), __ne__(), __radd__(), __repr__(), __rmul__(), __setitem__(), __setslice__(), append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

Inherited from _abcoll.Sequence

__iter__(), __reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__()

3.6.2 Properties

Name	Description
Inherited from object	
__class__	

3.6.3 Class Variables

Name	Description
Inherited from UserList.UserList	
__abstractmethods__, __hash__	

3.7 Class OverrideWarner

UserDict.UserDict
└ **SCons.Builder.OverrideWarner**

A class for warning about keyword arguments that we use as overrides in a Builder call.

This class exists to handle the fact that a single Builder call can actually invoke multiple builders. This class only emits the warnings once, no matter how many Builders are invoked.

3.7.1 Methods

__init__(*self*, *dict*)

Overrides: UserDict.UserDict.__init__

warn(*self*)

Inherited from UserDict.UserDict

__cmp__(), __contains__(), __delitem__(), __getitem__(), __len__(), __repr__(), __setitem__(), clear(), copy(), fromkeys(), get(), has_key(), items(), iteritems(), iterkeys(), itervalues(), keys(), pop(), popitem(), setdefault(), update(), values()

3.8 Class EmitterProxy

object ─┐
SCons.Builder.EmitterProxy

This is a callable class that can act as a Builder emitter. It holds on to a string that is a key into an Environment dictionary, and will look there at actual build time to see if it holds a callable. If so, we will call that as the actual emitter.

3.8.1 Methods

__init__(*self*, *var*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, *target*, *source*, *env*)

__cmp__(*self*, *other*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.8.2 Properties

Name	Description
Inherited from object	
__class__	

3.9 Class BuilderBase

object ─┐
SCons.Builder.BuilderBase

Base class for Builders, objects that create output nodes (files) from input nodes (files).

3.9.1 Methods

__init__(*self*, *action*=`None`, *prefix*=`''`, *suffix*=`''`, *src_suffix*=`''`, *target_factory*=`None`, *source_factory*=`None`, *target_scanner*=`None`, *source_scanner*=`None`, *emitter*=`None`, *multi*=`0`, *env*=`None`, *single_source*=`0`, *name*=`None`, *chdir*=`<class 'SCons.Builder._Null'>`, *is_explicit*=`1`, *src_builder*=`None`, *ensure_suffix*=`False`, ***overrides*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__nonzero__(*self*)

get_name(*self*, *env*)

Attempts to get the name of the Builder.

Look at the BUILDERS variable of env, expecting it to be a dictionary containing this Builder, and return the key of the dictionary. If there's no key, then return a directly-configured name (if there is one) or the name of the class (by default).

__cmp__(*self*, *other*)

splitext(*self*, *path*, *env*=`None`)

__call__(*self*, *env*, *target*=`None`, *source*=`None`, *chdir*=`<class 'SCons.Builder._Null'>`, ***kw*)

adjust_suffix(*self*, *suff*)

get_prefix(*self*, *env*, *sources*=`[]`)

set_suffix(*self*, *suffix*)

get_suffix(*self*, *env*, *sources*=`[]`)

set_src_suffix(*self*, *src_suffix*)

get_src_suffix(*self*, *env*)

Get the first src_suffix in the list of src_suffixes.

add_emitter(*self*, *suffix*, *emitter*)

Add a suffix-emitter mapping to this Builder.

This assumes that emitter has been initialized with an appropriate dictionary type, and will throw a TypeError if not, so the caller is responsible for knowing that this is an appropriate method to call for the Builder in question.

add_src_builder(*self*, *builder*)

Add a new Builder to the list of src_builders.

This requires wiping out cached values so that the computed lists of source suffixes get re-calculated.

src_builder_sources(*self*, *env*, *source*, *overwarn*={})

get_src_builders(*self*, *env*)

Returns the list of source Builders for this Builder.

This exists mainly to look up Builders referenced as strings in the 'BUILDER' variable of the construction environment and cache the result.

subst_src_suffixes(*self*, *env*)

The suffix list may contain construction variable expansions, so we have to evaluate the individual strings. To avoid doing this over and over, we memoize the results for each construction environment.

src_suffixes(*self*, *env*)

Returns the list of source suffixes for all src_builders of this Builder.

This is essentially a recursive descent of the src_builder "tree." (This value isn't cached because there may be changes in a src_builder many levels deep that we can't see.)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.9.2 Properties

Name	Description
Inherited from object	
__class__	

3.9.3 Class Variables

Name	Description
__metaclass__	**Value:** `SCons.Memoize.Memoized_Metaclass`
memoizer_counters	**Value:** `[]`

3.10 Class CompositeBuilder

object
SCons.Util.Proxy
SCons.Builder.CompositeBuilder

A Builder Proxy whose main purpose is to always have a DictCmdGenerator as its action, and to provide access to the DictCmdGenerator's add_action() method.

3.10.1 Methods

__init__(*self*, *builder*, *cmdgen*)

Wrap an object as a Proxy object Overrides: object.__init__ extit(inherited documentation)

__call__(*...*)

A Python Descriptor class that delegates attribute fetches to an underlying wrapped subject of a Proxy. Typical use:

class Foo(Proxy): __str__ = Delegate('__str__')

add_action(*self*, *suffix*, *action*)

Inherited from SCons.Util.Proxy(Section 36.5)

__cmp__(), __getattr__(), get()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.10.2 Properties

Name	Description
Inherited from object	
__class__	

4 Module SCons.CacheDir

CacheDir support

4.1 Functions

CacheRetrieveFunc(*target*, *source*, *env*)

CacheRetrieveString(*target*, *source*, *env*)

CachePushFunc(*target*, *source*, *env*)

4.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/CacheDir.py 5134 2010/08/16 23:02:40 bd...`
__doc__	**Value:** `...`
cache_enabled	**Value:** `True`
cache_debug	**Value:** `False`
cache_force	**Value:** `False`
cache_show	**Value:** `False`
CacheRetrieve	**Value:** `SCons.Action.Action(CacheRetrieveFunc, CacheRetrieveString)`
CacheRetrieveSilent	**Value:** `SCons.Action.Action(CacheRetrieveFunc, None)`
CachePush	**Value:** `SCons.Action.Action(CachePushFunc, None)`
__package__	**Value:** `'SCons'`

4.3 Class CacheDir

object ─┐
SCons.CacheDir.CacheDir

4.3.1 Methods

__init__(*self*, *path*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

CacheDebug(*self*, *fmt*, *target*, *cachefile*)

is_enabled(*self*)

cachepath(*self*, *node*)

retrieve(*self*, *node*)

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in built().

Note that there's a special trick here with the execute flag (one that's not normally done for other actions). Basically if the user requested a no_exec (-n) build, then SCons.Action.execute_actions is set to 0 and when any action is called, it does its showing but then just returns zero instead of actually calling the action execution operation. The problem for caching is that if the file does NOT exist in cache then the CacheRetrieveString won't return anything to show for the task, but the Action.__call__ won't call CacheRetrieveFunc; instead it just returns zero, which makes the code below think that the file *was* successfully retrieved from the cache, therefore it doesn't do any subsequent building. However, the CacheRetrieveString didn't print anything because it didn't actually exist in the cache, and no more build actions will be performed, so the user just sees nothing. The fix is to tell Action.__call__ to always execute the CacheRetrieveFunc and then have the latter explicitly check SCons.Action.execute_actions itself.

push(*self*, *node*)

push_if_forced(*self*, *node*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

4.3.2 Properties

Name	Description
Inherited from object	
__class__	

5 Module SCons.Conftest

SCons.Conftest

Autoconf-like configuration support; low level implementation of tests.

5.1 Functions

CheckBuilder(*context*, *text*=`None`, *language*=`None`)

Configure check to see if the compiler works. Note that this uses the current value of compiler and linker flags, make sure $CFLAGS, $CPPFLAGS and $LIBS are set correctly. "language" should be "C" or "C++" and is used to select the compiler. Default is "C". "text" may be used to specify the code to be build. Returns an empty string for success, an error message for failure.

CheckCC(*context*)

Configure check for a working C compiler.

This checks whether the C compiler, as defined in the $CC construction variable, can compile a C source file. It uses the current $CCCOM value too, so that it can test against non working flags.

CheckSHCC(*context*)

Configure check for a working shared C compiler.

This checks whether the C compiler, as defined in the $SHCC construction variable, can compile a C source file. It uses the current $SHCCCOM value too, so that it can test against non working flags.

CheckCXX(*context*)

Configure check for a working CXX compiler.

This checks whether the CXX compiler, as defined in the $CXX construction variable, can compile a CXX source file. It uses the current $CXXCOM value too, so that it can test against non working flags.

CheckSHCXX(*context*)

Configure check for a working shared CXX compiler.

This checks whether the CXX compiler, as defined in the $SHCXX construction variable, can compile a CXX source file. It uses the current $SHCXXCOM value too, so that it can test against non working flags.

CheckFunc(*context*, *function_name*, *header*=`None`, *language*=`None`)

Configure check for a function "function_name". "language" should be "C" or "C++" and is used to select the compiler. Default is "C". Optional "header" can be defined to define a function prototype, include a header file or anything else that comes before main(). Sets HAVE_function_name in context.havedict according to the result. Note that this uses the current value of compiler and linker flags, make sure $CFLAGS, $CPPFLAGS and $LIBS are set correctly. Returns an empty string for success, an error message for failure.

CheckHeader(*context*, *header_name*, *header*=`None`, *language*=`None`, *include_quotes*=`None`)

Configure check for a C or C++ header file "header_name". Optional "header" can be defined to do something before including the header file (unusual, supported for consistency). "language" should be "C" or "C++" and is used to select the compiler. Default is "C". Sets HAVE_header_name in context.havedict according to the result. Note that this uses the current value of compiler and linker flags, make sure $CFLAGS and $CPPFLAGS are set correctly. Returns an empty string for success, an error message for failure.

CheckType(*context*, *type_name*, *fallback*=`None`, *header*=`None`, *language*=`None`)

Configure check for a C or C++ type "type_name". Optional "header" can be defined to include a header file. "language" should be "C" or "C++" and is used to select the compiler. Default is "C". Sets HAVE_type_name in context.havedict according to the result. Note that this uses the current value of compiler and linker flags, make sure $CFLAGS, $CPPFLAGS and $LIBS are set correctly. Returns an empty string for success, an error message for failure.

CheckTypeSize(*context*, *type_name*, *header*=`None`, *language*=`None`, *expect*=`None`)

This check can be used to get the size of a given type, or to check whether the type is of expected size.

Arguments:

- **type** (**str**)

  the type to check

- **includes** (**sequence**)

  list of headers to include in the test code before testing the type

- **language** (**str**)

  'C' or 'C++'

- **expect** (**int**)

  if given, will test wether the type has the given number of bytes. If not given, will automatically find the size.

Returns:

status (**int**)

0 if the check failed, or the found size of the type if the check succeeded.

CheckDeclaration(*context*, *symbol*, *includes*=`None`, *language*=`None`)

Checks whether symbol is declared.

Use the same test as autoconf, that is test whether the symbol is defined as a macro or can be used as an r-value.

Arguments:

symbol (**str**)

the symbol to check

includes (**str**)

Optional "header" can be defined to include a header file.

language (**str**)

only C and C++ supported.

Returns:

status (**bool**)

True if the check failed, False if succeeded.

CheckLib(*context*, *libs*, *func_name*=`None`, *header*=`None`, *extra_libs*=`None`, *call*=`None`, *language*=`None`, *autoadd*=`1`, *append*=`True`)

Configure check for a C or C++ libraries "libs". Searches through the list of libraries, until one is found where the test succeeds. Tests if "func_name" or "call" exists in the library. Note: if it exists in another library the test succeeds anyway! Optional "header" can be defined to include a header file. If not given a default prototype for "func_name" is added. Optional "extra_libs" is a list of library names to be added after "lib_name" in the build command. To be used for libraries that "lib_name" depends on. Optional "call" replaces the call to "func_name" in the test code. It must consist of complete C statements, including a trailing ";". Both "func_name" and "call" arguments are optional, and in that case, just linking against the libs is tested. "language" should be "C" or "C++" and is used to select the compiler. Default is "C". Note that this uses the current value of compiler and linker flags, make sure $CFLAGS, $CPPFLAGS and $LIBS are set correctly. Returns an empty string for success, an error message for failure.

5.2 Variables

Name	Description
LogInputFiles	**Value:** `1`
LogErrorMessages	**Value:** `1`
__package__	**Value:** `'SCons'`

6 Module SCons.Debug

SCons.Debug

Code for debugging SCons internal things. Shouldn't be needed by most users.

6.1 Functions

logInstanceCreation(*instance*, *name*=`None`)

string_to_classes(*s*)

fetchLoggedInstances(*classes*=`'*'`)

countLoggedInstances(*classes*, *file*=`sys.stderr`)

listLoggedInstances(*classes*, *file*=`sys.stderr`)

dumpLoggedInstances(*classes*, *file*=`sys.stderr`)

memory()

caller_stack(**backlist*)

caller_trace(*back*=`0`)

dump_caller_counts(*file*=`sys.stderr`)

func_shorten(*func_tuple*)

Trace(*msg*, *file*=`None`, *mode*=`'w'`, *tstamp*=`None`)

Write a trace message to a file. Whenever a file is specified, it becomes the default for the next call to Trace().

6.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Debug.py 5134 2010/08/16 23:02:40 bdeegan'`
tracked_classes	**Value:** `{}`
caller_bases	**Value:** `{}`
caller_dicts	**Value:** `{}`
shorten_list	**Value:** `[('/scons/SCons/', 1), ('/src/engine/SCons/', 1), ('/usr/...`
TraceFP	**Value:** `{}`
TraceDefault	**Value:** `'/dev/tty'`
TimeStampDefault	**Value:** `None`
StartTime	**Value:** `1282025346.51`
PreviousTime	**Value:** `1282025346.51`
__package__	**Value:** `'SCons'`

7 Module SCons.Defaults

SCons.Defaults

Builders and other things for the local site. Here's where we'll duplicate the functionality of autoconf until we move it into the installation procedure or use something like qmconf.

The code that reads the registry to find MSVC components was borrowed from distutils.msvccompiler.

7.1 Functions

DefaultEnvironment(**args*, ***kw*)

Initial public entry point for creating the default construction Environment.

After creating the environment, we overwrite our name (DefaultEnvironment) with the _fetch_DefaultEnvironment() function, which more efficiently returns the initialized default construction environment without checking for its existence.

(This function still exists with its _default_check because someone else (*cough* Script/__init__.py *cough*) may keep a reference to this function. So we can't use the fully functional idiom of having the name originally be a something that *only* creates the construction environment and then overwrites the name.)

StaticObjectEmitter(*target*, *source*, *env*)

SharedObjectEmitter(*target*, *source*, *env*)

SharedFlagChecker(*source*, *target*, *env*)

get_paths_str(*dest*)

chmod_func(*dest*, *mode*)

chmod_strfunc(*dest*, *mode*)

copy_func(*dest*, *src*)

delete_func(*dest*, *must_exist*=0)

delete_strfunc(*dest*, *must_exist*=0)

mkdir_func(*dest*)

move_func(*dest*, *src*)

touch_func(*dest*)

processDefines(*defs*)

process defines, resolving strings, lists, dictionaries, into a list of strings

7.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Defaults.py 5134 2010/08/16 23:02:40 bd...`
SharedCheck	**Value:** `SCons.Action.Action(SharedFlagChecker, None)`
CScan	**Value:** `SCons.Defaults.CScan`
DScan	**Value:** `SCons.Tool.DScanner`
LaTeXScan	**Value:** `SCons.Tool.LaTeXScanner`
ObjSourceScan	**Value:** `SCons.Tool.SourceFileScanner`
ProgScan	**Value:** `SCons.Tool.ProgramScanner`
DirScanner	**Value:** `SCons.Defaults.DirScanner`
DirEntryScanner	**Value:** `SCons.Scanner.Dir.DirEntryScanner()`
CAction	**Value:** `SCons.Action.Action("$CCCOM", "$CCCOMSTR")`
ShCAction	**Value:** `SCons.Action.Action("$SHCCCOM", "$SHCCCOMSTR")`
CXXAction	**Value:** `SCons.Action.Action("$CXXCOM", "$CXXCOMSTR")`
ShCXXAction	**Value:** `SCons.Action.Action("$SHCXXCOM", "$SHCXXCOMSTR")`
ASAction	**Value:** `SCons.Action.Action("$ASCOM", "$ASCOMSTR")`

continued on next page

Name	Description
ASPPAction	**Value:** `SCons.Action.Action("$ASPPCOM", "$ASPPCOMSTR")`
LinkAction	**Value:** `SCons.Action.Action("$LINKCOM", "$LINKCOMSTR")`
ShLinkAction	**Value:** `SCons.Action.Action("$SHLINKCOM", "$SHLINKCOMSTR")`
LdModuleLinkAction	**Value:** `SCons.Action.Action("$LDMODULECOM", "$LDMODULECOMSTR")`
Chmod	**Value:** `SCons.Defaults.Chmod`
Copy	**Value:** `SCons.Defaults.Copy`
Delete	**Value:** `SCons.Defaults.Delete`
Mkdir	**Value:** `SCons.Defaults.Mkdir`
Move	**Value:** `SCons.Defaults.Move`
Touch	**Value:** `SCons.Defaults.Touch`
ConstructionEnvironment	**Value:** `{'BUILDERS': {}, 'CONFIGUREDIR': '#/.sconf_temp', 'CONFIG...`
__package__	**Value:** `'SCons'`

7.3 Class NullCmdGenerator

object ─┐
SCons.Defaults.NullCmdGenerator

This is a callable class that can be used in place of other command generators if you don't want them to do anything.

The __call__ method for this class simply returns the thing you instantiated it with.

Example usage: env["DO_NOTHING"] = NullCmdGenerator env["LINKCOM"] = "${DO_NOTHING('$LINK $SOURCES $TARGET')}"

7.3.1 Methods

__init__(*self*, *cmd*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, *target*, *source*, *env*, *for_signature*=`None`)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

7.3.2 Properties

Name	Description
Inherited from object	
__class__	

7.4 Class Variable_Method_Caller

object ─┐
SCons.Defaults.Variable_Method_Caller

A class for finding a construction variable on the stack and calling one of its methods.

We use this to support "construction variables" in our string eval()s that actually stand in for methods--specifically, use of "RDirs" in call to _concat that should actually execute the "TARGET.RDirs" method. (We used to support this by creating a little "build dictionary" that mapped RDirs to the method, but this got in the way of Memoizing construction environments, because we had to create new environment objects to hold the variables.)

7.4.1 Methods

__init__(*self*, *variable*, *method*)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, **args*, ***kw*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

7.4.2 Properties

Name	Description
Inherited from object	
__class__	

8 Module SCons.Environment

SCons.Environment

Base class for construction Environments. These are the primary objects used to communicate dependency and construction information to the build engine.

Keyword arguments supplied when the construction Environment is created are construction variables used to initialize the Environment

8.1 Functions

alias_builder(*env*, *target*, *source*)

apply_tools(*env*, *tools*, *toolpath*)

copy_non_reserved_keywords(*dict*)

is_valid_construction_var(*varstr*)

Return if the specified string is a legitimate construction variable.

default_decide_source(*dependency*, *target*, *prev_ni*)

default_decide_target(*dependency*, *target*, *prev_ni*)

default_copy_from_cache(*src*, *dst*)

NoSubstitutionProxy(*subject*)

8.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Environment.py 5134 2010/08/16 23:02:40...`
CleanTargets	**Value:** `{}`
CalculatorArgs	**Value:** `{}`
AliasBuilder	**Value:** `SCons.Builder.Builder(action= alias_builder, target_facto...`

continued on next page

Name	Description
reserved_construction_var_-names	**Value:** `['CHANGED_SOURCES', 'CHANGED_TARGETS', 'SOURCE', 'SOURCES...`
future_reserved_constructi-on_var_names	**Value:** `[]`
__package__	**Value:** `'SCons'`

8.3 Class MethodWrapper

object
└ **SCons.Environment.MethodWrapper**

Known Subclasses: SCons.Environment.BuilderWrapper

A generic Wrapper class that associates a method (which can actually be any callable) with an object. As part of creating this MethodWrapper object an attribute with the specified (by default, the name of the supplied method) is added to the underlying object. When that new "method" is called, our __call__() method adds the object as the first argument, simulating the Python behavior of supplying "self" on method calls.

We hang on to the name by which the method was added to the underlying base class so that we can provide a method to "clone" ourselves onto a new underlying object being copied (without which we wouldn't need to save that info).

8.3.1 Methods

__init__(*self*, *object*, *method*, *name*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, **args*, ***kwargs*)

clone(*self*, *new_object*)

Returns an object that re-binds the underlying "method" to the specified new object.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.3.2 Properties

Name	Description
Inherited from object	
__class__	

8.4 Class BuilderWrapper

object
SCons.Environment.MethodWrapper
SCons.Environment.BuilderWrapper

A MethodWrapper subclass that that associates an environment with a Builder.

This mainly exists to wrap the __call__() function so that all calls to Builders can have their argument lists massaged in the same way (treat a lone argument as the source, treat two arguments as target then source, make sure both target and source are lists) without having to have cut-and-paste code to do it.

As a bit of obsessive backwards compatibility, we also intercept attempts to get or set the "env" or "builder" attributes, which were the names we used before we put the common functionality into the MethodWrapper base class. We'll keep this around for a while in case people shipped Tool modules that reached into the wrapper (like the Tool/qt.py module does, or did). There shouldn't be a lot attribute fetching or setting on these, so a little extra work shouldn't hurt.

8.4.1 Methods

__call__(*self*, *target*=`None`, *source*=`<class 'SCons.Environment._Null'>`, **args*, ***kw*)

Overrides: SCons.Environment.MethodWrapper.__call__

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

x.__setattr__('name', value) <==> x.name = value Overrides: object.__setattr__ extit(inherited documentation)

Inherited from SCons.Environment.MethodWrapper(Section 8.3)

__init__(), clone()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

8.4.2 Properties

Name	Description
Inherited from object	
__class__	

8.5 Class BuilderDict

UserDict.UserDict
└ **SCons.Environment.BuilderDict**

This is a dictionary-like class used by an Environment to hold the Builders. We need to do this because every time someone changes the Builders in the Environment's BUILDERS dictionary, we must update the Environment's attributes.

8.5.1 Methods

__init__(*self*, *dict*, *env*)

Overrides: UserDict.UserDict.__init__

__semi_deepcopy__(*self*)

__setitem__(*self*, *item*, *val*)

Overrides: UserDict.UserDict.__setitem__

__delitem__(*self*, *item*)

Overrides: UserDict.UserDict.__delitem__

update(*self*, *dict*)

Overrides: UserDict.UserDict.update

Inherited from UserDict.UserDict

__cmp__(), __contains__(), __getitem__(), __len__(), __repr__(), clear(), copy(), fromkeys(), get(), has_key(), items(), iteritems(), iterkeys(), itervalues(), keys(), pop(), popitem(), setdefault(), values()

8.6 Class SubstitutionEnvironment

object ─┐

SCons.Environment.SubstitutionEnvironment

Known Subclasses: SCons.Environment.Base

Base class for different flavors of construction environments.

This class contains a minimal set of methods that handle contruction variable expansion and conversion of strings to Nodes, which may or may not be actually useful as a stand-alone class. Which methods ended up in this class is pretty arbitrary right now. They're basically the ones which we've empirically determined are common to the different construction environment subclasses, and most of the others that use or touch the underlying dictionary of construction variables.

Eventually, this class should contain all the methods that we determine are necessary for a "minimal" interface to the build engine. A full "native Python" SCons environment has gotten pretty heavyweight with all of the methods and Tools and construction variables we've jammed in there, so it would be nice to have a lighter weight alternative for interfaces that don't need all of the bells and whistles. (At some point, we'll also probably rename this class "Base," since that more reflects what we want this class to become, but because we've released comments that tell people to subclass Environment.Base to create their own flavors of construction environment, we'll save that for a future refactoring when this class actually becomes useful.)

8.6.1 Methods

__init__(*self*, ***kw*)

Initialization of an underlying SubstitutionEnvironment class. Overrides: object.__init__

__cmp__(*self*, *other*)

__delitem__(*self*, *key*)

__getitem__(*self*, *key*)

__setitem__(*self*, *key*, *value*)

get(*self*, *key*, *default*=`None`)

Emulates the get() method of dictionaries.

has_key(*self*, *key*)

__contains__(*self*, *key*)

items(*self*)

arg2nodes(*self*, *args*, *node_factory*=`<class 'SCons.Environment._Null'>`, *lookup_list*=`<class 'SCons.Environment._Null'>`, ***kw*)

gvars(*self*)

lvars(*self*)

subst(*self*, *string*, *raw*=0, *target*=None, *source*=None, *conv*=None, *executor*=None)

Recursively interpolates construction variables from the Environment into the specified string, returning the expanded result. Construction variables are specified by a $ prefix in the string and begin with an initial underscore or alphabetic character followed by any number of underscores or alphanumeric characters. The construction variable names may be surrounded by curly braces to separate the name from trailing characters.

subst_kw(*self*, *kw*, *raw*=0, *target*=None, *source*=None)

subst_list(*self*, *string*, *raw*=0, *target*=None, *source*=None, *conv*=None, *executor*=None)

Calls through to SCons.Subst.scons_subst_list(). See the documentation for that function.

subst_path(*self*, *path*, *target*=None, *source*=None)

Substitute a path list, turning EntryProxies into Nodes and leaving Nodes (and other objects) as-is.

subst_target_source(*self*, *string*, *raw*=0, *target*=None, *source*=None, *conv*=None, *executor*=None)

Recursively interpolates construction variables from the Environment into the specified string, returning the expanded result. Construction variables are specified by a $ prefix in the string and begin with an initial underscore or alphabetic character followed by any number of underscores or alphanumeric characters. The construction variable names may be surrounded by curly braces to separate the name from trailing characters.

backtick(*self*, *command*)

AddMethod(*self*, *function*, *name*=`None`)

Adds the specified function as a method of this construction environment with the specified name. If the name is omitted, the default name is the name of the function itself.

RemoveMethod(*self*, *function*)

Removes the specified function's MethodWrapper from the added_methods list, so we don't re-bind it when making a clone.

Override(*self*, *overrides*)

Produce a modified environment whose variables are overriden by the overrides dictionaries. "overrides" is a dictionary that will override the variables of this environment.

This function is much more efficient than Clone() or creating a new Environment because it doesn't copy the construction environment dictionary, it just wraps the underlying construction environment, and doesn't even create a wrapper object if there are no overrides.

ParseFlags(*self*, **flags*)

Parse the set of flags and return a dict with the flags placed in the appropriate entry. The flags are treated as a typical set of command-line flags for a GNU-like toolchain and used to populate the entries in the dict immediately below. If one of the flag strings begins with a bang (exclamation mark), it is assumed to be a command and the rest of the string is executed; the result of that evaluation is then added to the dict.

MergeFlags(*self*, *args*, *unique*=`1`, *dict*=`None`)

Merge the dict in args into the construction variables of this env, or the passed-in dict. If args is not a dict, it is converted into a dict using ParseFlags. If unique is not set, the flags are appended rather than merged.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.6.2 Properties

Name	Description
Inherited from object	
__class__	

8.6.3 Class Variables

Name	Description
__metaclass__	**Value:** `SCons.Memoize.Memoized_Metaclass`

8.7 Class Base

```
object ─┐
        │
SCons.Environment.SubstitutionEnvironment ─┐
                                           │
                                   SCons.Environment.Base
```

Known Subclasses: SCons.Environment.OverrideEnvironment, SCons.Script.SConscript'.SConsEnvironment

Base class for "real" construction Environments. These are the primary objects used to communicate dependency and construction information to the build engine.

Keyword arguments supplied when the construction Environment is created are construction variables used to initialize the Environment.

8.7.1 Methods

Action(*self*, **args*, ***kw*)

AddPostAction(*self*, *files*, *action*)

AddPreAction(*self*, *files*, *action*)

Alias(*self*, *target*, *source*=`[]`, *action*=`None`, ***kw*)

AlwaysBuild(*self*, **targets*)

Append(*self*, ***kw*)

Append values to existing construction variables in an Environment.

AppendENVPath(*self*, *name*, *newpath*, *envname*=`'ENV'`, *sep*=`':'`, *delete_existing*=`1`)

Append path elements to the path 'name' in the 'ENV' dictionary for this environment. Will only add any particular path once, and will normpath and normcase all paths to help assure this. This can also handle the case where the env variable is a list instead of a string.

If delete_existing is 0, a newpath which is already in the path will not be moved to the end (it will be left where it is).

AppendUnique(*self*, *delete_existing*=`0`, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there. If delete_existing is 1, removes existing values first, so values move to end.

BuildDir(*self*, **args*, ***kw*)

Builder(*self*, ***kw*)

CacheDir(*self*, *path*)

Clean(*self*, *targets*, *files*)

Clone(*self*, *tools*=`[]`, *toolpath*=`None`, *parse_flags*=`None`, ***kw*)

Return a copy of a construction Environment. The copy is like a Python "deep copy"--that is, independent copies are made recursively of each objects--except that a reference is copied when an object is not deep-copyable (like a function). There are no references to any mutable objects in the original Environment.

Command(*self*, *target*, *source*, *action*, ***kw*)

Builds the supplied target files from the supplied source files using the supplied action. Action may be any type that the Builder constructor will accept for an action.

Configure(*self*, **args*, ***kw*)

Copy(*self*, **args*, ***kw*)

Decider(*self*, *function*)

Depends(*self*, *target*, *dependency*)

Explicity specify that 'target's depend on 'dependency'.

Detect(*self*, *progs*)

Return the first available program in progs.

Dictionary(*self*, **args*)

Dir(*self*, *name*, **args*, ***kw*)

Dump(*self*, *key*=`None`)

Using the standard Python pretty printer, dump the contents of the scons build environment to stdout.

If the key passed in is anything other than None, then that will be used as an index into the build environment dictionary and whatever is found there will be fed into the pretty printer. Note that this key is case sensitive.

Entry(*self*, *name*, **args*, ***kw*)

Environment(*self*, ***kw*)

Execute(*self*, *action*, **args*, ***kw*)

Directly execute an action through an Environment

File(*self*, *name*, **args*, ***kw*)

FindFile(*self*, *file*, *dirs*)

FindInstalledFiles(*self*)

returns the list of all targets of the Install and InstallAs Builder.

FindIxes(*self*, *paths*, *prefix*, *suffix*)

Search a list of paths for something that matches the prefix and suffix.

paths - the list of paths or nodes. prefix - construction variable for the prefix. suffix - construction variable for the suffix.

FindSourceFiles(*self*, *node*=`'.'`)

returns a list of all source files.

Flatten(*self*, *sequence*)

GetBuildPath(*self*, *files*)

Glob(*self*, *pattern*, *ondisk*=`True`, *source*=`False`, *strings*=`False`)

Ignore(*self*, *target*, *dependency*)

Ignore a dependency.

Literal(*self*, *string*)

Local(*self*, **targets*)

NoCache(*self*, **targets*)

Tags a target so that it will not be cached

NoClean(*self*, **targets*)

Tags a target so that it will not be cleaned by `-c`

ParseConfig(*self*, *command*, *function*=`None`, *unique*=`1`)

Use the specified function to parse the output of the command in order to modify the current environment. The 'command' can be a string or a list of strings representing a command and its arguments. 'Function' is an optional argument that takes the environment, the output of the command, and the unique flag. If no function is specified, MergeFlags, which treats the output as the result of a typical 'X-config' command (i.e. gtk-config), will merge the output into the appropriate variables.

ParseDepends(*self*, *filename*, *must_exist*=`None`, *only_one*=`0`)

Parse a mkdep-style file for explicit dependencies. This is completely abusable, and should be unnecessary in the "normal" case of proper SCons configuration, but it may help make the transition from a Make hierarchy easier for some people to swallow. It can also be genuinely useful when using a tool that can write a .d file, but for which writing a scanner would be too complicated.

Platform(*self*, *platform*)

Precious(*self*, **targets*)

Prepend(*self*, ***kw*)

Prepend values to existing construction variables in an Environment.

PrependENVPath(*self*, *name*, *newpath*, *envname*='ENV', *sep*=':', *delete_existing*=1)

Prepend path elements to the path 'name' in the 'ENV' dictionary for this environment. Will only add any particular path once, and will normpath and normcase all paths to help assure this. This can also handle the case where the env variable is a list instead of a string.

If delete_existing is 0, a newpath which is already in the path will not be moved to the front (it will be left where it is).

PrependUnique(*self*, *delete_existing*=0, ***kw*)

Prepend values to existing construction variables in an Environment, if they're not already there. If delete_existing is 1, removes existing values first, so values move to front.

Replace(*self*, ***kw*)

Replace existing construction variables in an Environment with new construction variables and/or values.

ReplaceIxes(*self*, *path*, *old_prefix*, *old_suffix*, *new_prefix*, *new_suffix*)

Replace old_prefix with new_prefix and old_suffix with new_suffix.

env - Environment used to interpolate variables. path - the path that will be modified. old_prefix - construction variable for the old prefix. old_suffix - construction variable for the old suffix. new_prefix - construction variable for the new prefix. new_suffix - construction variable for the new suffix.

Repository(*self*, **dirs*, ***kw*)

Requires(*self*, *target*, *prerequisite*)

Specify that 'prerequisite' must be built before 'target', (but 'target' does not actually depend on 'prerequisite' and need not be rebuilt if it changes).

SConsignFile(*self*, *name*=`'.sconsign'`, *dbm_module*=`None`)

Scanner(*self*, **args*, ***kw*)

SetDefault(*self*, ***kw*)

SideEffect(*self*, *side_effect*, *target*)

Tell scons that side_effects are built as side effects of building targets.

SourceCode(*self*, *entry*, *builder*)

Arrange for a source code builder for (part of) a tree.

SourceSignatures(*self*, *type*)

Split(*self*, *arg*)

```
This function converts a string or list into a list of strings
or Nodes.  This makes things easier for users by allow-
ing files to
be specified as a white-space separated list to be split.
The input rules are:
    - A single string containing names sepa-
rated by spaces. These will be
      split apart at the spaces.
    - A single Node instance
    - A list containing either strings or Node in-
stances. Any strings
      in the list are not split at spaces.
In all cases, the function returns a list of Nodes and strings.
```

TargetSignatures(*self*, *type*)

Tool(*self*, *tool*, *toolpath*=`None`, ***kw*)

Value(*self*, *value*, *built_value*=`None`)

VariantDir(*self*, *variant_dir*, *src_dir*, *duplicate*=`1`)

WhereIs(*self*, *prog*, *path*=`None`, *pathext*=`None`, *reject*=`[]`)

Find prog in the path.

__init__(*self*, *platform*=`None`, *tools*=`None`, *toolpath*=`None`, *variables*=`None`, *parse_flags*=`None`, ***kw*)

Initialization of a basic SCons construction environment, including setting up special construction variables like BUILDER, PLATFORM, etc., and searching for and applying available Tools.

Note that we do *not* call the underlying base class (SubsitutionEnvironment) initialization, because we need to initialize things in a very specific order that doesn't work with the much simpler base class initialization. Overrides: object.__init__

get_CacheDir(*self*)

get_builder(*self*, *name*)

Fetch the builder with the specified name from the environment.

get_factory(*self*, *factory*, *default*=`'File'`)

Return a factory function for creating Nodes for this construction environment.

get_scanner(*self*, *skey*)

Find the appropriate scanner given a key (usually a file suffix).

get_src_sig_type(*self*)

get_tgt_sig_type(*self*)

scanner_map_delete(*self*, *kw*=`None`)

Delete the cached scanner map (if we need to).

Inherited from SCons.Environment.SubstitutionEnvironment(Section 8.6)

AddMethod(), MergeFlags(), Override(), ParseFlags(), RemoveMethod(), __cmp__(), __contains__(), __delitem__(), __getitem__(), __setitem__(), arg2nodes(), backtick(), get(), gvars(), has_key(), items(), lvars(), subst(), subst_kw(), subst_list(), subst_path(), subst_target_source()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.7.2 Properties

Name	Description
Inherited from object	
__class__	

8.7.3 Class Variables

Name	Description
memoizer_counters	**Value:** `[]`
Inherited from SCons.Environment.SubstitutionEnvironment (Section 8.6)	
__metaclass__	

8.8 Class OverrideEnvironment

object
SCons.Environment.SubstitutionEnvironment
SCons.Environment.Base
SCons.Environment.OverrideEnvironment

A proxy that overrides variables in a wrapped construction environment by returning values from an overrides dictionary in preference to values from the underlying subject environment.

This is a lightweight (I hope) proxy that passes through most use of attributes to the underlying Environment.Base class, but has just enough additional methods defined to act like a real construction environment with overridden values. It can wrap either a Base construction environment, or another OverrideEnvironment, which can in turn nest arbitrary OverrideEnvironments...

Note that we do *not* call the underlying base class (SubsitutionEnvironment) initialization, because we get most of those from proxying the attributes of the subject construction environment. But because we subclass SubstitutionEnvironment, this class also has inherited arg2nodes() and subst*() methods; those methods can't be proxied because they need *this* object's methods to fetch the values from the overrides dictionary.

8.8.1 Methods

__init__(*self*, *subject*, *overrides*={})

Initialization of a basic SCons construction environment, including setting up special construction variables like BUILDER, PLATFORM, etc., and searching for and applying available Tools.

