

SCons

API Documentation

August 12, 2008

Contents

Contents	1
1 Package SCons	2
1.1 Modules	2
1.2 Variables	4
2 Module SCons.Action	5
2.1 Functions	5
2.2 Variables	6
2.3 Class ActionBase	6
2.3.1 Methods	6
2.4 Class CommandAction	7
2.4.1 Methods	7
2.5 Class CommandGeneratorAction	8
2.5.1 Methods	8
2.6 Class LazyAction	9
2.6.1 Methods	9
2.7 Class FunctionAction	10
2.7.1 Methods	10
2.8 Class ListAction	11
2.8.1 Methods	11
2.9 Class ActionCaller	12
2.9.1 Methods	12
2.10 Class ActionFactory	12
2.10.1 Methods	13
3 Module SCons.Builder	14
3.1 Functions	15
3.2 Variables	15
3.3 Class DictCmdGenerator	15
3.3.1 Methods	16
3.4 Class CallableSelector	17
3.4.1 Methods	17
3.5 Class DictEmitter	19
3.5.1 Methods	19
3.6 Class ListEmitter	20
3.6.1 Methods	20
3.7 Class OverrideWarner	22
3.7.1 Methods	22

3.8	Class EmitterProxy	23
3.8.1	Methods	23
3.9	Class BuilderBase	23
3.9.1	Methods	23
3.9.2	Class Variables	25
3.10	Class CompositeBuilder	25
3.10.1	Methods	25
4	Module SCons.CacheDir	27
4.1	Functions	27
4.2	Variables	27
4.3	Class CacheDir	27
4.3.1	Methods	27
5	Module SCons.Conftest	29
5.1	Functions	29
5.2	Variables	30
6	Module SCons.Debug	31
6.1	Functions	31
6.2	Variables	31
7	Module SCons.Defaults	33
7.1	Functions	33
7.2	Variables	33
7.3	Class NullCmdGenerator	34
7.3.1	Methods	35
7.4	Class Variable_Method_Caller	35
7.4.1	Methods	35
8	Module SCons.Environment	36
8.1	Functions	36
8.2	Variables	36
8.3	Class MethodWrapper	36
8.3.1	Methods	37
8.4	Class BuilderWrapper	37
8.4.1	Methods	37
8.5	Class BuilderDict	38
8.5.1	Methods	38
8.6	Class SubstitutionEnvironment	39
8.6.1	Methods	39
8.6.2	Class Variables	41
8.7	Class Base	41
8.7.1	Methods	42
8.7.2	Class Variables	48
8.8	Class OverrideEnvironment	48
8.8.1	Methods	49
8.8.2	Class Variables	55
8.9	Class Base	56
8.9.1	Methods	56
8.9.2	Class Variables	63
9	Module SCons.Errors	64
9.1	Variables	64

9.2	Class BuildError	64
9.2.1	Methods	64
9.2.2	Properties	65
9.3	Class InternalError	66
9.3.1	Methods	66
9.3.2	Properties	67
9.4	Class UserError	67
9.4.1	Methods	67
9.4.2	Properties	69
9.5	Class StopError	69
9.5.1	Methods	69
9.5.2	Properties	70
9.6	Class EnvironmentError	71
9.6.1	Methods	71
9.6.2	Properties	72
9.7	Class ExplicitExit	72
9.7.1	Methods	72
9.7.2	Properties	74
9.8	Class TaskmasterException	74
9.8.1	Methods	74
9.8.2	Properties	75
10	Module SCons.Executor	76
10.1	Variables	76
10.2	Class Executor	76
10.2.1	Methods	76
10.2.2	Class Variables	78
10.3	Class Null	78
10.3.1	Methods	78
10.3.2	Class Variables	79
11	Module SCons.Job	81
11.1	Variables	81
11.2	Class InterruptState	81
11.2.1	Methods	81
11.3	Class Jobs	81
11.3.1	Methods	81
11.4	Class Serial	82
11.4.1	Methods	82
11.5	Class Worker	82
11.5.1	Methods	82
11.5.2	Properties	84
11.6	Class ThreadPool	84
11.6.1	Methods	84
11.7	Class Parallel	84
11.7.1	Methods	84
12	Module SCons.Memoize	86
12.1	Functions	87
12.2	Variables	88
12.3	Class Counter	88
12.3.1	Methods	88
12.4	Class CountValue	88

12.4.1	Methods	88
12.5	Class CountDict	89
12.5.1	Methods	89
12.6	Class Memoizer	89
12.6.1	Methods	89
12.7	Class Memoized_Metaclass	89
12.7.1	Methods	90
12.7.2	Properties	91
13	Package SCons.Node	92
13.1	Modules	92
13.2	Functions	92
13.3	Variables	92
13.4	Class NodeInfoBase	93
13.4.1	Methods	93
13.4.2	Class Variables	93
13.5	Class BuildInfoBase	93
13.5.1	Methods	94
13.5.2	Class Variables	94
13.6	Class Node	94
13.6.1	Methods	94
13.6.2	Class Variables	101
13.7	Class NodeList	101
13.7.1	Methods	102
13.8	Class Walker	103
13.8.1	Methods	103
14	Module SCons.Node.Alias	104
14.1	Variables	104
14.2	Class AliasNameSpace	104
14.2.1	Methods	104
14.3	Class AliasNodeInfo	105
14.3.1	Methods	105
14.3.2	Class Variables	106
14.4	Class AliasBuildInfo	106
14.4.1	Methods	106
14.4.2	Class Variables	106
14.5	Class Alias	106
14.5.1	Methods	106
14.5.2	Class Variables	114
15	Module SCons.Node.FS	115
15.1	Functions	115
15.2	Variables	116
15.3	Class DiskChecker	117
15.3.1	Methods	117
15.4	Class EntryProxy	117
15.4.1	Methods	117
15.4.2	Class Variables	118
15.5	Class Base	118
15.5.1	Methods	118
15.5.2	Class Variables	127
15.6	Class Entry	127

15.6.1	Methods	128
15.6.2	Class Variables	137
15.7	Class LocalFS	137
15.7.1	Methods	137
15.7.2	Class Variables	138
15.8	Class FS	138
15.8.1	Methods	138
15.8.2	Class Variables	140
15.9	Class DirNodeInfo	141
15.9.1	Methods	141
15.9.2	Class Variables	141
15.10	Class DirBuildInfo	141
15.10.1	Methods	142
15.10.2	Class Variables	142
15.11	Class Dir	142
15.11.1	Methods	142
15.11.2	Class Variables	153
15.12	Class RootDir	153
15.12.1	Methods	154
15.12.2	Class Variables	164
15.13	Class FileInfo	165
15.13.1	Methods	165
15.13.2	Class Variables	165
15.14	Class FileBuildInfo	165
15.14.1	Methods	165
15.14.2	Class Variables	166
15.15	Class File	166
15.15.1	Methods	166
15.15.2	Class Variables	177
15.16	Class Finder	178
15.16.1	Methods	178
15.16.2	Class Variables	178
16	Module SCons.Node.Python	179
16.1	Variables	179
16.2	Class ValueNodeInfo	179
16.2.1	Methods	179
16.2.2	Class Variables	179
16.3	Class ValueBuildInfo	180
16.3.1	Methods	180
16.3.2	Class Variables	180
16.4	Class Value	180
16.4.1	Methods	180
16.4.2	Class Variables	188
17	Module SCons.PathList	189
17.1	Functions	189
17.2	Variables	189
18	Module SCons.SConf	190
18.1	Functions	190
18.2	Variables	191
18.3	Class SConfWarning	191

18.3.1	Methods	191
18.3.2	Properties	193
18.4	Class SConfError	193
18.4.1	Methods	193
18.4.2	Properties	194
18.5	Class ConfigureDryRunError	195
18.5.1	Methods	195
18.5.2	Properties	196
18.6	Class ConfigureCacheError	196
18.6.1	Methods	197
18.6.2	Properties	198
18.7	Class SConfBuildInfo	198
18.7.1	Methods	198
18.7.2	Class Variables	199
18.8	Class Streamer	199
18.8.1	Methods	199
18.9	Class SConfBuildTask	199
18.9.1	Methods	200
18.10	Class SConfBase	202
18.10.1	Methods	202
18.11	Class CheckContext	203
18.11.1	Methods	204
19	Module SCons.SConsign	205
19.1	Functions	205
19.2	Variables	205
19.3	Class SConsignEntry	205
19.3.1	Methods	205
19.3.2	Class Variables	206
19.4	Class Base	206
19.4.1	Methods	206
19.5	Class DB	206
19.5.1	Methods	207
19.6	Class Dir	207
19.6.1	Methods	207
19.7	Class DirFile	208
19.7.1	Methods	208
19.8	Class DB	209
19.8.1	Methods	209
20	Package SCons.Scanner	210
20.1	Modules	210
20.2	Functions	210
20.3	Variables	210
20.4	Class FindPathDirs	210
20.4.1	Methods	210
20.5	Class Base	211
20.5.1	Methods	211
20.6	Class Selector	212
20.6.1	Methods	213
20.7	Class Current	214
20.7.1	Methods	215
20.8	Class Classic	216

20.8.1	Methods	216
20.9	Class ClassicCPP	217
20.9.1	Methods	217
21	Module SCons.Scanner.C	219
21.1	Functions	219
21.2	Variables	219
21.3	Class SConsCPPScanner	219
21.3.1	Methods	219
21.4	Class SConsCPPScannerWrapper	222
21.4.1	Methods	222
22	Module SCons.Scanner.D	223
22.1	Functions	223
22.2	Variables	223
22.3	Class D	223
22.3.1	Methods	223
23	Module SCons.Scanner.Dir	225
23.1	Functions	225
23.2	Variables	225
24	Module SCons.Scanner.Fortran	226
24.1	Functions	226
24.2	Variables	226
24.3	Class F90Scanner	226
24.3.1	Methods	226
25	Module SCons.Scanner.IDL	228
25.1	Functions	228
25.2	Variables	228
26	Module SCons.Scanner.LaTeX	229
26.1	Functions	229
26.2	Variables	229
26.3	Class LaTeX	229
26.3.1	Methods	229
27	Module SCons.Scanner.Prog	231
27.1	Functions	231
27.2	Variables	231
28	Package SCons.Script	232
28.1	Modules	232
28.2	Functions	232
28.3	Variables	232
28.4	Class TargetList	238
28.4.1	Methods	238
29	Module SCons.Script.Interactive	240
29.1	Functions	240
29.2	Variables	240
29.3	Class SConsInteractiveCmd	240
29.3.1	Methods	240

29.3.2 Class Variables	242
30 Module SCons.Script.Main	244
30.1 Functions	244
30.2 Variables	244
30.3 Class SConsPrintHelpException	245
30.3.1 Methods	245
30.3.2 Properties	246
30.4 Class Progressor	247
30.4.1 Methods	247
30.4.2 Class Variables	247
30.5 Class BuildTask	247
30.5.1 Methods	248
30.5.2 Class Variables	250
30.6 Class CleanTask	250
30.6.1 Methods	250
30.7 Class QuestionTask	252
30.7.1 Methods	253
30.8 Class TreePrinter	255
30.8.1 Methods	255
30.9 Class FakeOptionParser	255
30.9.1 Methods	255
30.9.2 Class Variables	255
30.10 Class Stats	255
30.10.1 Methods	255
30.11 Class CountStats	256
30.11.1 Methods	256
30.12 Class MemStats	256
30.12.1 Methods	256
31 Module SCons.Script.SConscript'	257
31.1 Functions	257
31.2 Variables	257
31.3 Class SConscriptReturn	258
31.3.1 Methods	258
31.3.2 Properties	259
31.4 Class Frame	259
31.4.1 Methods	259
31.5 Class SConsEnvironment	260
31.5.1 Methods	260
31.5.2 Class Variables	267
31.6 Class DefaultEnvironmentCall	267
31.6.1 Methods	267
32 Module SCons.Sig	268
32.1 Variables	268
32.2 Class MD5Null	268
32.2.1 Methods	268
32.3 Class TimeStampNull	269
32.3.1 Methods	269
33 Module SCons.Subst	270
33.1 Functions	270