Note that we do *not* call the underlying base class (SubsitutionEnvironment) initialization, because we need to initialize things in a very specific order that doesn't work with the much simpler base class initialization. Overrides: object.__init__ extit(inherited documentation)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

x.__setattr__('name', value) <==> x.name = value Overrides: object.__setattr__ extit(inherited documentation)

__getitem__(*self*, *key*)

Overrides: SCons.Environment.SubstitutionEnvironment.__getitem__

__setitem__(*self*, *key*, *value*)

Overrides: SCons.Environment.SubstitutionEnvironment.__setitem__

__delitem__(*self*, *key*)

Overrides: SCons.Environment.SubstitutionEnvironment.__delitem__

get(*self*, *key*, *default*=`None`)

Emulates the get() method of dictionaries. Overrides: SCons.Environment.SubstitutionEnvironment.get

has_key(*self*, *key*)

Overrides: SCons.Environment.SubstitutionEnvironment.has_key

__contains__(*self*, *key*)

Overrides: SCons.Environment.SubstitutionEnvironment.__contains__

Dictionary(*self*)

Emulates the items() method of dictionaries. Overrides: SCons.Environment.Base.Dictionary

items(*self*)

Emulates the items() method of dictionaries. Overrides: SCons.Environment.SubstitutionEnvironment.items

gvars(*self*)

Overrides: SCons.Environment.SubstitutionEnvironment.gvars

lvars(*self*)

Overrides: SCons.Environment.SubstitutionEnvironment.lvars

Replace(*self*, ***kw*)

Replace existing construction variables in an Environment with new construction variables and/or values. Overrides: SCons.Environment.Base.Replace extit(inherited documentation)

Inherited from SCons.Environment.Base(Section 8.9)

Action(), AddPostAction(), AddPreAction(), Alias(), AlwaysBuild(), Append(), AppendENVPath(), AppendUnique(), BuildDir(), Builder(), CacheDir(), Clean(), Clone(), Command(), Configure(), Copy(), Decider(), Depends(), Detect(), Dir(), Dump(), Entry(), Environment(), Execute(), File(), FindFile(), FindInstalledFiles(), FindIxes(), FindSourceFiles(), Flatten(), GetBuildPath(), Glob(), Ignore(), Literal(), Local(), NoCache(), NoClean(), ParseConfig(), ParseDepends(), Platform(), Precious(), Prepend(), PrependENVPath(), PrependUnique(), ReplaceIxes(), Repository(), Requires(), SConsignFile(), Scanner(), SetDefault(), SideEffect(), SourceCode(), SourceSignatures(), Split(), TargetSignatures(), Tool(), Value(), VariantDir(), WhereIs(), get_CacheDir(), get_builder(), get_factory(), get_scanner(), get_src_sig_type(), get_tgt_sig_type(), scanner_map_delete()

Inherited from SCons.Environment.SubstitutionEnvironment(Section 8.6)

AddMethod(), MergeFlags(), Override(), ParseFlags(), RemoveMethod(), __cmp__(), arg2nodes(), backtick(), subst(), subst_kw(), subst_list(), subst_path(), subst_target_source()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __sizeof__(), __str__(), __subclasshook__()

8.8.2 Properties

Name	Description
Inherited from object	
__class__	

8.8.3 Class Variables

Name	Description
Inherited from SCons.Environment.Base (Section 8.9)	
memoizer_counters	
Inherited from SCons.Environment.SubstitutionEnvironment (Section 8.6)	
__metaclass__	

8.9 Class Base

object

SCons.Environment.SubstitutionEnvironment

SCons.Environment.Base

Known Subclasses: SCons.Environment.OverrideEnvironment, SCons.Script.SConscript'.SConsEnvironment

Base class for "real" construction Environments. These are the primary objects used to communicate dependency and construction information to the build engine.

Keyword arguments supplied when the construction Environment is created are construction variables used to initialize the Environment.

8.9.1 Methods

Action(*self*, **args*, ***kw*)

AddPostAction(*self*, *files*, *action*)

AddPreAction(*self*, *files*, *action*)

Alias(*self*, *target*, *source*=`[]`, *action*=`None`, ***kw*)

AlwaysBuild(*self*, **targets*)

Append(*self*, ***kw*)

Append values to existing construction variables in an Environment.

AppendENVPath(*self*, *name*, *newpath*, *envname*=`'ENV'`, *sep*=`':'`, *delete_existing*=`1`)

Append path elements to the path 'name' in the 'ENV' dictionary for this environment. Will only add any particular path once, and will normpath and normcase all paths to help assure this. This can also handle the case where the env variable is a list instead of a string.

If delete_existing is 0, a newpath which is already in the path will not be moved to the end (it will be left where it is).

AppendUnique(*self*, *delete_existing*=`0`, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there. If delete_existing is 1, removes existing values first, so values move to end.

BuildDir(*self*, **args*, ***kw*)

Builder(*self*, ***kw*)

CacheDir(*self*, *path*)

Clean(*self*, *targets*, *files*)

Clone(*self*, *tools*=`[]`, *toolpath*=`None`, *parse_flags*=`None`, ***kw*)

Return a copy of a construction Environment. The copy is like a Python "deep copy"--that is, independent copies are made recursively of each objects--except that a reference is copied when an object is not deep-copyable (like a function). There are no references to any mutable objects in the original Environment.

Command(*self*, *target*, *source*, *action*, ***kw*)

Builds the supplied target files from the supplied source files using the supplied action. Action may be any type that the Builder constructor will accept for an action.

Configure(*self*, **args*, ***kw*)

Copy(*self*, **args*, ***kw*)

Decider(*self*, *function*)

Depends(*self*, *target*, *dependency*)

Explicity specify that 'target's depend on 'dependency'.

Detect(*self*, *progs*)

Return the first available program in progs.

Dictionary(*self*, **args*)

Dir(*self*, *name*, **args*, ***kw*)

Dump(*self*, *key*=`None`)

Using the standard Python pretty printer, dump the contents of the scons build environment to stdout.

If the key passed in is anything other than None, then that will be used as an index into the build environment dictionary and whatever is found there will be fed into the pretty printer. Note that this key is case sensitive.

Entry(*self*, *name*, **args*, ***kw*)

Environment(*self*, ***kw*)

Execute(*self*, *action*, **args*, ***kw*)

Directly execute an action through an Environment

File(*self*, *name*, **args*, ***kw*)

FindFile(*self*, *file*, *dirs*)

FindInstalledFiles(*self*)

returns the list of all targets of the Install and InstallAs Builder.

FindIxes(*self*, *paths*, *prefix*, *suffix*)

Search a list of paths for something that matches the prefix and suffix.

paths - the list of paths or nodes. prefix - construction variable for the prefix. suffix - construction variable for the suffix.

FindSourceFiles(*self*, *node*=`'.'`)

returns a list of all source files.

Flatten(*self*, *sequence*)

GetBuildPath(*self*, *files*)

Glob(*self*, *pattern*, *ondisk*=`True`, *source*=`False`, *strings*=`False`)

Ignore(*self*, *target*, *dependency*)

Ignore a dependency.

Literal(*self*, *string*)

Local(*self*, **targets*)

NoCache(*self*, **targets*)

Tags a target so that it will not be cached

NoClean(*self*, **targets*)

Tags a target so that it will not be cleaned by -c

ParseConfig(*self*, *command*, *function*=`None`, *unique*=`1`)

Use the specified function to parse the output of the command in order to modify the current environment. The 'command' can be a string or a list of strings representing a command and its arguments. 'Function' is an optional argument that takes the environment, the output of the command, and the unique flag. If no function is specified, MergeFlags, which treats the output as the result of a typical 'X-config' command (i.e. gtk-config), will merge the output into the appropriate variables.

ParseDepends(*self*, *filename*, *must_exist*=`None`, *only_one*=`0`)

Parse a mkdep-style file for explicit dependencies. This is completely abusable, and should be unnecessary in the "normal" case of proper SCons configuration, but it may help make the transition from a Make hierarchy easier for some people to swallow. It can also be genuinely useful when using a tool that can write a .d file, but for which writing a scanner would be too complicated.

Platform(*self*, *platform*)

Precious(*self*, **targets*)

Prepend(*self*, ***kw*)

Prepend values to existing construction variables in an Environment.

PrependENVPath(*self*, *name*, *newpath*, *envname*=`'ENV'`, *sep*=`':'`, *delete_existing*=`1`)

Prepend path elements to the path 'name' in the 'ENV' dictionary for this environment. Will only add any particular path once, and will normpath and normcase all paths to help assure this. This can also handle the case where the env variable is a list instead of a string.

If delete_existing is 0, a newpath which is already in the path will not be moved to the front (it will be left where it is).

PrependUnique(*self*, *delete_existing*=`0`, ***kw*)

Prepend values to existing construction variables in an Environment, if they're not already there. If delete_existing is 1, removes existing values first, so values move to front.

Replace(*self*, ***kw*)

Replace existing construction variables in an Environment with new construction variables and/or values.

ReplaceIxes(*self*, *path*, *old_prefix*, *old_suffix*, *new_prefix*, *new_suffix*)

Replace old_prefix with new_prefix and old_suffix with new_suffix.

env - Environment used to interpolate variables. path - the path that will be modified. old_prefix - construction variable for the old prefix. old_suffix - construction variable for the old suffix. new_prefix - construction variable for the new prefix. new_suffix - construction variable for the new suffix.

Repository(*self*, **dirs*, ***kw*)

Requires(*self*, *target*, *prerequisite*)

Specify that 'prerequisite' must be built before 'target', (but 'target' does not actually depend on 'prerequisite' and need not be rebuilt if it changes).

SConsignFile(*self*, *name*=`'.sconsign'`, *dbm_module*=`None`)

Scanner(*self*, **args*, ***kw*)

SetDefault(*self*, ***kw*)

SideEffect(*self*, *side_effect*, *target*)

Tell scons that side_effects are built as side effects of building targets.

SourceCode(*self*, *entry*, *builder*)

Arrange for a source code builder for (part of) a tree.

SourceSignatures(*self*, *type*)

Split(*self*, *arg*)

```
This function converts a string or list into a list of strings
or Nodes.  This makes things easier for users by allow-
ing files to
be specified as a white-space separated list to be split.
The input rules are:
    - A single string containing names sepa-
rated by spaces. These will be
      split apart at the spaces.
    - A single Node instance
    - A list containing either strings or Node in-
stances. Any strings
      in the list are not split at spaces.
In all cases, the function returns a list of Nodes and strings.
```

TargetSignatures(*self*, *type*)

Tool(*self*, *tool*, *toolpath*=`None`, ***kw*)

Value(*self*, *value*, *built_value*=`None`)

VariantDir(*self*, *variant_dir*, *src_dir*, *duplicate*=`1`)

WhereIs(*self*, *prog*, *path*=`None`, *pathext*=`None`, *reject*=`[]`)

Find prog in the path.

__init__(*self*, *platform*=`None`, *tools*=`None`, *toolpath*=`None`, *variables*=`None`, *parse_flags*=`None`, ***kw*)

Initialization of a basic SCons construction environment, including setting up special construction variables like BUILDER, PLATFORM, etc., and searching for and applying available Tools.

Note that we do *not* call the underlying base class (SubsitutionEnvironment) initialization, because we need to initialize things in a very specific order that doesn't work with the much simpler base class initialization. Overrides: object.__init__

get_CacheDir(*self*)

get_builder(*self*, *name*)

Fetch the builder with the specified name from the environment.

get_factory(*self*, *factory*, *default*=`'File'`)

Return a factory function for creating Nodes for this construction environment.

get_scanner(*self*, *skey*)

Find the appropriate scanner given a key (usually a file suffix).

get_src_sig_type(*self*)

get_tgt_sig_type(*self*)

scanner_map_delete(*self*, *kw*=`None`)

Delete the cached scanner map (if we need to).

Inherited from SCons.Environment.SubstitutionEnvironment(Section 8.6)

AddMethod(), MergeFlags(), Override(), ParseFlags(), RemoveMethod(), __cmp__(), __contains__(), __delitem__(), __getitem__(), __setitem__(), arg2nodes(), backtick(), get(), gvars(), has_key(), items(), lvars(), subst(), subst_kw(), subst_list(), subst_path(), subst_target_source()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.9.2 Properties

Name	Description
Inherited from object	
__class__	

8.9.3 Class Variables

Name	Description
memoizer_counters	**Value:** `[]`
Inherited from SCons.Environment.SubstitutionEnvironment (Section 8.6)	
__metaclass__	

9 Module SCons.Errors

SCons.Errors

This file contains the exception classes used to handle internal and user errors in SCons.

9.1 Functions

convert_to_BuildError(*status*, *exc_info*=`None`)

Convert any return code a BuildError Exception.

'status' can either be a return code or an Exception. The buildError.status we set here will normally be used as the exit status of the "scons" process.

9.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Errors.py 5134 2010/08/16 23:02:40 bdee...`
__package__	**Value:** `'SCons'`

9.3 Class BuildError

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.BuildError

```
Errors occuring while building.

BuildError have the following attributes:

    Information about the cause of the build error:
    -----------------------------------------------

    errstr : a description of the error message
```

```
status : the return code of the action that caused the build
         error. Must be set to a non-zero value even if the
         build error is not due to an action returning a
         non-zero returned code.

exitstatus : SCons exit status due to this build error.
             Must be nonzero unless due to an explicit Exit()
             call.  Not always the same as status, since
             actions return a status code that should be
             respected, but SCons typically exits with 2
             irrespective of the return value of the failed
             action.

filename : The name of the file or directory that caused the
           build error. Set to None if no files are associated with
           this error. This might be different from the target
           being built. For example, failure to create the
           directory in which the target file will appear. It
           can be None if the error is not due to a particular
           filename.

exc_info : Info about exception that caused the build
           error. Set to (None, None, None) if this build
           error is not due to an exception.


Information about the cause of the location of the error:
---------------------------------------------------------

node : the error occured while building this target node(s)

executor : the executor that caused the build to fail (might
           be None if the build failures is not due to the
           executor failing)

action : the action that caused the build to fail (might be
         None if the build failures is not due to the an
         action failure)

command : the command line for the action that caused the
          build to fail (might be None if the build failures
          is not due to the an action failure)
```

9.3.1 Methods

__init__(*self*, *node*=None, *errstr*='Unknown error', *status*=2, *exitstatus*=2, *filename*=None, *executor*=None, *action*=None, *command*=None, *exc_info*=(None, None, None))

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

9.3.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

9.4 Class InternalError

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.InternalError

9.4.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

9.4.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

9.5 Class UserError

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

Known Subclasses: SCons.SConf.SConfError, SCons.Warnings.Warning

9.5.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

9.5.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

9.6 Class StopError

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.StopError

9.6.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

9.6.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

9.7 Class EnvironmentError

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.EnvironmentError

9.7.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

9.7.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

9.8 Class MSVCError

object
exceptions.BaseException
exceptions.Exception
exceptions.StandardError
exceptions.EnvironmentError
exceptions.IOError
SCons.Errors.MSVCError

9.8.1 Methods

Inherited from exceptions.IOError

__init__(), __new__()

Inherited from exceptions.EnvironmentError

__reduce__(), __str__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

9.8.2 Properties

Name	Description
Inherited from exceptions.EnvironmentError	
errno, filename, strerror	
Inherited from exceptions.BaseException	
args, message	
Inherited from object	

continued on next page

Name	Description
__class__	

9.9 Class ExplicitExit

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.ExplicitExit

9.9.1 Methods

__init__(*self*, *node*=`None`, *status*=`None`, **args*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

9.9.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

10 Module SCons.Executor

SCons.Executor

A module for executing actions with specific lists of target and source Nodes.

10.1 Functions

rfile(*node*)

A function to return the results of a Node's rfile() method, if it exists, and the Node itself otherwise (if it's a Value Node, e.g.).

GetBatchExecutor(*key*)

AddBatchExecutor(*key*, *executor*)

get_NullEnvironment()

Use singleton pattern for Null Environments.

10.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Executor.py 5134 2010/08/16 23:02:40 bd...`
nullenv	**Value:** `None`
__package__	**Value:** `'SCons'`

10.3 Class Batch

object ─┐
  SCons.Executor.Batch

Remembers exact association between targets and sources of executor.

10.3.1 Methods

__init__(*self*, *targets*=`[]`, *sources*=`[]`)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

10.3.2 Properties

Name	Description
Inherited from object	
__class__	

10.4 Class TSList

object

_abcoll.Sized

object

_abcoll.Iterable

object

_abcoll.Container

_abcoll.Sequence

_abcoll.MutableSequence

UserList.UserList

SCons.Executor.TSList

A class that implements $TARGETS or $SOURCES expansions by wrapping an executor

Method. This class is used in the Executor.lvars() to delay creation of NodeList objects until they're needed.

Note that we subclass collections.UserList purely so that the is_Sequence() function will identify an object of this class as a list during variable expansion. We're not really using any collections.UserList methods in practice.

10.4.1 Methods

__init__(*self*, *func*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__getattr__(*self*, *attr*)

__getitem__(*self*, *i*)

Overrides: _abcoll.Sequence.__getitem__

__getslice__(*self*, *i*, *j*)

Overrides: UserList.UserList.__getslice__

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

Inherited from UserList.UserList

__add__(), __cmp__(), __contains__(), __delitem__(), __delslice__(), __eq__(), __ge__(), __gt__(), __iadd__(), __imul__(), __le__(), __len__(), __lt__(), __mul__(), __ne__(), __radd__(), __rmul__(), __setitem__(), __setslice__(), append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

Inherited from _abcoll.Sequence

__iter__(), __reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__()

10.4.2 Properties

Name	Description
Inherited from object	
__class__	

10.4.3 Class Variables

Name	Description
Inherited from UserList.UserList	
__abstractmethods__, __hash__	

10.5 Class TSObject

object ─┐
SCons.Executor.TSObject

A class that implements $TARGET or $SOURCE expansions by wrapping an Executor method.

10.5.1 Methods

__init__(*self*, *func*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__getattr__(*self*, *attr*)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __subclasshook__()

10.5.2 Properties

Name	Description
Inherited from object	
__class__	

10.6 Class Executor

object
└ **SCons.Executor.Executor**

A class for controlling instances of executing an action.

This largely exists to hold a single association of an action, environment, list of environment override dictionaries, targets and sources for later processing as needed.

10.6.1 Methods

__init__(*self*, *action*, *env*=`None`, *overridelist*=`[{}]`, *targets*=`[]`, *sources*=`[]`, *builder_kw*=`{}`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

get_lvars(*self*)

get_action_targets(*self*)

set_action_list(*self*, *action*)

get_action_list(*self*)

get_all_targets(*self*)

Returns all targets for all batches of this Executor.

get_all_sources(*self*)

Returns all sources for all batches of this Executor.

get_all_children(*self*)

Returns all unique children (dependencies) for all batches of this Executor.

The Taskmaster can recognize when it's already evaluated a Node, so we don't have to make this list unique for its intended canonical use case, but we expect there to be a lot of redundancy (long lists of batched .cc files #including the same .h files over and over), so removing the duplicates once up front should save the Taskmaster a lot of work.

get_all_prerequisites(*self*)

Returns all unique (order-only) prerequisites for all batches of this Executor.

get_action_side_effects(*self*)

Returns all side effects for all batches of this Executor used by the underlying Action.

get_build_env(*self*)

Fetch or create the appropriate build Environment for this Executor.

get_build_scanner_path(*self*, *scanner*)

Fetch the scanner path for this executor's targets and sources.

get_kw(*self*, *kw*=`{}`)

do_nothing(*self*, *target*, *kw*)

do_execute(*self*, *target*, *kw*)

Actually execute the action list.

__call__(*self*, *target*, ***kw*)

cleanup(*self*)

add_sources(*self*, *sources*)

Add source files to this Executor's list. This is necessary for "multi" Builders that can be called repeatedly to build up a source file list for a given target.

get_sources(*self*)

add_batch(*self*, *targets*, *sources*)

Add pair of associated target and source to this Executor's list. This is necessary for "batch" Builders that can be called repeatedly to build up a list of matching target and source files that will be used in order to update multiple target files at once from multiple corresponding source files, for tools like MSVC that support it.

prepare(*self*)

Preparatory checks for whether this Executor can go ahead and (try to) build its targets.

add_pre_action(*self*, *action*)

add_post_action(*self*, *action*)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

nullify(*self*)

get_contents(*self*)

Fetch the signature contents. This is the main reason this class exists, so we can compute this once and cache it regardless of how many target or source Nodes there are.

get_timestamp(*self*)

Fetch a time stamp for this Executor. We don't have one, of course (only files do), but this is the interface used by the timestamp module.

scan_targets(*self*, *scanner*)

scan_sources(*self*, *scanner*)

scan(*self*, *scanner*, *node_list*)

Scan a list of this Executor's files (targets or sources) for implicit dependencies and update all of the targets with them. This essentially short-circuits an N*M scan of the sources for each individual target, which is a hell of a lot more efficient.

get_unignored_sources(*self*, *node*, *ignore*=`()`)

get_implicit_deps(*self*)

Return the executor's implicit dependencies, i.e. the nodes of the commands to be executed.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

10.6.2 Properties

Name	Description
Inherited from object	
__class__	

10.6.3 Class Variables

Name	Description
memoizer_counters	**Value:** []

10.6.4 Instance Variables

Name	Description
my_str	**Value:** <function my_str at 0x131b140>

10.7 Class Null

object ─┐

SCons.Executor.Null

A null Executor, with a null build Environment, that does nothing when the rest of the methods call it.

This might be able to disapper when we refactor things to disassociate Builders from Nodes entirely, so we're not going to worry about unit tests for this--at least for now.

10.7.1 Methods

__init__(*self*, **args*, ***kw*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

get_build_env(*self*)

get_build_scanner_path(*self*)

cleanup(*self*)

prepare(*self*)

get_unignored_sources(*self*, **args*, ***kw*)

get_action_targets(*self*)

get_action_list(*self*)

get_all_targets(*self*)

get_all_sources(*self*)

get_all_children(*self*)

get_all_prerequisites(*self*)

get_action_side_effects(*self*)

__call__(*self*, **args*, ***kw*)

get_contents(*self*)

add_pre_action(*self*, *action*)

add_post_action(*self*, *action*)

set_action_list(*self*, *action*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

10.7.2 Properties

Name	Description
Inherited from object	
__class__	

11 Module SCons.Job

SCons.Job

This module defines the Serial and Parallel classes that execute tasks to complete a build. The Jobs class provides a higher level interface to start, stop, and wait on jobs.

11.1 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Job.py 5134 2010/08/16 23:02:40 bdeegan'`
explicit_stack_size	**Value:** `None`
default_stack_size	**Value:** `256`
interrupt_msg	**Value:** `'Build interrupted.'`
__package__	**Value:** `'SCons'`

11.2 Class InterruptState

object
SCons.Job.InterruptState

11.2.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

set(*self*)

__call__(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.2.2 Properties

Name	Description
Inherited from object	
__class__	

11.3 Class Jobs

object ─┐
SCons.Job.Jobs

An instance of this class initializes N jobs, and provides methods for starting, stopping, and waiting on all N jobs.

11.3.1 Methods

__init__(*self*, *num*, *taskmaster*)

create 'num' jobs using the given taskmaster.

If 'num' is 1 or less, then a serial job will be used, otherwise a parallel job with 'num' worker threads will be used.

The 'num_jobs' attribute will be set to the actual number of jobs allocated. If more than one job is requested but the Parallel class can't do it, it gets reset to 1. Wrapping interfaces that care should check the value of 'num_jobs' after initialization. Overrides: object.__init__

run(*self*, *postfunc*=`<function <lambda> at 0x18c6aa0>`)

Run the jobs.

postfunc() will be invoked after the jobs has run. It will be invoked even if the jobs are interrupted by a keyboard interrupt (well, in fact by a signal such as either SIGINT, SIGTERM or SIGHUP). The execution of postfunc() is protected against keyboard interrupts and is guaranteed to run to completion.

were_interrupted(*self*)

Returns whether the jobs were interrupted by a signal.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.3.2 Properties

Name	Description
Inherited from object	
__class__	

11.4 Class Serial

object ─┐
SCons.Job.Serial

This class is used to execute tasks in series, and is more efficient than Parallel, but is only appropriate for non-parallel builds. Only one instance of this class should be in existence at a time.

This class is not thread safe.

11.4.1 Methods

__init__(*self*, *taskmaster*)

Create a new serial job given a taskmaster.

The taskmaster's next_task() method should return the next task that needs to be executed, or None if there are no more tasks. The taskmaster's executed() method will be called for each task when it is successfully executed or failed() will be called if it failed to execute (e.g. execute() raised an exception). Overrides: object.__init__

start(*self*)

Start the job. This will begin pulling tasks from the taskmaster and executing them, and return when there are no more tasks. If a task fails to execute (i.e. execute() raises an exception), then the job will stop.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.4.2 Properties

Name	Description
Inherited from object	
__class__	

11.5 Class Worker

object

threading._Verbose

threading.Thread

SCons.Job.Worker

A worker thread waits on a task to be posted to its request queue, dequeues the task, executes it, and posts a tuple including the task and a boolean indicating whether the task executed successfully.

11.5.1 Methods

__init__(*self*, *requestQueue*, *resultsQueue*, *interrupted*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

run(*self*)

Overrides: threading.Thread.run

Inherited from threading.Thread

__repr__(), getName(), isAlive(), isDaemon(), is_alive(), join(), setDaemon(), setName(), start()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.5.2 Properties

Name	Description
Inherited from threading.Thread	
daemon, ident, name	
Inherited from object	
__class__	

11.6 Class ThreadPool

object
└ **SCons.Job.ThreadPool**

This class is responsible for spawning and managing worker threads.

11.6.1 Methods

__init__(*self*, *num*, *stack_size*, *interrupted*)

Create the request and reply queues, and 'num' worker threads.

One must specify the stack size of the worker threads. The stack size is specified in kilobytes. Overrides: object.__init__

put(*self*, *task*)

Put task into request queue.

get(*self*)

Remove and return a result tuple from the results queue.

preparation_failed(*self*, *task*)

cleanup(*self*)

Shuts down the thread pool, giving each worker thread a chance to shut down gracefully.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.6.2 Properties

Name	Description
Inherited from object	
__class__	

11.7 Class Parallel

object
└ **SCons.Job.Parallel**

This class is used to execute tasks in parallel, and is somewhat less efficient than Serial, but is appropriate for parallel builds.

This class is thread safe.

11.7.1 Methods

__init__(*self*, *taskmaster*, *num*, *stack_size*)

Create a new parallel job given a taskmaster.

The taskmaster's next_task() method should return the next task that needs to be executed, or None if there are no more tasks. The taskmaster's executed() method will be called for each task when it is successfully executed or failed() will be called if the task failed to execute (i.e. execute() raised an exception).

Note: calls to taskmaster are serialized, but calls to execute() on distinct tasks are not serialized, because that is the whole point of parallel jobs: they can execute multiple tasks simultaneously. Overrides: object.__init__

start(*self*)

Start the job. This will begin pulling tasks from the taskmaster and executing them, and return when there are no more tasks. If a task fails to execute (i.e. execute() raises an exception), then the job will stop.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.7.2 Properties

Name	Description
Inherited from object	
__class__	

12 Module SCons.Memoize

```
Memoizer

A metaclass implementation to count hits and misses of the computed
values that various methods cache in memory.

Use of this modules assumes that wrapped methods be coded to cache their
values in a consistent way.  Here is an example of wrapping a method
that returns a computed value, with no input parameters:

    memoizer_counters = []                                    # Memoization

    memoizer_counters.append(SCons.Memoize.CountValue('foo'))   # Memoization

    def foo(self):

        try:                                                    # Memoization
            return self._memo['foo']                            # Memoization
        except KeyError:                                        # Memoization
            pass                                                # Memoization

        result = self.compute_foo_value()

        self._memo['foo'] = result                              # Memoization

        return result

Here is an example of wrapping a method that will return different values
based on one or more input arguments:

    def _bar_key(self, argument):                               # Memoization
        return argument                                         # Memoization

    memoizer_counters.append(SCons.Memoize.CountDict('bar', _bar_key)) # Memoization

    def bar(self, argument):

        memo_key = argument                                     # Memoization
        try:                                                    # Memoization
            memo_dict = self._memo['bar']                       # Memoization
        except KeyError:                                        # Memoization
            memo_dict = {}                                      # Memoization
            self._memo['dict'] = memo_dict                      # Memoization
        else:                                                   # Memoization
```

```
        try:                                              # Memoization
            return memo_dict[memo_key]                    # Memoization
        except KeyError:                                  # Memoization
            pass                                          # Memoization

    result = self.compute_bar_value(argument)

    memo_dict[memo_key] = result                          # Memoization

    return result
```

```
At one point we avoided replicating this sort of logic in all the methods
by putting it right into this module, but we've moved away from that at
present (see the "Historical Note," below.).

Deciding what to cache is tricky, because different configurations
can have radically different performance tradeoffs, and because the
tradeoffs involved are often so non-obvious.  Consequently, deciding
whether or not to cache a given method will likely be more of an art than
a science, but should still be based on available data from this module.
Here are some VERY GENERAL guidelines about deciding whether or not to
cache return values from a method that's being called a lot:

    --  The first question to ask is, "Can we change the calling code
        so this method isn't called so often?"  Sometimes this can be
        done by changing the algorithm.  Sometimes the *caller* should
        be memoized, not the method you're looking at.

    --  The memoized function should be timed with multiple configurations
        to make sure it doesn't inadvertently slow down some other
        configuration.

    --  When memoizing values based on a dictionary key composed of
        input arguments, you don't need to use all of the arguments
        if some of them don't affect the return values.

Historical Note:  The initial Memoizer implementation actually handled
the caching of values for the wrapped methods, based on a set of generic
algorithms for computing hashable values based on the method's arguments.
This collected caching logic nicely, but had two drawbacks:

    Running arguments through a generic key-conversion mechanism is slower
    (and less flexible) than just coding these things directly.  Since the
    methods that need memoized values are generally performance-critical,
    slowing them down in order to collect the logic isn't the right
```

```
tradeoff.

Use of the memoizer really obscured what was being called, because
all the memoized methods were wrapped with re-used generic methods.
This made it more difficult, for example, to use the Python profiler
to figure out how to optimize the underlying methods.
```

12.1 Functions

Dump(*title*=None)

EnableMemoization()

12.2 Variables

Name	Description
__revision__	**Value:** 'src/engine/SCons/Memoize.py 5134 2010/08/16 23:02:40 bde...
__doc__	**Value:** """Memoi...
use_memoizer	**Value:** None
CounterList	**Value:** []
__package__	**Value:** 'SCons'

12.3 Class Counter

object ┐
SCons.Memoize.Counter

Known Subclasses: SCons.Memoize.CountDict, SCons.Memoize.CountValue

Base class for counting memoization hits and misses.

We expect that the metaclass initialization will have filled in the .name attribute that represents the name of the function being counted.

12.3.1 Methods

__init__(*self*, *method_name*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__

display(*self*)

__cmp__(*self*, *other*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.3.2 Properties

Name	Description
Inherited from object	
__class__	

12.4 Class CountValue

object
SCons.Memoize.Counter
SCons.Memoize.CountValue

A counter class for simple, atomic memoized values.

A CountValue object should be instantiated in a class for each of the class's methods that memoizes its return value by simply storing the return value in its _memo dictionary.

We expect that the metaclass initialization will fill in the .underlying_method attribute with the method that we're wrapping. We then call the underlying_method method after counting whether its memoized value has already been set (a hit) or not (a miss).

12.4.1 Methods

__call__(*self*, **args*, ***kw*)

Inherited from SCons.Memoize.Counter(Section 12.3)

__cmp__(), __init__(), display()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.4.2 Properties

Name	Description
Inherited from object	
__class__	

12.5 Class CountDict

object ─┐

SCons.Memoize.Counter ─┐

SCons.Memoize.CountDict

A counter class for memoized values stored in a dictionary, with keys based on the method's input arguments.

A CountDict object is instantiated in a class for each of the class's methods that memoizes its return value in a dictionary, indexed by some key that can be computed from one or more of its input arguments.

We expect that the metaclass initialization will fill in the .underlying_method attribute with the method that we're wrapping. We then call the underlying_method method after counting whether the computed key value is already present in the memoization dictionary (a hit) or not (a miss).

12.5.1 Methods

__init__(*self*, *method_name*, *keymaker*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__

__call__(*self*, **args*, ***kw*)

Inherited from SCons.Memoize.Counter(Section 12.3)

__cmp__(), display()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.5.2 Properties

Name	Description
Inherited from object	
__class__	

12.6 Class Memoizer

object ─┐
SCons.Memoize.Memoizer

Object which performs caching of method calls for its 'primary' instance.

12.6.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.6.2 Properties

Name	Description
Inherited from object	
__class__	

12.7 Class Memoized_Metaclass

object
  type
    SCons.Memoize.Memoized_Metaclass

12.7.1 Methods

__init__(*cls*, *name*, *bases*, *cls_dict*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature **Return Value**
the object's type

Overrides: object.__init__ extit(inherited documentation)

Inherited from type

__call__(), __cmp__(), __delattr__(), __eq__(), __ge__(), __getattribute__(), __gt__(), __hash__(), __le__(), __lt__(), __ne__(), __new__(), __repr__(), __setattr__(), __subclasses__(), mro()

Inherited from object

__format__(), __reduce__(), __reduce_ex__(), __sizeof__(), __str__(), __subclasshook__()

12.7.2 Properties

Name	Description
Inherited from type	
__abstractmethods__, __base__, __bases__, __basicsize__, __dictoffset__, __flags__, __instancecheck__, __itemsize__, __mro__, __name__, __subclasscheck__, __weakrefoffset__	
Inherited from object	
__class__	

13 Package SCons.Node

SCons.Node

The Node package for the SCons software construction utility.

This is, in many ways, the heart of SCons.

A Node is where we encapsulate all of the dependency information about any thing that SCons can build, or about any thing which SCons can use to build some other thing. The canonical "thing," of course, is a file, but a Node can also represent something remote (like a web page) or something completely abstract (like an Alias).

Each specific type of "thing" is specifically represented by a subclass of the Node base class: Node.FS.File for files, Node.Alias for aliases, etc. Dependency information is kept here in the base class, and information specific to files/aliases/etc. is in the subclass. The goal, if we've done this correctly, is that any type of "thing" should be able to depend on any other type of "thing."

13.1 Modules

- **Alias**: scons.Node.Alias
  (Section 14, p. 124)
- **FS**: scons.Node.FS
  (Section 15, p. 130)
- **Python**: scons.Node.Python
  (Section 16, p. 173)

13.2 Functions

classname(*obj*)

Annotate(*node*)

get_children(*node, parent*)

ignore_cycle(*node, stack*)

do_nothing(*node, parent*)

13.3 Variables

Name	Description
__revision__	**Value:** '`src/engine/SCons/Node/__init__.py 5134 2010/08/16 23:02:...`'
no_state	**Value:** `0`
pending	**Value:** `1`
executing	**Value:** `2`
up_to_date	**Value:** `3`
executed	**Value:** `4`
failed	**Value:** `5`
StateString	**Value:** `{0: 'no_state', 1: 'pending', 2: 'executing', 3: 'up_to_d...`
implicit_cache	**Value:** `0`
implicit_deps_unchanged	**Value:** `0`
implicit_deps_changed	**Value:** `0`
arg2nodes_lookups	**Value:** `[<bound method AliasNameSpace.lookup of {}>]`
__package__	**Value:** `'SCons.Node'`

13.4 Class NodeInfoBase

object ─┐
SCons.Node.NodeInfoBase

Known Subclasses: SCons.Node.Alias.AliasNodeInfo, SCons.Node.FS.DirNodeInfo, SCons.Node.FS.FileNodeInfo, SCons.Node.Python.ValueNodeInfo

The generic base class for signature information for a Node.

Node subclasses should subclass NodeInfoBase to provide their own logic for dealing with their own Node-specific signature information.

13.4.1 Methods

__init__(*self*, *node*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

convert(*self*, *node*, *val*)

format(*self*, *field_list*=`None`, *names*=`0`)

merge(*self*, *other*)

update(*self*, *node*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.4.2 Properties

Name	Description
Inherited from object	
__class__	

13.4.3 Class Variables

Name	Description
current_version_id	**Value:** `1`

13.5 Class BuildInfoBase

object
└ **SCons.Node.BuildInfoBase**

Known Subclasses: SCons.Node.Alias.AliasBuildInfo, SCons.Node.FS.DirBuildInfo, SCons.Node.FS.FileBuildInfo, SCons.Node.Python.ValueBuildInfo

The generic base class for build information for a Node.

This is what gets stored in a .sconsign file for each target file. It contains a NodeInfo instance for this node (signature information that's specific to the type of Node) and direct attributes for the generic build stuff we have to track: sources, explicit dependencies, implicit dependencies, and action information.

13.5.1 Methods

__init__(*self*, *node*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

merge(*self*, *other*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.5.2 Properties

Name	Description
Inherited from object	
__class__	

13.5.3 Class Variables

Name	Description
current_version_id	**Value:** `1`

13.6 Class Node

object
    SCons.Node.Node

Known Subclasses: SCons.Node.Alias.Alias, SCons.Node.FS.Base, SCons.Node.Python.Value

The base Node class, for entities that we know how to build, or use to build other Nodes.

13.6.1 Methods

Decider(*self*, *function*)

__init__(*self*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

add_dependency(*self*, *depend*)

Adds dependencies.

add_ignore(*self*, *depend*)

Adds dependencies to ignore.

add_prerequisite(*self*, *prerequisite*)

Adds prerequisites

add_source(*self*, *source*)

Adds sources.

add_to_implicit(*self*, *deps*)

add_to_waiting_parents(*self*, *node*)

Returns the number of nodes added to our waiting parents list: 1 if we add a unique waiting parent, 0 if not. (Note that the returned values are intended to be used to increment a reference count, so don't think you can "clean up" this function by using True and False instead...)

add_to_waiting_s_e(*self*, *node*)

add_wkid(*self*, *wkid*)

Add a node to the list of kids waiting to be evaluated

all_children(*self*, *scan*=`1`)

Return a list of all the node's direct children.

alter_targets(*self*)

Return a list of alternate targets for this Node.

build(*self*, ***kw*)

Actually build the node.

This is called by the Taskmaster after it's decided that the Node is out-of-date and must be rebuilt, and after the prepare() method has gotten everything, uh, prepared.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in built().

builder_set(*self*, *builder*)

built(*self*)

Called just after this node is successfully built.

changed(*self*, *node*=`None`)

Returns if the node is up-to-date with respect to the BuildInfo stored last time it was built. The default behavior is to compare it against our own previously stored BuildInfo, but the stored BuildInfo from another Node (typically one in a Repository) can be used instead.

Note that we now *always* check every dependency. We used to short-circuit the check by returning as soon as we detected any difference, but we now rely on checking every dependency to make sure that any necessary Node information (for example, the content signature of an #included .h file) is updated.

changed_since_last_build(*self*, *target*, *prev_ni*)

Must be overridden in a specific subclass to return True if this Node (a dependency) has changed since the last time it was used to build the specified target. prev_ni is this Node's state (for example, its file timestamp, length, maybe content signature) as of the last time the target was built.

Note that this method is called through the dependency, not the target, because a dependency Node must be able to use its own logic to decide if it changed. For example, File Nodes need to obey if we're configured to use timestamps, but Python Value Nodes never use timestamps and always use the content. If this method were called through the target, then each Node's implementation of this method would have to have more complicated logic to handle all the different Node types on which it might depend.

children(*self*, *scan*=1)

Return a list of the node's direct children, minus those that are ignored by this node.

children_are_up_to_date(*self*)

Alternate check for whether the Node is current: If all of our children were up-to-date, then this Node was up-to-date, too.