33.2	Variables	271
33.3	Class Literal	271
33.3.1	Methods	271
33.4	Class SpecialAttrWrapper	271
33.4.1	Methods	272
33.5	Class CmdStringHolder	272
33.5.1	Methods	272
33.6	Class NLWrapper	275
33.6.1	Methods	275
33.7	Class Targets_or_Sources	275
33.7.1	Methods	275
33.8	Class Target_or_Source	277
33.8.1	Methods	277
34	Module SCons.Taskmaster	278
34.1	Functions	278
34.2	Variables	278
34.3	Class Stats	279
34.3.1	Methods	279
34.4	Class Task	279
34.4.1	Methods	279
34.5	Class Taskmaster	281
34.5.1	Methods	281
35	Module SCons.Util	283
35.1	Functions	283
35.2	Variables	287
35.3	Class CallableComposite	288
35.3.1	Methods	288
35.4	Class NodeList	289
35.4.1	Methods	290
35.5	Class DisplayEngine	291
35.5.1	Methods	291
35.6	Class mystr	292
35.6.1	Methods	292
35.6.2	Properties	300
35.7	Class Proxy	300
35.7.1	Methods	300
35.8	Class _NoError	300
35.8.1	Methods	301
35.8.2	Properties	302
35.9	Class CLVar	302
35.9.1	Methods	302
35.10	Class OrderedDict	304
35.10.1	Methods	304
35.11	Class Selector	305
35.11.1	Methods	305
35.12	Class LogicalLines	307
35.12.1	Methods	307
35.13	Class UniqueList	307
35.13.1	Methods	307
35.14	Class Unbuffered	309
35.14.1	Methods	309

35.15	Class Null	309
35.15.1	Methods	309
36	Package SCons.Variables	310
36.1	Modules	310
36.2	Variables	310
36.3	Class Variables	310
36.3.1	Methods	310
36.3.2	Class Variables	312
37	Module SCons.Variables.BoolVariable'	313
37.1	Functions	313
38	Module SCons.Variables.EnumVariable'	314
38.1	Functions	314
39	Module SCons.Variables.ListVariable'	315
39.1	Functions	315
40	Module SCons.Variables.PackageVariable'	316
40.1	Functions	316
41	Module SCons.Variables.PathVariable'	317
41.1	Variables	317
42	Module SCons.Warnings	319
42.1	Functions	319
42.2	Variables	319
42.3	Class Warning	320
42.3.1	Methods	320
42.3.2	Properties	321
42.4	Class CacheWriteErrorWarning	322
42.4.1	Methods	322
42.4.2	Properties	323
42.5	Class CorruptSConsignWarning	323
42.5.1	Methods	324
42.5.2	Properties	325
42.6	Class DependencyWarning	325
42.6.1	Methods	325
42.6.2	Properties	326
42.7	Class DeprecatedWarning	327
42.7.1	Methods	327
42.7.2	Properties	328
42.8	Class DeprecatedCopyWarning	329
42.8.1	Methods	329
42.8.2	Properties	330
42.9	Class DeprecatedSourceSignaturesWarning	331
42.9.1	Methods	331
42.9.2	Properties	332
42.10	Class DeprecatedTargetSignaturesWarning	333
42.10.1	Methods	333
42.10.2	Properties	334
42.11	Class DuplicateEnvironmentWarning	335
42.11.1	Methods	335

42.11.2 Properties	336
42.12 Class LinkWarning	336
42.12.1 Methods	337
42.12.2 Properties	338
42.13 Class MisleadingKeywordsWarning	338
42.13.1 Methods	338
42.13.2 Properties	339
42.14 Class MissingSConscriptWarning	340
42.14.1 Methods	340
42.14.2 Properties	341
42.15 Class NoMD5ModuleWarning	342
42.15.1 Methods	342
42.15.2 Properties	343
42.16 Class NoMetaclassSupportWarning	343
42.16.1 Methods	344
42.16.2 Properties	345
42.17 Class NoObjectCountWarning	345
42.17.1 Methods	345
42.17.2 Properties	346
42.18 Class NoParallelSupportWarning	347
42.18.1 Methods	347
42.18.2 Properties	348
42.19 Class PythonVersionWarning	349
42.19.1 Methods	349
42.19.2 Properties	350
42.20 Class ReservedVariableWarning	351
42.20.1 Methods	351
42.20.2 Properties	352
42.21 Class StackSizeWarning	352
42.21.1 Methods	353
42.21.2 Properties	354
42.22 Class FortranCxxMixWarning	354
42.22.1 Methods	354
42.22.2 Properties	356
43 Package SCons.compat	357
43.1 Modules	357
43.2 Functions	358
43.3 Variables	358
44 Module SCons.compat._scons_UserString	359
44.1 Functions	359
44.2 Variables	359
44.3 Class UserString	359
44.3.1 Methods	359
45 Module SCons.compat._scons_hashlib	361
45.1 Functions	361
45.2 Variables	361
45.3 Class md5obj	361
45.3.1 Methods	361
45.3.2 Class Variables	361
45.4 Class md5obj	362

45.4.1	Methods	362
45.4.2	Class Variables	362
46	Module SCons.compat._scons_itertools	363
46.1	Functions	363
46.2	Variables	363
47	Module SCons.compat._scons_optparse	365
47.1	Variables	365
47.2	Class OptParseError	366
47.2.1	Methods	366
47.2.2	Properties	367
47.3	Class OptionError	367
47.3.1	Methods	368
47.3.2	Properties	369
47.4	Class OptionConflictError	369
47.4.1	Methods	369
47.4.2	Properties	371
47.5	Class OptionValueError	371
47.5.1	Methods	371
47.5.2	Properties	372
47.6	Class BadOptionError	373
47.6.1	Methods	373
47.6.2	Properties	374
47.7	Class HelpFormatter	374
47.7.1	Methods	375
47.7.2	Class Variables	376
47.8	Class IndentedHelpFormatter	376
47.8.1	Methods	376
47.8.2	Class Variables	377
47.9	Class TitledHelpFormatter	377
47.9.1	Methods	377
47.9.2	Class Variables	378
47.10	Class Option	378
47.10.1	Methods	378
47.10.2	Class Variables	379
47.11	Class Values	379
47.11.1	Methods	379
47.12	Class OptionContainer	379
47.12.1	Methods	380
47.13	Class OptionGroup	381
47.13.1	Methods	381
47.14	Class OptionParser	381
47.14.1	Methods	383
47.14.2	Class Variables	385
48	Module SCons.compat._scons_sets	386
48.1	Class BaseSet	386
48.1.1	Methods	386
48.1.2	Properties	389
48.2	Class ImmutableSet	389
48.2.1	Methods	390
48.2.2	Properties	392

48.3 Class Set	393
48.3.1 Methods	393
48.3.2 Properties	397
49 Module SCons.compat._scons_sets15	398
49.1 Class Set	398
49.1.1 Methods	398
50 Module SCons.compat._scons_shlex	400
50.1 Functions	400
50.2 Class shlex	400
50.2.1 Methods	400
51 Module SCons.compat._scons_subprocess	401
51.1 Functions	407
51.2 Variables	407
51.3 Class CalledProcessError	408
51.3.1 Methods	408
51.3.2 Properties	409
51.4 Class Popen	409
51.4.1 Methods	409
51.4.2 Properties	411
52 Module SCons.compat._scons_textwrap	412
52.1 Functions	412
52.2 Class TextWrapper	412
52.2.1 Methods	413
52.2.2 Class Variables	413
53 Module SCons.compat.builtins	414
53.1 Functions	414
53.2 Variables	414
54 Module SCons.cpp	416
54.1 Functions	416
54.2 Variables	416
54.3 Class FunctionEvaluator	416
54.3.1 Methods	416
54.4 Class PreProcessor	417
54.4.1 Methods	417
54.5 Class DumbPreProcessor	419
54.5.1 Methods	419
55 Module SCons.dblite	423
55.1 Functions	423
55.2 Variables	423
55.3 Class dblite	423
55.3.1 Methods	423
56 Module SCons.exitfuncs	424
56.1 Functions	424
56.2 Variables	424
57 Module md5	425

57.1 Variables	425
--------------------------	-----

1 Package SCons

SCons

The main package for the SCons software construction utility. **Version:** 1.0.0

Date: 2008/08/12 07:31:01

1.1 Modules

- **Action:** SCons.Action
(Section 2, p. 5)
- **Builder:** SCons.Builder
Builder object subsystem.
(Section 3, p. 14)
- **CacheDir:** CacheDir support
(Section 4, p. 27)
- **Conftest:** SCons.Conftest
(Section 5, p. 29)
- **Debug:** SCons.Debug
(Section 6, p. 31)
- **Defaults:** SCons.Defaults
(Section 7, p. 33)
- **Environment:** SCons.Environment
(Section 8, p. 36)
- **Errors:** SCons.Errors
(Section 9, p. 64)
- **Executor:** SCons.Executor
(Section 10, p. 76)
- **Job:** SCons.Job
(Section 11, p. 81)
- **Memoize:** Memoizer
A metaclass implementation to count hits and misses of the computed values that various methods cache in memory.
(Section 12, p. 86)
- **Node:** SCons.Node
(Section 13, p. 92)
 - **Alias:** sconsn.Node.Alias
(Section 14, p. 104)
 - **FS:** sconsn.Node.FS
(Section 15, p. 115)
 - **Python:** sconsn.Node.Python
(Section 16, p. 179)
- **PathList:** SCons.PathList
(Section 17, p. 189)
- **SConf:** SCons.SConf
(Section 18, p. 190)
- **SConsign:** SCons.SConsign
(Section 19, p. 205)
- **Scanner:** SCons.Scanner
(Section 20, p. 210)
 - **C:** SCons.Scanner.C
(Section 21, p. 219)

- **D**: SCons.Scanner.D
(Section 22, p. 223)
- **Dir** (Section 23, p. 225)
- **Fortran**: SCons.Scanner.Fortran
(Section 24, p. 226)
- **IDL**: SCons.Scanner.IDL
(Section 25, p. 228)
- **LaTeX**: SCons.Scanner.LaTeX
(Section 26, p. 229)
- **Prog** (Section 27, p. 231)
- **Script**: SCons.Script
(Section 28, p. 232)
 - **Interactive**: SCons interactive mode
(Section 29, p. 240)
 - **Main**: SCons.Script
(Section 30, p. 244)
 - **SConscript'**: SCons.Script.SConscript
(Section 31, p. 257)
- **Sig**: Place-holder for the old SCons.Sig module hierarchy
(Section 32, p. 268)
- **Subst**: SCons.Subst
(Section 33, p. 270)
- **Taskmaster**: Generic Taskmaster module for the SCons build engine.
(Section 34, p. 278)
- **Util**: SCons.Util
(Section 35, p. 283)
- **Variables**: engine.SCons.Variables
(Section 36, p. 310)
 - **BoolVariable'**: engine.SCons.Variables.BoolVariable
This file defines the option type for SCons implementing true/false values.
(Section 37, p. 313)
 - **EnumVariable'**: engine.SCons.Variables.EnumVariable
This file defines the option type for SCons allowing only specified input-values.
(Section 38, p. 314)
 - **ListVariable'**: engine.SCons.Variables.ListVariable
This file defines the option type for SCons implementing 'lists'.
(Section 39, p. 315)
 - **PackageVariable'**: engine.SCons.Variables.PackageVariable
This file defines the option type for SCons implementing 'package activation'.
(Section 40, p. 316)
 - **PathVariable'**: SCons.Variables.PathVariable
This file defines an option type for SCons implementing path settings.
(Section 41, p. 317)
- **Warnings**: SCons.Warnings
(Section 42, p. 319)
- **compat**: SCons compatibility package for old Python versions
(Section 43, p. 357)
 - **_scons_UserString**: A user-defined wrapper around string objects
(Section 44, p. 359)
 - **_scons_hashlib**: hashlib backwards-compatibility module for older (pre-2.5) Python versions
(Section 45, p. 361)
 - **_scons_itertools**: Implementations of itertools functions for Python versions that don't have iterators.