The SCons.Node.Alias and SCons.Node.Python.Value subclasses rebind their current() method to this method.

clear(*self*)

Completely clear a Node of all its cached state (so that it can be re-evaluated by interfaces that do continuous integration builds).

clear_memoized_values(*self*)

del_binfo(*self*)

Delete the build info from this node.

disambiguate(*self*, *must_exist*=`None`)

do_not_store_info(*self*)

env_set(*self*, *env*, *safe*=`0`)

executor_cleanup(*self*)

Let the executor clean up any cached information.

exists(*self*)

Does this node exists?

explain(*self*)

for_signature(*self*)

Return a string representation of the Node that will always be the same for this particular Node, no matter what. This is by contrast to the __str__() method, which might, for instance, return a relative path for a file Node. The purpose of this method is to generate a value to be used in signature calculation for the command line used to build a target, and we use this method instead of str() to avoid unnecessary rebuilds. This method does not need to return something that would actually work in a command line; it can return any kind of nonsense, so long as it does not change.

get_abspath(*self*)

Return an absolute path to the Node. This will return simply str(Node) by default, but for Node types that have a concept of relative path, this might return something different.

get_binfo(*self*)

Fetch a node's build information.

node - the node whose sources will be collected cache - alternate node to use for the signature cache returns - the build signature

This no longer handles the recursive descent of the node's children's signatures. We expect that they're already built and updated by someone else, if that's what's wanted.

get_build_env(*self*)

Fetch the appropriate Environment to build this node.

get_build_scanner_path(*self*, *scanner*)

Fetch the appropriate scanner path for this node.

get_builder(*self*, *default_builder*=`None`)

Return the set builder, or a specified default value

get_cachedir_csig(*self*)

get_csig(*self*)

get_env(*self*)

get_env_scanner(*self*, *env*, *kw*=`{}`)

get_executor(*self*, *create*=`1`)

Fetch the action executor for this node. Create one if there isn't already one, and requested to do so.

get_found_includes(*self*, *env*, *scanner*, *path*)

Return the scanned include lines (implicit dependencies) found in this node.

The default is no implicit dependencies. We expect this method to be overridden by any subclass that can be scanned for implicit dependencies.

get_implicit_deps(*self*, *env*, *scanner*, *path*)

Return a list of implicit dependencies for this node.

This method exists to handle recursive invocation of the scanner on the implicit dependencies returned by the scanner, if the scanner's recursive flag says that we should.

get_ninfo(*self*)

get_source_scanner(*self*, *node*)

Fetch the source scanner for the specified node

NOTE: "self" is the target being built, "node" is the source file for which we want to fetch the scanner.

Implies self.has_builder() is true; again, expect to only be called from locations where this is already verified.

This function may be called very often; it attempts to cache the scanner found to improve performance.

get_state(*self*)

get_stored_implicit(*self*)

Fetch the stored implicit dependencies

get_stored_info(*self*)

get_string(*self*, *for_signature*)

This is a convenience function designed primarily to be used in command generators (i.e., CommandGeneratorActions or Environment variables that are callable), which are called with a for_signature argument that is nonzero if the command generator is being called to generate a signature for the command line, which determines if we should rebuild or not.

Such command generators should use this method in preference to str(Node) when converting a Node to a string, passing in the for_signature parameter, such that we will call Node.for_signature() or str(Node) properly, depending on whether we are calculating a signature or actually constructing a command line.

get_subst_proxy(*self*)

This method is expected to return an object that will function exactly like this Node, except that it implements any additional special features that we would like to be in effect for Environment variable substitution. The principle use is that some Nodes would like to implement a __getattr__() method, but putting that in the Node type itself has a tendency to kill performance. We instead put it in a proxy and return it from this method. It is legal for this method to return self if no new functionality is needed for Environment substitution.

get_suffix(*self*)

get_target_scanner(*self*)

has_builder(*self*)

Return whether this Node has a builder or not.

In Boolean tests, this turns out to be a *lot* more efficient than simply examining the builder attribute directly ("if node.builder: ..."). When the builder attribute is examined directly, it ends up calling __getattr__ for both the __len__ and __nonzero__ attributes on instances of our Builder Proxy class(es), generating a bazillion extra calls and slowing things down immensely.

has_explicit_builder(*self*)

Return whether this Node has an explicit builder

This allows an internal Builder created by SCons to be marked non-explicit, so that it can be overridden by an explicit builder that the user supplies (the canonical example being directories).

is_derived(*self*)

Returns true iff this node is derived (i.e. built).

This should return true only for nodes whose path should be in the variant directory when duplicate=0 and should contribute their build signatures when they are used as source files to other derived files. For example: source with source builders are not derived in this sense, and hence should not return true.

is_literal(*self*)

Always pass the string representation of a Node to the command interpreter literally.

is_up_to_date(*self*)

Default check for whether the Node is current: unknown Node subtypes are always out of date, so they will always get built.

make_ready(*self*)

Get a Node ready for evaluation.

This is called before the Taskmaster decides if the Node is up-to-date or not. Overriding this method allows for a Node subclass to be disambiguated if necessary, or for an implicit source builder to be attached.

missing(*self*)

multiple_side_effect_has_builder(*self*)

Return whether this Node has a builder or not.

In Boolean tests, this turns out to be a *lot* more efficient than simply examining the builder attribute directly ("if node.builder: ..."). When the builder attribute is examined directly, it ends up calling __getattr__ for both the __len__ and __nonzero__ attributes on instances of our Builder Proxy class(es), generating a bazillion extra calls and slowing things down immensely.

new_binfo(*self*)

new_ninfo(*self*)

postprocess(*self*)

Clean up anything we don't need to hang onto after we've been built.

prepare(*self*)

Prepare for this Node to be built.

This is called after the Taskmaster has decided that the Node is out-of-date and must be rebuilt, but before actually calling the method to build the Node.

This default implementation checks that explicit or implicit dependencies either exist or are derived, and initializes the BuildInfo structure that will hold the information about how this node is, uh, built.

(The existence of source files is checked separately by the Executor, which aggregates checks for all of the targets built by a specific action.)

Overriding this method allows for for a Node subclass to remove the underlying file from the file system. Note that subclass methods should call this base class method to get the child check and the BuildInfo structure.

push_to_cache(*self*)

Try to push a node into a cache

remove(*self*)

Remove this Node: no-op by default.

render_include_tree(*self*)

Return a text representation, suitable for displaying to the user, of the include tree for the sources of this node.

reset_executor(*self*)

Remove cached executor; forces recompute when needed.

retrieve_from_cache(*self*)

Try to retrieve the node's content from a cache

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in built().

Returns true iff the node was successfully retrieved.

rexists(*self*)

Does this node exist locally or in a repositiory?

scan(*self*)

Scan this node's dependents for implicit dependencies.

scanner_key(*self*)

select_scanner(*self*, *scanner*)

Selects a scanner for this Node.

This is a separate method so it can be overridden by Node subclasses (specifically, Node.FS.Dir) that *must* use their own Scanner and don't select one the Scanner.Selector that's configured for the target.

set_always_build(*self*, *always_build*=1)

Set the Node's always_build value.

set_executor(*self*, *executor*)

Set the action executor for this node.

set_explicit(*self*, *is_explicit*)

set_nocache(*self*, *nocache*=1)

Set the Node's nocache value.

set_noclean(*self*, *noclean*=1)

Set the Node's noclean value.

set_precious(*self*, *precious*=1)

Set the Node's precious value.

set_specific_source(*self*, *source*)

set_state(*self*, *state*)

state_has_changed(*self*, *target*, *prev_ni*)

store_info(*self*)

Make the build signature permanent (that is, store it in the .sconsign file or equivalent).

visited(*self*)

Called just after this node has been visited (with or without a build).

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.6.2 Properties

Name	Description
Inherited from object	
__class__	

13.6.3 Class Variables

Name	Description
__metaclass__	**Value:** `SCons.Memoize.Memoized_Metaclass`
memoizer_counters	**Value:** `[]`

13.7 Class NodeList

object

_abcoll.Sized

object

_abcoll.Iterable

object

_abcoll.Container

_abcoll.Sequence

_abcoll.MutableSequence

UserList.UserList

SCons.Node.NodeList

13.7.1 Methods

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from UserList.UserList

__add__(), __cmp__(), __contains__(), __delitem__(), __delslice__(), __eq__(), __ge__(), __getitem__(), __getslice__(), __gt__(), __iadd__(), __imul__(), __init__(), __le__(), __len__(), __lt__(), __mul__(), __ne__(), __radd__(), __repr__(), __rmul__(), __setitem__(), __setslice__(), append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

Inherited from _abcoll.Sequence

__iter__(), __reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__()

13.7.2 Properties

Name	Description
Inherited from object	
__class__	

13.7.3 Class Variables

Name	Description
Inherited from UserList.UserList	
__abstractmethods__, __hash__	

13.8 Class Walker

object
└─ **SCons.Node.Walker**

An iterator for walking a Node tree.

This is depth-first, children are visited before the parent. The Walker object can be initialized with any node, and returns the next node on the descent with each get_next() call. 'kids_func' is an optional function that will be called to get the children of a node instead of calling 'children'. 'cycle_func' is an optional function that will be called when a cycle is detected.

This class does not get caught in node cycles caused, for example, by C header file include loops.

13.8.1 Methods

__init__(*self*, *node*, *kids_func*=<function get_children at 0x13fade8>, *cycle_func*=<function ignore_cycle at 0x13fae60>, *eval_func*=<function do_nothing at 0x13faed8>)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

get_next(*self*)

Return the next node for this walk of the tree.

This function is intentionally iterative, not recursive, to sidestep any issues of stack size limitations.

is_done(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.8.2 Properties

Name	Description
Inherited from object	
__class__	

14 Module SCons.Node.Alias

scons.Node.Alias

Alias nodes.

This creates a hash of global Aliases (dummy targets).

14.1 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Node/Alias.py 5134 2010/08/16 23:02:40 ...`
default_ans	**Value:** `{}`
__package__	**Value:** `'SCons.Node'`

14.2 Class AliasNameSpace

UserDict.UserDict
└ **SCons.Node.Alias.AliasNameSpace**

14.2.1 Methods

Alias(*self*, *name*, ***kw*)

lookup(*self*, *name*, ***kw*)

Inherited from UserDict.UserDict

__cmp__(), __contains__(), __delitem__(), __getitem__(), __init__(), __len__(), __repr__(), __setitem__(), clear(), copy(), fromkeys(), get(), has_key(), items(), iteritems(), iterkeys(), itervalues(), keys(), pop(), popitem(), setdefault(), update(), values()

14.3 Class AliasNodeInfo

```
object ─┐
        │
SCons.Node.NodeInfoBase ─┐
                         │
                    SCons.Node.Alias.AliasNodeInfo
```

The generic base class for signature information for a Node.

Node subclasses should subclass NodeInfoBase to provide their own logic for dealing with their own Node-specific signature information.

14.3.1 Methods

str_to_node(*self*, *s*)

Inherited from SCons.Node.NodeInfoBase(Section 13.4)

__init__(), convert(), format(), merge(), update()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

14.3.2 Properties

Name	Description
Inherited from object	
__class__	

14.3.3 Class Variables

Name	Description
current_version_id	**Value:** `1`
field_list	**Value:** `['csig']`

14.4 Class AliasBuildInfo

object
SCons.Node.BuildInfoBase
SCons.Node.Alias.AliasBuildInfo

The generic base class for build information for a Node.

This is what gets stored in a .sconsign file for each target file. It contains a NodeInfo instance for this node (signature information that's specific to the type of Node) and direct attributes for the generic build stuff we have to track: sources, explicit dependencies, implicit dependencies, and action information.

14.4.1 Methods

Inherited from SCons.Node.BuildInfoBase(Section 13.5)

__init__(), merge()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

14.4.2 Properties

Name	Description
Inherited from object	
__class__	

14.4.3 Class Variables

Name	Description
current_version_id	**Value:** `1`

14.5 Class Alias

object
SCons.Node.Node
SCons.Node.Alias.Alias

14.5.1 Methods

__init__(*self*, *name*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

str_for_display(*self*)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

make_ready(*self*)

Get a Node ready for evaluation.

This is called before the Taskmaster decides if the Node is up-to-date or not. Overriding this method allows for a Node subclass to be disambiguated if necessary, or for an implicit source builder to be attached. Overrides: SCons.Node.Node.make_ready extit(inherited documentation)

really_build(*self*, ***kw*)

Actually build the node.

This is called by the Taskmaster after it's decided that the Node is out-of-date and must be rebuilt, and after the prepare() method has gotten everything, uh, prepared.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in built().

is_up_to_date(*self*)

Alternate check for whether the Node is current: If all of our children were up-to-date, then this Node was up-to-date, too.

The SCons.Node.Alias and SCons.Node.Python.Value subclasses rebind their current() method to this method. Overrides: SCons.Node.Node.is_up_to_date

is_under(*self*, *dir*)

get_contents(*self*)

The contents of an alias is the concatenation of the content signatures of all its sources.

sconsign(*self*)

An Alias is not recorded in .sconsign files

changed_since_last_build(*self*, *target*, *prev_ni*)

Must be overridden in a specific subclass to return True if this Node (a dependency) has changed since the last time it was used to build the specified target. prev_ni is this Node's state (for example, its file timestamp, length, maybe content signature) as of the last time the target was built.

Note that this method is called through the dependency, not the target, because a dependency Node must be able to use its own logic to decide if it changed. For example, File Nodes need to obey if we're configured to use timestamps, but Python Value Nodes never use timestamps and always use the content. If this method were called through the target, then each Node's implementation of this method would have to have more complicated logic to handle all the different Node types on which it might depend. Overrides: SCons.Node.Node.changed_since_last_build extit(inherited documentation)

build(*self*)

A "builder" for aliases. Overrides: SCons.Node.Node.build

convert(*self*)

get_csig(*self*)

Generate a node's content signature, the digested signature of its content.

node - the node cache - alternate node to use for the signature cache returns - the content signature Overrides: SCons.Node.Node.get_csig

Inherited from SCons.Node.Node(Section 13.6)

Decider(), add_dependency(), add_ignore(), add_prerequisite(), add_source(), add_to_implicit(), add_to_waiting_parents(), add_to_waiting_s_e(), add_wkid(), all_children(), alter_targets(), builder_set(), built(), changed(), children(), children_are_up_to_date(), clear(), clear_memoized_values(), del_binfo(), disambiguate(), do_not_store_info(), env_set(), executor_cleanup(), exists(), explain(), for_signature(), get_abspath(), get_binfo(), get_build_env(), get_build_scanner_path(), get_builder(), get_cachedir_csig(), get_env(), get_env_scanner(), get_executor(), get_found_includes(), get_implicit_deps(), get_ninfo(), get_source_scanner(), get_state(), get_stored_implicit(), get_stored_info(), get_string(), get_subst_proxy(), get_suffix(), get_target_scanner(), has_builder(), has_explicit_builder(), is_derived(), is_literal(), missing(), multiple_side_effect_has_builder(), new_binfo(), new_ninfo(), postprocess(), prepare(), push_to_cache(), remove(), render_include_tree(), reset_executor(), retrieve_from_cache(), rexists(), scan(), scanner_key(), select_scanner(), set_always_build(), set_executor(), set_explicit(), set_nocache(), set_noclean(), set_precious(), set_specific_source(), set_state(), state_has_changed(), store_info(), visited()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

14.5.2 Properties

Name	Description
Inherited from object	
__class__	

14.5.3 Class Variables

Name	Description
Inherited from SCons.Node.Node (Section 13.6)	
__metaclass__, memoizer_counters	

15 Module SCons.Node.FS

scons.Node.FS

File system nodes.

These Nodes represent the canonical external objects that people think of when they think of building software: files and directories.

This holds a "default_fs" variable that should be initialized with an FS that can be used by scripts or modules looking for the canonical default.

15.1 Functions

save_strings(*val*)

initialize_do_splitdrive()

initialize_normpath_check()

Initialize the normpath_check regular expression.

This function is used by the unit tests to re-initialize the pattern when testing for behavior with different values of os.sep.

set_duplicate(*duplicate*)

LinkFunc(*target*, *source*, *env*)

LocalString(*target*, *source*, *env*)

UnlinkFunc(*target*, *source*, *env*)

MkdirFunc(*target*, *source*, *env*)

get_MkdirBuilder()

get_DefaultSCCSBuilder()

get_DefaultRCSBuilder()

do_diskcheck_match(*node*, *predicate*, *errorfmt*)

ignore_diskcheck_match(*node*, *predicate*, *errorfmt*)

do_diskcheck_rcs(*node*, *name*)

ignore_diskcheck_rcs(*node*, *name*)

do_diskcheck_sccs(*node*, *name*)

ignore_diskcheck_sccs(*node*, *name*)

set_diskcheck(*list*)

diskcheck_types()

has_glob_magic(*s*)

get_default_fs()

find_file(*filename*, *paths*, *verbose*=None)

```
find_file(str, [Dir()]) -> [nodes]

filename - a filename to find
paths - a list of directory path *nodes* to search in.  Can be
        represented as a list, a tuple, or a callable that is
        called with no arguments and returns the list or tuple.

returns - the node created from the found file.

Find a node corresponding to either a derived file or a file
that exists already.

Only the first file found is returned, and none is returned
if no file is found.
```

invalidate_node_memos(*targets*)

Invalidate the memoized values of all Nodes (files or directories) that are associated with the given entries. Has been added to clear the cache of nodes affected by a direct execution of an action (e.g. Delete/Copy/Chmod). Existing Node caches become inconsistent if the action is run through Execute(). The argument `targets` can be a single Node object or filename, or a sequence of Nodes/filenames.

15.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Node/FS.py 5134 2010/08/16 23:02:40 bde...`
do_store_info	**Value:** `True`
default_max_drift	**Value:** `172800`
Save_Strings	**Value:** `None`
do_splitdrive	**Value:** `False`
needs_normpath_check	**Value:** `re.compile(r'.*/|\.$\|\.\.$')`
Valid_Duplicates	**Value:** `['hard-soft-copy', 'soft-hard-copy', 'hard-copy', 'soft-c...`
Link_Funcs	**Value:** `[]`
Link	**Value:** `SCons.Action.Action(LinkFunc, None)`
LocalCopy	**Value:** `SCons.Action.Action(LinkFunc, LocalString)`
Unlink	**Value:** `SCons.Action.Action(UnlinkFunc, None)`
Mkdir	**Value:** `SCons.Action.Action(MkdirFunc, None, presub= None)`
MkdirBuilder	**Value:** `None`
DefaultSCCSBuilder	**Value:** `None`
DefaultRCSBuilder	**Value:** `None`
diskcheck_match	**Value:** `DiskChecker('match', do_diskcheck_match, ignore_diskcheck...`
diskcheck_rcs	**Value:** `DiskChecker('rcs', do_diskcheck_rcs, ignore_diskcheck_rcs)`
diskcheck_sccs	**Value:** `DiskChecker('sccs', do_diskcheck_sccs, ignore_diskcheck_s...`

continued on next page

Name	Description
diskcheckers	**Value:** `[diskcheck_match, diskcheck_rcs, diskcheck_sccs,]`
glob_magic_check	**Value:** `re.compile(r'[\*\?\[]')`
default_fs	**Value:** `None`
__package__	**Value:** `'SCons.Node'`

15.3 Class EntryProxyAttributeError

object
exceptions.BaseException
exceptions.Exception
exceptions.StandardError
exceptions.AttributeError
SCons.Node.FS.EntryProxyAttributeError

An AttributeError subclass for recording and displaying the name of the underlying Entry involved in an AttributeError exception.

15.3.1 Methods

__init__(*self*, *entry_proxy*, *attribute*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.AttributeError

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

15.3.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

15.4 Class DiskChecker

object ─┐
SCons.Node.FS.DiskChecker

15.4.1 Methods

__init__(*self*, *type*, *do*, *ignore*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, **args*, ***kw*)

set(*self*, *list*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.4.2 Properties

Name	Description
Inherited from object	
__class__	

15.5 Class EntryProxy

object
SCons.Util.Proxy
SCons.Node.FS.EntryProxy

15.5.1 Methods

__str__(...)

A Python Descriptor class that delegates attribute fetches to an underlying wrapped subject of a Proxy. Typical use:

class Foo(Proxy): __str__ = Delegate('__str__')

Overrides: object.__str__

__getattr__(*self*, *name*)

Retrieve an attribute from the wrapped object. If the named attribute doesn't exist, AttributeError is raised Overrides: SCons.Util.Proxy.__getattr__ extit(inherited documentation)

Inherited from SCons.Util.Proxy(Section 36.5)

__cmp__(), __init__(), get()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

15.5.2 Properties

Name	Description
Inherited from object	
__class__	

15.5.3 Class Variables

Name	Description
dictSpecialAttrs	**Value:** `{"base": __get_base_path, "posix": __get_posix_path, "win...`

15.6 Class Base

```
object ─┐
        │
SCons.Node.Node ─┐
                 │
                 SCons.Node.FS.Base
```

Known Subclasses: SCons.Node.FS.Dir, SCons.Node.FS.Entry, SCons.Node.FS.File

A generic class for file system entries. This class is for when we don't know yet whether the entry being looked up is a file or a directory. Instances of this class can morph into either Dir or File objects by a later, more precise lookup.

Note: this class does not define __cmp__ and __hash__ for efficiency reasons. SCons does a lot of comparing of Node.FS.{Base,Entry,File,Dir} objects, so those operations must be as fast as possible, which means we want to use Python's built-in object identity comparisons.

15.6.1 Methods

__init__(*self*, *name*, *directory*, *fs*)

Initialize a generic Node.FS.Base object.

Call the superclass initialization, take care of setting up our relative and absolute paths, identify our parent directory, and indicate that this node should use signatures. Overrides: object.__init__

str_for_display(*self*)

must_be_same(*self*, *klass*)

This node, which already existed, is being looked up as the specified klass. Raise an exception if it isn't.

get_dir(*self*)

get_suffix(*self*)

Overrides: SCons.Node.Node.get_suffix

rfile(*self*)

__str__(*self*)

A Node.FS.Base object's string representation is its path name. Overrides: object.__str__

rstr(*self*)

A Node.FS.Base object's string representation is its path name.

stat(*self*)

exists(*self*)

Does this node exists? Overrides: SCons.Node.Node.exists extit(inherited documentation)

rexists(*self*)

Does this node exist locally or in a repositiory? Overrides: SCons.Node.Node.rexists extit(inherited documentation)

getmtime(*self*)

getsize(*self*)

isdir(*self*)

isfile(*self*)

islink(*self*)

is_under(*self*, *dir*)

set_local(*self*)

srcnode(*self*)

If this node is in a build path, return the node corresponding to its source file. Otherwise, return ourself.

get_path(*self*, *dir*=`None`)

Return path relative to the current working directory of the Node.FS.Base object that owns us.

set_src_builder(*self*, *builder*)

Set the source code builder for this node.

src_builder(*self*)

Fetch the source code builder for this node.

If there isn't one, we cache the source code builder specified for the directory (which in turn will cache the value from its parent directory, and so on up to the file system root).

get_abspath(*self*)

Get the absolute path of the file. Overrides: SCons.Node.Node.get_abspath

for_signature(*self*)

Return a string representation of the Node that will always be the same for this particular Node, no matter what. This is by contrast to the __str__() method, which might, for instance, return a relative path for a file Node. The purpose of this method is to generate a value to be used in signature calculation for the command line used to build a target, and we use this method instead of str() to avoid unnecessary rebuilds. This method does not need to return something that would actually work in a command line; it can return any kind of nonsense, so long as it does not change. Overrides: SCons.Node.Node.for_signature extit(inherited documentation)

get_subst_proxy(*self*)

This method is expected to return an object that will function exactly like this Node, except that it implements any additional special features that we would like to be in effect for Environment variable substitution. The principle use is that some Nodes would like to implement a __getattr__() method, but putting that in the Node type itself has a tendency to kill performance. We instead put it in a proxy and return it from this method. It is legal for this method to return self if no new functionality is needed for Environment substitution. Overrides: SCons.Node.Node.get_subst_proxy extit(inherited documentation)

target_from_source(*self*, *prefix*, *suffix*, *splitext*=`<function splitext at 0x13029b0>`)

Generates a target entry that corresponds to this entry (usually a source file) with the specified prefix and suffix.

Note that this method can be overridden dynamically for generated files that need different behavior. See Tool/swig.py for an example.

Rfindalldirs(*self*, *pathlist*)

Return all of the directories for a given path list, including corresponding “backing” directories in any repositories.

The Node lookups are relative to this Node (typically a directory), so memoizing result saves cycles from looking up the same path for each target in a given directory.

RDirs(*self, pathlist*)

Search for a list of directories in the Repository list.

rentry(*self*)

Inherited from SCons.Node.Node(Section 13.6)

Decider(), add_dependency(), add_ignore(), add_prerequisite(), add_source(), add_to_implicit(), add_to_waiting_parents(), add_to_waiting_s_e(), add_wkid(), all_children(), alter_targets(), build(), builder_set(), built(), changed(), changed_since_last_build(), children(), children_are_up_to_date(), clear(), clear_memoized_values(), del_binfo(), disambiguate(), do_not_store_info(), env_set(), executor_cleanup(), explain(), get_binfo(), get_build_env(), get_build_scanner_path(), get_builder(), get_cachedir_csig(), get_csig(), get_env(), get_env_scanner(), get_executor(), get_found_includes(), get_implicit_deps(), get_ninfo(), get_source_scanner(), get_state(), get_stored_implicit(), get_stored_info(), get_string(), get_target_scanner(), has_builder(), has_explicit_builder(), is_derived(), is_literal(), is_up_to_date(), make_ready(), missing(), multiple_side_effect_has_builder(), new_binfo(), new_ninfo(), postprocess(), prepare(), push_to_cache(), remove(), render_include_tree(), reset_executor(), retrieve_from_cache(), scan(), scanner_key(), select_scanner(), set_always_build(), set_executor(), set_explicit(), set_nocache(), set_noclean(), set_precious(), set_specific_source(), set_state(), state_has_changed(), store_info(), visited()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

15.6.2 Properties

Name	Description
Inherited from object	
__class__	

15.6.3 Class Variables

Name	Description
memoizer_counters	**Value:** `[]`
Inherited from SCons.Node.Node (Section 13.6)	
__metaclass__	

15.7 Class Entry

object ─┐
SCons.Node.Node ─┐
SCons.Node.FS.Base ─┐
SCons.Node.FS.Entry

This is the class for generic Node.FS entries--that is, things that could be a File or a Dir, but we're just not sure yet. Consequently, the methods in this class really exist just to transform their associated object into the right class when the time comes, and then call the same-named method in the transformed class.

15.7.1 Methods

diskcheck_match(*self*)

disambiguate(*self*, *must_exist*=`None`)

Overrides: SCons.Node.Node.disambiguate

rfile(*self*)

We're a generic Entry, but the caller is actually looking for a File at this point, so morph into one. Overrides: SCons.Node.FS.Base.rfile

scanner_key(*self*)

Overrides: SCons.Node.Node.scanner_key

get_contents(*self*)

Fetch the contents of the entry. Returns the exact binary contents of the file.

get_text_contents(*self*)

Fetch the decoded text contents of a Unicode encoded Entry.

Since this should return the text contents from the file system, we check to see into what sort of subclass we should morph this Entry.

must_be_same(*self*, *klass*)

Called to make sure a Node is a Dir. Since we're an Entry, we can morph into one. Overrides: SCons.Node.FS.Base.must_be_same

exists(*self*)

Return if the Entry exists. Check the file system to see what we should turn into first. Assume a file if there's no directory. Overrides: SCons.Node.Node.exists

rel_path(*self*, *other*)

new_ninfo(*self*)

Overrides: SCons.Node.Node.new_ninfo

changed_since_last_build(*self*, *target*, *prev_ni*)

Must be overridden in a specific subclass to return True if this Node (a dependency) has changed since the last time it was used to build the specified target. prev_ni is this Node's state (for example, its file timestamp, length, maybe content signature) as of the last time the target was built.

Note that this method is called through the dependency, not the target, because a dependency Node must be able to use its own logic to decide if it changed. For example, File Nodes need to obey if we're configured to use timestamps, but Python Value Nodes never use timestamps and always use the content. If this method were called through the target, then each Node's implementation of this method would have to have more complicated logic to handle all the different Node types on which it might depend. Overrides: SCons.Node.Node.changed_since_last_build extit(inherited documentation)

get_subst_proxy(*self*)

This method is expected to return an object that will function exactly like this Node, except that it implements any additional special features that we would like to be in effect for Environment variable substitution. The principle use is that some Nodes would like to implement a __getattr__() method, but putting that in the Node type itself has a tendency to kill performance. We instead put it in a proxy and return it from this method. It is legal for this method to return self if no new functionality is needed for Environment substitution.
Overrides: SCons.Node.Node.get_subst_proxy extit(inherited documentation)

Inherited from SCons.Node.FS.Base(Section 15.6)

RDirs(), Rfindalldirs(), __init__(), __str__(), for_signature(), get_abspath(), get_dir(), get_path(), get_suffix(), getmtime(), getsize(), is_under(), isdir(), isfile(), islink(), rentry(), rexists(), rstr(), set_local(), set_src_builder(), src_builder(), srcnode(), stat(), str_for_display(), target_from_source()

Inherited from SCons.Node.Node(Section 13.6)

Decider(), add_dependency(), add_ignore(), add_prerequisite(), add_source(), add_to_implicit(), add_to_waiting_parents(), add_to_waiting_s_e(), add_wkid(), all_children(), alter_targets(), build(), builder_set(), built(), changed(), children(), children_are_up_to_date(), clear(), clear_memoized_values(), del_binfo(), do_not_store_info(), env_set(), executor_cleanup(), explain(), get_binfo(), get_build_env(), get_build_scanner_path(), get_builder(), get_cachedir_csig(), get_csig(), get_env(), get_env_scanner(), get_executor(), get_found_includes(), get_implicit_deps(), get_ninfo(), get_source_scanner(), get_state(), get_stored_implicit(), get_stored_info(), get_string(), get_target_scanner(), has_builder(), has_explicit_builder(), is_derived(), is_literal(), is_up_to_date(), make_ready(), missing(), multiple_side_effect_has_builder(), new_binfo(), postprocess(), prepare(), push_to_cache(), remove(), render_include_tree(), reset_executor(), retrieve_from_cache(), scan(), select_scanner(), set_always_build(), set_executor(), set_explicit(), set_nocache(), set_noclean(), set_precious(), set_specific_source(), set_state(), state_has_changed(), store_info(), visited()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

15.7.2 Properties

Name	Description
Inherited from object	
__class__	

15.7.3 Class Variables

Name	Description
Inherited from SCons.Node.FS.Base (Section 15.6)	
memoizer_counters	
Inherited from SCons.Node.Node (Section 13.6)	
__metaclass__	

15.8 Class LocalFS

object ─┐
SCons.Node.FS.LocalFS

Known Subclasses: SCons.Node.FS.FS

15.8.1 Methods

chmod(*self*, *path*, *mode*)

copy(*self*, *src*, *dst*)

copy2(*self*, *src*, *dst*)

exists(*self*, *path*)

getmtime(*self*, *path*)

getsize(*self*, *path*)

isdir(*self*, *path*)

isfile(*self*, *path*)

link(*self*, *src*, *dst*)

lstat(*self*, *path*)

listdir(*self*, *path*)

makedirs(*self*, *path*)

mkdir(*self*, *path*)

rename(*self*, *old*, *new*)

stat(*self*, *path*)

symlink(*self*, *src*, *dst*)

open(*self*, *path*)

unlink(*self*, *path*)

islink(*self*, *path*)

readlink(*self*, *file*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.8.2 Properties

Name	Description
Inherited from object	
__class__	

15.8.3 Class Variables

Name	Description
__metaclass__	**Value:** `SCons.Memoize.Memoized_Metaclass`

15.9 Class FS

object
SCons.Node.FS.LocalFS
SCons.Node.FS.FS

15.9.1 Methods

__init__(*self*, *path*=`None`)

Initialize the Node.FS subsystem.

The supplied path is the top of the source tree, where we expect to find the top-level build file. If no path is supplied, the current directory is the default.

The path argument must be a valid absolute path. Overrides: object.__init__

set_SConstruct_dir(*self*, *dir*)

get_max_drift(*self*)

set_max_drift(*self*, *max_drift*)

getcwd(*self*)

chdir(*self*, *dir*, *change_os_dir*=`0`)

Change the current working directory for lookups. If change_os_dir is true, we will also change the "real" cwd to match.

get_root(*self*, *drive*)

Returns the root directory for the specified drive, creating it if necessary.

Entry(*self*, *name*, *directory*=`None`, *create*=`1`)

Look up or create a generic Entry node with the specified name. If the name is a relative path (begins with ./, ../, or a file name), then it is looked up relative to the supplied directory node, or to the top level directory of the FS (supplied at construction time) if no directory is supplied.

File(*self*, *name*, *directory*=`None`, *create*=`1`)

Look up or create a File node with the specified name. If the name is a relative path (begins with ./, ../, or a file name), then it is looked up relative to the supplied directory node, or to the top level directory of the FS (supplied at construction time) if no directory is supplied.

This method will raise TypeError if a directory is found at the specified path.

Dir(*self*, *name*, *directory*=`None`, *create*=`True`)

Look up or create a Dir node with the specified name. If the name is a relative path (begins with ./, ../, or a file name), then it is looked up relative to the supplied directory node, or to the top level directory of the FS (supplied at construction time) if no directory is supplied.

This method will raise TypeError if a normal file is found at the specified path.

VariantDir(*self*, *variant_dir*, *src_dir*, *duplicate*=`1`)

Link the supplied variant directory to the source directory for purposes of building files.

Repository(*self*, **dirs*)

Specify Repository directories to search.

variant_dir_target_climb(*self*, *orig*, *dir*, *tail*)

Create targets in corresponding variant directories

Climb the directory tree, and look up path names relative to any linked variant directories we find.

Even though this loops and walks up the tree, we don't memoize the return value because this is really only used to process the command-line targets.

Glob(*self*, *pathname*, *ondisk*=`True`, *source*=`True`, *strings*=`False`, *cwd*=`None`)

Globs

This is mainly a shim layer

Inherited from SCons.Node.FS.LocalFS(Section 15.8)

chmod(), copy(), copy2(), exists(), getmtime(), getsize(), isdir(), isfile(), islink(), link(), listdir(), lstat(), makedirs(), mkdir(), open(), readlink(), rename(), stat(), symlink(), unlink()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.9.2 Properties

Name	Description
Inherited from object	
__class__	

15.9.3 Class Variables

Name	Description
memoizer_counters	**Value:** `[]`
Inherited from SCons.Node.FS.LocalFS (Section 15.8)	
__metaclass__	

15.10 Class DirNodeInfo

object
SCons.Node.NodeInfoBase
SCons.Node.FS.DirNodeInfo

The generic base class for signature information for a Node.

Node subclasses should subclass NodeInfoBase to provide their own logic for dealing with their own Node-specific signature information.

15.10.1 Methods

str_to_node(*self*, *s*)

Inherited from SCons.Node.NodeInfoBase(Section 13.4)

__init__(), convert(), format(), merge(), update()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.10.2 Properties

Name	Description
Inherited from object	
__class__	

15.10.3 Class Variables

Name	Description
current_version_id	**Value:** `1`
fs	**Value:** `None`

15.11 Class DirBuildInfo

object
SCons.Node.BuildInfoBase
SCons.Node.FS.DirBuildInfo

The generic base class for build information for a Node.

This is what gets stored in a .sconsign file for each target file. It contains a NodeInfo instance for this node (signature information that's specific to the type of Node) and direct attributes for the generic build stuff we have to track: sources, explicit dependencies, implicit dependencies, and action information.

15.11.1 Methods

Inherited from SCons.Node.BuildInfoBase(Section 13.5)

__init__(), merge()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.11.2 Properties

Name	Description
Inherited from object	
__class__	

15.11.3 Class Variables

Name	Description
current_version_id	**Value:** `1`

15.12 Class Dir

object
SCons.Node.Node
SCons.Node.FS.Base
SCons.Node.FS.Dir

Known Subclasses: SCons.Node.FS.RootDir

A class for directories in a file system.

15.12.1 Methods

__init__(*self*, *name*, *directory*, *fs*)

Initialize a generic Node.FS.Base object.

Call the superclass initialization, take care of setting up our relative and absolute paths, identify our parent directory, and indicate that this node should use signatures. Overrides: object.__init__ extit(inherited documentation)

diskcheck_match(*self*)

Entry(*self*, *name*)

Looks up or creates an entry node named 'name' relative to this directory.

Dir(*self*, *name*, *create*=`True`)

Looks up or creates a directory node named 'name' relative to this directory.

File(*self*, *name*)

Looks up or creates a file node named 'name' relative to this directory.

link(*self*, *srcdir*, *duplicate*)

Set this directory as the variant directory for the supplied source directory.

getRepositories(*self*)

Returns a list of repositories for this directory.

get_all_rdirs(*self*)

addRepository(*self*, *dir*)

up(*self*)

rel_path(*self*, *other*)

Return a path to "other" relative to this directory.

get_env_scanner(*self*, *env*, *kw*={})

Overrides: SCons.Node.Node.get_env_scanner

get_target_scanner(*self*)

Overrides: SCons.Node.Node.get_target_scanner

get_found_includes(*self*, *env*, *scanner*, *path*)

Return this directory's implicit dependencies.

We don't bother caching the results because the scan typically shouldn't be requested more than once (as opposed to scanning .h file contents, which can be requested as many times as the files is #included by other files).
Overrides: SCons.Node.Node.get_found_includes

prepare(*self*)

Prepare for this Node to be built.

This is called after the Taskmaster has decided that the Node is out-of-date and must be rebuilt, but before actually calling the method to build the Node.

This default implementation checks that explicit or implicit dependencies either exist or are derived, and initializes the BuildInfo structure that will hold the information about how this node is, uh, built.

(The existence of source files is checked separately by the Executor, which aggregates checks for all of the targets built by a specific action.)

Overriding this method allows for for a Node subclass to remove the underlying file from the file system. Note that subclass methods should call this base class method to get the child check and the BuildInfo structure. Overrides: SCons.Node.Node.prepare extit(inherited documentation)

build(*self*, ***kw*)

A null "builder" for directories. Overrides: SCons.Node.Node.build

multiple_side_effect_has_builder(*self*)

Return whether this Node has a builder or not.