- (Section 46, p. 363)
- **_scons_optparse**: optparse - a powerful, extensible, and easy-to-use option parser.
(Section 47, p. 365)
- **_scons_sets**: Classes to represent arbitrary sets (including sets of sets).
(Section 48, p. 386)
- **_scons_sets15** (Section 49, p. 398)
- **_scons_shlex**: A lexical analyzer class for simple shell-like syntaxes.
(Section 50, p. 400)
- **_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.
(Section 51, p. 401)
- **_scons_textwrap**: Text wrapping and filling.
(Section 52, p. 412)
- **builtins**: Compatibility idioms for `__builtin__` names
(Section 53, p. 414)
- **cpp**: SCons C Pre-Processor module
(Section 54, p. 416)
- **dblite** (Section 55, p. 423)
- **exitfuncs**: SCons.exitfuncs
(Section 56, p. 424)

1.2 Variables

Name	Description
<code>__build__</code>	Value: 'r3266'
<code>__buildsys__</code>	Value: 'bangkok'
<code>__developer__</code>	Value: 'knight'
<code>__revision__</code>	Value: 'src/engine/SCons/__init__.py 3266 2008/08/12 07:31:01 kn...'

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. This is what gets MD5 checksumm’ed 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.

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(<i>n</i>)

default_exitstatfunc (<i>s</i>)
--

remove_set_lineno_codes (<i>x</i>)

Action (<i>act</i> , * <i>args</i> , ** <i>kw</i>)

A factory for action objects.

2.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Action.py 3266 2008/08/12 07:31:01 knight'
<code>print_actions</code>	Value: False
<code>execute_actions</code>	Value: False
<code>print_actions_presub</code>	Value: 0
<code>default_ENV</code>	Value: False
<code>SET_LINENO</code>	Value: <code>dis.SET_LINENO</code>
<code>HAVE_ARGUMENT</code>	Value: <code>dis.HAVE_ARGUMENT</code>

2.3 Class *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__ (<i>self</i> , <i>other</i>)

genstring (<i>self</i> , <i>target</i> , <i>source</i> , <i>env</i>)

__add__ (<i>self</i> , <i>other</i>)

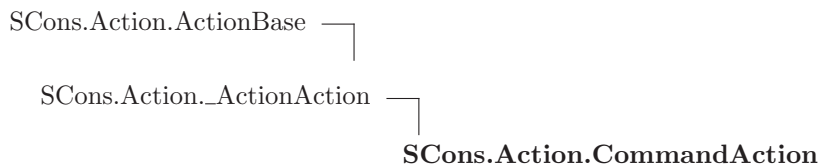
__radd__ (<i>self</i> , <i>other</i>)
--

presub_lines (<i>self</i> , <i>env</i>)
--

get_executor (<i>self</i> , <i>env</i> , <i>overrides</i> , <i>tlist</i> , <i>slist</i> , <i>executor_kw</i>)
--

Return the Executor for this Action.

2.4 Class `CommandAction`



Known Subclasses: `SCons.Action.LazyAction`

Class for command-execution actions.

2.4.1 Methods

`__init__(self, cmd, cmdstr=False, *args, **kw)`
 Overrides: `SCons.Action._ActionAction.__init__`

`__str__(self)`

`process(self, target, source, env)`

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

`execute(self, target, source, env)`

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_contents(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.

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

`__add__(self, other)`

`__call__(self, target, source, env, exitstatfunc=<class SCons.Action._Null at 0xc99770>, presub=<class SCons.Action._Null at 0xc99770>, show=<class SCons.Action._Null at 0xc99770>, execute=<class SCons.Action._Null at 0xc99770>, chdir=<class SCons.Action._Null at 0xc99770>)`

`__cmp__(self, other)`

`__radd__(self, other)`

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

```
get_executor(self, env, overrides, tlist, slist, executor_kw)
```

Return the Executor for this Action.

```
presub_lines(self, env)
```

```
print_cmd_line(self, s, target, source, env)
```

2.5 Class CommandGeneratorAction

SCons.Action.ActionBase



SCons.Action.CommandGeneratorAction

Known Subclasses: SCons.Action.LazyAction

Class for command-generator actions.

2.5.1 Methods

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

```
__str__(self)
```

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

Overrides: SCons.Action.ActionBase.genstring

```
__call__(self, target, source, env, exitstatfunc=<class SCons.Action._Null at 0xc99770>,
presub=<class SCons.Action._Null at 0xc99770>, show=<class SCons.Action._Null at
0xc99770>, execute=<class SCons.Action._Null at 0xc99770>, chdir=<class
SCons.Action._Null at 0xc99770>)
```

```
get_contents(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.

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

```
__add__(self, other)
```

```
__cmp__(self, other)
```

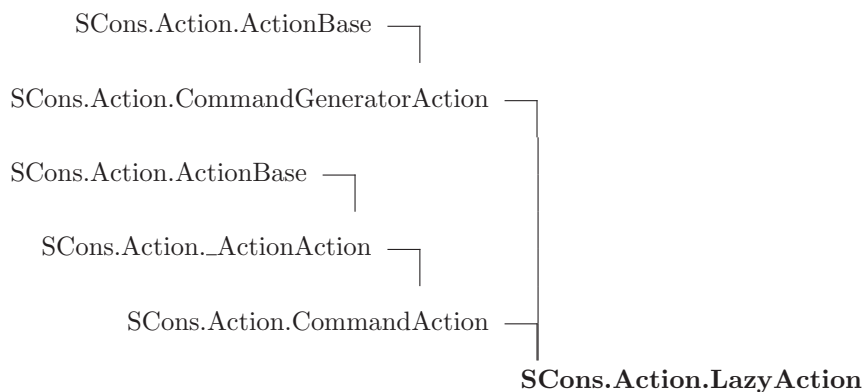
```
__radd__(self, other)
```

```
get_executor(self, env, overrides, tlist, slist, executor_kw)
```

Return the Executor for this Action.