In Boolean tests, this turns out to be a *lot* more efficient than simply examining the builder attribute directly ("if node.builder: ..."). When the builder attribute is examined directly, it ends up calling __getattr__ for both the __len__ and __nonzero__ attributes on instances of our Builder Proxy class(es), generating a bazillion extra calls and slowing things down immensely. Overrides: SCons.Node.Node.multiple_side_effect_has_builder extit(inherited documentation)

alter_targets(*self*)

Return any corresponding targets in a variant directory. Overrides: SCons.Node.Node.alter_targets

scanner_key(*self*)

A directory does not get scanned. Overrides: SCons.Node.Node.scanner_key

get_text_contents(*self*)

We already emit things in text, so just return the binary version.

get_contents(*self*)

Return content signatures and names of all our children separated by new-lines. Ensure that the nodes are sorted.

get_csig(*self*)

Compute the content signature for Directory nodes. In general, this is not needed and the content signature is not stored in the DirNodeInfo. However, if get_contents on a Dir node is called which has a child directory, the child directory should return the hash of its contents. Overrides: SCons.Node.Node.get_csig

do_duplicate(*self*, *src*)

changed_since_last_build(*self*, *target*, *prev_ni*)

Must be overridden in a specific subclass to return True if this Node (a dependency) has changed since the last time it was used to build the specified target. prev_ni is this Node's state (for example, its file timestamp, length, maybe content signature) as of the last time the target was built.

Note that this method is called through the dependency, not the target, because a dependency Node must be able to use its own logic to decide if it changed. For example, File Nodes need to obey if we're configured to use timestamps, but Python Value Nodes never use timestamps and always use the content. If this method were called through the target, then each Node's implementation of this method would have to have more complicated logic to handle all the different Node types on which it might depend. Overrides: SCons.Node.Node.changed_since_last_build extit(inherited documentation)

is_up_to_date(*self*)

If any child is not up-to-date, then this directory isn't, either. Overrides: SCons.Node.Node.is_up_to_date

rdir(*self*)

sconsign(*self*)

Return the .sconsign file info for this directory, creating it first if necessary.

srcnode(*self*)

Dir has a special need for srcnode()...if we have a srcdir attribute set, then that *is* our srcnode. Overrides: SCons.Node.FS.Base.srcnode

get_timestamp(*self*)

Return the latest timestamp from among our children

entry_abspath(*self*, *name*)

entry_labspath(*self*, *name*)

entry_path(*self*, *name*)

entry_tpath(*self*, *name*)

entry_exists_on_disk(*self*, *name*)

srcdir_list(*self*)

srcdir_duplicate(*self*, *name*)

srcdir_find_file(*self*, *filename*)

dir_on_disk(*self*, *name*)

file_on_disk(*self*, *name*)

walk(*self*, *func*, *arg*)

Walk this directory tree by calling the specified function for each directory in the tree.

This behaves like the os.path.walk() function, but for in-memory Node.FS.Dir objects. The function takes the same arguments as the functions passed to os.path.walk():

> func(arg, dirname, fnames)

Except that "dirname" will actually be the directory *Node*, not the string. The '.' and '..' entries are excluded from fnames. The fnames list may be modified in-place to filter the subdirectories visited or otherwise impose a specific order. The "arg" argument is always passed to func() and may be used in any way (or ignored, passing None is common).

glob(*self*, *pathname*, *ondisk*=`True`, *source*=`False`, *strings*=`False`)

Returns a list of Nodes (or strings) matching a specified pathname pattern.

Pathname patterns follow UNIX shell semantics: * matches any-length strings of any characters, ? matches any character, and [] can enclose lists or ranges of characters. Matches do not span directory separators.

The matches take into account Repositories, returning local Nodes if a corresponding entry exists in a Repository (either an in-memory Node or something on disk).

By defafult, the glob() function matches entries that exist on-disk, in addition to in-memory Nodes. Setting the "ondisk" argument to False (or some other non-true value) causes the glob() function to only match in-memory Nodes. The default behavior is to return both the on-disk and in-memory Nodes.

The "source" argument, when true, specifies that corresponding source Nodes must be returned if you're globbing in a build directory (initialized with VariantDir()). The default behavior is to return Nodes local to the VariantDir().

The "strings" argument, when true, returns the matches as strings, not Nodes. The strings are path names relative to this directory.

The underlying algorithm is adapted from the glob.glob() function in the Python library (but heavily modified), and uses fnmatch() under the covers.

Inherited from SCons.Node.FS.Base(Section 15.6)

RDirs(), Rfindalldirs(), __str__(), exists(), for_signature(), get_abspath(), get_dir(), get_path(), get_subst_proxy(), get_suffix(), getmtime(), getsize(), is_under(), isdir(), isfile(), islink(), must_be_same(), rentry(), rexists(), rfile(), rstr(), set_local(), set_src_builder(), src_builder(), stat(), str_for_display(), target_from_source()

Inherited from SCons.Node.Node(Section 13.6)

Decider(), add_dependency(), add_ignore(), add_prerequisite(), add_source(), add_to_implicit(), add_to_waiting_parents(), add_to_waiting_s_e(), add_wkid(), all_children(), builder_set(), built(), changed(), children(), children_are_up_to_date(), clear(), clear_memoized_values(), del_binfo(), disambiguate(), do_not_store_info(), env_set(), executor_cleanup(), explain(), get_binfo(), get_build_env(), get_build_scanner_path(), get_builder(), get_cachedir_csig(), get_env(), get_executor(), get_implicit_deps(), get_ninfo(), get_source_scanner(), get_state(), get_stored_implicit(), get_stored_info(), get_string(), has_builder(), has_explicit_builder(), is_derived(), is_literal(), make_ready(), missing(), new_binfo(), new_ninfo(), postprocess(), push_to_cache(), remove(), render_include_tree(), reset_executor(), retrieve_from_cache(), scan(), select_scanner(), set_always_build(), set_executor(), set_explicit(), set_nocache(), set_noclean(), set_precious(), set_specific_source(), set_state(), state_has_changed(), store_info(), visited()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

15.12.2 Properties

Name	Description
Inherited from object	
__class__	

15.12.3 Class Variables

Name	Description
memoizer_counters	**Value:** `[]`
Inherited from SCons.Node.Node (Section 13.6)	
__metaclass__	

15.13 Class RootDir

object
SCons.Node.Node
SCons.Node.FS.Base
SCons.Node.FS.Dir
SCons.Node.FS.RootDir

A class for the root directory of a file system.

This is the same as a Dir class, except that the path separator ('/' or '') is actually part of the name, so we don't need to add a separator when creating the path names of entries within this directory.

15.13.1 Methods

__init__(*self*, *name*, *fs*)

Initialize a generic Node.FS.Base object.

Call the superclass initialization, take care of setting up our relative and absolute paths, identify our parent directory, and indicate that this node should use signatures. Overrides: object.__init__ extit(inherited documentation)

must_be_same(*self*, *klass*)

This node, which already existed, is being looked up as the specified klass. Raise an exception if it isn't. Overrides: SCons.Node.FS.Base.must_be_same extit(inherited documentation)

__str__(*self*)

A Node.FS.Base object's string representation is its path name. Overrides: object.__str__ extit(inherited documentation)

entry_abspath(*self*, *name*)

Overrides: SCons.Node.FS.Dir.entry_abspath

entry_labspath(*self*, *name*)

Overrides: SCons.Node.FS.Dir.entry_labspath

entry_path(*self*, *name*)

Overrides: SCons.Node.FS.Dir.entry_path

entry_tpath(*self*, *name*)

Overrides: SCons.Node.FS.Dir.entry_tpath

is_under(*self*, *dir*)

Overrides: SCons.Node.FS.Base.is_under

up(*self*)

Overrides: SCons.Node.FS.Dir.up

get_dir(*self*)

Overrides: SCons.Node.FS.Base.get_dir

src_builder(*self*)

Fetch the source code builder for this node.

If there isn't one, we cache the source code builder specified for the directory (which in turn will cache the value from its parent directory, and so on up to the file system root). Overrides: SCons.Node.FS.Base.src_builder extit(inherited documentation)

Inherited from SCons.Node.FS.Dir(Section 15.12)

Dir(), Entry(), File(), addRepository(), alter_targets(), build(), changed_since_last_build(), dir_on_disk(), diskcheck_match(), do_duplicate(), entry_exists_on_disk(), file_on_disk(), getRepositories(), get_all_rdirs(), get_contents(), get_csig(), get_env_scanner(), get_found_includes(), get_target_scanner(), get_text_contents(), get_timestamp(), glob(), is_up_to_date(), link(), multiple_side_effect_has_builder(), prepare(), rdir(), rel_path(), scanner_key(), sconsign(), srcdir_duplicate(), srcdir_find_file(), srcdir_list(), srcnode(), walk()

Inherited from SCons.Node.FS.Base(Section 15.6)

RDirs(), Rfindalldirs(), exists(), for_signature(), get_abspath(), get_path(), get_subst_proxy(), get_suffix(), getmtime(), getsize(), isdir(), isfile(), islink(), rentry(), rexists(), rfile(), rstr(), set_local(), set_src_builder(), stat(), str_for_display(), target_from_source()

Inherited from SCons.Node.Node(Section 13.6)

Decider(), add_dependency(), add_ignore(), add_prerequisite(), add_source(), add_to_implicit(), add_to_waiting_parents(), add_to_waiting_s_e(), add_wkid(), all_children(), builder_set(), built(), changed(), children(), children_are_up_to_date(), clear(), clear_memoized_values(), del_binfo(), disambiguate(), do_not_store_info(), env_set(), executor_cleanup(), explain(), get_binfo(), get_build_env(), get_build_scanner_path(), get_builder(), get_cachedir_csig(), get_env(), get_executor(), get_implicit_deps(), get_ninfo(), get_source_scanner(), get_state(), get_stored_implicit(), get_stored_info(), get_string(), has_builder(), has_explicit_builder(), is_derived(), is_literal(), make_ready(), missing(), new_binfo(), new_ninfo(), postprocess(), push_to_cache(), remove(), render_include_tree(), reset_executor(), retrieve_from_cache(), scan(), select_scanner(), set_always_build(), set_executor(), set_explicit(), set_nocache(), set_noclean(), set_precious(), set_specific_source(), set_state(), state_has_changed(), store_info(), visited()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

15.13.2 Properties

Name	Description
Inherited from object	
__class__	

15.13.3 Class Variables

Name	Description
Inherited from SCons.Node.FS.Dir (Section 15.12)	
memoizer_counters	
Inherited from SCons.Node.Node (Section 13.6)	
__metaclass__	

15.14 Class FileNodeInfo

object

SCons.Node.NodeInfoBase

SCons.Node.FS.FileNodeInfo

The generic base class for signature information for a Node.

Node subclasses should subclass NodeInfoBase to provide their own logic for dealing with their own Node-specific signature information.

15.14.1 Methods

str_to_node(*self*, *s*)

Inherited from SCons.Node.NodeInfoBase(Section 13.4)

__init__(), convert(), format(), merge(), update()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.14.2 Properties

Name	Description
Inherited from object	
__class__	

15.14.3 Class Variables

Name	Description
current_version_id	**Value:** `1`
field_list	**Value:** `['csig', 'timestamp', 'size']`
fs	**Value:** `None`

15.15 Class FileBuildInfo

```
object ─┐
        │
SCons.Node.BuildInfoBase ─┐
                          │
                SCons.Node.FS.FileBuildInfo
```

Known Subclasses: SCons.SConf.SConfBuildInfo

The generic base class for build information for a Node.

This is what gets stored in a .sconsign file for each target file. It contains a NodeInfo instance for this node (signature information that's specific to the type of Node) and direct attributes for the generic build stuff we have to track: sources, explicit dependencies, implicit dependencies, and action information.

15.15.1 Methods

convert_from_sconsign(*self*, *dir*, *name*)

Converts a newly-read FileBuildInfo object for in-SCons use

For normal up-to-date checking, we don't have any conversion to perform--but we're leaving this method here to make that clear.

convert_to_sconsign(*self*)

Converts this FileBuildInfo object for writing to a .sconsign file

This replaces each Node in our various dependency lists with its usual string representation: relative to the top-level SConstruct directory, or an absolute path if it's outside.

format(*self*, *names*=`0`)

prepare_dependencies(*self*)

Prepares a FileBuildInfo object for explaining what changed

The bsources, bdepends and bimplicit lists have all been stored on disk as paths relative to the top-level SConstruct directory. Convert the strings to actual Nodes (for use by the --debug=explain code and --implicit-cache).

Inherited from SCons.Node.BuildInfoBase(Section 13.5)

__init__(), merge()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.15.2 Properties

Name	Description
Inherited from object	
__class__	

15.15.3 Class Variables

Name	Description
current_version_id	**Value:** `1`

15.16 Class File

```
object ─┐
        │
SCons.Node.Node ─┐
                 │
  SCons.Node.FS.Base ─┐
                      │
                   SCons.Node.FS.File
```

A class for files in a file system.

15.16.1 Methods

diskcheck_match(*self*)

__init__(*self*, *name*, *directory*, *fs*)

Initialize a generic Node.FS.Base object.

Call the superclass initialization, take care of setting up our relative and absolute paths, identify our parent directory, and indicate that this node should use signatures. Overrides: object.__init__ extit(inherited documentation)

Entry(*self*, *name*)

Create an entry node named 'name' relative to the directory of this file.

Dir(*self*, *name*, *create*=`True`)

Create a directory node named 'name' relative to the directory of this file.

Dirs(*self*, *pathlist*)

Create a list of directories relative to the SConscript directory of this file.

File(*self*, *name*)

Create a file node named 'name' relative to the directory of this file.

scanner_key(*self*)

Overrides: SCons.Node.Node.scanner_key

get_contents(*self*)

get_text_contents(*self*)

get_content_hash(*self*)

Compute and return the MD5 hash for this file.

get_size(*self*)

get_timestamp(*self*)

store_info(*self*)

Make the build signature permanent (that is, store it in the .sconsign file or equivalent). Overrides: SCons.Node.Node.store_info extit(inherited documentation)

convert_old_entry(*self*, *old_entry*)

get_stored_info(*self*)

Overrides: SCons.Node.Node.get_stored_info

get_stored_implicit(*self*)

Fetch the stored implicit dependencies Overrides: SCons.Node.Node.get_stored_implicit extit(inherited documentation)

rel_path(*self*, *other*)

get_found_includes(*self*, *env*, *scanner*, *path*)

Return the included implicit dependencies in this file. Cache results so we only scan the file once per path regardless of how many times this information is requested. Overrides: SCons.Node.Node.get_found_includes

push_to_cache(*self*)

Try to push the node into a cache Overrides: SCons.Node.Node.push_to_cache

retrieve_from_cache(*self*)

Try to retrieve the node's content from a cache

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in built().

Returns true iff the node was successfully retrieved. Overrides: SCons.Node.Node.retrieve_from_cache

visited(*self*)

Called just after this node has been visited (with or without a build). Overrides: SCons.Node.Node.visited extit(inherited documentation)

find_src_builder(*self*)

has_src_builder(*self*)

Return whether this Node has a source builder or not.

If this Node doesn't have an explicit source code builder, this is where we figure out, on the fly, if there's a transparent source code builder for it.

Note that if we found a source builder, we also set the self.builder attribute, so that all of the methods that actually *build* this file don't have to do anything different.

alter_targets(*self*)

Return any corresponding targets in a variant directory. Overrides: SCons.Node.Node.alter_targets

make_ready(*self*)

Get a Node ready for evaluation.

This is called before the Taskmaster decides if the Node is up-to-date or not. Overriding this method allows for a Node subclass to be disambiguated if necessary, or for an implicit source builder to be attached. Overrides: SCons.Node.Node.make_ready extit(inherited documentation)

prepare(*self*)

Prepare for this file to be created. Overrides: SCons.Node.Node.prepare

remove(*self*)

Remove this file. Overrides: SCons.Node.Node.remove

do_duplicate(*self*, *src*)

exists(*self*)

Does this node exists? Overrides: SCons.Node.Node.exists extit(inherited documentation)

get_max_drift_csig(*self*)

Returns the content signature currently stored for this node if it's been unmodified longer than the max_drift value, or the max_drift value is 0. Returns None otherwise.

get_csig(*self*)

Generate a node's content signature, the digested signature of its content.

node - the node cache - alternate node to use for the signature cache returns - the content signature Overrides: SCons.Node.Node.get_csig

builder_set(*self*, *builder*)

Overrides: SCons.Node.Node.builder_set

changed_content(*self*, *target*, *prev_ni*)

changed_state(*self*, *target*, *prev_ni*)

changed_timestamp_then_content(*self*, *target*, *prev_ni*)

changed_timestamp_newer(*self*, *target*, *prev_ni*)

changed_timestamp_match(*self*, *target*, *prev_ni*)

decide_source(*self*, *target*, *prev_ni*)

Must be overridden in a specific subclass to return True if this Node (a dependency) has changed since the last time it was used to build the specified target. prev_ni is this Node's state (for example, its file timestamp, length, maybe content signature) as of the last time the target was built.

Note that this method is called through the dependency, not the target, because a dependency Node must be able to use its own logic to decide if it changed. For example, File Nodes need to obey if we're configured to use timestamps, but Python Value Nodes never use timestamps and always use the content. If this method were called through the target, then each Node's implementation of this method would have to have more complicated logic to handle all the different Node types on which it might depend.

decide_target(*self*, *target*, *prev_ni*)

changed_since_last_build(*self*, *target*, *prev_ni*)

Must be overridden in a specific subclass to return True if this Node (a dependency) has changed since the last time it was used to build the specified target. prev_ni is this Node's state (for example, its file timestamp, length, maybe content signature) as of the last time the target was built.

Note that this method is called through the dependency, not the target, because a dependency Node must be able to use its own logic to decide if it changed. For example, File Nodes need to obey if we're configured to use timestamps, but Python Value Nodes never use timestamps and always use the content. If this method were called through the target, then each Node's implementation of this method would have to have more complicated logic to handle all the different Node types on which it might depend. Overrides: SCons.Node.Node.changed_since_last_build extit(inherited documentation)

is_up_to_date(*self*)

Default check for whether the Node is current: unknown Node subtypes are always out of date, so they will always get built. Overrides: SCons.Node.Node.is_up_to_date extit(inherited documentation)

rfile(*self*)

Overrides: SCons.Node.FS.Base.rfile

rstr(*self*)

A Node.FS.Base object's string representation is its path name. Overrides: SCons.Node.FS.Base.rstr extit(inherited documentation)

get_cachedir_csig(*self*)

Fetch a Node's content signature for purposes of computing another Node's cachesig.

This is a wrapper around the normal get_csig() method that handles the somewhat obscure case of using CacheDir with the -n option. Any files that don't exist would normally be "built" by fetching them from the cache, but the normal get_csig() method will try to open up the local file, which doesn't exist because the -n option meant we didn't actually pull the file from cachedir. But since the file *does* actually exist in the cachedir, we can use its contents for the csig. Overrides: SCons.Node.Node.get_cachedir_csig

get_cachedir_bsig(*self*)

Inherited from SCons.Node.FS.Base(Section 15.6)

RDirs(), Rfindalldirs(), __str__(), for_signature(), get_abspath(), get_dir(), get_path(), get_subst_proxy(), get_suffix(), getmtime(), getsize(), is_under(), isdir(), isfile(), islink(), must_be_same(), rentry(), rexists(), set_local(), set_src_builder(), src_builder(), srcnode(), stat(), str_for_display(), target_from_source()

Inherited from SCons.Node.Node(Section 13.6)

Decider(), add_dependency(), add_ignore(), add_prerequisite(), add_source(), add_to_implicit(), add_to_waiting_parents(), add_to_waiting_s_e(), add_wkid(), all_children(), build(), built(), changed(), children(), children_are_up_to_date(), clear(), clear_memoized_values(), del_binfo(), disambiguate(), do_not_store_info(), env_set(), executor_cleanup(), explain(), get_binfo(), get_build_env(), get_build_scanner_path(), get_builder(), get_env(), get_env_scanner(), get_executor(), get_implicit_deps(), get_ninfo(), get_source_scanner(), get_state(), get_string(), get_target_scanner(), has_builder(), has_explicit_builder(), is_derived(), is_literal(), missing(), multiple_side_effect_has_builder(), new_binfo(), new_ninfo(), postprocess(), render_include_tree(), reset_executor(), scan(), select_scanner(), set_always_build(), set_executor(), set_explicit(), set_nocache(), set_noclean(), set_precious(), set_specific_source(), set_state(), state_has_changed()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

15.16.2 Properties

Name	Description
Inherited from object	
__class__	

15.16.3 Class Variables

Name	Description
memoizer_counters	**Value:** `[]`
md5_chunksize	**Value:** `64`
convert_copy_attrs	**Value:** `['bsources', 'bimplicit', 'bdepends', 'bact', 'bactsig', ...`
convert_sig_attrs	**Value:** `['bsourcesigs', 'bimplicitsigs', 'bdependsigs']`

continued on next page

Name	Description
Inherited from SCons.Node.Node (Section 13.6)	
__metaclass__	

15.17 Class FileFinder

object ┐
 SCons.Node.FS.FileFinder

15.17.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

filedir_lookup(*self*, *p*, *fd*=`None`)

A helper method for find_file() that looks up a directory for a file we're trying to find. This only creates the Dir Node if it exists on-disk, since if the directory doesn't exist we know we won't find any files in it... :-)

It would be more compact to just use this as a nested function with a default keyword argument (see the commented-out version below), but that doesn't work unless you have nested scopes, so we define it here just so this work under Python 1.5.2.

find_file(*self*, *filename*, *paths*, *verbose*=None)

```
find_file(str, [Dir()]) -> [nodes]

filename - a filename to find
paths - a list of directory path *nodes* to search in.  Can be
        represented as a list, a tuple, or a callable that is
        called with no arguments and returns the list or tuple.

returns - the node created from the found file.

Find a node corresponding to either a derived file or a file
that exists already.

Only the first file found is returned, and none is returned
if no file is found.
```

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.17.2 Properties

Name	Description
Inherited from object	
__class__	

15.17.3 Class Variables

Name	Description
__metaclass__	**Value:** `SCons.Memoize.Memoized_Metaclass`
memoizer_counters	**Value:** `[]`

16 Module SCons.Node.Python

scons.Node.Python

Python nodes.

16.1 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Node/Python.py 5134 2010/08/16 23:02:40...`
__package__	**Value:** `'SCons.Node'`

16.2 Class ValueNodeInfo

object ─┐
SCons.Node.NodeInfoBase ─┐
SCons.Node.Python.ValueNodeInfo

The generic base class for signature information for a Node.

Node subclasses should subclass NodeInfoBase to provide their own logic for dealing with their own Node-specific signature information.

16.2.1 Methods

str_to_node(*self*, *s*)

Inherited from SCons.Node.NodeInfoBase(Section 13.4)

__init__(), convert(), format(), merge(), update()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.2.2 Properties

Name	Description
Inherited from object	
__class__	

16.2.3 Class Variables

Name	Description
current_version_id	**Value:** `1`
field_list	**Value:** `['csig']`

16.3 Class ValueBuildInfo

object
SCons.Node.BuildInfoBase
SCons.Node.Python.ValueBuildInfo

The generic base class for build information for a Node.

This is what gets stored in a .sconsign file for each target file. It contains a NodeInfo instance for this node (signature information that's specific to the type of Node) and direct attributes for the generic build stuff we have to track: sources, explicit dependencies, implicit dependencies, and action information.

16.3.1 Methods

Inherited from SCons.Node.BuildInfoBase(Section 13.5)

__init__(), merge()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

16.3.2 Properties

Name	Description
Inherited from object	
__class__	

16.3.3 Class Variables

Name	Description
current_version_id	**Value:** `1`

16.4 Class Value

object
SCons.Node.Node
SCons.Node.Python.Value

A class for Python variables, typically passed on the command line or generated by a script, but not from a file or some other source.

16.4.1 Methods

__init__(*self*, *value*, *built_value*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

str_for_display(*self*)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

make_ready(*self*)

Get a Node ready for evaluation.

This is called before the Taskmaster decides if the Node is up-to-date or not. Overriding this method allows for a Node subclass to be disambiguated if necessary, or for an implicit source builder to be attached. Overrides: SCons.Node.Node.make_ready extit(inherited documentation)

build(*self*, **kw*)

Actually build the node.

This is called by the Taskmaster after it's decided that the Node is out-of-date and must be rebuilt, and after the prepare() method has gotten everything, uh, prepared.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in built(). Overrides: SCons.Node.Node.build extit(inherited documentation)

is_up_to_date(*self*)

Alternate check for whether the Node is current: If all of our children were up-to-date, then this Node was up-to-date, too.

The SCons.Node.Alias and SCons.Node.Python.Value subclasses rebind their current() method to this method. Overrides: SCons.Node.Node.is_up_to_date

is_under(*self*, *dir*)

write(*self*, *built_value*)

Set the value of the node.

read(*self*)

Return the value. If necessary, the value is built.

get_text_contents(*self*)

By the assumption that the node.built_value is a deterministic product of the sources, the contents of a Value are the concatenation of all the contents of its sources. As the value need not be built when get_contents() is called, we cannot use the actual node.built_value.

get_contents(*self*)

By the assumption that the node.built_value is a deterministic product of the sources, the contents of a Value are the concatenation of all the contents of its sources. As the value need not be built when get_contents() is called, we cannot use the actual node.built_value.

changed_since_last_build(*self*, *target*, *prev_ni*)

Must be overridden in a specific subclass to return True if this Node (a dependency) has changed since the last time it was used to build the specified target. prev_ni is this Node's state (for example, its file timestamp, length, maybe content signature) as of the last time the target was built.

Note that this method is called through the dependency, not the target, because a dependency Node must be able to use its own logic to decide if it changed. For example, File Nodes need to obey if we're configured to use timestamps, but Python Value Nodes never use timestamps and always use the content. If this method were called through the target, then each Node's implementation of this method would have to have more complicated logic to handle all the different Node types on which it might depend. Overrides: SCons.Node.Node.changed_since_last_build extit(inherited documentation)

get_csig(*self*, *calc*=`None`)

Because we're a Python value node and don't have a real timestamp, we get to ignore the calculator and just use the value contents. Overrides: SCons.Node.Node.get_csig

Inherited from SCons.Node.Node(Section 13.6)

Decider(), add_dependency(), add_ignore(), add_prerequisite(), add_source(), add_to_implicit(), add_to_waiting_parents(), add_to_waiting_s_e(), add_wkid(), all_children(), alter_targets(), builder_set(), built(), changed(), children(), children_are_up_to_date(), clear(), clear_memoized_values(), del_binfo(), disambiguate(), do_not_store_info(), env_set(), executor_cleanup(), exists(), explain(), for_signature(), get_abspath(), get_binfo(), get_build_env(), get_build_scanner_path(), get_builder(), get_cachedir_csig(), get_env(), get_env_scanner(), get_executor(), get_found_includes(), get_implicit_deps(), get_ninfo(), get_source_scanner(), get_state(), get_stored_implicit(), get_stored_info(), get_string(), get_subst_proxy(), get_suffix(), get_target_scanner(), has_builder(), has_explicit_builder(), is_derived(), is_literal(), missing(), multiple_side_effect_has_builder(), new_binfo(), new_ninfo(), postprocess(), prepare(), push_to_cache(), remove(), render_include_tree(), reset_executor(), retrieve_from_cache(), rexists(), scan(), scanner_key(), select_scanner(), set_always_build(), set_executor(), set_explicit(), set_nocache(), set_noclean(), set_precious(), set_specific_source(), set_state(), state_has_changed(), store_info(), visited()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

16.4.2 Properties

Name	Description
Inherited from object	
__class__	

16.4.3 Class Variables

Name	Description
Inherited from SCons.Node.Node (Section 13.6)	
__metaclass__, memoizer_counters	

17 Module SCons.PathList

SCons.PathList

A module for handling lists of directory paths (the sort of things that get set as CPPPATH, LIBPATH, etc.) with as much caching of data and efficiency as we can while still keeping the evaluation delayed so that we Do the Right Thing (almost) regardless of how the variable is specified.

17.1 Functions

node_conv(*obj*)

This is the "string conversion" routine that we have our substitutions use to return Nodes, not strings. This relies on the fact that an EntryProxy object has a get() method that returns the underlying Node that it wraps, which is a bit of architectural dependence that we might need to break or modify in the future in response to additional requirements.

PathList(*pathlist*)

Returns the cached _PathList object for the specified pathlist, creating and caching a new object as necessary.

17.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/PathList.py 5134 2010/08/16 23:02:40 bd...`
__doc__	**Value:** `"""SCons.PathL...`
TYPE_STRING_NO_SUBST	**Value:** `0`
TYPE_STRING_SUBST	**Value:** `1`
TYPE_OBJECT	**Value:** `2`
__package__	**Value:** `'SCons'`

18 Module SCons.SConf

SCons.SConf

Autoconf-like configuration support.

18.1 Functions

SetBuildType(*type*)

SetCacheMode(*mode*)

Set the Configure cache mode. mode must be one of “auto”, “force”, or “cache”.

SetProgressDisplay(*display*)

Set the progress display to use (called from SCons.Script)

CreateConfigHBuilder(*env*)

Called just before the building targets phase begins.

SConf(**args*, ***kw*)

CheckFunc(*context*, *function_name*, *header*=`None`, *language*=`None`)

CheckType(*context*, *type_name*, *includes*=`''`, *language*=`None`)

CheckTypeSize(*context*, *type_name*, *includes*=`''`, *language*=`None`, *expect*=`None`)

CheckDeclaration(*context*, *declaration*, *includes*=`''`, *language*=`None`)

createIncludesFromHeaders(*headers*, *leaveLast*, *include_quotes*=`'""'`)

CheckHeader(*context*, *header*, *include_quotes*=`'<>'`, *language*=`None`)

A test for a C or C++ header file.

CheckCC(*context*)

CheckCXX(*context*)

CheckSHCC(*context*)

CheckSHCXX(*context*)

CheckCHeader(*context*, *header*, *include_quotes*=`'""'`)

A test for a C header file.

CheckCXXHeader(*context*, *header*, *include_quotes*=`'""'`)

A test for a C++ header file.

CheckLib(*context*, *library*=`None`, *symbol*=`'main'`, *header*=`None`, *language*=`None`, *autoadd*=`1`)

A test for a library. See also CheckLibWithHeader. Note that library may also be None to test whether the given symbol compiles without flags.

CheckLibWithHeader(*context*, *libs*, *header*, *language*, *call*=`None`, *autoadd*=`1`)

Another (more sophisticated) test for a library. Checks, if library and header is available for language (may be 'C' or 'CXX'). Call maybe be a valid expression _with_ a trailing ';'. As in CheckLib, we support library=None, to test if the call compiles without extra link flags.

18.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/SConf.py 5134 2010/08/16 23:02:40 bdeegan'`
build_type	**Value:** `None`
build_types	**Value:** `['clean', 'help']`
dryrun	**Value:** `0`
AUTO	**Value:** `0`
FORCE	**Value:** `1`
CACHE	**Value:** `2`
cache_mode	**Value:** `0`
progress_display	**Value:** `DisplayEngine()`
SConfFS	**Value:** `None`
sconf_global	**Value:** `None`
__package__	**Value:** `'SCons'`

18.3 Class SConfWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.SConf.SConfWarning

18.3.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

18.3.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

18.4 Class SConfError

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.SConf.SConfError

Known Subclasses: SCons.SConf.ConfigureCacheError, SCons.SConf.ConfigureDryRunError

18.4.1 Methods

__init__(*self*, *msg*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

18.4.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

18.5 Class ConfigureDryRunError

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.SConf.SConfError
SCons.SConf.ConfigureDryRunError

Raised when a file or directory needs to be updated during a Configure process, but the user requested a dry-run

18.5.1 Methods

__init__(*self*, *target*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

18.5.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

18.6 Class ConfigureCacheError

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.SConf.SConfError
SCons.SConf.ConfigureCacheError

Raised when a use explicitely requested the cache feature, but the test is run the first time.

18.6.1 Methods

__init__(*self*, *target*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

18.6.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

18.7 Class SConfBuildInfo

object
SCons.Node.BuildInfoBase
SCons.Node.FS.FileBuildInfo
SCons.SConf.SConfBuildInfo

Special build info for targets of configure tests. Additional members are result (did the builder succeed last time?) and string, which contains messages of the original build phase.

18.7.1 Methods

set_build_result(*self*, *result*, *string*)

Inherited from SCons.Node.FS.FileBuildInfo(Section 15.15)

convert_from_sconsign(), convert_to_sconsign(), format(), prepare_dependencies()

Inherited from SCons.Node.BuildInfoBase(Section 13.5)

__init__(), merge()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

18.7.2 Properties

Name	Description
Inherited from object	
__class__	

18.7.3 Class Variables

Name	Description
result	**Value:** `None`
string	**Value:** `None`
Inherited from SCons.Node.FS.FileBuildInfo (Section 15.15)	
current_version_id	

18.8 Class Streamer

object ─┐
SCons.SConf.Streamer

'Sniffer' for a file-like writable object. Similar to the unix tool tee.

18.8.1 Methods

__init__(*self*, *orig*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

write(*self*, *str*)

writelines(*self*, *lines*)

getvalue(*self*)

Return everything written to orig since the Streamer was created.

flush(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

18.8.2 Properties

Name	Description
Inherited from object	
__class__	

18.9 Class SConfBuildTask

object
SCons.Taskmaster.Task
SCons.Taskmaster.AlwaysTask
SCons.SConf.SConfBuildTask

This is almost the same as SCons.Script.BuildTask. Handles SConfErrors correctly and knows about the current cache_mode.

18.9.1 Methods

display(*self*, *message*)

Hook to allow the calling interface to display a message.

This hook gets called as part of preparing a task for execution (that is, a Node to be built). As part of figuring out what Node should be built next, the actually target list may be altered, along with a message describing the alteration. The calling interface can subclass Task and provide a concrete implementation of this method to see those messages. Overrides: SCons.Taskmaster.Task.display extit(inherited documentation)

display_cached_string(*self*, *bi*)

Logs the original builder messages, given the SConfBuildInfo instance bi.

failed(*self*)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure(). Overrides: SCons.Taskmaster.Task.failed extit(inherited documentation)

collect_node_states(*self*)

execute(*self*)

Called to execute the task.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in prepare(), executed() or failed(). Overrides: SCons.Taskmaster.Task.execute extit(inherited documentation)

Inherited from SCons.Taskmaster.AlwaysTask(Section 35.5)

needs_execute()

Inherited from SCons.Taskmaster.Task(Section 35.4)

__init__(), exc_clear(), exc_info(), exception_set(), executed(), executed_with_callbacks(), executed_without_callbacks(), fail_continue(), fail_stop(), get_target(), make_ready(), make_ready_all(), make_ready_current(), postprocess(), prepare(), trace_message()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

18.9.2 Properties

Name	Description
Inherited from object	
__class__	

18.10 Class SConfBase

object ┐
 SCons.SConf.SConfBase

This is simply a class to represent a configure context. After creating a SConf object, you can call any tests. After finished with your tests, be sure to call the Finish() method, which returns the modified environment. Some words about caching: In most cases, it is not necessary to cache Test results explicitely. Instead, we use the scons dependency checking mechanism. For example, if one wants to compile a test program (SConf.TryLink), the compiler is only called, if the program dependencies have changed. However, if the program could not be compiled in a former SConf run, we need to explicitely cache this error.

18.10.1 Methods

__init__(*self*, *env*, *custom_tests*=`{}`, *conf_dir*=`'$CONFIGUREDIR'`, *log_file*=`'$CONFIGURELOG'`, *config_h*=`None`, *_depth*=`0`)

Constructor. Pass additional tests in the custom_tests-dictinary, e.g. custom_tests={'CheckPrivate':MyPrivateTest}, where MyPrivateTest defines a custom test. Note also the conf_dir and log_file arguments (you may want to build tests in the VariantDir, not in the SourceDir) Overrides: object.__init__

Finish(*self*)

Call this method after finished with your tests: env = sconf.Finish()

Define(*self*, *name*, *value*=`None`, *comment*=`None`)

Define a pre processor symbol name, with the optional given value in the current config header.

If value is None (default), then #define name is written. If value is not none, then #define name value is written.

comment is a string which will be put as a C comment in the header, to explain the meaning of the value (appropriate C comments /* and */ will be put automatically.

BuildNodes(*self*, *nodes*)

Tries to build the given nodes immediately. Returns 1 on success, 0 on error.

pspawn_wrapper(*self*, *sh*, *escape*, *cmd*, *args*, *env*)

Wrapper function for handling piped spawns.

This looks to the calling interface (in Action.py) like a “normal” spawn, but associates the call with the PSPAWN variable from the construction environment and with the streams to which we want the output logged. This gets slid into the construction environment as the SPAWN variable so Action.py doesn't have to know or care whether it's spawning a piped command or not.

TryBuild(*self*, *builder*, *text*=`None`, *extension*=`''`)

Low level TryBuild implementation. Normally you don't need to call that - you can use TryCompile / TryLink / TryRun instead

TryAction(*self*, *action*, *text*=`None`, *extension*=`''`)

Tries to execute the given action with optional source file contents <text> and optional source file extension <extension>, Returns the status (0 : failed, 1 : ok) and the contents of the output file.

TryCompile(*self*, *text*, *extension*)

Compiles the program given in text to an env.Object, using extension as file extension (e.g. '.c'). Returns 1, if compilation was successful, 0 otherwise. The target is saved in self.lastTarget (for further processing).

TryLink(*self*, *text*, *extension*)

Compiles the program given in text to an executable env.Program, using extension as file extension (e.g. '.c'). Returns 1, if compilation was successful, 0 otherwise. The target is saved in self.lastTarget (for further processing).

TryRun(*self*, *text*, *extension*)

Compiles and runs the program given in text, using extension as file extension (e.g. '.c'). Returns (1, outputStr) on success, (0, '') otherwise. The target (a file containing the program's stdout) is saved in self.lastTarget (for further processing).

AddTest(*self*, *test_name*, *test_instance*)

Adds test_class to this SConf instance. It can be called with self.test_name(...)

AddTests(*self*, *tests*)

Adds all the tests given in the tests dictionary to this SConf instance

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

18.10.2 Properties

Name	Description
Inherited from object	
__class__	

18.11 Class CheckContext

object
└ **SCons.SConf.CheckContext**

Provides a context for configure tests. Defines how a test writes to the screen and log file.

A typical test is just a callable with an instance of CheckContext as first argument:

def CheckCustom(context, ...) context.Message('Checking my weird test ... ') ret = myWeirdTestFunction(...) context.Result(ret)

Often, myWeirdTestFunction will be one of context.TryCompile/context.TryLink/context.TryRun. The results of those are cached, for they are only rebuild, if the dependencies have changed.

18.11.1 Methods

__init__(*self*, *sconf*)

Constructor. Pass the corresponding SConf instance. Overrides: object.__init__

Message(*self*, *text*)

Inform about what we are doing right now, e.g. 'Checking for SOMETHING ... '

Result(*self*, *res*)

Inform about the result of the test. res may be an integer or a string. In case of an integer, the written text will be 'yes' or 'no'. The result is only displayed when self.did_show_result is not set.

TryBuild(*self*, **args*, ***kw*)

TryAction(*self*, **args*, ***kw*)

TryCompile(*self*, **args*, ***kw*)

TryLink(*self*, **args*, ***kw*)

TryRun(*self*, **args*, ***kw*)

__getattr__(*self*, *attr*)

BuildProg(*self*, *text*, *ext*)

CompileProg(*self*, *text*, *ext*)

CompileSharedObject(*self*, *text*, *ext*)

RunProg(*self*, *text*, *ext*)

AppendLIBS(*self*, *lib_name_list*)

PrependLIBS(*self*, *lib_name_list*)

SetLIBS(*self*, *val*)

Display(*self*, *msg*)

Log(*self*, *msg*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

18.11.2 Properties

Name	Description
Inherited from object	
__class__	

19 Module SCons.SConsign

SCons.SConsign

Writing and reading information to the .sconsign file or files.

19.1 Functions

corrupt_dblite_warning(*filename*)

Get_DataBase(*dir*)

Reset()

Reset global state. Used by unit tests that end up using SConsign multiple times to get a clean slate for each test.

write()

File(*name*, *dbm_module*=`None`)

Arrange for all signatures to be stored in a global .sconsign.db* file.

19.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/SConsign.py 5134 2010/08/16 23:02:40 bd...`
sig_files	**Value:** `[]`
DataBase	**Value:** `{}`
DB_Name	**Value:** `'.sconsign'`
DB_sync_list	**Value:** `[]`
__package__	**Value:** `'SCons'`

19.3 Class SConsignEntry

object ─┐
SCons.SConsign.SConsignEntry

Wrapper class for the generic entry in a .sconsign file. The Node subclass populates it with attributes as it pleases.

XXX As coded below, we do expect a '.binfo' attribute to be added, but we'll probably generalize this in the next refactorings.

19.3.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

convert_to_sconsign(*self*)

convert_from_sconsign(*self*, *dir*, *name*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

19.3.2 Properties

Name	Description
Inherited from object	
__class__	

19.3.3 Class Variables

Name	Description
current_version_id	**Value:** `1`

19.4 Class Base

object ─┐
SCons.SConsign.Base

Known Subclasses: SCons.SConsign.DB, SCons.SConsign.Dir

This is the controlling class for the signatures for the collection of entries associated with a specific directory. The actual directory association will be maintained by a subclass that is specific to the underlying storage method. This class provides a common set of methods for fetching and storing the individual bits of information that make up signature entry.

19.4.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

---

get_entry(*self*, *filename*)

Fetch the specified entry attribute.

---

set_entry(*self*, *filename*, *obj*)

Set the entry.

---

do_not_set_entry(*self*, *filename*, *obj*)

---

store_info(*self*, *filename*, *node*)

---

do_not_store_info(*self*, *filename*, *node*)

---

merge(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

19.4.2 Properties

Name	Description
Inherited from object	
__class__	

19.5 Class DB

object

SCons.SConsign.Base

SCons.SConsign.DB

A Base subclass that reads and writes signature information from a global .sconsign.db* file--the actual file suffix is determined by the database module.

19.5.1 Methods

__init__(*self*, *dir*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

write(*self*, *sync*=1)

Inherited from SCons.SConsign.Base(Section 19.4)

do_not_set_entry(), do_not_store_info(), get_entry(), merge(), set_entry(), store_info()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

19.5.2 Properties

Name	Description
Inherited from object	
__class__	

19.6 Class Dir

object
SCons.SConsign.Base
SCons.SConsign.Dir

Known Subclasses: SCons.SConsign.DirFile

19.6.1 Methods

__init__(*self*, *fp*=None, *dir*=None)

fp - file pointer to read entries from Overrides: object.__init__

Inherited from SCons.SConsign.Base(Section 19.4)

do_not_set_entry(), do_not_store_info(), get_entry(), merge(), set_entry(), store_info()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

19.6.2 Properties

Name	Description
Inherited from object	
__class__	

19.7 Class DirFile

object
SCons.SConsign.Base
SCons.SConsign.Dir
SCons.SConsign.DirFile

Encapsulates reading and writing a per-directory .sconsign file.

19.7.1 Methods

__init__(*self*, *dir*)

dir - the directory for the file Overrides: object.__init__

write(*self*, *sync*=1)

Write the .sconsign file to disk.

Try to write to a temporary file first, and rename it if we succeed. If we can't write to the temporary file, it's probably because the directory isn't writable (and if so, how did we build anything in this directory, anyway?), so try to write directly to the .sconsign file as a backup. If we can't rename, try to copy the temporary contents back to the .sconsign file. Either way, always try to remove the temporary file at the end.

Inherited from SCons.SConsign.Base(Section 19.4)

do_not_set_entry(), do_not_store_info(), get_entry(), merge(), set_entry(), store_info()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

19.7.2 Properties

Name	Description
Inherited from object	
__class__	

19.8 Class DB

object

SCons.SConsign.Base

SCons.SConsign.DB

A Base subclass that reads and writes signature information from a global .sconsign.db* file--the actual file suffix is determined by the database module.

19.8.1 Methods

__init__(*self*, *dir*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

write(*self*, *sync*=1)

Inherited from SCons.SConsign.Base(Section 19.4)

do_not_set_entry(), do_not_store_info(), get_entry(), merge(), set_entry(), store_info()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

19.8.2 Properties

Name	Description
Inherited from object	
__class__	

20 Package SCons.Scanner

SCons.Scanner

The Scanner package for the SCons software construction utility.

20.1 Modules

20.2 Functions

Scanner(*function*, **args*, ***kw*)

Public interface factory function for creating different types of Scanners based on the different types of "functions" that may be supplied.

TODO: Deprecate this some day. We've moved the functionality inside the Base class and really don't need this factory function any more. It was, however, used by some of our Tool modules, so the call probably ended up in various people's custom modules patterned on SCons code.

20.3 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/__init__.py 5134 2010/08/16 23:...`

continued on next page

Name	Description
__package__	**Value:** `'SCons.Scanner'`

20.4 Class FindPathDirs

object ─┐
SCons.Scanner.FindPathDirs

A class to bind a specific *PATH variable name to a function that will return all of the *path directories.

20.4.1 Methods

__init__(*self*, *variable*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, *env*, *dir*=`None`, *target*=`None`, *source*=`None`, *argument*=`None`)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

20.4.2 Properties

Name	Description
Inherited from object	
__class__	

20.5 Class Base

object ─┐
SCons.Scanner.Base

Known Subclasses: SCons.Scanner.Current, SCons.Scanner.Selector, SCons.Scanner.LaTeX.LaTeX

The base class for dependency scanners. This implements straightforward, single-pass scanning of a single file.

20.5.1 Methods

__call__(*self*, *node*, *env*, *path*=`()`)

This method scans a single object. 'node' is the node that will be passed to the scanner function, and 'env' is the environment that will be passed to the scanner function. A list of direct dependency nodes for the specified node will be returned.

__cmp__(*self*, *other*)

__hash__(*self*)

hash(x) Overrides: object.__hash__ extit(inherited documentation)

__init__(*self*, *function*, *name*='NONE', *argument*=<class 'SCons.Scanner._Null'>, *skeys*=<class 'SCons.Scanner._Null'>, *path_function*=None, *node_class*=<class 'SCons.Node.FS.Base'>, *node_factory*=None, *scan_check*=None, *recursive*=None)

Construct a new scanner object given a scanner function.

'function' - a scanner function taking two or three arguments and returning a list of strings.

'name' - a name for identifying this scanner object.

'argument' - an optional argument that, if specified, will be passed to both the scanner function and the path_function.

'skeys' - an optional list argument that can be used to determine which scanner should be used for a given Node. In the case of File nodes, for example, the 'skeys' would be file suffixes.

'path_function' - a function that takes four or five arguments (a construction environment, Node for the directory containing the SConscript file that defined the primary target, list of target nodes, list of source nodes, and optional argument for this instance) and returns a tuple of the directories that can be searched for implicit dependency files. May also return a callable() which is called with no args and returns the tuple (supporting Bindable class).

'node_class' - the class of Nodes which this scan will return. If node_class is None, then this scanner will not enforce any Node conversion and will return the raw results from the underlying scanner function.

'node_factory' - the factory function to be called to translate the raw results returned by the scanner function into the expected node_class objects.

'scan_check' - a function to be called to first check whether this node really needs to be scanned.

'recursive' - specifies that this scanner should be invoked recursively on all of the implicit dependencies it returns (the canonical example being #include lines in C source files). May be a callable, which will be called to filter the list of nodes found to select a subset for recursive scanning (the canonical example being only recursively scanning subdirectories within a directory).

The scanner function's first argument will be a Node that should be scanned for dependencies, the second argument will be an Environment object, the third argument will be the tuple of paths returned by the path_function, and the fourth argument will be the value passed into 'argument', and the returned list should contain the Nodes for all the direct dependencies of the file.

Examples:

s = Scanner(my_scanner_function)

s = Scanner(function = my_scanner_function)

s = Scanner(function = my_scanner_function, argument = 'foo') Overrides: object.__init__

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

add_scanner(*self*, *skey*, *scanner*)

add_skey(*self*, *skey*)

Add a skey to the list of skeys

get_skeys(*self*, *env*=`None`)

path(*self*, *env*, *dir*=`None`, *target*=`None`, *source*=`None`)

recurse_nodes(*self*, *nodes*)

select(*self*, *node*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

20.5.2 Properties

Name	Description
Inherited from object	
__class__	

20.6 Class Selector

object ┐
SCons.Scanner.Base ┐
SCons.Scanner.Selector

A class for selecting a more specific scanner based on the scanner_key() (suffix) for a specific Node.

TODO: This functionality has been moved into the inner workings of the Base class, and this class will be deprecated at some point. (It was never exposed directly as part of the public interface, although it is used by the Scanner() factory function that was used by various Tool modules and therefore was likely a template for custom modules that may be out there.)

20.6.1 Methods

__init__(*self*, *dict*, **args*, ***kw*)

Construct a new scanner object given a scanner function.

'function' - a scanner function taking two or three arguments and returning a list of strings.

'name' - a name for identifying this scanner object.

'argument' - an optional argument that, if specified, will be passed to both the scanner function and the path_function.

'skeys' - an optional list argument that can be used to determine which scanner should be used for a given Node. In the case of File nodes, for example, the 'skeys' would be file suffixes.

'path_function' - a function that takes four or five arguments (a construction environment, Node for the directory containing the SConscript file that defined the primary target, list of target nodes, list of source nodes, and optional argument for this instance) and returns a tuple of the directories that can be searched for implicit dependency files. May also return a callable() which is called with no args and returns the tuple (supporting Bindable class).

'node_class' - the class of Nodes which this scan will return. If node_class is None, then this scanner will not enforce any Node conversion and will return the raw results from the underlying scanner function.

'node_factory' - the factory function to be called to translate the raw results returned by the scanner function into the expected node_class objects.

'scan_check' - a function to be called to first check whether this node really needs to be scanned.

'recursive' - specifies that this scanner should be invoked recursively on all of the implicit dependencies it returns (the canonical example being #include lines in C source files). May be a callable, which will be called to filter the list of nodes found to select a subset for recursive scanning (the canonical example being only recursively scanning subdirectories within a directory).

The scanner function's first argument will be a Node that should be scanned for dependencies, the second argument will be an Environment object, the third argument will be the tuple of paths returned by the path_function, and the fourth argument will be the value passed into 'argument', and the returned list should contain the Nodes for all the direct dependencies of the file.

Examples:

s = Scanner(my_scanner_function)

s = Scanner(function = my_scanner_function)

s = Scanner(function = my_scanner_function, argument = 'foo') Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, *node*, *env*, *path*=`()`)

This method scans a single object. 'node' is the node that will be passed to the scanner function, and 'env' is the environment that will be passed to the scanner function. A list of direct dependency nodes for the specified node will be returned. Overrides: SCons.Scanner.Base.__call__ extit(inherited documentation)

select(*self*, *node*)

Overrides: SCons.Scanner.Base.select

add_scanner(*self*, *skey*, *scanner*)

Overrides: SCons.Scanner.Base.add_scanner

Inherited from SCons.Scanner.Base(Section 20.5)

__cmp__(), __hash__(), __str__(), add_skey(), get_skeys(), path(), recurse_nodes()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

20.6.2 Properties

Name	Description
Inherited from object	
__class__	

20.7 Class Current

object

SCons.Scanner.Base

SCons.Scanner.Current

Known Subclasses: SCons.Scanner.Classic

A class for scanning files that are source files (have no builder) or are derived files and are current (which implies that they exist, either locally or in a repository).

20.7.1 Methods

__init__(*self*, **args*, ***kw*)

Construct a new scanner object given a scanner function.

'function' - a scanner function taking two or three arguments and returning a list of strings.

'name' - a name for identifying this scanner object.

'argument' - an optional argument that, if specified, will be passed to both the scanner function and the path_function.

'skeys' - an optional list argument that can be used to determine which scanner should be used for a given Node. In the case of File nodes, for example, the 'skeys' would be file suffixes.

'path_function' - a function that takes four or five arguments (a construction environment, Node for the directory containing the SConscript file that defined the primary target, list of target nodes, list of source nodes, and optional argument for this instance) and returns a tuple of the directories that can be searched for implicit dependency files. May also return a callable() which is called with no args and returns the tuple (supporting Bindable class).

'node_class' - the class of Nodes which this scan will return. If node_class is None, then this scanner will not enforce any Node conversion and will return the raw results from the underlying scanner function.

'node_factory' - the factory function to be called to translate the raw results returned by the scanner function into the expected node_class objects.

'scan_check' - a function to be called to first check whether this node really needs to be scanned.

'recursive' - specifies that this scanner should be invoked recursively on all of the implicit dependencies it returns (the canonical example being #include lines in C source files). May be a callable, which will be called to filter the list of nodes found to select a subset for recursive scanning (the canonical example being only recursively scanning subdirectories within a directory).

The scanner function's first argument will be a Node that should be scanned for dependencies, the second argument will be an Environment object, the third argument will be the tuple of paths returned by the path_function, and the fourth argument will be the value passed into 'argument', and the returned list should contain the Nodes for all the direct dependencies of the file.

Examples:

s = Scanner(my_scanner_function)

s = Scanner(function = my_scanner_function)

s = Scanner(function = my_scanner_function, argument = 'foo') Overrides: object.__init__ extit(inherited documentation)

Inherited from SCons.Scanner.Base(Section 20.5)

__call__(), __cmp__(), __hash__(), __str__(), add_scanner(), add_skey(), get_skeys(), path(), recurse_nodes(), select()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

20.7.2 Properties

Name	Description
Inherited from object	
__class__	

20.8 Class Classic

object

SCons.Scanner.Base

SCons.Scanner.Current

SCons.Scanner.Classic

Known Subclasses: SCons.Scanner.ClassicCPP, SCons.Scanner.D.D, SCons.Scanner.Fortran.F90Scanner

A Scanner subclass to contain the common logic for classic CPP-style include scanning, but which can be customized to use different regular expressions to find the includes.

Note that in order for this to work "out of the box" (without overriding the find_include() and sort_key() methods), the regular expression passed to the constructor must return the name of the include file in group 0.

20.8.1 Methods

__init__(*self*, *name*, *suffixes*, *path_variable*, *regex*, **args*, ***kw*)

Construct a new scanner object given a scanner function.

'function' - a scanner function taking two or three arguments and returning a list of strings.

'name' - a name for identifying this scanner object.

'argument' - an optional argument that, if specified, will be passed to both the scanner function and the path_function.

'skeys' - an optional list argument that can be used to determine which scanner should be used for a given Node. In the case of File nodes, for example, the 'skeys' would be file suffixes.

'path_function' - a function that takes four or five arguments (a construction environment, Node for the directory containing the SConscript file that defined the primary target, list of target nodes, list of source nodes, and optional argument for this instance) and returns a tuple of the directories that can be searched for implicit dependency files. May also return a callable() which is called with no args and returns the tuple (supporting Bindable class).

'node_class' - the class of Nodes which this scan will return. If node_class is None, then this scanner will not enforce any Node conversion and will return the raw results from the underlying scanner function.

'node_factory' - the factory function to be called to translate the raw results returned by the scanner function into the expected node_class objects.

'scan_check' - a function to be called to first check whether this node really needs to be scanned.

'recursive' - specifies that this scanner should be invoked recursively on all of the implicit dependencies it returns (the canonical example being #include lines in C source files). May be a callable, which will be called to filter the list of nodes found to select a subset for recursive scanning (the canonical example being only recursively scanning subdirectories within a directory).

The scanner function's first argument will be a Node that should be scanned for dependencies, the second argument will be an Environment object, the third argument will be the tuple of paths returned by the path_function, and the fourth argument will be the value passed into 'argument', and the returned list should contain the Nodes for all the direct dependencies of the file.

Examples:

s = Scanner(my_scanner_function)

s = Scanner(function = my_scanner_function)

s = Scanner(function = my_scanner_function, argument = 'foo') Overrides: object.__init__ extit(inherited documentation)

find_include(*self*, *include*, *source_dir*, *path*)

find_include_names(*self*, *node*)

scan(*self*, *node*, *path*=`()`)

sort_key(*self*, *include*)

Inherited from SCons.Scanner.Base(Section 20.5)

__call__(), __cmp__(), __hash__(), __str__(), add_scanner(), add_skey(), get_skeys(), path(), recurse_nodes(), select()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

20.8.2 Properties

Name	Description
Inherited from object	
__class__	

20.9 Class ClassicCPP

object
SCons.Scanner.Base
SCons.Scanner.Current
SCons.Scanner.Classic
SCons.Scanner.ClassicCPP

A Classic Scanner subclass which takes into account the type of bracketing used to include the file, and uses classic CPP rules for searching for the files based on the bracketing.

Note that in order for this to work, the regular expression passed to the constructor must return the leading bracket in group 0, and the contained filename in group 1.

20.9.1 Methods

find_include(*self*, *include*, *source_dir*, *path*)
Overrides: SCons.Scanner.Classic.find_include

sort_key(*self*, *include*)
Overrides: SCons.Scanner.Classic.sort_key

Inherited from SCons.Scanner.Classic(Section 20.8)

__init__(), find_include_names(), scan()

Inherited from SCons.Scanner.Base(Section 20.5)

__call__(), __cmp__(), __hash__(), __str__(), add_scanner(), add_skey(), get_skeys(), path(), recurse_nodes(), select()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

20.9.2 Properties

Name	Description
Inherited from object	
__class__	

21 Module SCons.Scanner.C

SCons.Scanner.C

This module implements the depenency scanner for C/C++ code.

21.1 Functions

dictify_CPPDEFINES(*env*)

CScanner()

Return a prototype Scanner instance for scanning source files that use the C pre-processor

21.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/C.py 5134 2010/08/16 23:02:40 b...`
__package__	**Value:** `'SCons.Scanner'`

21.3 Class SConsCPPScanner

object

SCons.cpp.PreProcessor

SCons.Scanner.C.SConsCPPScanner

SCons-specific subclass of the cpp.py module's processing.

We subclass this so that: 1) we can deal with files represented by Nodes, not strings; 2) we can keep track of the files that are missing.

21.3.1 Methods

__init__(*self*, **args*, ***kw*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

initialize_result(*self*, *fname*)

Overrides: SCons.cpp.PreProcessor.initialize_result

finalize_result(*self*, *fname*)

Overrides: SCons.cpp.PreProcessor.finalize_result

find_include_file(*self*, *t*)

Finds the #include file for a given preprocessor tuple. Overrides: SCons.cpp.PreProcessor.find_include_file extit(inherited documentation)

read_file(*self*, *file*)

Overrides: SCons.cpp.PreProcessor.read_file

Inherited from SCons.cpp.PreProcessor(Section 52.4)

__call__(), all_include(), do_define(), do_elif(), do_else(), do_endif(), do_if(), do_ifdef(), do_ifndef(), do_import(), do_include(), do_include_next(), do_nothing(), do_undef(), eval_expression(), process_contents(), resolve_include(), restore(), save(), scons_current_file(), start_handling_includes(), stop_handling_includes(), tupleize()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

21.3.2 Properties

Name	Description
Inherited from object	
__class__	

21.4 Class SConsCPPScannerWrapper

object ─┐

SCons.Scanner.C.SConsCPPScannerWrapper

The SCons wrapper around a cpp.py scanner.

This is the actual glue between the calling conventions of generic SCons scanners, and the (subclass of) cpp.py class that knows how to look for #include lines with reasonably real C-preprocessor-like evaluation of #if/#ifdef/#else/#elif lines.

21.4.1 Methods

__init__(*self*, *name*, *variable*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, *node*, *env*, *path*=`()`)

recurse_nodes(*self*, *nodes*)

select(*self*, *node*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

21.4.2 Properties

Name	Description
Inherited from object	
__class__	

22 Module SCons.Scanner.D

SCons.Scanner.D

Scanner for the Digital Mars "D" programming language.

Coded by Andy Friesen 17 Nov 2003

22.1 Functions

DScanner()

Return a prototype Scanner instance for scanning D source files

22.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/D.py 5134 2010/08/16 23:02:40 b...`
__package__	**Value:** `'SCons.Scanner'`

22.3 Class D

object
SCons.Scanner.Base
SCons.Scanner.Current
SCons.Scanner.Classic
SCons.Scanner.D.D

22.3.1 Methods

__init__(*self*)

Construct a new scanner object given a scanner function.

'function' - a scanner function taking two or three arguments and returning a list of strings.

'name' - a name for identifying this scanner object.

'argument' - an optional argument that, if specified, will be passed to both the scanner function and the path_function.

'skeys' - an optional list argument that can be used to determine which scanner should be used for a given Node. In the case of File nodes, for example, the 'skeys' would be file suffixes.

'path_function' - a function that takes four or five arguments (a construction environment, Node for the directory containing the SConscript file that defined the primary target, list of target nodes, list of source nodes, and optional argument for this instance) and returns a tuple of the directories that can be searched for implicit dependency files. May also return a callable() which is called with no args and returns the tuple (supporting Bindable class).

'node_class' - the class of Nodes which this scan will return. If node_class is None, then this scanner will not enforce any Node conversion and will return the raw results from the underlying scanner function.

'node_factory' - the factory function to be called to translate the raw results returned by the scanner function into the expected node_class objects.

'scan_check' - a function to be called to first check whether this node really needs to be scanned.

'recursive' - specifies that this scanner should be invoked recursively on all of the implicit dependencies it returns (the canonical example being #include lines in C source files). May be a callable, which will be called to filter the list of nodes found to select a subset for recursive scanning (the canonical example being only recursively scanning subdirectories within a directory).

The scanner function's first argument will be a Node that should be scanned for dependencies, the second argument will be an Environment object, the third argument will be the tuple of paths returned by the path_function, and the fourth argument will be the value passed into 'argument', and the returned list should contain the Nodes for all the direct dependencies of the file.

Examples:

s = Scanner(my_scanner_function)

s = Scanner(function = my_scanner_function)

s = Scanner(function = my_scanner_function, argument = 'foo') Overrides: object.__init__ extit(inherited documentation)

find_include(*self*, *include*, *source_dir*, *path*)
Overrides: SCons.Scanner.Classic.find_include

find_include_names(*self*, *node*)
Overrides: SCons.Scanner.Classic.find_include_names

Inherited from SCons.Scanner.Classic(Section 20.8)

scan(), sort_key()

Inherited from SCons.Scanner.Base(Section 20.5)

__call__(), __cmp__(), __hash__(), __str__(), add_scanner(), add_skey(), get_skeys(), path(), recurse_nodes(), select()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

22.3.2 Properties

Name	Description
Inherited from object	
__class__	

23 Module SCons.Scanner.Dir

23.1 Functions

only_dirs(*nodes*)

DirScanner(***kw*)

Return a prototype Scanner instance for scanning directories for on-disk files

---

DirEntryScanner(***kw*)

Return a prototype Scanner instance for "scanning" directory Nodes for their in-memory entries

---

do_not_scan(*k*)

---

scan_on_disk(*node*, *env*, *path*=`()`)

Scans a directory for on-disk files and directories therein.

Looking up the entries will add these to the in-memory Node tree representation of the file system, so all we have to do is just that and then call the in-memory scanning function.

---

scan_in_memory(*node*, *env*, *path*=`()`)

"Scans" a Node.FS.Dir for its in-memory entries.

23.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/Dir.py 5134 2010/08/16 23:02:40...`
skip_entry	**Value:** `{'.': 1, '..': 1, '.sconsign': 1, '.sconsign.bak': 1, '.s...`

continued on next page

Name	Description
skip_entry_list	**Value:** `['.', '..', '.sconsign', '.sconsign.dblite', '.sconsign.d...`
__package__	**Value:** `'SCons.Scanner'`
skip	**Value:** `'.sconsign.db'`

24 Module SCons.Scanner.Fortran

SCons.Scanner.Fortran

This module implements the dependency scanner for Fortran code.

24.1 Functions

FortranScan(*path_variable*=`'FORTRANPATH'`)

Return a prototype Scanner instance for scanning source files for Fortran USE & INCLUDE statements

24.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/Fortran.py 5134 2010/08/16 23:0...`
__package__	**Value:** `'SCons.Scanner'`

24.3 Class F90Scanner

object
└ SCons.Scanner.Base
  └ SCons.Scanner.Current
    └ SCons.Scanner.Classic
      └ **SCons.Scanner.Fortran.F90Scanner**

A Classic Scanner subclass for Fortran source files which takes into account both USE and INCLUDE statements. This scanner will work for both F77 and F90 (and beyond) compilers.

Currently, this scanner assumes that the include files do not contain USE statements. To enable the ability to deal with USE statements in include files, add logic right after the module names are found to loop over each include file, search for and locate each

USE statement, and append each module name to the list of dependencies. Caching the search results in a common dictionary somewhere so that the same include file is not searched multiple times would be a smart thing to do.

24.3.1 Methods

__init__(*self*, *name*, *suffixes*, *path_variable*, *use_regex*, *incl_regex*, *def_regex*, **args*, ***kw*)

Construct a new scanner object given a scanner function.

'function' - a scanner function taking two or three arguments and returning a list of strings.

'name' - a name for identifying this scanner object.

'argument' - an optional argument that, if specified, will be passed to both the scanner function and the path_function.

'skeys' - an optional list argument that can be used to determine which scanner should be used for a given Node. In the case of File nodes, for example, the 'skeys' would be file suffixes.

'path_function' - a function that takes four or five arguments (a construction environment, Node for the directory containing the SConscript file that defined the primary target, list of target nodes, list of source nodes, and optional argument for this instance) and returns a tuple of the directories that can be searched for implicit dependency files. May also return a callable() which is called with no args and returns the tuple (supporting Bindable class).

'node_class' - the class of Nodes which this scan will return. If node_class is None, then this scanner will not enforce any Node conversion and will return the raw results from the underlying scanner function.

'node_factory' - the factory function to be called to translate the raw results returned by the scanner function into the expected node_class objects.

'scan_check' - a function to be called to first check whether this node really needs to be scanned.

'recursive' - specifies that this scanner should be invoked recursively on all of the implicit dependencies it returns (the canonical example being #include lines in C source files). May be a callable, which will be called to filter the list of nodes found to select a subset for recursive scanning (the canonical example being only recursively scanning subdirectories within a directory).

The scanner function's first argument will be a Node that should be scanned for dependencies, the second argument will be an Environment object, the third argument will be the tuple of paths returned by the path_function, and the fourth argument will be the value passed into 'argument', and the returned list should contain the Nodes for all the direct dependencies of the file.

Examples:

s = Scanner(my_scanner_function)

s = Scanner(function = my_scanner_function)

s = Scanner(function = my_scanner_function, argument = 'foo') Overrides: object.__init__ extit(inherited documentation)

scan(*self*, *node*, *env*, *path*=())

Overrides: SCons.Scanner.Classic.scan

Inherited from SCons.Scanner.Classic(Section 20.8)

find_include(), find_include_names(), sort_key()

Inherited from SCons.Scanner.Base(Section 20.5)

__call__(), __cmp__(), __hash__(), __str__(), add_scanner(), add_skey(), get_skeys(), path(), recurse_nodes(), select()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

24.3.2 Properties

Name	Description
Inherited from object	
__class__	

25 Module SCons.Scanner.IDL

SCons.Scanner.IDL

This module implements the depenency scanner for IDL (Interface Definition Language) files.

25.1 Functions

IDLScan()

Return a prototype Scanner instance for scanning IDL source files

25.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/IDL.py 5134 2010/08/16 23:02:40...`
__package__	**Value:** `'SCons.Scanner'`

26 Module SCons.Scanner.LaTeX

SCons.Scanner.LaTeX

This module implements the dependency scanner for LaTeX code.

26.1 Functions

modify_env_var(*env*, *var*, *abspath*)

LaTeXScanner()

Return a prototype Scanner instance for scanning LaTeX source files when built with latex.

PDFLaTeXScanner()

Return a prototype Scanner instance for scanning LaTeX source files when built with pdflatex.

26.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/LaTeX.py 5134 2010/08/16 23:02:...`
TexGraphics	**Value:** `['.eps', '.ps']`
LatexGraphics	**Value:** `['.pdf', '.png', '.jpg', '.gif', '.tif']`
__package__	**Value:** `'SCons.Scanner'`

26.3 Class FindENVPathDirs

object ─┐
SCons.Scanner.LaTeX.FindENVPathDirs

A class to bind a specific *PATH variable name to a function that will return all of the *path directories.

26.3.1 Methods

__init__(*self*, *variable*)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, *env*, *dir*=`None`, *target*=`None`, *source*=`None`, *argument*=`None`)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

26.3.2 Properties

Name	Description
Inherited from object	
__class__	

26.4 Class LaTeX

object
SCons.Scanner.Base
SCons.Scanner.LaTeX.LaTeX

```
Class for scanning LaTeX files for included files.

Unlike most scanners, which use regular expressions that just
return the included file name, this returns a tuple consisting
of the keyword for the inclusion ("include", "includegraphics",
"input", or "bibliography"), and then the file name itself.
Based on a quick look at LaTeX documentation, it seems that we
should append .tex suffix for the "include" keywords, append .tex if
there is no extension for the "input" keyword, and need to add .bib
for the "bibliography" keyword that does not accept extensions by itself.

Finally, if there is no extension for an "includegraphics" keyword
```

```
latex will append .ps or .eps to find the file, while pdftex may use .pdf,
.jpg, .tif, .mps, or .png.

The actual subset and search order may be altered by
DeclareGraphicsExtensions command. This complication is ignored.
The default order corresponds to experimentation with teTeX
    $ latex --version
    pdfeTeX 3.141592-1.21a-2.2 (Web2C 7.5.4)
    kpathsea version 3.5.4
The order is:
    ['.eps', '.ps'] for latex
    ['.png', '.pdf', '.jpg', '.tif'].

Another difference is that the search path is determined by the type
of the file being searched:
env['TEXINPUTS'] for "input" and "include" keywords
env['TEXINPUTS'] for "includegraphics" keyword
env['TEXINPUTS'] for "lstinputlisting" keyword
env['BIBINPUTS'] for "bibliography" keyword
env['BSTINPUTS'] for "bibliographystyle" keyword

FIXME: also look for the class or style in document[class|style]{}
FIXME: also look for the argument of bibliographystyle{}
```

26.4.1 Methods

__init__(*self*, *name*, *suffixes*, *graphics_extensions*, **args*, ***kw*)

Construct a new scanner object given a scanner function.

'function' - a scanner function taking two or three arguments and returning a list of strings.

'name' - a name for identifying this scanner object.

'argument' - an optional argument that, if specified, will be passed to both the scanner function and the path_function.

'skeys' - an optional list argument that can be used to determine which scanner should be used for a given Node. In the case of File nodes, for example, the 'skeys' would be file suffixes.

'path_function' - a function that takes four or five arguments (a construction environment, Node for the directory containing the SConscript file that defined the primary target, list of target nodes, list of source nodes, and optional argument for this instance) and returns a tuple of the directories that can be searched for implicit dependency files. May also return a callable() which is called with no args and returns the tuple (supporting Bindable class).

'node_class' - the class of Nodes which this scan will return. If node_class is None, then this scanner will not enforce any Node conversion and will return the raw results from the underlying scanner function.

'node_factory' - the factory function to be called to translate the raw results returned by the scanner function into the expected node_class objects.

'scan_check' - a function to be called to first check whether this node really needs to be scanned.

'recursive' - specifies that this scanner should be invoked recursively on all of the implicit dependencies it returns (the canonical example being #include lines in C source files). May be a callable, which will be called to filter the list of nodes found to select a subset for recursive scanning (the canonical example being only recursively scanning subdirectories within a directory).

The scanner function's first argument will be a Node that should be scanned for dependencies, the second argument will be an Environment object, the third argument will be the tuple of paths returned by the path_function, and the fourth argument will be the value passed into 'argument', and the returned list should contain the Nodes for all the direct dependencies of the file.

Examples:

s = Scanner(my_scanner_function)

s = Scanner(function = my_scanner_function)

s = Scanner(function = my_scanner_function, argument = 'foo') Overrides: object.__init__ extit(inherited documentation)

sort_key(*self*, *include*)

find_include(*self*, *include*, *source_dir*, *path*)

canonical_text(*self*, *text*)

Standardize an input TeX-file contents.

Currently:

- removes comments, unwrapping comment-wrapped lines.

scan(*self*, *node*)

scan_recurse(*self*, *node*, *path*=())

do a recursive scan of the top level target file This lets us search for included files based on the directory of the main file just as latex does

Inherited from SCons.Scanner.Base(Section 20.5)

__call__(), __cmp__(), __hash__(), __str__(), add_scanner(), add_skey(), get_skeys(), path(), recurse_nodes(), select()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

26.4.2 Properties

Name	Description
Inherited from object	
__class__	

26.4.3 Class Variables

Name	Description
keyword_paths	**Value:** `{'bibliography': 'BIBINPUTS', 'bibliographystyle': 'BSTIN...`

continued on next page

Name	Description
env_variables	**Value:** `['BIBINPUTS', 'TEXINPUTS', 'BSTINPUTS']`

27 Module SCons.Scanner.Prog

27.1 Functions

ProgramScanner(*******kw*)

Return a prototype Scanner instance for scanning executable files for static-lib dependencies

scan(*node*, *env*, *libpath*=`()`)

This scanner scans program files for static-library dependencies. It will search the LIBPATH environment variable for libraries specified in the LIBS variable, returning any files it finds as dependencies.

27.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/Prog.py 5134 2010/08/16 23:02:4...`
print_find_libs	**Value:** `None`
__package__	**Value:** `'SCons.Scanner'`

28 Module SCons.Scanner.RC

SCons.Scanner.RC

This module implements the depenency scanner for RC (Interface Definition Language) files.

28.1 Functions

RCScan()

Return a prototype Scanner instance for scanning RC source files

28.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Scanner/RC.py 5134 2010/08/16 23:02:40 ...`
__package__	**Value:** `'SCons.Scanner'`

29 Package SCons.Script

SCons.Script

This file implements the main() function used by the scons script.

Architecturally, this *is* the scons script, and will likely only be called from the external "scons" wrapper. Consequently, anything here should not be, or be considered, part of the build engine. If it's something that we expect other software to want to use, it should go in some other module. If it's specific to the "scons" script invocation, it goes here.