```
presub_lines(self, env)
```

2.6 Class *LazyAction*



2.6.1 Methods

`__init__(self, var, *args, **kw)`
 Overrides: `SCons.Action.CommandGeneratorAction.__init__`

`get_parent_class(self, env)`

`__call__(self, target, source, env, *args, **kw)`
 Overrides: `SCons.Action.CommandGeneratorAction.__call__`

`get_contents(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.CommandGeneratorAction.get_contents` `exitit`(inherited documentation)

`__add__(self, other)`

`__cmp__(self, other)`

`__radd__(self, other)`

`__str__(self)`

`execute(self, target, source, env)`
 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.

`genstring(self, target, source, env)`
 Overrides: `SCons.Action.ActionBase.genstring`

```
get_executor(self, env, overrides, tlist, slist, executor_kw)
```

Return the Executor for this Action.

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

```
presub_lines(self, env)
```

```
print_cmd_line(self, s, target, source, env)
```

```
process(self, target, source, env)
```

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

2.7 Class *FunctionAction*

SCons.Action.ActionBase

SCons.Action._ActionAction

SCons.Action.FunctionAction

Class for Python function actions.

2.7.1 Methods

```
__init__(self, execfunction, cmdstr=<class SCons.Action._Null at 0xc99770>, *args, **kw)
```

Overrides: SCons.Action._ActionAction.__init__

```
function_name(self)
```

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

```
__str__(self)
```

```
execute(self, target, source, env)
```

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

Return the signature contents of this callable action.

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

```
__add__(self, other)
```

```
__call__(self, target, source, env, exitstatfunc=<class SCons.Action._Null at 0xc99770>,
presub=<class SCons.Action._Null at 0xc99770>, show=<class SCons.Action._Null at
0xc99770>, execute=<class SCons.Action._Null at 0xc99770>, chdir=<class
SCons.Action._Null at 0xc99770>)
```

```
__cmp__(self, other)
```

```
__radd__(self, other)
```

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

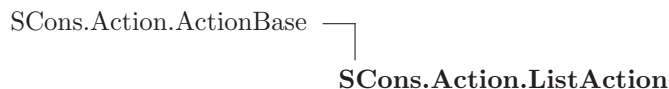
```
get_executor(self, env, overrides, tlist, slist, executor_kw)
```

Return the Executor for this Action.

```
presub_lines(self, env)
```

```
print_cmd_line(self, s, target, source, env)
```

2.8 Class ListAction



Class for lists of other actions.

2.8.1 Methods

```
__init__(self, list)
```

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

Overrides: SCons.Action.ActionBase.genstring

```
__str__(self)
```

```
presub_lines(self, env)
```

Overrides: SCons.Action.ActionBase.presub_lines

```
get_contents(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 at 0xc99770>,
presub=<class SCons.Action._Null at 0xc99770>, show=<class SCons.Action._Null at
0xc99770>, execute=<class SCons.Action._Null at 0xc99770>, chdir=<class
SCons.Action._Null at 0xc99770>)
```



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

```
__add__(self, other)
```

```
__cmp__(self, other)
```

```
__radd__(self, other)
```

```
get_executor(self, env, overrides, tlist, slist, executor_kw)
```

Return the Executor for this Action.

2.9 Class 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)
```

```
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)
```

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

```
__str__(self)
```

2.10 Class 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

<code>__init__(self, actfunc, strfunc, convert=<function <lambda> at 0xd2e7d0>)</code>
--

<code>__call__(self, *args, **kw)</code>
--

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

Builder(kw)**

A factory for builder objects.

3.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Builder.py 3266 2008/08/12 07:31:01 kni...'
<code>misleading_keywords</code>	Value: {'sources': 'source', 'targets': 'target'}

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=False, source_ext_match=False)`

Overrides: SCons.Util.OrderedDict.__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__

`__cmp__(self, dict)`

`__contains__(self, key)`

`__delitem__(self, key)`

Overrides: UserDict.UserDict.__delitem__

`__getitem__(self, key)`

`__len__(self)`

`__repr__(self)`

`__setitem__(self, key, item)`

Overrides: UserDict.UserDict.__setitem__

`clear(self)`

Overrides: UserDict.UserDict.clear

`copy(self)`

Overrides: UserDict.UserDict.copy

`fromkeys(cls, iterable, value=False)`

`get(self, key, failobj=False)`

`has_key(self, key)`

`items(self)`

Overrides: UserDict.UserDict.items

`iteritems(self)`

iterkeys(*self*)**itervalues**(*self*)**keys**(*self*)

Overrides: UserDict.UserDict.keys

pop(*self*, *key*, **args*)**popitem**(*self*)

Overrides: UserDict.UserDict.popitem

setdefault(*self*, *key*, *failobj*=False)

Overrides: UserDict.UserDict.setdefault

update(*self*, *dict*)

Overrides: UserDict.UserDict.update

values(*self*)