29.1 Modules

- **Interactive**: SCons interactive mode
  (Section 30, p. 251)
- **Main**: SCons.Script
  (Section 31, p. 254)
- **SConscript'**: SCons.Script.SConscript
  (Section 32, p. 268)

29.2 Functions

HelpFunction(*text*)

Variables(*files*=`[]`, *args*=`{}`)

Options(*files*=`[]`, *args*=`{}`)

29.3 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Script/__init__.py 5134 2010/08/16 23:0...`
start_time	**Value:** `1282025354.29`
call_stack	**Value:** `[]`
PathVariable	**Value:** `SCons.Variables.PathVariable`
PathOption	**Value:** `SCons.Options.PathOption`
Chmod	**Value:** `SCons.Defaults.Chmod`
Copy	**Value:** `SCons.Defaults.Copy`
Delete	**Value:** `SCons.Defaults.Delete`

continued on next page

Name	Description
Mkdir	**Value:** `SCons.Defaults.Mkdir`
Move	**Value:** `SCons.Defaults.Move`
Touch	**Value:** `SCons.Defaults.Touch`
CScanner	**Value:** `SCons.Defaults.CScan`
DScanner	**Value:** `SCons.Tool.DScanner`
DirScanner	**Value:** `SCons.Defaults.DirScanner`
ProgramScanner	**Value:** `SCons.Tool.ProgramScanner`
SourceFileScanner	**Value:** `SCons.Tool.SourceFileScanner`
CScan	**Value:** `SCons.Defaults.CScan`
ARGUMENTS	**Value:** `{}`
ARGLIST	**Value:** `[]`
BUILD_TARGETS	**Value:** `[]`
COMMAND_LINE_TAR-GETS	**Value:** `[]`
DEFAULT_TARGETS	**Value:** `[]`
help_text	**Value:** `None`
sconscript_reading	**Value:** `0`
GlobalDefaultEnvironmen-tFunctions	**Value:** `['Default', 'EnsurePythonVersion', 'EnsureSConsVersion', ...`
GlobalDefaultBuilders	**Value:** `['CFile', 'CXXFile', 'DVI', 'Jar', 'Java', 'JavaH', 'Libr...`
SConscript	**Value:** `_SCon-script.DefaultEnvironmentCall('SConscript')`
Command	**Value:** `_SCon-script.DefaultEnvironmentCall('Command', subst= 1)`
AddPostAction	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
AddPreAction	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Alias	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
AlwaysBuild	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`

continued on next page

Name	Description
BuildDir	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
CFile	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
CXXFile	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
CacheDir	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Clean	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
DVI	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Decider	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Default	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Depends	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Dir	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
EnsurePythonVersion	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
EnsureSConsVersion	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Entry	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`

continued on next page

Name	Description
Execute	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Exit	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Export	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
File	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
FindFile	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
FindInstalledFiles	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
FindSourceFiles	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Flatten	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
GetBuildPath	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
GetLaunchDir	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Glob	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Help	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Ignore	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`

continued on next page

Name	Description
Import	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Install	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
InstallAs	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Jar	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Java	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
JavaH	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Library	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Literal	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Local	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
M4	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
MSVSProject	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
NoCache	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
NoClean	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`

continued on next page

Name	Description
Object	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
PCH	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
PDF	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Package	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
ParseDepends	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
PostScript	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Precious	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Program	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
RES	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
RMIC	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Repository	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Requires	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
SConscriptChdir	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`

continued on next page

Name	Description
SConsignFile	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
SharedLibrary	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
SharedObject	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
SideEffect	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
SourceCode	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
SourceSignatures	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Split	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
StaticLibrary	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
StaticObject	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Tag	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
Tar	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
TargetSignatures	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`
TypeLibrary	**Value:** `<SCons.Script.SConscript.DefaultEnvironmentCall object at...`

continued on next page

Name	Description
Value	**Value:** <SCons.Script.SConscript.DefaultEnvironmentCall object at...
VariantDir	**Value:** <SCons.Script.SConscript.DefaultEnvironmentCall object at...
Zip	**Value:** <SCons.Script.SConscript.DefaultEnvironmentCall object at...
__package__	**Value:** 'SCons.Script'

29.4 Class TargetList

object

_abcoll.Sized

object

_abcoll.Iterable

object

_abcoll.Container

_abcoll.Sequence

_abcoll.MutableSequence

UserList.UserList

SCons.Script.TargetList

29.4.1 Methods

Inherited from UserList.UserList

__add__(), __cmp__(), __contains__(), __delitem__(), __delslice__(), __eq__(), __ge__(), __getitem__(), __getslice__(), __gt__(), __iadd__(), __imul__(), __init__(), __le__(), __len__(), __lt__(), __mul__(), __ne__(), __radd__(), __repr__(), __rmul__(), __setitem__(), __setslice__(), append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

Inherited from _abcoll.Sequence

__iter__(), __reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__()

29.4.2 Properties

Name	Description
Inherited from object	
__class__	

29.4.3 Class Variables

Name	Description
Inherited from UserList.UserList	
__abstractmethods__, __hash__	

30 Module SCons.Script.Interactive

SCons interactive mode

30.1 Functions

interact(*fs*, *parser*, *options*, *targets*, *target_top*)

30.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Script/Interactive.py 5134 2010/08/16 2...`
__doc__	**Value:** ...
__package__	**Value:** `'SCons.Script'`

30.3 Class SConsInteractiveCmd

cmd.Cmd

SCons.Script.Interactive.SConsInteractiveCmd

```
build [TARGETS]         Build the specified TARGETS and their dependencies.
                        'b' is a synonym.
clean [TARGETS]         Clean (remove) the specified TARGETS and their
                        dependencies.  'c' is a synonym.
exit                    Exit SCons interactive mode.
help [COMMAND]          Prints help for the specified COMMAND.  'h' and
                        '?' are synonyms.
shell [COMMANDLINE]     Execute COMMANDLINE in a subshell.  'sh' and '!'
                        are synonyms.
version                 Prints SCons version information.
```

30.3.1 Methods

__init__(*self*, ***kw*)

Instantiate a line-oriented interpreter framework.

The optional argument 'completekey' is the readline name of a completion key; it defaults to the Tab key. If completekey is not None and the readline module is available, command completion is done automatically. The optional arguments stdin and stdout specify alternate input and output file objects; if not specified, sys.stdin and sys.stdout are used. Overrides: cmd.Cmd.__init__ extit(inherited documentation)

default(*self*, *argv*)

Called on an input line when the command prefix is not recognized.

If this method is not overridden, it prints an error message and returns. Overrides: cmd.Cmd.default extit(inherited documentation)

onecmd(*self*, *line*)

Interpret the argument as though it had been typed in response to the prompt.

This may be overridden, but should not normally need to be; see the precmd() and postcmd() methods for useful execution hooks. The return value is a flag indicating whether interpretation of commands by the interpreter should stop. Overrides: cmd.Cmd.onecmd extit(inherited documentation)

do_build(*self*, *argv*)

build [TARGETS] Build the specified TARGETS and their dependencies. 'b' is a synonym.

do_clean(*self*, *argv*)

clean [TARGETS] Clean (remove) the specified TARGETS and their dependencies. 'c' is a synonym.

do_EOF(*self*, *argv*)

do_exit(*self*, *argv*)

exit Exit SCons interactive mode.

do_help(*self*, *argv*)

help [COMMAND] Prints help for the specified COMMAND. 'h' and '?' are synonyms. Overrides: cmd.Cmd.do_help

do_shell(*self*, *argv*)

shell [COMMANDLINE] Execute COMMANDLINE in a subshell. 'sh' and '!' are synonyms.

do_version(*self*, *argv*)

version Prints SCons version information.

Inherited from cmd.Cmd

cmdloop(), columnize(), complete(), complete_help(), completedefault(), completenames(), emptyline(), get_names(), parseline(), postcmd(), postloop(), precmd(), preloop(), print_topics()

30.3.2 Class Variables

Name	Description
synonyms	**Value:** `{'b': 'build', 'c': 'clean', 'h': 'help', 'scons': 'build...`
Inherited from cmd.Cmd doc_header, doc_leader, identchars, intro, lastcmd, misc_header, nohelp, prompt, ruler, undoc_header, use_rawinput	

31 Module SCons.Script.Main

SCons.Script

This file implements the main() function used by the scons script.

Architecturally, this *is* the scons script, and will likely only be called from the external "scons" wrapper. Consequently, anything here should not be, or be considered, part of the build engine. If it's something that we expect other software to want to use, it should go in some other module. If it's specific to the "scons" script invocation, it goes here.

31.1 Functions

fetch_win32_parallel_msg()

Progress(**args*, ***kw*)

GetBuildFailures()

python_version_string()

python_version_unsupported(*version*=`(2, 6, 4, 'final', 0)`)

python_version_deprecated(*version*=`(2, 6, 4, 'final', 0)`)

AddOption(**args*, ***kw*)

GetOption(*name*)

SetOption(*name*, *value*)

find_deepest_user_frame(*tb*)

Find the deepest stack frame that is not part of SCons.

Input is a "pre-processed" stack trace in the form returned by traceback.extract_tb() or traceback.extract_stack()

version_string(*label*, *module*)

main()

31.2 Variables

Name	Description
unsupported_python_version	**Value:** `(2, 3, 0)`
deprecated_python_version	**Value:** `(2, 4, 0)`
__revision__	**Value:** `'src/engine/SCons/Script/Main.py 5134 2010/08/16 23:02:40...`
display	**Value:** `DisplayEngine()`
progress_display	**Value:** `SCons.Util.DisplayEngine()`
first_command_start	**Value:** `None`
last_command_end	**Value:** `None`
ProgressObject	**Value:** `Null(0x02100850)`
print_objects	**Value:** `0`
print_memoizer	**Value:** `0`
print_stacktrace	**Value:** `0`
print_time	**Value:** `0`
sconscript_time	**Value:** `0`
cumulative_command_time	**Value:** `0`
exit_status	**Value:** `0`
this_build_status	**Value:** `0`
num_jobs	**Value:** `None`
delayed_warnings	**Value:** `[]`
OptionsParser	**Value:** `FakeOptionParser()`
count_stats	**Value:** `CountStats()`
memory_stats	**Value:** `MemStats()`
__package__	**Value:** `'SCons.Script'`

31.3 Class SConsPrintHelpException

object
exceptions.BaseException
exceptions.Exception
SCons.Script.Main.SConsPrintHelpException

31.3.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

31.3.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

31.4 Class Progressor

object
SCons.Script.Main.Progressor

31.4.1 Methods

__init__(*self*, *obj*, *interval*=`1`, *file*=`None`, *overwrite*=`False`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

write(*self*, *s*)

erase_previous(*self*)

spinner(*self*, *node*)

string(*self*, *node*)

replace_string(*self*, *node*)

__call__(*self*, *node*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.4.2 Properties

Name	Description
Inherited from object	
__class__	

31.4.3 Class Variables

Name	Description
prev	**Value:** `''`
count	**Value:** `0`
target_string	**Value:** `'$TARGET'`

31.5 Class BuildTask

object

SCons.Taskmaster.Task

SCons.Taskmaster.OutOfDateTask

SCons.Script.Main.BuildTask

An SCons build task.

31.5.1 Methods

display(*self*, *message*)

Hook to allow the calling interface to display a message.

This hook gets called as part of preparing a task for execution (that is, a Node to be built). As part of figuring out what Node should be built next, the actually target list may be altered, along with a message describing the alteration. The calling interface can subclass Task and provide a concrete implementation of this method to see those messages. Overrides: SCons.Taskmaster.Task.display extit(inherited documentation)

prepare(*self*)

Called just before the task is executed.

This is mainly intended to give the target Nodes a chance to unlink underlying files and make all necessary directories before the Action is actually called to build the targets. Overrides: SCons.Taskmaster.Task.prepare extit(inherited documentation)

needs_execute(*self*)

Returns True (indicating this Task should be executed) if this Task's target state indicates it needs executing, which has already been determined by an earlier up-to-date check. Overrides: SCons.Taskmaster.Task.needs_execute

execute(*self*)

Called to execute the task.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in prepare(), executed() or failed(). Overrides: SCons.Taskmaster.Task.execute extit(inherited documentation)

do_failed(*self*, *status*=2)

executed(*self*)

Called when the task has been successfully executed and the Taskmaster instance wants to call the Node's callback methods.

This may have been a do-nothing operation (to preserve build order), so we must check the node's state before deciding whether it was "built", in which case we call the appropriate Node method. In any event, we always call "visited()", which will handle any post-visit actions that must take place regardless of whether or not the target was an actual built target or a source Node. Overrides: SCons.Taskmaster.Task.executed extit(inherited documentation)

failed(*self*)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure(). Overrides: SCons.Taskmaster.Task.failed extit(inherited documentation)

postprocess(*self*)

Post-processes a task after it's been executed.

This examines all the targets just built (or not, we don't care if the build was successful, or even if there was no build because everything was up-to-date) to see if they have any waiting parent Nodes, or Nodes waiting on a common side effect, that can be put back on the candidates list. Overrides: SCons.Taskmaster.Task.postprocess extit(inherited documentation)

make_ready(*self*)

Make a task ready for execution Overrides: SCons.Taskmaster.Task.make_ready

Inherited from SCons.Taskmaster.Task(Section 35.4)

__init__(), exc_clear(), exc_info(), exception_set(), executed_with_callbacks(), executed_without_callbacks(), fail_continue(), fail_stop(), get_target(), make_ready_all(), make_ready_current(), trace_message()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.5.2 Properties

Name	Description
Inherited from object	
__class__	

31.5.3 Class Variables

Name	Description
progress	**Value:** `Null(0x02100850)`

31.6 Class CleanTask

object
SCons.Taskmaster.Task
SCons.Taskmaster.AlwaysTask
SCons.Script.Main.CleanTask

An SCons clean task.

31.6.1 Methods

fs_delete(*self*, *path*, *pathstr*, *remove*=1)

show(*self*)

remove(*self*)

Called to execute the task.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in prepare(), executed() or failed().

execute(*self*)

Called to execute the task.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in prepare(), executed() or failed(). Overrides: SCons.Taskmaster.Task.execute extit(inherited documentation)

executed(*self*)

Called when the task has been successfully executed and the Taskmaster instance doesn't want to call the Node's callback methods. Overrides: SCons.Taskmaster.Task.executed

make_ready(*self*)

Marks all targets in a task ready for execution.

This is used when the interface needs every target Node to be visited--the canonical example being the "scons -c" option. Overrides: SCons.Taskmaster.Task.make_ready

prepare(*self*)

Called just before the task is executed.

This is mainly intended to give the target Nodes a chance to unlink underlying files and make all necessary directories before the Action is actually called to build the targets. Overrides: SCons.Taskmaster.Task.prepare extit(inherited documentation)

Inherited from SCons.Taskmaster.AlwaysTask(Section 35.5)

needs_execute()

Inherited from SCons.Taskmaster.Task(Section 35.4)

__init__(), display(), exc_clear(), exc_info(), exception_set(), executed_with_callbacks(), executed_without_callbacks(), fail_continue(), fail_stop(), failed(), get_target(), make_ready_all(), make_ready_current(), postprocess(), trace_message()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.6.2 Properties

Name	Description
Inherited from object	
__class__	

31.7 Class QuestionTask

object
SCons.Taskmaster.Task
SCons.Taskmaster.AlwaysTask
SCons.Script.Main.QuestionTask

An SCons task for the -q (question) option.

31.7.1 Methods

prepare(*self*)

Called just before the task is executed.

This is mainly intended to give the target Nodes a chance to unlink underlying files and make all necessary directories before the Action is actually called to build the targets. Overrides: SCons.Taskmaster.Task.prepare extit(inherited documentation)

execute(*self*)

Called to execute the task.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in prepare(), executed() or failed(). Overrides: SCons.Taskmaster.Task.execute extit(inherited documentation)

executed(*self*)

Called when the task has been successfully executed and the Taskmaster instance wants to call the Node's callback methods.

This may have been a do-nothing operation (to preserve build order), so we must check the node's state before deciding whether it was "built", in which case we call the appropriate Node method. In any event, we always call "visited()", which will handle any post-visit actions that must take place regardless of whether or not the target was an actual built target or a source Node. Overrides: SCons.Taskmaster.Task.executed extit(inherited documentation)

Inherited from SCons.Taskmaster.AlwaysTask(Section 35.5)

needs_execute()

Inherited from SCons.Taskmaster.Task(Section 35.4)

__init__(), display(), exc_clear(), exc_info(), exception_set(), executed_with_callbacks(), executed_without_callbacks(), fail_continue(), fail_stop(), failed(), get_target(), make_ready(), make_ready_all(), make_ready_current(), postprocess(), trace_message()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.7.2 Properties

Name	Description
Inherited from object	
__class__	

31.8 Class TreePrinter

object
└ **SCons.Script.Main.TreePrinter**

31.8.1 Methods

__init__(*self*, *derived*=`False`, *prune*=`False`, *status*=`False`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

get_all_children(*self*, *node*)

get_derived_children(*self*, *node*)

display(*self*, *t*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.8.2 Properties

Name	Description
Inherited from object	
__class__	

31.9 Class FakeOptionParser

object ─┐
SCons.Script.Main.FakeOptionParser

A do-nothing option parser, used for the initial OptionsParser variable.

During normal SCons operation, the OptionsParser is created right away by the main() function. Certain tests scripts however, can introspect on different Tool modules, the initialization of which can try to add a new, local option to an otherwise uninitialized OptionsParser object. This allows that introspection to happen without blowing up.

31.9.1 Methods

add_local_option(*self*, **args*, ***kw*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.9.2 Properties

Name	Description
Inherited from object	
__class__	

31.9.3 Class Variables

Name	Description
values	**Value:** `FakeOptionValues()`

31.10 Class Stats

object ─┐
SCons.Script.Main.Stats

Known Subclasses: SCons.Script.Main.CountStats, SCons.Script.Main.MemStats

31.10.1 Methods

__init__(*self*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

enable(*self*, *outfp*)

do_nothing(*self*, **args*, ***kw*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.10.2 Properties

Name	Description
Inherited from object	
__class__	

31.11 Class CountStats

object
SCons.Script.Main.Stats
SCons.Script.Main.CountStats

31.11.1 Methods

do_append(*self*, *label*)

do_print(*self*)

Inherited from SCons.Script.Main.Stats(Section 31.10)

__init__(), do_nothing(), enable()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.11.2 Properties

Name	Description
Inherited from object	
__class__	

31.12 Class MemStats

object
SCons.Script.Main.Stats
SCons.Script.Main.MemStats

31.12.1 Methods

do_append(*self*, *label*)

do_print(*self*)

Inherited from SCons.Script.Main.Stats(Section 31.10)

__init__(), do_nothing(), enable()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

31.12.2 Properties

Name	Description
Inherited from object	
__class__	

32 Module SCons.Script.SConscript'

SCons.Script.SConscript

This module defines the Python API provided to SConscript and SConstruct files.

32.1 Functions

get_calling_namespaces()

Return the locals and globals for the function that called into this module in the current call stack.

compute_exports(*exports*)

Compute a dictionary of exports given one of the parameters to the Export() function or the exports argument to SConscript().

Return(**vars*, ***kw*)

SConscript_exception(*file*=`sys.stderr`)

Print an exception stack trace just for the SConscript file(s). This will show users who have Python errors where the problem is, without cluttering the output with all of the internal calls leading up to where we exec the SConscript.

annotate(*node*)

Annotate a node with the stack frame describing the SConscript file and line number that created it.

Configure(**args*, ***kw*)

get_DefaultEnvironmentProxy()

BuildDefaultGlobals()

Create a dictionary containing all the default globals for SConstruct and SConscript files.

32.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Script/SConscript.py 5134 2010/08/16 23...`
launch_dir	**Value:** `'/home/bdeegan/scons/checkpoints/2.0'`
GlobalDict	**Value:** `None`
global_exports	**Value:** `{}`
sconscript_chdir	**Value:** `1`
call_stack	**Value:** `[]`
stack_bottom	**Value:** `'% Stack boTTom %'`
__package__	**Value:** `'SCons.Script'`

32.3 Class SConscriptReturn

object
exceptions.BaseException
exceptions.Exception
SCons.Script.SConscript'.SConscriptReturn

32.3.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

32.3.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

32.4 Class Frame

object ┐
SCons.Script.SConscript'.Frame

A frame on the SConstruct/SConscript call stack

32.4.1 Methods

__init__(*self*, *fs*, *exports*, *sconscript*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

32.4.2 Properties

Name	Description
Inherited from object	
__class__	

32.5 Class SConsEnvironment

object
SCons.Environment.SubstitutionEnvironment
SCons.Environment.Base
SCons.Script.SConscript'.SConsEnvironment

An Environment subclass that contains all of the methods that are particular to the wrapper SCons interface and which aren't (or shouldn't be) part of the build engine itself.

Note that not all of the methods of this class have corresponding global functions, there are some private methods.

32.5.1 Methods

Configure(*self*, **args*, ***kw*)

Overrides: SCons.Environment.Base.Configure

Default(*self*, **targets*)

EnsureSConsVersion(*self*, *major*, *minor*, *revision*=0)

Exit abnormally if the SCons version is not late enough.

EnsurePythonVersion(*self*, *major*, *minor*)

Exit abnormally if the Python version is not late enough.

Exit(*self*, *value*=0)

Export(*self*, **vars*, ***kw*)

GetLaunchDir(*self*)

GetOption(*self*, *name*)

Help(*self*, *text*)

Import(*self*, **vars*)

SConscript(*self*, **ls*, ***kw*)

SConscriptChdir(*self*, *flag*)

SetOption(*self*, *name*, *value*)

Inherited from SCons.Environment.Base(Section 8.9)

Action(), AddPostAction(), AddPreAction(), Alias(), AlwaysBuild(), Append(), AppendENVPath(), AppendUnique(), BuildDir(), Builder(), CacheDir(), Clean(), Clone(), Command(), Copy(), Decider(), Depends(), Detect(), Dictionary(), Dir(), Dump(), Entry(), Environment(), Execute(), File(), FindFile(), FindInstalledFiles(), FindIxes(), FindSourceFiles(), Flatten(), GetBuildPath(), Glob(), Ignore(), Literal(), Local(), NoCache(), NoClean(), ParseConfig(), ParseDepends(), Platform(), Precious(), Prepend(), PrependENVPath(), PrependUnique(), Replace(), ReplaceIxes(), Repository(), Requires(), SConsignFile(), Scanner(), SetDefault(), SideEffect(), SourceCode(), SourceSignatures(), Split(), TargetSignatures(), Tool(), Value(), VariantDir(), WhereIs(), __init__(), get_CacheDir(), get_builder(), get_factory(), get_scanner(), get_src_sig_type(), get_tgt_sig_type(), scanner_map_delete()

Inherited from SCons.Environment.SubstitutionEnvironment(Section 8.6)

AddMethod(), MergeFlags(), Override(), ParseFlags(), RemoveMethod(), __cmp__(), __contains__(), __delitem__(), __getitem__(), __setitem__(), arg2nodes(), backtick(), get(), gvars(), has_key(), items(), lvars(), subst(), subst_kw(), subst_list(), subst_path(), subst_target_source()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

32.5.2 Properties

Name	Description
Inherited from object	
__class__	

32.5.3 Class Variables

Name	Description
Inherited from SCons.Environment.Base (Section 8.9)	
memoizer_counters	
Inherited from SCons.Environment.SubstitutionEnvironment (Section 8.6)	
__metaclass__	

32.6 Class DefaultEnvironmentCall

object
└ **SCons.Script.SConscript'.DefaultEnvironmentCall**

A class that implements "global function" calls of Environment methods by fetching the specified method from the DefaultEnvironment's class. Note that this uses an intermediate proxy class instead of calling the DefaultEnvironment method directly so that the proxy can override the subst() method and thereby prevent expansion of construction variables (since from the user's point of view this was called as a global function, with no associated construction environment).

32.6.1 Methods

__init__(*self*, *method_name*, *subst*=0)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, **args*, ***kw*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

32.6.2 Properties

Name	Description
Inherited from object	
__class__	

33 Module SCons.Sig

Place-holder for the old SCons.Sig module hierarchy

This is no longer used, but code out there (such as the NSIS module on the SCons wiki) may try to import SCons.Sig. If so, we generate a warning that points them to the line that caused the import, and don't die.

If someone actually tried to use the sub-modules or functions within the package (for example, SCons.Sig.MD5.signature()), then they'll still get an AttributeError, but at least they'll know where to start looking.

33.1 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Sig.py 5134 2010/08/16 23:02:40 bdeegan'`
__doc__	**Value:** `"""Place-holder for the old SCons.Sig module hierar...`
msg	**Value:** `'The SCons.Sig module no longer exists.\n Remove the f...`
default_calc	**Value:** `None`
default_module	**Value:** `None`
MD5	**Value:** `MD5Null()`
TimeStamp	**Value:** `TimeStampNull()`
__package__	**Value:** `'SCons'`

33.2 Class MD5Null

object ─┐
SCons.Util.Null ─┐
SCons.Sig.MD5Null

33.2.1 Methods

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

Inherited from SCons.Util.Null(Section 36.15)

__call__(), __delattr__(), __getattr__(), __init__(), __new__(), __nonzero__(), __setattr__()

Inherited from object

__format__(), __getattribute__(), __hash__(), __reduce__(), __reduce_ex__(), __sizeof__(), __str__(), __subclasshook__()

33.2.2 Properties

Name	Description
Inherited from object	
__class__	

33.3 Class TimeStampNull

object
SCons.Util.Null
SCons.Sig.TimeStampNull

33.3.1 Methods

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

Inherited from SCons.Util.Null(Section 36.15)

__call__(), __delattr__(), __getattr__(), __init__(), __new__(), __nonzero__(), __setattr__()

Inherited from object

__format__(), __getattribute__(), __hash__(), __reduce__(), __reduce_ex__(), __sizeof__(), __str__(), __subclasshook__()

33.3.2 Properties

Name	Description
Inherited from object	
__class__	

34 Module SCons.Subst

SCons.Subst

SCons string substitution.

34.1 Functions

SetAllowableExceptions(**excepts*)

raise_exception(*exception*, *target*, *s*)

quote_spaces(*arg*)

Generic function for putting double quotes around any string that has white space in it.

escape_list(*mylist*, *escape_func*)

Escape a list of arguments by running the specified escape_func on every object in the list that has an escape() method.

subst_dict(*target*, *source*)

Create a dictionary for substitution of special construction variables.

This translates the following special arguments:

target - the target (object or array of objects), used to generate the TARGET and TARGETS construction variables

source - the source (object or array of objects), used to generate the SOURCES and SOURCE construction variables

scons_subst(*strSubst*, *env*, *mode*=`1`, *target*=`None`, *source*=`None`, *gvars*=`{}`, *lvars*=`{}`, *conv*=`None`)

Expand a string or list containing construction variable substitutions.

This is the work-horse function for substitutions in file names and the like. The companion scons_subst_list() function (below) handles separating command lines into lists of arguments, so see that function if that's what you're looking for.

scons_subst_list(*strSubst*, *env*, *mode*=`1`, *target*=`None`, *source*=`None`, *gvars*=`{}`, *lvars*=`{}`, *conv*=`None`)

Substitute construction variables in a string (or list or other object) and separate the arguments into a command list.

The companion scons_subst() function (above) handles basic substitutions within strings, so see that function instead if that's what you're looking for.

scons_subst_once(*strSubst*, *env*, *key*)

Perform single (non-recursive) substitution of a single construction variable keyword.

This is used when setting a variable when copying or overriding values in an Environment. We want to capture (expand) the old value before we override it, so people can do things like:

env2 = env.Clone(CCFLAGS = '$CCFLAGS -g')

We do this with some straightforward, brute-force code here...

34.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Subst.py 5134 2010/08/16 23:02:40 bdeegan'`
AllowableExceptions	**Value:** `(<type 'exceptions.IndexError'>, <type 'exceptions.NameEr...`
NullNodesList	**Value:** `Null(0x01326590)`

continued on next page

Name	Description
SUBST_CMD	**Value:** `0`
SUBST_RAW	**Value:** `1`
SUBST_SIG	**Value:** `2`
__package__	**Value:** `'SCons'`

34.3 Class Literal

object ┐
SCons.Subst.Literal

A wrapper for a string. If you use this object wrapped around a string, then it will be interpreted as literal. When passed to the command interpreter, all special characters will be escaped.

34.3.1 Methods

__init__(*self*, *lstr*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

escape(*self*, *escape_func*)

for_signature(*self*)

is_literal(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

34.3.2 Properties

Name	Description
Inherited from object	
__class__	

34.4 Class SpecialAttrWrapper

object
└ **SCons.Subst.SpecialAttrWrapper**

This is a wrapper for what we call a 'Node special attribute.' This is any of the attributes of a Node that we can reference from Environment variable substitution, such as $TARGET.abspath or $SOURCES[1].filebase. We implement the same methods as Literal so we can handle special characters, plus a for_signature method, such that we can return some canonical string during signature calculation to avoid unnecessary rebuilds.

34.4.1 Methods

__init__(*self*, *lstr*, *for_signature*=`None`)

The for_signature parameter, if supplied, will be the canonical string we return from for_signature(). Else we will simply return lstr. Overrides: object.__init__

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

escape(*self*, *escape_func*)

for_signature(*self*)

is_literal(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __subclasshook__()

34.4.2 Properties

Name	Description
Inherited from object	
__class__	

34.5 Class CmdStringHolder

object

_abcoll.Sized

object

_abcoll.Iterable

object

_abcoll.Container

_abcoll.Sequence

UserString.UserString

SCons.Subst.CmdStringHolder

This is a special class used to hold strings generated by scons_subst() and scons_subst_list(). It defines a special method escape(). When passed a function with an escape algorithm for a particular platform, it will return the contained string with the proper escape sequences inserted.

34.5.1 Methods

__init__(*self*, *cmd*, *literal*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

is_literal(*self*)

escape(*self*, *escape_func*, *quote_func*=`<function quote_spaces at 0x1328398>`)

Escape the string with the supplied function. The function is expected to take an arbitrary string, then return it with all special characters escaped and ready for passing to the command interpreter.

After calling this function, the next call to str() will return the escaped string.

Inherited from UserString.UserString

__add__(), __cmp__(), __complex__(), __contains__(), __float__(), __getitem__(), __getslice__(), __hash__(), __int__(), __len__(), __long__(), __mod__(), __mul__(), __radd__(), __repr__(), __rmul__(), __str__(), capitalize(), center(), count(), decode(), encode(), endswith(), expandtabs(), find(), index(), isalnum(), isalpha(), isdecimal(), isdigit(), islower(), isnumeric(), isspace(), istitle(), isupper(), join(), ljust(), lower(), lstrip(), partition(), replace(), rfind(), rindex(), rjust(), rpartition(), rsplit(), rstrip(), split(), splitlines(), startswith(), strip(), swapcase(), title(), translate(), upper(), zfill()

Inherited from _abcoll.Sequence

__iter__(), __reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__()

34.5.2 Properties

Name	Description
Inherited from object	
__class__	

34.5.3 Class Variables

Name	Description
Inherited from UserString.UserString	
__abstractmethods__	

34.6 Class NLWrapper

object ┐
SCons.Subst.NLWrapper

A wrapper class that delays turning a list of sources or targets into a NodeList until it's needed. The specified function supplied when the object is initialized is responsible for turning raw nodes into proxies that implement the special attributes like .abspath, .source, etc. This way, we avoid creating those proxies just "in case" someone is going to use $TARGET or the like, and only go through the trouble if we really have to.

In practice, this might be a wash performance-wise, but it's a little cleaner conceptually...

34.6.1 Methods

__init__(*self*, *list*, *func*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

34.6.2 Properties

Name	Description
Inherited from object	
__class__	

34.7 Class Targets_or_Sources

```
object
  _abcoll.Sized
object
  _abcoll.Iterable
object
  _abcoll.Container
    _abcoll.Sequence
      _abcoll.MutableSequence
        UserList.UserList
          SCons.Subst.Targets_or_Sources
```

A class that implements $TARGETS or $SOURCES expansions by in turn wrapping a NLWrapper. This class handles the different methods used to access the list, calling the NLWrapper to create proxies on demand.

Note that we subclass collections.UserList purely so that the is_Sequence() function will identify an object of this class as a list during variable expansion. We're not really using any collections.UserList methods in practice.

34.7.1 Methods

__init__(*self*, *nl*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__getattr__(*self*, *attr*)

__getitem__(*self*, *i*)

Overrides: _abcoll.Sequence.__getitem__

__getslice__(*self*, *i*, *j*)

Overrides: UserList.UserList.__getslice__

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

Inherited from UserList.UserList

__add__(), __cmp__(), __contains__(), __delitem__(), __delslice__(), __eq__(), __ge__(), __gt__(), __iadd__(), __imul__(), __le__(), __len__(), __lt__(), __mul__(), __ne__(), __radd__(), __rmul__(), __setitem__(), __setslice__(), append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

Inherited from _abcoll.Sequence

__iter__(), __reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__()

34.7.2 Properties

Name	Description
Inherited from object	
__class__	

34.7.3 Class Variables

Name	Description
Inherited from UserList.UserList	
__abstractmethods__, __hash__	

34.8 Class Target_or_Source

object ─┐
SCons.Subst.Target_or_Source

A class that implements $TARGET or $SOURCE expansions by in turn wrapping a NLWrapper. This class handles the different methods used to access an individual proxy Node, calling the NLWrapper to create a proxy on demand.

34.8.1 Methods

__init__(*self*, *nl*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__getattr__(*self*, *attr*)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __subclasshook__()

34.8.2 Properties

Name	Description
Inherited from object	
__class__	

34.9 Class NullNodeList

object
SCons.Util.Null
SCons.Util.NullSeq
SCons.Subst.NullNodeList

34.9.1 Methods

__call__(*self*, **args*, ***kwargs*)

Overrides: SCons.Util.Null.__call__

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from SCons.Util.NullSeq(Section 36.16)

__delitem__(), __getitem__(), __iter__(), __len__(), __setitem__()

Inherited from SCons.Util.Null(Section 36.15)

__delattr__(), __getattr__(), __init__(), __new__(), __nonzero__(), __repr__(), __setattr__()

Inherited from object

__format__(), __getattribute__(), __hash__(), __reduce__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

34.9.2 Properties

Name	Description
Inherited from object	
__class__	

35 Module SCons.Taskmaster

Generic Taskmaster module for the SCons build engine.

This module contains the primary interface(s) between a wrapping user interface and the SCons build engine. There are two key classes here:

Taskmaster This is the main engine for walking the dependency graph and calling things to decide what does or doesn't need to be built.

Task This is the base class for allowing a wrapping interface to decide what does or doesn't actually need to be done. The intention is for a wrapping interface to subclass this as appropriate for different types of behavior it may need.

The canonical example is the SCons native Python interface, which has Task subclasses that handle its specific behavior, like printing "'foo' is up to date" when a top-level target doesn't need to be built, and handling the -c option by removing targets as its "build" action. There is also a separate subclass for suppressing this output when the -q option is used.

The Taskmaster instantiates a Task object for each (set of) target(s) that it decides need to be evaluated and/or built.

35.1 Functions

dump_stats()

find_cycle(*stack*, *visited*)

35.2 Variables

Name	Description
__doc__	**Value:** `...`
__revision__	**Value:** `'src/engine/SCons/Taskmaster.py 5134 2010/08/16 23:02:40 ...`
StateString	**Value:** `{0: 'no_state', 1: 'pending', 2: 'executing', 3: 'up_to_d...`
NODE_NO_STATE	**Value:** `0`
NODE_PENDING	**Value:** `1`
NODE_EXECUTING	**Value:** `2`
NODE_UP_TO_DATE	**Value:** `3`
NODE_EXECUTED	**Value:** `4`

continued on next page

Name	Description
NODE_FAILED	**Value:** `5`
CollectStats	**Value:** `None`
StatsNodes	**Value:** `[]`
fmt	**Value:** `'%(considered)3d %(already_handled)3d %(problem)3d %(chil...`
__package__	**Value:** `'SCons'`

35.3 Class Stats

```
object ─┐
        SCons.Taskmaster.Stats
```

A simple class for holding statistics about the disposition of a Node by the Taskmaster. If we're collecting statistics, each Node processed by the Taskmaster gets one of these attached, in which case the Taskmaster records its decision each time it processes the Node. (Ideally, that's just once per Node.)

35.3.1 Methods

__init__(*self*)

Instantiates a Taskmaster.Stats object, initializing all appropriate counters to zero. Overrides: object.__init__

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

35.3.2 Properties

Name	Description
Inherited from object	
__class__	

35.4 Class Task

object ┐
    SCons.Taskmaster.Task

Known Subclasses: SCons.Taskmaster.AlwaysTask, SCons.Taskmaster.OutOfDateTask

Default SCons build engine task.

This controls the interaction of the actual building of node and the rest of the engine.

This is expected to handle all of the normally-customizable aspects of controlling a build, so any given application *should* be able to do what it wants by sub-classing this class and overriding methods as appropriate. If an application needs to customze something by sub-classing Taskmaster (or some other build engine class), we should first try to migrate that functionality into this class.

Note that it's generally a good idea for sub-classes to call these methods explicitly to update state, etc., rather than roll their own interaction with Taskmaster from scratch.

35.4.1 Methods

__init__(*self*, *tm*, *targets*, *top*, *node*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

trace_message(*self*, *method*, *node*, *description*=`'node'`)

display(*self*, *message*)

Hook to allow the calling interface to display a message.

This hook gets called as part of preparing a task for execution (that is, a Node to be built). As part of figuring out what Node should be built next, the actually target list may be altered, along with a message describing the alteration. The calling interface can subclass Task and provide a concrete implementation of this method to see those messages.

prepare(*self*)

Called just before the task is executed.

This is mainly intended to give the target Nodes a chance to unlink underlying files and make all necessary directories before the Action is actually called to build the targets.

get_target(*self*)

Fetch the target being built or updated by this task.

needs_execute(*self*)

execute(*self*)

Called to execute the task.

This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in prepare(), executed() or failed().

executed_without_callbacks(*self*)

Called when the task has been successfully executed and the Taskmaster instance doesn't want to call the Node's callback methods.

executed_with_callbacks(*self*)

Called when the task has been successfully executed and the Taskmaster instance wants to call the Node's callback methods.

This may have been a do-nothing operation (to preserve build order), so we must check the node's state before deciding whether it was "built", in which case we call the appropriate Node method. In any event, we always call "visited()", which will handle any post-visit actions that must take place regardless of whether or not the target was an actual built target or a source Node.

executed(*self*)

Called when the task has been successfully executed and the Taskmaster instance wants to call the Node's callback methods.

This may have been a do-nothing operation (to preserve build order), so we must check the node's state before deciding whether it was "built", in which case we call the appropriate Node method. In any event, we always call "visited()", which will handle any post-visit actions that must take place regardless of whether or not the target was an actual built target or a source Node.

failed(*self*)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

fail_stop(*self*)

Explicit stop-the-build failure.

This sets failure status on the target nodes and all of their dependent parent nodes.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

fail_continue(*self*)

Explicit continue-the-build failure.

This sets failure status on the target nodes and all of their dependent parent nodes.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

make_ready_all(*self*)

Marks all targets in a task ready for execution.

This is used when the interface needs every target Node to be visited--the canonical example being the "scons -c" option.

make_ready_current(*self*)

Marks all targets in a task ready for execution if any target is not current.

This is the default behavior for building only what's necessary.

make_ready(*self*)

Marks all targets in a task ready for execution if any target is not current.

This is the default behavior for building only what's necessary.

postprocess(*self*)

Post-processes a task after it's been executed.

This examines all the targets just built (or not, we don't care if the build was successful, or even if there was no build because everything was up-to-date) to see if they have any waiting parent Nodes, or Nodes waiting on a common side effect, that can be put back on the candidates list.

exc_info(*self*)

Returns info about a recorded exception.

exc_clear(*self*)

Clears any recorded exception.

This also changes the "exception_raise" attribute to point to the appropriate do-nothing method.

exception_set(*self*, *exception*=`None`)

Records an exception to be raised at the appropriate time.

This also changes the "exception_raise" attribute to point to the method that will, in fact

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

35.4.2 Properties

Name	Description
Inherited from object	
__class__	

35.5 Class AlwaysTask

object
SCons.Taskmaster.Task
SCons.Taskmaster.AlwaysTask

Known Subclasses: SCons.SConf.SConfBuildTask, SCons.Script.Main.CleanTask, SCons.Script.Main.QuestionTask

35.5.1 Methods

needs_execute(*self*)

Always returns True (indicating this Task should always be executed).

Subclasses that need this behavior (as opposed to the default of only executing Nodes that are out of date w.r.t. their dependencies) can use this as follows:

class MyTaskSubclass(SCons.Taskmaster.Task):
needs_execute = SCons.Taskmaster.Task.execute_always

Overrides: SCons.Taskmaster.Task.needs_execute

Inherited from SCons.Taskmaster.Task(Section 35.4)

__init__(), display(), exc_clear(), exc_info(), exception_set(), execute(), executed(), executed_with_callbacks(), executed_without_callbacks(), fail_continue(), fail_stop(), failed(), get_target(), make_ready(), make_ready_all(), make_ready_current(), postprocess(), prepare(), trace_message()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

35.5.2 Properties

Name	Description
Inherited from object	
__class__	

35.6 Class OutOfDateTask

object
SCons.Taskmaster.Task
SCons.Taskmaster.OutOfDateTask

Known Subclasses: SCons.Script.Main.BuildTask

35.6.1 Methods

needs_execute(*self*)

Returns True (indicating this Task should be executed) if this Task's target state indicates it needs executing, which has already been determined by an earlier up-to-date check. Overrides: SCons.Taskmaster.Task.needs_execute

Inherited from SCons.Taskmaster.Task(Section 35.4)

__init__(), display(), exc_clear(), exc_info(), exception_set(), execute(), executed(), executed_with_callbacks(), executed_without_callbacks(), fail_continue(), fail_stop(), failed(), get_target(), make_ready(), make_ready_all(), make_ready_current(), postprocess(), prepare(), trace_message()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

35.6.2 Properties

Name	Description
Inherited from object	
__class__	

35.7 Class Taskmaster

object ┐
SCons.Taskmaster.Taskmaster

The Taskmaster for walking the dependency DAG.

35.7.1 Methods

__init__(*self*, *targets*=`[]`, *tasker*=`None`, *order*=`None`, *trace*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

find_next_candidate(*self*)

Returns the next candidate Node for (potential) evaluation.

The candidate list (really a stack) initially consists of all of the top-level (command line) targets provided when the Taskmaster was initialized. While we walk the DAG, visiting Nodes, all the children that haven't finished processing get pushed on to the candidate list. Each child can then be popped and examined in turn for whether *their* children are all up-to-date, in which case a Task will be created for their actual evaluation and potential building.

Here is where we also allow candidate Nodes to alter the list of Nodes that should be examined. This is used, for example, when invoking SCons in a source directory. A source directory Node can return its corresponding build directory Node, essentially saying, "Hey, you really need to build this thing over here instead."

no_next_candidate(*self*)

Stops Taskmaster processing by not returning a next candidate.

Note that we have to clean-up the Taskmaster candidate list because the cycle detection depends on the fact all nodes have been processed somehow.

trace_message(*self*, *message*)

trace_node(*self*, *node*)

next_task(*self*)

Returns the next task to be executed.

This simply asks for the next Node to be evaluated, and then wraps it in the specific Task subclass with which we were initialized.

will_not_build(*self*, *nodes*, *node_func*=`<function <lambda> at 0x18d1d70>`)

Perform clean-up about nodes that will never be built. Invokes a user defined function on all of these nodes (including all of their parents).

stop(*self*)

Stops the current build completely.

cleanup(*self*)

Check for dependency cycles.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

35.7.2 Properties

Name	Description
Inherited from object	
__class__	

36 Module SCons.Util

SCons.Util

Various utility functions go here.

36.1 Functions

dictify(*keys*, *values*, *result*={})

rightmost_separator(*path*, *sep*)

containsAny(*str*, *set*)

Check whether sequence str contains ANY of the items in set.

containsAll(*str*, *set*)

Check whether sequence str contains ALL of the items in set.

containsOnly(*str*, *set*)

Check whether sequence str contains ONLY items in set.

splitext(*path*)

Same as os.path.splitext() but faster.

updrive(*path*)

Make the drive letter (if any) upper case. This is useful because Windows is inconsitent on the case of the drive letter, which can cause inconsistencies when calculating command signatures.

get_environment_var(*varstr*)

Given a string, first determine if it looks like a reference to a single environment variable, like "$FOO" or "${FOO}". If so, return that variable with no decorations ("FOO"). If not, return None.

render_tree(*root*, *child_func*, *prune*=0, *margin*=[0], *visited*={})

```
Render a tree of nodes into an ASCII tree view.
root - the root node of the tree
child_func - the function called to get the children of a node
prune - don't visit the same node twice
margin - the format of the left margin to use for chil-
dren of root.
   1 results in a pipe, and 0 results in no pipe.
visited - a dictionary of visited nodes in the cur-
rent branch if not prune,
   or in the whole tree if prune.
```

IDX(*N*)

print_tree(*root*, *child_func*, *prune*=0, *showtags*=0, *margin*=[0], *visited*={})

```
Print a tree of nodes.  This is like render_tree, ex-
cept it prints
lines directly instead of creating a string representa-
tion in memory,
so that huge trees can be printed.

root - the root node of the tree
child_func - the function called to get the children of a node
prune - don't visit the same node twice
showtags - print status information to the left of each node line
margin - the format of the left margin to use for chil-
dren of root.
   1 results in a pipe, and 0 results in no pipe.
visited - a dictionary of visited nodes in the cur-
rent branch if not prune,
   or in the whole tree if prune.
```

is_Dict(*obj*, *isinstance*=`<built-in function isinstance>`, *DictTypes*=`dict, UserDict`)

is_List(*obj*, *isinstance*=`<built-in function isinstance>`, *ListTypes*=`(<type 'list'>, <class 'UserList.UserList'>)`)

is_Sequence(*obj*, *isinstance*=`<built-in function isinstance>`, *SequenceTypes*=`(<type 'list'>, <type 'tuple'>, <class 'UserList.UserList'>)`)

is_Tuple(*obj*, *isinstance*=`<built-in function isinstance>`, *tuple*=`<type 'tuple'>`)

is_String(*obj*, *isinstance*=`<built-in function isinstance>`, *StringTypes*=`(<type 'str'>, <type 'unicode'>, <class 'UserString.UserS...`)

is_Scalar(*obj*, *isinstance*=`<built-in function isinstance>`, *StringTypes*=`(<type 'str'>, <type 'unicode'>, <class 'UserString.UserS...`, *SequenceTypes*=`(<type 'list'>, <type 'tuple'>, <class 'UserList.UserList'>)`)

do_flatten(*sequence*, *result*, *isinstance*=`<built-in function isinstance>`, *StringTypes*=`(<type 'str'>, <type 'unicode'>, <class 'UserString.UserS...`, *SequenceTypes*=`(<type 'list'>, <type 'tuple'>, <class 'UserList.UserList'>)`)

flatten(*obj*, *isinstance*=`<built-in function isinstance>`, *StringTypes*=`(<type 'str'>, <type 'unicode'>, <class 'UserString.UserS...`, *SequenceTypes*=`(<type 'list'>, <type 'tuple'>, <class 'UserList.UserList'>)`, *do_flatten*=`<function do_flatten at 0x1306398>`)

Flatten a sequence to a non-nested list.

Flatten() converts either a single scalar or a nested sequence to a non-nested list. Note that flatten() considers strings to be scalars instead of sequences like Python would.

flatten_sequence(*sequence*, *isinstance*=<built-in function isinstance>, *StringTypes*=(<type 'str'>, <type 'unicode'>, <class 'UserString.UserS..., *SequenceTypes*=(<type 'list'>, <type 'tuple'>, <class 'UserList.UserList'>), *do_flatten*=<function do_flatten at 0x1306398>)

Flatten a sequence to a non-nested list.

Same as flatten(), but it does not handle the single scalar case. This is slightly more efficient when one knows that the sequence to flatten can not be a scalar.

to_String(*s*, *isinstance*=<built-in function isinstance>, *str*=<type 'str'>, *UserString*=<class 'UserString.UserString'>, *BaseStringTypes*=(<type 'str'>, <type 'unicode'>))

to_String_for_subst(*s*, *isinstance*=<built-in function isinstance>, *str*=<type 'str'>, *to_String*=<function to_String at 0x1306500>, *BaseStringTypes*=(<type 'str'>, <type 'unicode'>), *SequenceTypes*=(<type 'list'>, <type 'tuple'>, <class 'UserList.UserList'>), *UserString*=<class 'UserString.UserString'>)

to_String_for_signature(*obj*, *to_String_for_subst*=<function to_String_for_subst at 0x1306578>, *AttributeError*=<type 'exceptions.AttributeError'>)

semi_deepcopy(*x*)

RegGetValue(*root*, *key*)

RegOpenKeyEx(*root*, *key*)

WhereIs(*file*, *path*=None, *pathext*=None, *reject*=[])

PrependPath(*oldpath*, *newpath*, *sep*=`':'`, *delete_existing*=`1`, *canonicalize*=`None`)

This prepends newpath elements to the given oldpath. Will only add any particular path once (leaving the first one it encounters and ignoring the rest, to preserve path order), and will os.path.normpath and os.path.normcase all paths to help assure this. This can also handle the case where the given old path variable is a list instead of a string, in which case a list will be returned instead of a string.

Example: Old Path: "/foo/bar:/foo" New Path: "/biz/boom:/foo" Result: "/biz/boom:/foo:/foo/bar"

If delete_existing is 0, then adding a path that exists will not move it to the beginning; it will stay where it is in the list.

If canonicalize is not None, it is applied to each element of newpath before use.

AppendPath(*oldpath*, *newpath*, *sep*=`':'`, *delete_existing*=`1`, *canonicalize*=`None`)

This appends new path elements to the given old path. Will only add any particular path once (leaving the last one it encounters and ignoring the rest, to preserve path order), and will os.path.normpath and os.path.normcase all paths to help assure this. This can also handle the case where the given old path variable is a list instead of a string, in which case a list will be returned instead of a string.

Example: Old Path: "/foo/bar:/foo" New Path: "/biz/boom:/foo" Result: "/foo/bar:/biz/boom:/foo"

If delete_existing is 0, then adding a path that exists will not move it to the end; it will stay where it is in the list.

If canonicalize is not None, it is applied to each element of newpath before use.

get_native_path(*path*)

Transforms an absolute path into a native path for the system. Non-Cygwin version, just leave the path alone.

Split(*arg*)

case_sensitive_suffixes(*s1*, *s2*)

adjustixes(*fname*, *pre*, *suf*, *ensure_suffix*=`False`)

unique(*s*)

Return a list of the elements in s, but without duplicates.

For example, unique([1,2,3,1,2,3]) is some permutation of [1,2,3], unique("abcabc") some permutation of ["a", "b", "c"], and unique(([1, 2], [2, 3], [1, 2])) some permutation of [[2, 3], [1, 2]].

For best speed, all sequence elements should be hashable. Then unique() will usually work in linear time.

If not possible, the sequence elements should enjoy a total ordering, and if list(s).sort() doesn't raise TypeError it's assumed that they do enjoy a total ordering. Then unique() will usually work in O(N*log2(N)) time.

If that's not possible either, the sequence elements must support equality-testing. Then unique() will usually work in quadratic time.

uniquer(*seq*, *idfun*=`None`)

uniquer_hashables(*seq*)

make_path_relative(*path*)

makes an absolute path name to a relative pathname.

AddMethod(*obj*, *function*, *name*=None)

```
Adds either a bound method to an instance or an unbound method to
a class. If name is ommited the name of the specified function
is used by default.
Example:
  a = A()
  def f(self, x, y):
    self.z = x + y
  AddMethod(f, A, "add")
  a.add(2, 4)
  print a.z
  AddMethod(lambda self, i: self.l[i], a, "listIndex")
  print a.listIndex(5)
```

RenameFunction(*function*, *name*)

Returns a function identical to the specified function, but with the specified name.

MD5signature(*s*)

MD5filesignature(*fname*, *chunksize*=65536)

MD5collect(*signatures*)

Collects a list of signatures into an aggregate signature.

signatures - a list of signatures returns - the aggregate signature

silent_intern(*x*)

Perform sys.intern() on the passed argument and return the result. If the input is ineligible (e.g. a unicode string) the original argument is returned and no exception is thrown.

36.2 Variables

Name	Description
DictTypes	**Value:** dict, UserDict
ListTypes	**Value:** (<type ’list’>, <class ’UserList.UserList’>)
SequenceTypes	**Value:** (<type ’list’>, <type ’tuple’>, <class ’UserList.UserList’>)
StringTypes	**Value:** (<type ’str’>, <type ’unicode’>, <class ’UserString.UserS...
BaseStringTypes	**Value:** (<type ’str’>, <type ’unicode’>)
d	**Value:** {}
can_read_reg	**Value:** 0
hkey_mod	**Value:** win32con
RegEnumKey	**Value:** win32api.RegEnumKey
RegEnumValue	**Value:** win32api.RegEnumValue
RegQueryValueEx	**Value:** win32api.RegQueryValueEx
HKEY_CLASSES_ROOT	**Value:** None
HKEY_LOCAL_MACHIN-E	**Value:** None
HKEY_CURRENT_USER	**Value:** None
HKEY_USERS	**Value:** None
display	**Value:** DisplayEngine()
md5	**Value:** True
__package__	**Value:** ’SCons’

36.3 Class NodeList

```
object ─┐
        _abcoll.Sized ─┐
object ─┐              │
        _abcoll.Iterable ─┤
object ─┐              │
        _abcoll.Container ─┘
                _abcoll.Sequence ─┐
                        _abcoll.MutableSequence ─┐
                                UserList.UserList ─┐
                                        SCons.Util.NodeList
```

This class is almost exactly like a regular list of Nodes (actually it can hold any object), with one important difference. If you try to get an attribute from this list, it will return that attribute from every item in the list. For example:

```
>>> someList = NodeList([ ' foo ', ' bar ' ])
>>> someList.strip()
[ 'foo', 'bar' ]
```

36.3.1 Methods

__nonzero__(*self*)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

__iter__(*self*)

Overrides: _abcoll.Iterable.__iter__

__call__(*self*, **args*, ***kwargs*)

__getattr__(*self*, *name*)

Inherited from UserList.UserList

__add__(), __cmp__(), __contains__(), __delitem__(), __delslice__(), __eq__(), __ge__(), __getitem__(), __getslice__(), __gt__(), __iadd__(), __imul__(), __init__(), __le__(), __len__(), __lt__(), __mul__(), __ne__(), __radd__(), __repr__(), __rmul__(), __setitem__(), __setslice__(), append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

Inherited from _abcoll.Sequence

__reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__()

36.3.2 Properties

Name	Description
Inherited from object	
__class__	

36.3.3 Class Variables

Name	Description
Inherited from UserList.UserList	
__abstractmethods__, __hash__	

36.4 Class DisplayEngine

object

SCons.Util.DisplayEngine

36.4.1 Methods

__call__(*self*, *text*, *append_newline*=1)

set_mode(*self*, *mode*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

36.4.2 Properties

Name	Description
Inherited from object	
__class__	

36.4.3 Class Variables

Name	Description
print_it	**Value:** `True`

36.5 Class Proxy

object

SCons.Util.Proxy

Known Subclasses: SCons.Builder.CompositeBuilder, SCons.Node.FS.EntryProxy

A simple generic Proxy class, forwarding all calls to subject. So, for the benefit of the python newbie, what does this really mean? Well, it means that you can take an object, let's call it 'objA', and wrap it in this Proxy class, with a statement like this

proxyObj = Proxy(objA),

Then, if in the future, you do something like this

x = proxyObj.var1,

since Proxy does not have a 'var1' attribute (but presumably objA does), the request actually is equivalent to saying x = objA.var1

Inherit from this class to create a Proxy.

Note that, with new-style classes, this does *not* work transparently for Proxy subclasses that use special .__*__() method names, because those names are now bound to the class, not the individual instances. You now need to know in advance which .__*__() method names you want to pass on to the underlying Proxy object, and specifically delegate their calls like this:

class Foo(Proxy): __str__ = Delegate('__str__')

36.5.1 Methods

__init__(*self*, *subject*)

Wrap an object as a Proxy object Overrides: object.__init__

__getattr__(*self*, *name*)

Retrieve an attribute from the wrapped object. If the named attribute doesn't exist, AttributeError is raised

get(*self*)

Retrieve the entire wrapped object

__cmp__(*self*, *other*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

36.5.2 Properties

Name	Description
Inherited from object	
__class__	

36.6 Class Delegate

object ─┐
SCons.Util.Delegate

A Python Descriptor class that delegates attribute fetches to an underlying wrapped subject of a Proxy. Typical use:

class Foo(Proxy): __str__ = Delegate('__str__')

36.6.1 Methods

__init__(*self*, *attribute*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__get__(*self*, *obj*, *cls*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

36.6.2 Properties

Name	Description
Inherited from object	
__class__	

36.7 Class _NoError

object ─┐
exceptions.BaseException ─┐
exceptions.Exception ─┐
SCons.Util._NoError

36.7.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

36.7.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

36.8 Class WindowsError

object
exceptions.BaseException
exceptions.Exception
exceptions.StandardError
exceptions.EnvironmentError
exceptions.OSError
SCons.Util.WindowsError

36.8.1 Methods

Inherited from exceptions.OSError

__init__(), __new__()

Inherited from exceptions.EnvironmentError

__reduce__(), __str__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

36.8.2 Properties

Name	Description
Inherited from exceptions.EnvironmentError errno, filename, strerror	
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

36.9 Class CLVar

```
object ─┐
        │
   _abcoll.Sized ─┐
                  │
object ─┐         │
        │         │
 _abcoll.Iterable ─┤
                  │
object ─┐         │
        │         │
_abcoll.Container ─┤
                   │
     _abcoll.Sequence ─┐
                       │
  _abcoll.MutableSequence ─┐
                           │
          UserList.UserList ─┐
                             │
                     SCons.Util.CLVar
```

A class for command-line construction variables.

This is a list that uses Split() to split an initial string along white-space arguments, and similarly to split any strings that get added. This allows us to Do the Right Thing with Append() and Prepend() (as well as straight Python foo = env['VAR'] + 'arg1 arg2') regardless of whether a user adds a list or a string to a command-line construction variable.

36.9.1 Methods

__init__(*self*, *seq*=`[]`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__add__(*self*, *other*)

Overrides: UserList.UserList.__add__

__radd__(*self*, *other*)

Overrides: UserList.UserList.__radd__

__coerce__(*self*, *other*)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from UserList.UserList

__cmp__(), __contains__(), __delitem__(), __delslice__(), __eq__(), __ge__(), __getitem__(), __getslice__(), __gt__(), __iadd__(), __imul__(), __le__(), __len__(), __lt__(), __mul__(), __ne__(), __repr__(), __rmul__(), __setitem__(), __setslice__(), append(), count(), extend(), index(), insert(), pop(), remove(), reverse(), sort()

Inherited from _abcoll.Sequence

__iter__(), __reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__()

36.9.2 Properties

Name	Description
Inherited from object	
__class__	

36.9.3 Class Variables

Name	Description
Inherited from UserList.UserList	
__abstractmethods__, __hash__	

36.10 Class OrderedDict

UserDict.UserDict

SCons.Util.OrderedDict

Known Subclasses: SCons.Util.Selector

36.10.1 Methods

__init__(*self*, *dict*=None)

Overrides: UserDict.UserDict.__init__

__delitem__(*self*, *key*)

Overrides: UserDict.UserDict.__delitem__

__setitem__(*self*, *key*, *item*)

Overrides: UserDict.UserDict.__setitem__

clear(*self*)

Overrides: UserDict.UserDict.clear

copy(*self*)

Overrides: UserDict.UserDict.copy

items(*self*)

Overrides: UserDict.UserDict.items

keys(*self*)

Overrides: UserDict.UserDict.keys

popitem(*self*)

Overrides: UserDict.UserDict.popitem

setdefault(*self*, *key*, *failobj*=None)

Overrides: UserDict.UserDict.setdefault

update(*self*, *dict*)

Overrides: UserDict.UserDict.update

values(*self*)

Overrides: UserDict.UserDict.values

Inherited from UserDict.UserDict

__cmp__(), __contains__(), __getitem__(), __len__(), __repr__(), fromkeys(), get(), has_key(), iteritems(), iterkeys(), itervalues(), pop()

36.11 Class Selector

UserDict.UserDict
SCons.Util.OrderedDict
SCons.Util.Selector

Known Subclasses: SCons.Builder.CallableSelector, SCons.Builder.DictCmdGenerator, SCons.Builder.DictEmitter

A callable ordered dictionary that maps file suffixes to dictionary values. We preserve the order in which items are added so that get_suffix() calls always return the first suffix added.

36.11.1 Methods

__call__(*self*, *env*, *source*, *ext*=`None`)

Inherited from SCons.Util.OrderedDict(Section 36.10)

__delitem__(), __init__(), __setitem__(), clear(), copy(), items(), keys(), popitem(), setdefault(), update(), values()

Inherited from UserDict.UserDict

__cmp__(), __contains__(), __getitem__(), __len__(), __repr__(), fromkeys(), get(), has_key(), iteritems(), iterkeys(), itervalues(), pop()

36.12 Class LogicalLines

object
SCons.Util.LogicalLines

36.12.1 Methods

__init__(*self*, *fileobj*) x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

readline(*self*)

readlines(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

36.12.2 Properties

Name	Description
Inherited from object	
__class__	

36.13 Class UniqueList

```
object
  _abcoll.Sized
object
  _abcoll.Iterable
object
  _abcoll.Container
    _abcoll.Sequence
      _abcoll.MutableSequence
        UserList.UserList
          SCons.Util.UniqueList
```

36.13.1 Methods

__init__(*self*, *seq*=`[]`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__lt__(*self*, *other*)

Overrides: UserList.UserList.__lt__

__le__(*self*, *other*)

Overrides: UserList.UserList.__le__

__eq__(*self*, *other*)

Overrides: UserList.UserList.__eq__

__ne__(*self*, *other*)

Overrides: UserList.UserList.__ne__

__gt__(*self*, *other*)

Overrides: UserList.UserList.__gt__

__ge__(*self*, *other*)

Overrides: UserList.UserList.__ge__

__cmp__(*self*, *other*)

Overrides: UserList.UserList.__cmp__

__len__(*self*)

Overrides: _abcoll.Sized.__len__

__getitem__(*self*, *i*)

Overrides: _abcoll.Sequence.__getitem__

__setitem__(*self*, *i*, *item*)

Overrides: _abcoll.MutableSequence.__setitem__

__getslice__(*self*, *i*, *j*)

Overrides: UserList.UserList.__getslice__

__setslice__(*self*, *i*, *j*, *other*)

Overrides: UserList.UserList.__setslice__

__add__(*self*, *other*)

Overrides: UserList.UserList.__add__

__radd__(*self*, *other*)

Overrides: UserList.UserList.__radd__

__iadd__(*self*, *other*)

Overrides: _abcoll.MutableSequence.__iadd__

__mul__(*self*, *other*)

Overrides: UserList.UserList.__mul__

__rmul__(*self*, *other*)

Overrides: UserList.UserList.__rmul__

__imul__(*self*, *other*)

Overrides: UserList.UserList.__imul__

append(*self*, *item*)

Overrides: _abcoll.MutableSequence.append

insert(*self*, *i*)

Overrides: _abcoll.MutableSequence.insert

count(*self*, *item*)

Overrides: _abcoll.Sequence.count

index(*self*, *item*)

Overrides: _abcoll.Sequence.index

reverse(*self*)

Overrides: _abcoll.MutableSequence.reverse

sort(*self*, **args*, ***kwds*)

Overrides: UserList.UserList.sort

extend(*self*, *other*)
Overrides: _abcoll.MutableSequence.extend

Inherited from UserList.UserList

__contains__(), __delitem__(), __delslice__(), __repr__(), pop(), remove()

Inherited from _abcoll.Sequence

__iter__(), __reversed__()

Inherited from _abcoll.Sized

__subclasshook__()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__()

36.13.2 Properties

Name	Description
Inherited from object	
__class__	

36.13.3 Class Variables

Name	Description
Inherited from UserList.UserList	
__abstractmethods__, __hash__	

36.14 Class Unbuffered

```
object ─┐
        │
     SCons.Util.Unbuffered
```

A proxy class that wraps a file object, flushing after every write, and delegating everything else to the wrapped object.

36.14.1 Methods

__init__(*self*, *file*) x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

write(*self*, *arg*)

__getattr__(*self*, *attr*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

36.14.2 Properties

Name	Description
Inherited from object	
__class__	

36.15 Class Null

object ─┐
SCons.Util.Null

Known Subclasses: SCons.Sig.MD5Null, SCons.Sig.TimeStampNull, SCons.Util.NullSeq

Null objects always and reliably "do nothing."

36.15.1 Methods

__new__(*cls*, **args*, ***kwargs*) **Return Value** a new object with type S, a subtype of T Overrides: object.__new__ extit(inherited documentation)

__init__(*self*, **args*, ***kwargs*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__call__(*self*, **args*, ***kwargs*)

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

__nonzero__(*self*)

__getattr__(*self*, *name*)

__setattr__(*self*, *name*, *value*)

x.__setattr__('name', value) <==> x.name = value Overrides: object.__setattr__ extit(inherited documentation)

__delattr__(*self*, *name*)

x.__delattr__('name') <==> del x.name Overrides: object.__delattr__ extit(inherited documentation)

Inherited from object

__format__(), __getattribute__(), __hash__(), __reduce__(), __reduce_ex__(), __sizeof__(), __str__(), __subclasshook__()

36.15.2 Properties

Name	Description
Inherited from object	
__class__	

36.16 Class NullSeq

```
object ─┐
        │
SCons.Util.Null ─┐
                 │
                 SCons.Util.NullSeq
```

Known Subclasses: SCons.Subst.NullNodeList

36.16.1 Methods

__len__(*self*)

__iter__(*self*)

__getitem__(*self*, *i*)

__delitem__(*self*, *i*)

__setitem__(*self*, *i*, *v*)

Inherited from SCons.Util.Null(Section 36.15)

__call__(), __delattr__(), __getattr__(), __init__(), __new__(), __nonzero__(), __repr__(), __setattr__()

Inherited from object

__format__(), __getattribute__(), __hash__(), __reduce__(), __reduce_ex__(), __sizeof__(), __str__(), __subclasshook__()

36.16.2 Properties

Name	Description
Inherited from object	
__class__	

37 Package SCons.Variables

engine.SCons.Variables

This file defines the Variables class that is used to add user-friendly customizable variables to an SCons build.

37.1 Modules

- **BoolVariable** *(Section ??, p. ??)*
- **BoolVariable'**: engine.SCons.Variables.BoolVariable *(Section 38, p. 329)*
- **EnumVariable** *(Section ??, p. ??)*
- **EnumVariable'**: engine.SCons.Variables.EnumVariable *(Section 39, p. 330)*
- **ListVariable** *(Section ??, p. ??)*
- **ListVariable'**: engine.SCons.Variables.ListVariable *(Section 40, p. 332)*
- **PackageVariable** *(Section ??, p. ??)*
- **PackageVariable'**: engine.SCons.Variables.PackageVariable *(Section 41, p. 333)*
- **PathVariable** *(Section ??, p. ??)*
- **PathVariable'**: SCons.Variables.PathVariable *(Section 42, p. 334)*

37.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Variables/__init__.py 5134 2010/08/16 2...`
__package__	**Value:** `'SCons.Variables'`

37.3 Class Variables

object ─┐
  SCons.Variables.Variables

37.3.1 Methods

__init__(*self*, *files*=`[]`, *args*=`{}`, *is_global*=`1`)

files - [optional] List of option configuration files to load

(backward compatibility) If a single string is passed it is automatically placed in a file list

Overrides: object.__init__

---

keys(*self*)

Returns the keywords for the options

Add(*self*, *key*, *help*=`''`, *default*=`None`, *validator*=`None`, *converter*=`None`, ***kw*)

```
Add an option.

key - the name of the variable, or a list or tuple of arguments
help - optional help text for the options
default - optional default value
validator - optional function that is called to validate the op-
tion's value
            Called with (key, value, environment)
converter - optional function that is called to convert the op-
tion's value before
            putting it in the environment.
```

AddVariables(*self*, **optlist*)

```
Add a list of options.

Each list element is a tuple/list of arguments to be passed on
to the underlying method for adding options.

Example:
  opt.AddVariables(
    ('debug', '', 0),
    ('CC', 'The C compiler'),
    ('VALIDATE', 'An option for testing validation', 'notset',
     validator, None),
    )
```

Update(*self*, *env*, *args*=`None`)

Update an environment with the option variables.

env - the environment to update.

UnknownVariables(*self*)

Returns any options in the specified arguments lists that were not known, declared options in this object.

Save(*self*, *filename*, *env*)

Saves all the options in the given file. This file can then be used to load the options next run. This can be used to create an option cache file.

filename - Name of the file to save into env - the environment get the option values from

GenerateHelpText(*self*, *env*, *sort*=`None`)

Generate the help text for the options.

env - an environment that is used to get the current values of the options.

FormatVariableHelpText(*self*, *env*, *key*, *help*, *default*, *actual*, *aliases*=`[]`)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

37.3.2 Properties

Name	Description
Inherited from object	
__class__	

37.3.3 Class Variables

Name	Description
instance	Holds all the options, updates the environment with the variables, and renders the help text. **Value:** `None`
format	**Value:** `'\n%s: %s\n default: %s\n actual: %s\n'`
format_	**Value:** `'\n%s: %s\n default: %s\n actual: %s\n aliases: ...`

38 Module SCons.Variables.BoolVariable'

```
engine.SCons.Variables.BoolVariable

This file defines the option type for SCons implementing true/false values.

Usage example:

  opts = Variables()
  opts.Add(BoolVariable('embedded', 'build for an embedded system', 0))
  ...
  if env['embedded'] == 1:
    ...
```

38.1 Functions

BoolVariable(*key*, *help*, *default*)

The input parameters describe a boolen option, thus they are returned with the correct converter and validator appended. The 'help' text will by appended by '(yes|no) to show the valid valued. The result is usable for input to opts.Add().

39 Module SCons.Variables.EnumVariable'

```
engine.SCons.Variables.EnumVariable

This file defines the option type for SCons allowing only specified
input-values.

Usage example:

  opts = Variables()
  opts.Add(EnumVariable('debug', 'debug output and symbols', 'no',
                    allowed_values=('yes', 'no', 'full'),
                    map={}, ignorecase=2))
  ...
  if env['debug'] == 'full':
    ...
```

39.1 Functions

EnumVariable(*key*, *help*, *default*, *allowed_values*, *map*={}, *ignorecase*=0)

```
The input parameters describe a option with only certain values
allowed. They are returned with an appropriate converter and
validator appended. The result is usable for input to
Variables.Add().

'key' and 'default' are the values to be passed on to Variables.Add().

'help' will be appended by the allowed values automatically

'allowed_values' is a list of strings, which are al-
lowed as values
for this option.

The 'map'-dictionary may be used for converting the input value
into canonical values (eg. for aliases).

'ignorecase' defines the behaviour of the validator:

If ignorecase == 0, the validator/converter are case-sensitive.
If ignorecase == 1, the validator/converter are case-insensitive.
If ignorecase == 2, the validator/converter is case-
insensitive and
                    the converted value will always be lower-
case.

The 'validator' tests whether the value is in the list of allowed
values. The 'converter' converts input values according to the
given 'map'-dictionary (unmapped input values are returned
unchanged).
```

40 Module SCons.Variables.ListVariable'

```
engine.SCons.Variables.ListVariable

This file defines the option type for SCons implementing 'lists'.

A 'list' option may either be 'all', 'none' or a list of names
separated by comma. After the option has been processed, the option
value holds either the named list elements, all list elemens or no
list elements at all.

Usage example:

  list_of_libs = Split('x11 gl qt ical')

  opts = Variables()
  opts.Add(ListVariable('shared',
                      'libraries to build as shared libraries',
                      'all',
                      elems = list_of_libs))
  ...
  for lib in list_of_libs:
     if lib in env['shared']:
         env.SharedObject(...)
     else:
         env.Object(...)
```

40.1 Functions

ListVariable(*key*, *help*, *default*, *names*, *map*={})

The input parameters describe a 'package list' option, thus they are returned with the correct converter and validater appended. The result is usable for input to opts.Add() .

A 'package list' option may either be 'all', 'none' or a list of package names (separated by space).

41 Module SCons.Variables.PackageVariable'

```
engine.SCons.Variables.PackageVariable

This file defines the option type for SCons implementing 'package
activation'.

To be used whenever a 'package' may be enabled/disabled and the
package path may be specified.

Usage example:

  Examples:
      x11=no   (disables X11 support)
      x11=yes  (will search for the package installation dir)
      x11=/usr/local/X11 (will check this path for existance)

  To replace autoconf's --with-xxx=yyy

  opts = Variables()
  opts.Add(PackageVariable('x11',
                         'use X11 installed here (yes = search some places',
                         'yes'))
  ...
  if env['x11'] == True:
      dir = ... search X11 in some standard places ...
      env['x11'] = dir
  if env['x11']:
      ... build with x11 ...
```

41.1 Functions

PackageVariable(*key*, *help*, *default*, *searchfunc*=`None`)

The input parameters describe a 'package list' option, thus they are returned with the correct converter and validator appended. The result is usable for input to opts.Add() .

A 'package list' option may either be 'all', 'none' or a list of package names (seperated by space).

42 Module SCons.Variables.PathVariable'

```
SCons.Variables.PathVariable

This file defines an option type for SCons implementing path settings.

To be used whenever a a user-specified path override should be allowed.

Arguments to PathVariable are:
  option-name  = name of this option on the command line (e.g. "prefix")
  option-help  = help string for option
  option-dflt  = default value for this option
  validator    = [optional] validator for option value.  Predefined
                 validators are:

                     PathAccept -- accepts any path setting; no validation
                     PathIsDir  -- path must be an existing directory
                     PathIsDirCreate -- path must be a dir; will create
                     PathIsFile -- path must be a file
                     PathExists -- path must exist (any type) [default]

                 The validator is a function that is called and which
                 should return True or False to indicate if the path
                 is valid.  The arguments to the validator function
                 are: (key, val, env).  The key is the name of the
                 option, the val is the path specified for the option,
                 and the env is the env to which the Otions have been
                 added.

Usage example:

  Examples:
      prefix=/usr/local

  opts = Variables()

  opts = Variables()
  opts.Add(PathVariable('qtdir',
                      'where the root of Qt is installed',
                      qtdir, PathIsDir))
  opts.Add(PathVariable('qt_includes',
                      'where the Qt includes are installed',
                      '$qtdir/includes', PathIsDirCreate))
  opts.Add(PathVariable('qt_libraries',
                      'where the Qt library is installed',
```

```
'$qtdir/lib'))
```

42.1 Variables

Name	Description
PathVariable	**Value:** `SCons.Variables.PathVariable`

43 Module SCons.Warnings

SCons.Warnings

This file implements the warnings framework for SCons.

43.1 Functions

suppressWarningClass(*clazz*)

Suppresses all warnings that are of type clazz or derived from clazz.

enableWarningClass(*clazz*)

Enables all warnings that are of type clazz or derived from clazz.

warningAsException(*flag*=1)

Turn warnings into exceptions. Returns the old value of the flag.

warn(*clazz*, **args*)

process_warn_strings(*arguments*)

Process string specifications of enabling/disabling warnings, as passed to the --warn option or the SetOption('warn') function.

An argument to this option should be of the form <warning-class> or no-<warning-class>. The warning class is munged in order to get an actual class name from the classes above, which we need to pass to the {enable,disable}WarningClass() functions. The supplied <warning-class> is split on hyphens, each element is capitalized, then smushed back together. Then the string "Warning" is appended to get the class name.

For example, 'deprecated' will enable the DeprecatedWarning class. 'no-dependency' will disable the DependencyWarning class.

As a special case, --warn=all and --warn=no-all will enable or disable (respectively) the base Warning class of all warnings.