Overrides: UserDict.UserDict.values

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__

__cmp__(*self*, *dict*)**__contains__**(*self*, *key*)**__delitem__**(*self*, *key*)

Overrides: UserDict.UserDict.__delitem__

__getitem__(*self*, *key*)**__init__**(*self*, *dict*=False)

Overrides: UserDict.UserDict.__init__

`__len__(self)`

`__repr__(self)`

`__setitem__(self, key, item)`

Overrides: UserDict.UserDict.__setitem__

`clear(self)`

Overrides: UserDict.UserDict.clear

`copy(self)`

Overrides: UserDict.UserDict.copy

`fromkeys(cls, iterable, value=False)`

`get(self, key, failobj=False)`

`has_key(self, key)`

`items(self)`

Overrides: UserDict.UserDict.items

`iteritems(self)`

`iterkeys(self)`

`itervalues(self)`

`keys(self)`

Overrides: UserDict.UserDict.keys

`pop(self, key, *args)`

`popitem(self)`

Overrides: UserDict.UserDict.popitem

`setdefault(self, key, failobj=False)`

Overrides: UserDict.UserDict.setdefault

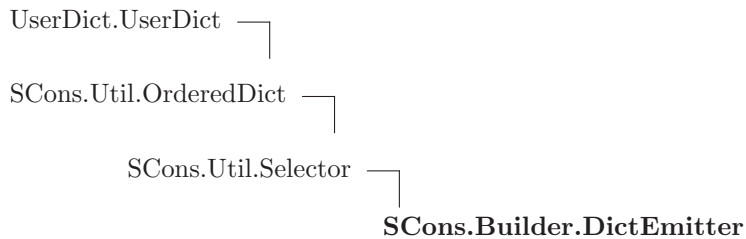
`update(self, dict)`

Overrides: UserDict.UserDict.update

`values(self)`

Overrides: UserDict.UserDict.values

3.5 Class 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__

`__cmp__(self, dict)`

`__contains__(self, key)`

`__delitem__(self, key)`

Overrides: UserDict.UserDict.__delitem__

`__getitem__(self, key)`

`__init__(self, dict=False)`

Overrides: UserDict.UserDict.__init__

`__len__(self)`

`__repr__(self)`

`__setitem__(self, key, item)`

Overrides: UserDict.UserDict.__setitem__

`clear(self)`

Overrides: UserDict.UserDict.clear

`copy(self)`

Overrides: UserDict.UserDict.copy

`fromkeys(cls, iterable, value=False)`

`get(self, key, failobj=False)`

`has_key(self, key)``items(self)`Overrides: `UserDict.UserDict.items``iteritems(self)``iterkeys(self)``itervalues(self)``keys(self)`Overrides: `UserDict.UserDict.keys``pop(self, key, *args)``popitem(self)`Overrides: `UserDict.UserDict.popitem``setdefault(self, key, failobj=False)`Overrides: `UserDict.UserDict.setdefault``update(self, dict)`Overrides: `UserDict.UserDict.update``values(self)`Overrides: `UserDict.UserDict.values`

3.6 Class `ListEmitter`



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

3.6.1 Methods

`__call__(self, target, source, env)``__add__(self, other)``__cmp__(self, other)``__contains__(self, item)``__delitem__(self, i)`

`__delslice__(self, i, j)``__eq__(self, other)``__ge__(self, other)``__getitem__(self, i)``__getslice__(self, i, j)``__gt__(self, other)``__iadd__(self, other)``__imul__(self, n)``__init__(self, initlist=False)``__le__(self, other)``__len__(self)``__lt__(self, other)``__mul__(self, n)``__ne__(self, other)``__radd__(self, other)``__repr__(self)``__rmul__(self, n)``__setitem__(self, i, item)``__setslice__(self, i, j, other)``append(self, item)``count(self, item)``extend(self, other)``index(self, item, *args)``insert(self, i, item)`

```
pop(self, i=-1)
```

```
remove(self, item)
```

```
reverse(self)
```

```
sort(self, *args, **kws)
```

3.7 Class 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)
```

```
__cmp__(self, dict)
```

```
__contains__(self, key)
```

```
__delitem__(self, key)
```

```
__getitem__(self, key)
```

```
__len__(self)
```

```
__repr__(self)
```

```
__setitem__(self, key, item)
```

```
clear(self)
```

```
copy(self)
```

```
fromkeys(cls, iterable, value=False)
```

```
get(self, key, failobj=False)
```

`has_key(self, key)``items(self)``iteritems(self)``iterkeys(self)``itervalues(self)``keys(self)``pop(self, key, *args)``popitem(self)``setdefault(self, key, failobj=False)``update(self, dict=False, **kwargs)``values(self)`

3.8 Class 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)``__call__(self, target, source, env)``__cmp__(self, other)`

3.9 Class BuilderBase

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

3.9.1 Methods

```
__init__(self, action=False, prefix='', suffix='', src_suffix='', target_factory=False,
source_factory=False, target_scanner=False, source_scanner=False, emitter=False, multi=0,
env=False, single_source=0, name=False, chdir=<class SCons.Builder._Null at 0xdeb9b0>,
is_explicit=False, src_builder=[], ensure_suffix=False, **overrides)
```

__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)

splittext(self, path, env=False)

get_single_executor(self, env, tlist, slist, executor_kw)

get_multi_executor(self, env, tlist, slist, executor_kw)

__call__(self, env, target=False, source=False, chdir=<class SCons.Builder._Null at 0xdeb9b0>, **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.)

3.9.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

3.10 Class CompositeBuilder

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: SCons.Util.Proxy.__init__ extit(inherited documentation)

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

__cmp__(*self*, *other*)

__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

4 Module *SCons.CacheDir*

CacheDir support

4.1 Functions

CacheRetrieveFunc (<i>target</i> , <i>source</i> , <i>env</i>)

CacheRetrieveString (<i>target</i> , <i>source</i> , <i>env</i>)

CachePushFunc (<i>target</i> , <i>source</i> , <i>env</i>)

4.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/CacheDir.py 3266 2008/08/12 07:31:01 kn...
<code>__doc__</code>	Value: ...
<code>cache_enabled</code>	Value: True
<code>cache_debug</code>	Value: False
<code>cache_force</code>	Value: False
<code>cache_show</code>	Value: False
<code>CacheRetrieve</code>	Value: <code>SCons.Action.Action(CacheRetrieveFunc, CacheRetrieveString)</code>
<code>CacheRetrieveSilent</code>	Value: <code>SCons.Action.Action(CacheRetrieveFunc, None)</code>
<code>CachePush</code>	Value: <code>SCons.Action.Action(CachePushFunc, None)</code>

4.3 Class *CacheDir*

4.3.1 Methods

<code>__init__</code> (<i>self</i> , <i>path</i>)
--

CacheDebug (<i>self</i> , <i>fmt</i> , <i>target</i> , <i>cachefile</i>)

is_enabled (<i>self</i>)

cachepath (<i>self</i> , <i>node</i>)
--

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*)

5 Module SCons.Conftest

SCons.Conftest

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

5.1 Functions

CheckBuilder(*context*, *text*=False, *language*=False)

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.

CheckFunc(*context*, *function_name*, *header*=False, *language*=False)

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*=False, *language*=False, *include_quotes*=False)

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*=False, *header*=False, *language*=False)

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=False*, *language=False*, *expect=False*)

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 whether 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=False*, *language=False*)

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=False*, *header=False*, *extra_libs=False*, *call=False*, *language=False*, *autoadd=False*)

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: False
LogErrorMessages	Value: False

6 Module SCons.Debug

SCons.Debug

Code for debugging SCons internal things. Not everything here is guaranteed to work all the way back to Python 1.5.2, and shouldn't be needed by most users.

6.1 Functions

```
logInstanceCreation(instance, name=False)
```

```
string_to_classes(s)
```

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