43.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/Warnings.py 5134 2010/08/16 23:02:40 bd...`
__package__	**Value:** `'SCons'`

43.3 Class Warning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning

Known Subclasses: SCons.SConf.SConfWarning, SCons.Warnings.CacheWriteErrorWarning, SCons.Warnings.WarningOnByDefault, SCons.Warnings.DependencyWarning, SCons.Warnings.DeprecatedWarning, SCons.Warnings.FutureDeprecatedWarning, SCons.Warnings.VisualStudioMissingWarning

43.3.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.3.2 Properties

Name	Description
Inherited from exceptions.BaseException	

continued on next page

Name	Description
args, message	
Inherited from object	
__class__	

43.4 Class WarningOnByDefault

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault

Known Subclasses: SCons.Warnings.CorruptSConsignWarning, SCons.Warnings.DuplicateEnvironmentWarning, SCons.Warnings.LinkWarning, SCons.Warnings.FutureReservedVariableWarning, SCons.Warnings.MisleadingKeywordsWarning, SCons.Warnings.MissingSConscriptWarning, SCons.Warnings.NoMD5ModuleWarning, SCons.Warnings.NoMetaclassSupportWarning, SCons.Warnings.NoObjectCountWarning, SCons.Warnings.NoParallelSupportWarning, SCons.Warnings.ReservedVariableWarning, SCons.Warnings.StackSizeWarning, SCons.Warnings.VisualCMissingWarning, SCons.Warnings.VisualVersionMismatch

43.4.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.4.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.5 Class CacheWriteErrorWarning

```
object ─┐
exceptions.BaseException ─┐
        exceptions.Exception ─┐
          SCons.Errors.UserError ─┐
             SCons.Warnings.Warning ─┐
                                     SCons.Warnings.CacheWriteErrorWarning
```

43.5.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.5.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	

continued on next page

Name	Description
Inherited from object	
__class__	

43.6 Class CorruptSConsignWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.CorruptSConsignWarning

43.6.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.6.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	

continued on next page

Name	Description
Inherited from object	
__class__	

43.7 Class DependencyWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DependencyWarning

43.7.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.7.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.8 Class DuplicateEnvironmentWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.DuplicateEnvironmentWarning

43.8.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.8.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.9 Class FutureReservedVariableWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.FutureReservedVariableWarning

43.9.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.9.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.10 Class LinkWarning

```
object
  └ exceptions.BaseException
      └ exceptions.Exception
          └ SCons.Errors.UserError
              └ SCons.Warnings.Warning
                  └ SCons.Warnings.WarningOnByDefault
                      └ SCons.Warnings.LinkWarning
```

Known Subclasses: SCons.Warnings.FortranCxxMixWarning

43.10.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.10.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.11 Class MisleadingKeywordsWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.MisleadingKeywordsWarning

43.11.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.11.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.12 Class MissingSConscriptWarning

```
object
  |
exceptions.BaseException
  |
  exceptions.Exception
    |
    SCons.Errors.UserError
      |
      SCons.Warnings.Warning
        |
        SCons.Warnings.WarningOnByDefault
          |
          SCons.Warnings.MissingSConscriptWarning
```

43.12.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.12.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.13 Class NoMD5ModuleWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.NoMD5ModuleWarning

43.13.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.13.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.14 Class NoMetaclassSupportWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.NoMetaclassSupportWarning

43.14.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.14.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.15 Class NoObjectCountWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.NoObjectCountWarning

43.15.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.15.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

43.16 Class NoParallelSupportWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.NoParallelSupportWarning

43.16.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.16.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.17 Class ReservedVariableWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.ReservedVariableWarning

43.17.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.17.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.18 Class StackSizeWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.StackSizeWarning

43.18.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.18.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.19 Class VisualCMissingWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.VisualCMissingWarning

43.19.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.19.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.20 Class VisualVersionMismatch

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.VisualVersionMismatch

43.20.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.20.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.21 Class VisualStudioMissingWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.VisualStudioMissingWarning

43.21.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.21.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.22 Class FortranCxxMixWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.LinkWarning
SCons.Warnings.FortranCxxMixWarning

43.22.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.22.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.23 Class FutureDeprecatedWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.FutureDeprecatedWarning

Known Subclasses: SCons.Warnings.DeprecatedSourceCodeWarning

43.23.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.23.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.24 Class DeprecatedWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning

Known Subclasses: SCons.Warnings.DeprecatedBuildDirWarning, SCons.Warnings.MandatoryDeprecatedWarning, SCons.Warnings.PythonVersionWarning, SCons.Warnings.TaskmasterNeedsExecuteWarning

43.24.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.24.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.25 Class MandatoryDeprecatedWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.MandatoryDeprecatedWarning

Known Subclasses: SCons.Warnings.DeprecatedBuilderKeywordsWarning, SCons.Warnings.DeprecatedCopyWarning, SCons.Warnings.DeprecatedDebugOptionsWarning, SCons.Warnings.DeprecatedOptionsWarning, SCons.Warnings.DeprecatedSigModuleWarning, SCons.Warnings.DeprecatedSourceSignaturesWarning, SCons.Warnings.DeprecatedTargetSignaturesWarning

43.25.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.25.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.26 Class PythonVersionWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.PythonVersionWarning

43.26.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.26.2 Properties

Name	Description
Inherited from exceptions.BaseException args, message	
Inherited from object __class__	

43.27 Class DeprecatedSourceCodeWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.FutureDeprecatedWarning
SCons.Warnings.DeprecatedSourceCodeWarning

43.27.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.27.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.28 Class DeprecatedBuildDirWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.DeprecatedBuildDirWarning

43.28.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.28.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.29 Class TaskmasterNeedsExecuteWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.TaskmasterNeedsExecuteWarning

43.29.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.29.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.30 Class DeprecatedCopyWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.MandatoryDeprecatedWarning
SCons.Warnings.DeprecatedCopyWarning

43.30.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.30.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.31 Class DeprecatedOptionsWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.MandatoryDeprecatedWarning
SCons.Warnings.DeprecatedOptionsWarning

43.31.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.31.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.32 Class DeprecatedSourceSignaturesWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.MandatoryDeprecatedWarning
SCons.Warnings.DeprecatedSourceSignaturesWarning

43.32.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.32.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.33 Class DeprecatedTargetSignaturesWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.MandatoryDeprecatedWarning
SCons.Warnings.DeprecatedTargetSignaturesWarning

43.33.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.33.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.34 Class DeprecatedDebugOptionsWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.MandatoryDeprecatedWarning
SCons.Warnings.DeprecatedDebugOptionsWarning

43.34.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.34.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.35 Class DeprecatedSigModuleWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.MandatoryDeprecatedWarning
SCons.Warnings.DeprecatedSigModuleWarning

43.35.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.35.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

43.36 Class DeprecatedBuilderKeywordsWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.DeprecatedWarning
SCons.Warnings.MandatoryDeprecatedWarning
SCons.Warnings.DeprecatedBuilderKeywordsWarning

43.36.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

43.36.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

44 Package SCons.compat

SCons compatibility package for old Python versions

This subpackage holds modules that provide backwards-compatible implementations of various things that we'd like to use in SCons but which only show up in later versions of Python than the early, old version(s) we still support.

Other code will not generally reference things in this package through the SCons.compat namespace. The modules included here add things to the builtins namespace or the global module list so that the rest of our code can use the objects and names imported here regardless of Python version.

Simply enough, things that go in the builtins name space come from our _scons_builtins module.

The rest of the things here will be in individual compatibility modules that are either: 1) suitably modified copies of the future modules that we want to use; or 2) backwards compatible re-implementations of the specific portions of a future module's API that we want to use.

GENERAL WARNINGS: Implementations of functions in the SCons.compat modules are *NOT* guaranteed to be fully compliant with these functions in later versions of Python. We are only concerned with adding functionality that we actually use in SCons, so be wary if you lift this code for other uses. (That said, making these more nearly the same as later, official versions is still a desirable goal, we just don't need to be obsessive about it.)

We name the compatibility modules with an initial '_scons_' (for example, _scons_subprocess.py is our compatibility module for subprocess) so that we can still try to import the real module name and fall back to our compatibility module if we get an ImportError. The import_as() function defined below loads the module as the "real" name (without the '_scons'), after which all of the "import {module}" statements in the rest of our code will find our pre-loaded compatibility module.

44.1 Modules

- **_scons_io**: io compatibility module for older (pre-2.6) Python versions
  (Section 49, p. 381)
- **_scons_sets**: Classes to represent arbitrary sets (including sets of sets).
  (Section 50, p. 382)
- **_scons_subprocess**: subprocess - Subprocesses with accessible I/O streams
  (Section 51, p. 391)

44.2 Functions

import_as(*module*, *name*)

Imports the specified module (from our local directory) as the specified name, returning the loaded module object.

rename_module(*new*, *old*)

Attempts to import the old module and load it under the new name. Used for purely cosmetic name changes in Python 3.x.

lexists(*path*)

44.3 Variables

Name	Description
__doc__	**Value:** `...`
__revision__	**Value:** `'src/engine/SCons/compat/__init__.py 5134 2010/08/16 23:0...`
__package__	**Value:** `'SCons.compat'`

45 Module SCons.compat._scons_builtins

Compatibility idioms for builtins names

This module adds names to the builtins module for things that we want to use in SCons but which don't show up until later Python versions than the earliest ones we support.

This module checks for the following builtins names:

all() any() sorted() memoryview()

Implementations of functions are *NOT* guaranteed to be fully compliant with these functions in later versions of Python. We are only concerned with adding functionality that we actually use in SCons, so be wary if you lift this code for other uses. (That said, making these more nearly the same as later, official versions is still a desirable goal, we just don't need to be obsessive about it.)

If you're looking at this with pydoc and various names don't show up in the FUNCTIONS or DATA output, that means those names are already built in to this version of Python and we don't need to add them from this module.

45.1 Functions

all(*iterable*)

Returns True if all elements of the iterable are true.

any(*iterable*)

Returns True if any element of the iterable is true.

sorted(*iterable*, *cmp*=`None`, *key*=`None`, *reverse*=`False`)

45.2 Variables

Name	Description
__doc__	**Value:** `...`
__revision__	**Value:** `'src/engine/SCons/compat/_scons_builtins.py 5134 2010/08/...`
__package__	**Value:** `'SCons.compat'`

45.3 Class memoryview

object ─┐
SCons.compat._scons_builtins.memoryview

45.3.1 Methods

__init__(*self*, *obj*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__getitem__(*self*, *indx*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

45.3.2 Properties

Name	Description
Inherited from object	
__class__	

46 Module SCons.compat._scons_collections

collections compatibility module for older (pre-2.4) Python versions

This does not not NOT (repeat, *NOT*) provide complete collections functionality. It only wraps the portions of collections functionality used by SCons, in an interface that looks enough like collections for our purposes.

46.1 Variables

Name	Description
__doc__	**Value:** `...`
__revision__	**Value:** `'src/engine/SCons/compat/_scons_collections.py 5134 2010/...`
__package__	**Value:** `'SCons.compat'`

47 Module SCons.compat._scons_dbm

dbm compatibility module for Python versions that don't have dbm.

This does not not NOT (repeat, *NOT*) provide complete dbm functionality. It's just a stub on which to hang just enough pieces of dbm functionality that the whichdb.whichdb() implementstation in the various 2.X versions of Python won't blow up even if dbm wasn't compiled in.

47.1 Functions

open(**args*, ***kw*)

47.2 Variables

Name	Description
__doc__	**Value:** `...`
__revision__	**Value:** `'src/engine/SCons/compat/_scons_dbm.py 5134 2010/08/16 23...`
__package__	**Value:** `None`

47.3 Class error

object
exceptions.BaseException
exceptions.Exception
SCons.compat._scons_dbm.error

47.3.1 Methods

Inherited from exceptions.Exception

__init__(), __new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

47.3.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

48 Module SCons.compat._scons_hashlib

hashlib backwards-compatibility module for older (pre-2.5) Python versions

This does not not NOT (repeat, *NOT*) provide complete hashlib functionality. It only wraps the portions of MD5 functionality used by SCons, in an interface that looks like hashlib (or enough for our purposes, anyway). In fact, this module will raise an ImportError if the underlying md5 module isn't available.

48.1 Functions

md5(*string*='')

48.2 Variables

Name	Description
__doc__	**Value:** `...`
__revision__	**Value:** `'src/engine/SCons/compat/_scons_hashlib.py 5134 2010/08/1...`
__package__	**Value:** `'SCons.compat'`
__warningregistry__	**Value:** `{('the md5 module is deprecated; use hashlib instead', <t...`

48.3 Class md5obj

object ┐
　　SCons.compat._scons_hashlib.md5obj

48.3.1 Methods

__init__(*self*, *name*, *string*='')

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

copy(*self*)

digest(*self*)

update(*self*, *arg*)

hexdigest(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

48.3.2 Properties

Name	Description
Inherited from object	
__class__	

48.3.3 Class Variables

Name	Description
md5_module	**Value:** `<module 'md5' from '/usr/lib/python2.6/md5.pyc'>`

48.4 Class md5obj

object
└ **SCons.compat._scons_hashlib.md5obj**

48.4.1 Methods

__init__(*self*, *name*, *string*='')

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__repr__(*self*)

repr(x) Overrides: object.__repr__ extit(inherited documentation)

copy(*self*)

digest(*self*)

update(*self*, *arg*)

hexdigest(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

48.4.2 Properties

Name	Description
Inherited from object	
__class__	

48.4.3 Class Variables

Name	Description
md5_module	**Value:** `<module 'md5' from '/usr/lib/python2.6/md5.pyc'>`

49 Module SCons.compat._scons_io

io compatibility module for older (pre-2.6) Python versions

This does not not NOT (repeat, *NOT*) provide complete io functionality. It only wraps the portions of io functionality used by SCons, in an interface that looks enough like io for our purposes.

49.1 Variables

Name	Description
__doc__	**Value:** `...`
__revision__	**Value:** `'src/engine/SCons/compat/_scons_io.py 5134 2010/08/16 23:...`
__package__	**Value:** `'SCons.compat'`

50 Module SCons.compat._scons_sets

Classes to represent arbitrary sets (including sets of sets).

This module implements sets using dictionaries whose values are ignored. The usual operations (union, intersection, deletion, etc.) are provided as both methods and operators.

Important: sets are not sequences! While they support 'x in s', 'len(s)', and 'for x in s', none of those operations are unique for sequences; for example, mappings support all three as well. The characteristic operation for sequences is subscripting with small integers: s[i], for i in range(len(s)). Sets don't support subscripting at all. Also, sequences allow multiple occurrences and their elements have a definite order; sets on the other hand don't record multiple occurrences and don't remember the order of element insertion (which is why they don't support s[i]).

The following classes are provided:

BaseSet -- All the operations common to both mutable and immutable sets. This is an abstract class, not meant to be directly instantiated.

Set -- Mutable sets, subclass of BaseSet; not hashable.

ImmutableSet -- Immutable sets, subclass of BaseSet; hashable. An iterable argument is mandatory to create an ImmutableSet.

_TemporarilyImmutableSet -- A wrapper around a Set, hashable, giving the same hash value as the immutable set equivalent would have. Do not use this class directly.

Only hashable objects can be added to a Set. In particular, you cannot really add a Set as an element to another Set; if you try, what is actually added is an ImmutableSet built from it (it compares equal to the one you tried adding).

When you ask if 'x in y' where x is a Set and y is a Set or ImmutableSet, x is wrapped into a _TemporarilyImmutableSet z, and what's tested is actually 'z in y'.

50.1 Class BaseSet

object
└ **SCons.compat._scons_sets.BaseSet**

Known Subclasses: SCons.compat._scons_sets.ImmutableSet, SCons.compat._scons_sets.Set, SCons.compat._scons_sets._TemporarilyImmutableSet

Common base class for mutable and immutable sets.

50.1.1 Methods

__init__(*self*)

This is an abstract class. Overrides: object.__init__

__len__(*self*)

Return the number of elements of a set.

__repr__(*self*)

Return string representation of a set.

This looks like 'Set([<list of elements>])'. Overrides: object.__repr__

__str__(*self*)

Return string representation of a set.

This looks like 'Set([<list of elements>])'. Overrides: object.__str__

__iter__(*self*)

Return an iterator over the elements or a set.

This is the keys iterator for the underlying dict.

__cmp__(*self, other*)

__eq__(*self, other*)

__ne__(*self, other*)

copy(*self*)

Return a shallow copy of a set.

__copy__(*self*)

Return a shallow copy of a set.

__deepcopy__(*self*, *memo*)

Return a deep copy of a set; used by copy module.

__or__(*self*, *other*)

Return the union of two sets as a new set.

(I.e. all elements that are in either set.)

union(*self*, *other*)

Return the union of two sets as a new set.

(I.e. all elements that are in either set.)

__and__(*self*, *other*)

Return the intersection of two sets as a new set.

(I.e. all elements that are in both sets.)

intersection(*self*, *other*)

Return the intersection of two sets as a new set.

(I.e. all elements that are in both sets.)

__xor__(*self*, *other*)

Return the symmetric difference of two sets as a new set.

(I.e. all elements that are in exactly one of the sets.)

symmetric_difference(*self*, *other*)

Return the symmetric difference of two sets as a new set.

(I.e. all elements that are in exactly one of the sets.)

__sub__(*self*, *other*)

Return the difference of two sets as a new Set.

(I.e. all elements that are in this set and not in the other.)

difference(*self*, *other*)

Return the difference of two sets as a new Set.

(I.e. all elements that are in this set and not in the other.)

__contains__(*self*, *element*)

Report whether an element is a member of a set.

(Called in response to the expression 'element in self'.)

issubset(*self*, *other*)

Report whether another set contains this set.

issuperset(*self*, *other*)

Report whether this set contains another set.

__le__(*self*, *other*)

Report whether another set contains this set.

__ge__(*self*, *other*)

Report whether this set contains another set.

__lt__(*self*, *other*)

__gt__(*self*, *other*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __subclasshook__()

50.1.2 Properties

Name	Description
Inherited from object	
__class__	

50.2 Class ImmutableSet

object
SCons.compat._scons_sets.BaseSet
SCons.compat._scons_sets.ImmutableSet

Immutable set class.

50.2.1 Methods

__init__(*self*, *iterable*=`None`)

Construct an immutable set from an optional iterable. Overrides: object.__init__

__hash__(*self*)

hash(x) Overrides: object.__hash__ extit(inherited documentation)

__getstate__(*self*)

__setstate__(*self*, *state*)

Inherited from SCons.compat._scons_sets.BaseSet(Section 50.1)

__and__(), __cmp__(), __contains__(), __copy__(), __deepcopy__(), __eq__(), __ge__(), __gt__(), __iter__(), __le__(), __len__(), __lt__(), __ne__(), __or__(), __repr__(), __str__(), __sub__(), __xor__(), copy(), difference(), intersection(), issubset(), issuperset(), symmetric_difference(), union()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __subclasshook__()

50.2.2 Properties

Name	Description
Inherited from object	
__class__	

50.3 Class Set

object
SCons.compat._scons_sets.BaseSet
SCons.compat._scons_sets.Set

Mutable set class.

50.3.1 Methods

__init__(*self*, *iterable*=`None`)

Construct a set from an optional iterable. Overrides: object.__init__

__getstate__(*self*)

__setstate__(*self*, *data*)

__hash__(*self*)

A Set cannot be hashed. Overrides: object.__hash__

__ior__(*self*, *other*)

Update a set with the union of itself and another.

union_update(*self*, *other*)

Update a set with the union of itself and another.

__iand__(*self*, *other*)

Update a set with the intersection of itself and another.

intersection_update(*self*, *other*)

Update a set with the intersection of itself and another.

__ixor__(*self*, *other*)

Update a set with the symmetric difference of itself and another.

symmetric_difference_update(*self*, *other*)

Update a set with the symmetric difference of itself and another.

__isub__(*self*, *other*)

Remove all elements of another set from this set.

difference_update(*self*, *other*)

Remove all elements of another set from this set.

update(*self*, *iterable*)

Add all values from an iterable (such as a list or file).

clear(*self*)

Remove all elements from this set.

add(*self*, *element*)

Add an element to a set.

This has no effect if the element is already present.

remove(*self*, *element*)

Remove an element from a set; it must be a member.

If the element is not a member, raise a KeyError.

discard(*self*, *element*)

Remove an element from a set if it is a member.

If the element is not a member, do nothing.

pop(*self*)

Remove and return an arbitrary set element.

__as_immutable__(*self*)

__as_temporarily_immutable__(*self*)

Inherited from SCons.compat._scons_sets.BaseSet(Section 50.1)

__and__(), __cmp__(), __contains__(), __copy__(), __deepcopy__(), __eq__(), __ge__(), __gt__(), __iter__(), __le__(), __len__(), __lt__(), __ne__(), __or__(), __repr__(), __str__(), __sub__(), __xor__(), copy(), difference(), intersection(), issubset(), issuperset(), symmetric_difference(), union()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __subclasshook__()

50.3.2 Properties

Name	Description
Inherited from object	
__class__	

51 Module SCons.compat._scons_subprocess

```
subprocess - Subprocesses with accessible I/O streams

This module allows you to spawn processes, connect to their
input/output/error pipes, and obtain their return codes.  This module
intends to replace several other, older modules and functions, like:

os.system
os.spawn*
os.popen*
popen2.*
commands.*

Information about how the subprocess module can be used to replace these
modules and functions can be found below.



Using the subprocess module
===========================
This module defines one class called Popen:

class Popen(args, bufsize=0, executable=None,
            stdin=None, stdout=None, stderr=None,
            preexec_fn=None, close_fds=False, shell=False,
            cwd=None, env=None, universal_newlines=False,
            startupinfo=None, creationflags=0):


Arguments are:

args should be a string, or a sequence of program arguments.  The
program to execute is normally the first item in the args sequence or
string, but can be explicitly set by using the executable argument.

On UNIX, with shell=False (default): In this case, the Popen class
uses os.execvp() to execute the child program.  args should normally
be a sequence.  A string will be treated as a sequence with the string
as the only item (the program to execute).

On UNIX, with shell=True: If args is a string, it specifies the
command string to execute through the shell.  If args is a sequence,
the first item specifies the command string, and any additional items
will be treated as additional shell arguments.
```

```
On Windows: the Popen class uses CreateProcess() to execute the child
program, which operates on strings.  If args is a sequence, it will be
converted to a string using the list2cmdline method.  Please note that
not all MS Windows applications interpret the command line the same
way: The list2cmdline is designed for applications using the same
rules as the MS C runtime.

bufsize, if given, has the same meaning as the corresponding argument
to the built-in open() function: 0 means unbuffered, 1 means line
buffered, any other positive value means use a buffer of
(approximately) that size.  A negative bufsize means to use the system
default, which usually means fully buffered.  The default value for
bufsize is 0 (unbuffered).

stdin, stdout and stderr specify the executed programs' standard
input, standard output and standard error file handles, respectively.
Valid values are PIPE, an existing file descriptor (a positive
integer), an existing file object, and None.  PIPE indicates that a
new pipe to the child should be created.  With None, no redirection
will occur; the child's file handles will be inherited from the
parent.  Additionally, stderr can be STDOUT, which indicates that the
stderr data from the applications should be captured into the same
file handle as for stdout.

If preexec_fn is set to a callable object, this object will be called
in the child process just before the child is executed.

If close_fds is true, all file descriptors except 0, 1 and 2 will be
closed before the child process is executed.

if shell is true, the specified command will be executed through the
shell.

If cwd is not None, the current directory will be changed to cwd
before the child is executed.

If env is not None, it defines the environment variables for the new
process.

If universal_newlines is true, the file objects stdout and stderr are
opened as a text files, but lines may be terminated by any of '\n',
the Unix end-of-line convention, '\r', the Macintosh convention or
'\r\n', the Windows convention.  All of these external representations
are seen as '\n' by the Python program.  Note: This feature is only
```

```
available if Python is built with universal newline support (the
default).  Also, the newlines attribute of the file objects stdout,
stdin and stderr are not updated by the communicate() method.

The startupinfo and creationflags, if given, will be passed to the
underlying CreateProcess() function.  They can specify things such as
appearance of the main window and priority for the new process.
(Windows only)


This module also defines two shortcut functions:

call(*popenargs, **kwargs):
    Run command with arguments.  Wait for command to complete, then
    return the returncode attribute.

    The arguments are the same as for the Popen constructor.  Example:

    retcode = call(["ls", "-l"])

check_call(*popenargs, **kwargs):
    Run command with arguments.  Wait for command to complete.  If the
    exit code was zero then return, otherwise raise
    CalledProcessError.  The CalledProcessError object will have the
    return code in the returncode attribute.

    The arguments are the same as for the Popen constructor.  Example:

    check_call(["ls", "-l"])

Exceptions
----------
Exceptions raised in the child process, before the new program has
started to execute, will be re-raised in the parent.  Additionally,
the exception object will have one extra attribute called
'child_traceback', which is a string containing traceback information
from the childs point of view.

The most common exception raised is OSError.  This occurs, for
example, when trying to execute a non-existent file.  Applications
should prepare for OSErrors.

A ValueError will be raised if Popen is called with invalid arguments.

check_call() will raise CalledProcessError, if the called process
```

```
returns a non-zero return code.


Security
--------
Unlike some other popen functions, this implementation will never call
/bin/sh implicitly.  This means that all characters, including shell
metacharacters, can safely be passed to child processes.


Popen objects
=============
Instances of the Popen class have the following methods:

poll()
    Check if child process has terminated.  Returns returncode
    attribute.

wait()
    Wait for child process to terminate.  Returns returncode attribute.

communicate(input=None)
    Interact with process: Send data to stdin.  Read data from stdout
    and stderr, until end-of-file is reached.  Wait for process to
    terminate.  The optional stdin argument should be a string to be
    sent to the child process, or None, if no data should be sent to
    the child.

    communicate() returns a tuple (stdout, stderr).

    Note: The data read is buffered in memory, so do not use this
    method if the data size is large or unlimited.

The following attributes are also available:

stdin
    If the stdin argument is PIPE, this attribute is a file object
    that provides input to the child process.  Otherwise, it is None.

stdout
    If the stdout argument is PIPE, this attribute is a file object
    that provides output from the child process.  Otherwise, it is
    None.

stderr
```

```
    If the stderr argument is PIPE, this attribute is file object that
    provides error output from the child process.  Otherwise, it is
    None.

pid
    The process ID of the child process.

returncode
    The child return code.  A None value indicates that the process
    hasn't terminated yet.  A negative value -N indicates that the
    child was terminated by signal N (UNIX only).


Replacing older functions with the subprocess module
====================================================
In this section, "a ==> b" means that b can be used as a replacement
for a.

Note: All functions in this section fail (more or less) silently if
the executed program cannot be found; this module raises an OSError
exception.

In the following examples, we assume that the subprocess module is
imported with "from subprocess import *".


Replacing /bin/sh shell backquote
---------------------------------
output=`mycmd myarg`
==>
output = Popen(["mycmd", "myarg"], stdout=PIPE).communicate()[0]


Replacing shell pipe line
-------------------------
output=`dmesg | grep hda`
==>
p1 = Popen(["dmesg"], stdout=PIPE)
p2 = Popen(["grep", "hda"], stdin=p1.stdout, stdout=PIPE)
output = p2.communicate()[0]


Replacing os.system()
---------------------
sts = os.system("mycmd" + " myarg")
```

```
==>
p = Popen("mycmd" + " myarg", shell=True)
pid, sts = os.waitpid(p.pid, 0)

Note:

* Calling the program through the shell is usually not required.

* It's easier to look at the returncode attribute than the
  exitstatus.

A more real-world example would look like this:

try:
    retcode = call("mycmd" + " myarg", shell=True)
    if retcode < 0:
        print >>sys.stderr, "Child was terminated by signal", -retcode
    else:
        print >>sys.stderr, "Child returned", retcode
except OSError, e:
    print >>sys.stderr, "Execution failed:", e


Replacing os.spawn*
-------------------
P_NOWAIT example:

pid = os.spawnlp(os.P_NOWAIT, "/bin/mycmd", "mycmd", "myarg")
==>
pid = Popen(["/bin/mycmd", "myarg"]).pid


P_WAIT example:

retcode = os.spawnlp(os.P_WAIT, "/bin/mycmd", "mycmd", "myarg")
==>
retcode = call(["/bin/mycmd", "myarg"])


Vector example:

os.spawnvp(os.P_NOWAIT, path, args)
==>
Popen([path] + args[1:])
```

```
Environment example:

os.spawnlpe(os.P_NOWAIT, "/bin/mycmd", "mycmd", "myarg", env)
==>
Popen(["/bin/mycmd", "myarg"], env={"PATH": "/usr/bin"})


Replacing os.popen*
-------------------
pipe = os.popen(cmd, mode='r', bufsize)
==>
pipe = Popen(cmd, shell=True, bufsize=bufsize, stdout=PIPE).stdout

pipe = os.popen(cmd, mode='w', bufsize)
==>
pipe = Popen(cmd, shell=True, bufsize=bufsize, stdin=PIPE).stdin


(child_stdin, child_stdout) = os.popen2(cmd, mode, bufsize)
==>
p = Popen(cmd, shell=True, bufsize=bufsize,
          stdin=PIPE, stdout=PIPE, close_fds=True)
(child_stdin, child_stdout) = (p.stdin, p.stdout)


(child_stdin,
 child_stdout,
 child_stderr) = os.popen3(cmd, mode, bufsize)
==>
p = Popen(cmd, shell=True, bufsize=bufsize,
          stdin=PIPE, stdout=PIPE, stderr=PIPE, close_fds=True)
(child_stdin,
 child_stdout,
 child_stderr) = (p.stdin, p.stdout, p.stderr)


(child_stdin, child_stdout_and_stderr) = os.popen4(cmd, mode, bufsize)
==>
p = Popen(cmd, shell=True, bufsize=bufsize,
          stdin=PIPE, stdout=PIPE, stderr=STDOUT, close_fds=True)
(child_stdin, child_stdout_and_stderr) = (p.stdin, p.stdout)


Replacing popen2.*
```

```
------------------
Note: If the cmd argument to popen2 functions is a string, the command
is executed through /bin/sh.  If it is a list, the command is directly
executed.

(child_stdout, child_stdin) = popen2.popen2("somestring", bufsize, mode)
==>
p = Popen(["somestring"], shell=True, bufsize=bufsize
          stdin=PIPE, stdout=PIPE, close_fds=True)
(child_stdout, child_stdin) = (p.stdout, p.stdin)


(child_stdout, child_stdin) = popen2.popen2(["mycmd", "myarg"], bufsize, mode)
==>
p = Popen(["mycmd", "myarg"], bufsize=bufsize,
          stdin=PIPE, stdout=PIPE, close_fds=True)
(child_stdout, child_stdin) = (p.stdout, p.stdin)

The popen2.Popen3 and popen3.Popen4 basically works as subprocess.Popen,
except that:

* subprocess.Popen raises an exception if the execution fails
* the capturestderr argument is replaced with the stderr argument.
* stdin=PIPE and stdout=PIPE must be specified.
* popen2 closes all filedescriptors by default, but you have to specify
  close_fds=True with subprocess.Popen.
```

51.1 Functions

call(**popenargs*, ***kwargs*)

Run command with arguments. Wait for command to complete, then return the returncode attribute.

The arguments are the same as for the Popen constructor. Example:

retcode = call([“ls”, “-l”])

check_call(**popenargs*, ***kwargs*)

Run command with arguments. Wait for command to complete. If the exit code was zero then return, otherwise raise CalledProcessError. The CalledProcessError object will have the return code in the returncode attribute.

The arguments are the same as for the Popen constructor. Example:

check_call([“ls”, “-l”])

51.2 Variables

Name	Description
PIPE	**Value:** `-1`
STDOUT	**Value:** `-2`

51.3 Class CalledProcessError

object ┐

exceptions.BaseException ┐

exceptions.Exception ┐

SCons.compat._scons_subprocess.CalledProcessError

This exception is raised when a process run by check_call() returns a non-zero exit status. The exit status will be stored in the returncode attribute.

51.3.1 Methods

__init__(*self*, *returncode*, *cmd*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__str__(*self*)

str(x) Overrides: object.__str__ extit(inherited documentation)

Inherited from exceptions.Exception

__new__()

Inherited from exceptions.BaseException

__delattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __unicode__()

Inherited from object

__format__(), __hash__(), __reduce_ex__(), __sizeof__(), __subclasshook__()

51.3.2 Properties

Name	Description
Inherited from exceptions.BaseException	
args, message	
Inherited from object	
__class__	

51.4 Class Popen

object
└ **SCons.compat._scons_subprocess.Popen**

51.4.1 Methods

__init__(*self*, *args*, *bufsize*=`0`, *executable*=`None`, *stdin*=`None`, *stdout*=`None`, *stderr*=`None`, *preexec_fn*=`None`, *close_fds*=`False`, *shell*=`False`, *cwd*=`None`, *env*=`None`, *universal_newlines*=`False`, *startupinfo*=`None`, *creationflags*=`0`)

Create new Popen instance. Overrides: object.__init__

__del__(*self*)

communicate(*self*, *input*=`None`)

Interact with process: Send data to stdin. Read data from stdout and stderr, until end-of-file is reached. Wait for process to terminate. The optional input argument should be a string to be sent to the child process, or None, if no data should be sent to the child.

communicate() returns a tuple (stdout, stderr).

poll(*self*, *_deadstate*=`None`)

Check if child process has terminated. Returns returncode attribute.

wait(*self*)

Wait for child process to terminate. Returns returncode attribute.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

51.4.2 Properties

Name	Description
Inherited from object	
__class__	

52 Module SCons.cpp

SCons C Pre-Processor module

52.1 Functions

CPP_to_Python_Ops_Sub(*m*)

CPP_to_Python(*s*)

Converts a C pre-processor expression into an equivalent Python expression that can be evaluated.

52.2 Variables

Name	Description
__doc__	**Value:** `...`
cpp_lines_dict	**Value:** `{('define'): '\\s+([_A-Za-z][_A-Za-z0-9_]*)(\\([^)]*\\))?...`
Table	**Value:** `{'define': re.compile(r'\s+([_A-Za-z][_A-Za-z0-9_]*)(\([^...`
e	**Value:** `'^\\s*#\\s*(elif|undef|include_next|endif|else|include|if...`
CPP_Expression	**Value:** `re.compile(r'(?m)^\s*#\s*(elif|undef|include_next|endif|e...`
CPP_to_Python_Ops_Dict	**Value:** `{'\r': '', '!': ' not ', '!=': ' != ', '&&': ' and ', ':'...`
CPP_to_Python_Ops_Expression	**Value:** `re.compile(r'|||&&|!=|!|\r|:|\?')`
CPP_to_Python_Eval_List	**Value:** `[[re.compile(r'defined\s+(\w+)'), '"\\1" in __dict__'], [...`
line_continuations	**Value:** `re.compile(r'\\\r?\n')`
function_name	**Value:** `re.compile(r'(\S+)\(([^\)]*)\)')`
function_arg_separator	**Value:** `re.compile(r',\s*')`
__package__	**Value:** `'SCons'`
x	**Value:** `'if'`

52.3 Class FunctionEvaluator

object ─┐
SCons.cpp.FunctionEvaluator

Handles delayed evaluation of a #define function call.

52.3.1 Methods

__init__(*self*, *name*, *args*, *expansion*)

Squirrels away the arguments and expansion value of a #define macro function for later evaluation when we must actually expand a value that uses it.
Overrides: object.__init__

__call__(*self*, **values*)

Evaluates the expansion of a #define macro function called with the specified values.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

52.3.2 Properties

Name	Description
Inherited from object	
__class__	

52.4 Class PreProcessor

object ─┐
SCons.cpp.PreProcessor

Known Subclasses: SCons.cpp.DumbPreProcessor, SCons.Scanner.C.SConsCPPScanner

The main workhorse class for handling C pre-processing.

52.4.1 Methods

__call__(*self*, *file*)

Pre-processes a file.

This is the main public entry point.

__init__(*self*, *current*=`'.'`, *cpppath*=`()`, *dict*=`{}`, *all*=`0`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

all_include(*self*, *t*)

do_define(*self*, *t*)

Default handling of a #define line.

do_elif(*self*, *t*)

Default handling of a #elif line.

do_else(*self*, *t*)

Default handling of a #else line.

do_endif(*self*, *t*)

Default handling of a #endif line.

do_if(*self*, *t*)

Default handling of a #if line.

do_ifdef(*self*, *t*)

Default handling of a #ifdef line.

do_ifndef(*self*, *t*)

Default handling of a #ifndef line.

do_import(*self*, *t*)

Default handling of a #import line.

do_include(*self*, *t*)

Default handling of a #include line.

do_include_next(*self*, *t*)

Default handling of a #include line.

do_nothing(*self*, *t*)

Null method for when we explicitly want the action for a specific preprocessor directive to do nothing.

do_undef(*self*, *t*)

Default handling of a #undef line.

eval_expression(*self*, *t*)

Evaluates a C preprocessor expression.

This is done by converting it to a Python equivalent and eval()ing it in the C preprocessor namespace we use to track #define values.

finalize_result(*self*, *fname*)

find_include_file(*self*, *t*)

Finds the #include file for a given preprocessor tuple.

initialize_result(*self*, *fname*)

process_contents(*self*, *contents*, *fname*=`None`)

Pre-processes a file contents.

This is the main internal entry point.

read_file(*self*, *file*)

resolve_include(*self*, *t*)

```
Resolve a tuple-ized #include line.

This handles recursive expansion of values without "" or <>
surrounding the name until an initial " or < is found, to handle
        #include FILE
where FILE is a #define somewhere else.
```

restore(*self*)

Pops the previous dispatch table off the stack and makes it the current one.

save(*self*)

Pushes the current dispatch table on the stack and re-initializes the current dispatch table to the default.

scons_current_file(*self*, *t*)

start_handling_includes(*self*, *t*=`None`)

Causes the PreProcessor object to start processing #import, #include and #include_next lines.

This method will be called when a #if, #ifdef, #ifndef or #elif evaluates True, or when we reach the #else in a #if, #ifdef, #ifndef or #elif block where a condition already evaluated False.

stop_handling_includes(*self*, *t*=`None`)

Causes the PreProcessor object to stop processing #import, #include and #include_next lines.

This method will be called when a #if, #ifdef, #ifndef or #elif evaluates False, or when we reach the #else in a #if, #ifdef, #ifndef or #elif block where a condition already evaluated True.

tupleize(*self*, *contents*)

Turns the contents of a file into a list of easily-processed tuples describing the CPP lines in the file.

The first element of each tuple is the line's preprocessor directive (#if, #include, #define, etc., minus the initial '#'). The remaining elements are specific to the type of directive, as pulled apart by the regular expression.

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

52.4.2 Properties

Name	Description
Inherited from object	
__class__	

52.5 Class DumbPreProcessor

object ─┐

SCons.cpp.PreProcessor ─┐

SCons.cpp.DumbPreProcessor

A preprocessor that ignores all #if/#elif/#else/#endif directives and just reports back *all* of the #include files (like the classic SCons scanner did).

This is functionally equivalent to using a regular expression to find all of the #include lines, only slower. It exists mainly as an example of how the main PreProcessor class can be sub-classed to tailor its behavior.

52.5.1 Methods

__init__(*self*, **args*, ***kw*)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

Inherited from SCons.cpp.PreProcessor(Section 52.4)

__call__(), all_include(), do_define(), do_elif(), do_else(), do_endif(), do_if(), do_ifdef(), do_ifndef(), do_import(), do_include(), do_include_next(), do_nothing(), do_undef(), eval_expression(), finalize_result(), find_include_file(), initialize_result(), process_contents(), read_file(), resolve_include(), restore(), save(), scons_current_file(), start_handling_includes(), stop_handling_includes(), tupleize()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

52.5.2 Properties

Name	Description
Inherited from object	
__class__	

53 Module SCons.dblite

53.1 Functions

corruption_warning(*filename*)

is_string(*s*)

unicode(*s*)

open(*file*, *flag*=`None`, *mode*=`438`)

53.2 Variables

Name	Description
keep_all_files	**Value:** `0`
ignore_corrupt_dbfiles	**Value:** `0`
dblite_suffix	**Value:** `'.dblite'`
tmp_suffix	**Value:** `'.tmp'`
__package__	**Value:** `'SCons'`

53.3 Class dblite

object ┐
SCons.dblite.dblite

53.3.1 Methods

__init__(*self*, *file_base_name*, *flag*, *mode*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: object.__init__ extit(inherited documentation)

__del__(*self*)

sync(*self*)

__getitem__(*self*, *key*)

__setitem__(*self*, *key*, *value*)

keys(*self*)

has_key(*self*, *key*)

__contains__(*self*, *key*)

iterkeys(*self*)

__iter__(*self*)

__len__(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

53.3.2 Properties

Name	Description
Inherited from object	
__class__	

54 Module SCons.exitfuncs

SCons.exitfuncs

Register functions which are executed when SCons exits for any reason.

54.1 Functions

register(*func*, **targs*, ***kargs*)

register a function to be executed upon normal program termination

func - function to be called at exit targs - optional arguments to pass to func kargs - optional keyword arguments to pass to func

54.2 Variables

Name	Description
__revision__	**Value:** `'src/engine/SCons/exitfuncs.py 5134 2010/08/16 23:02:40 b...`
x	**Value:** `sys.exitfunc`
__package__	**Value:** `'SCons'`

55 Module md5

55.1 Variables

Name	Description
__package__	**Value:** `None`
blocksize	**Value:** `1`
digest_size	**Value:** `16`

Index