```
countLoggedInstances(classes, file=sys.stdout)
```

```
listLoggedInstances(classes, file=sys.stdout)
```

```
dumpLoggedInstances(classes, file=sys.stdout)
```

```
memory()
```

```
caller_stack(*backlist)
```

```
caller_trace(back=0)
```

```
dump_caller_counts(file=sys.stdout)
```

```
func_shorten(func_tuple)
```

```
Trace(msg, file=False, mode='w')
```

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
<code>__revision__</code>	Value: 'src/engine/SCons/Debug.py 3266 2008/08/12 07:31:01 knight'
<code>tracked_classes</code>	Value: {'Action.CommandAction': [<i><weakref at 0x13c5d60; to 'inst...</i>
<code>caller_bases</code>	Value: {}
<code>caller_dicts</code>	Value: {}
<code>shorten_list</code>	Value: [(<i>'/scons/SCons/'</i> , 1), (<i>'/src/engine/SCons/'</i> , 1), (<i>'/usr/...</i>

continued on next page

Name	Description
TraceFP	Value: {}
TraceDefault	Value: '/dev/tty'

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)

touch_func(dest)

7.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Defaults.py 3266 2008/08/12 07:31:01 kn...
<code>SharedCheck</code>	Value: <code>SCons.Action.Action(SharedFlagChecker, None)</code>
<code>CScan</code>	Value: <code>SCons.Tool.CScanner</code>
<code>DScan</code>	Value: <code>SCons.Tool.DScanner</code>
<code>LaTeXScan</code>	Value: <code>SCons.Tool.LaTeXScanner</code>
<code>ObjSourceScan</code>	Value: <code>SCons.Tool.SourceFileScanner</code>
<code>ProgScan</code>	Value: <code>SCons.Tool.ProgramScanner</code>
<code>DirScanner</code>	Value: <code>SCons.Scanner.Dir.DirScanner()</code>
<code>DirEntryScanner</code>	Value: <code>SCons.Scanner.Dir.DirEntryScanner()</code>
<code>CAction</code>	Value: <code>SCons.Action.Action("\$CCCOM", "\$CCCOMSTR")</code>
<code>ShCAction</code>	Value: <code>SCons.Action.Action("\$SHCCCOM", "\$SHCCCOMSTR")</code>
<code>CXXAction</code>	Value: <code>SCons.Action.Action("\$CXXCOM", "\$CXXCOMSTR")</code>
<code>ShCXXAction</code>	Value: <code>SCons.Action.Action("\$SHCXXCOM", "\$SHCXXCOMSTR")</code>
<code>ASAction</code>	Value: <code>SCons.Action.Action("\$ASCOM", "\$ASCOMSTR")</code>
<code>ASPPAction</code>	Value: <code>SCons.Action.Action("\$ASPPCOM", "\$ASPPCOMSTR")</code>
<code>LinkAction</code>	Value: <code>SCons.Action.Action("\$LINKCOM", "\$LINKCOMSTR")</code>
<code>ShLinkAction</code>	Value: <code>SCons.Action.Action("\$SHLINKCOM", "\$SHLINKCOMSTR")</code>
<code>LdModuleLinkAction</code>	Value: <code>SCons.Action.Action("\$LDMODULECOM", "\$LDMODULECOMSTR")</code>
<code>Chmod</code>	Value: <code>ActionFactory(chmod_func, chmod_strfunc)</code>
<code>Copy</code>	Value: <code>ActionFactory(copy_func, lambda dest, src: 'Copy("%s", "%...</code>
<code>Delete</code>	Value: <code>ActionFactory(delete_func, delete_strfunc)</code>
<code>Mkdir</code>	Value: <code>ActionFactory(mkdir_func, lambda dir: 'Mkdir("%s)' % get_p...</code>
<code>Move</code>	Value: <code>ActionFactory(lambda dest, src: os.rename(src, dest), lam...</code>
<code>Touch</code>	Value: <code>ActionFactory(touch_func, lambda file: 'Touch("%s)' % get_...</code>
<code>ConstructionEnvironment</code>	Value: <code>{'BUILDERS': {}, 'CONFIGUREDIR': '#/.sconf_temp', 'CONFIG...</code>

7.3 Class 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)
```

```
__call__(self, target, source, env, for_signature=False)
```

7.4 Class 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)
```

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


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

<code>alias_builder(<i>env</i>, <i>target</i>, <i>source</i>)</code>
--

<code>apply_tools(<i>env</i>, <i>tools</i>, <i>toolpath</i>)</code>

<code>copy_non_reserved_keywords(<i>dict</i>)</code>
--

<code>is_valid_construction_var(<i>varstr</i>)</code>

Return if the specified string is a legitimate construction variable.

<code>build_source(<i>ss</i>, <i>result</i>)</code>

<code>default_decide_source(<i>dependency</i>, <i>target</i>, <i>prev_ni</i>)</code>
--

<code>default_decide_target(<i>dependency</i>, <i>target</i>, <i>prev_ni</i>)</code>
--

<code>default_copy_from_cache(<i>src</i>, <i>dst</i>)</code>
--

<code>NoSubstitutionProxy(<i>subject</i>)</code>
--

8.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Environment.py 3266 2008/08/12 07:31:01...
<code>CleanTargets</code>	Value: {}
<code>CalculatorArgs</code>	Value: {}
<code>AliasBuilder</code>	Value: SCons.Builder.Builder(action= alias_builder, target_facto...
<code>reserved_construction_var_names</code>	Value: ['TARGET', 'TARGETS', 'SOURCE', 'SOURCES']

8.3 Class 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=False)
```

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

```
clone(self, new_object)
```

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

8.4 Class BuilderWrapper

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=False, source=<class SCons.Environment._Null at 0x13ff0b0>, *args,
**kw)
```

Overrides: SCons.Environment.MethodWrapper.__call__

```
__repr__(self)
```

```
__str__(self)
```

```
__getattr__(self, name)
```

<code>__setattr__(self, name, value)</code>

<code>__init__(self, object, method, name=False)</code>

<code>clone(self, new_object)</code>

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

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

<code>__init__(self, dict, env)</code>
--

Overrides: UserDict.UserDict.__init__

<code>__semi_deepcopy__(self)</code>

<code>__setitem__(self, item, val)</code>

Overrides: UserDict.UserDict.__setitem__
--

<code>__delitem__(self, item)</code>

Overrides: UserDict.UserDict.__delitem__
--

<code>update(self, dict)</code>

Overrides: UserDict.UserDict.update

<code>__cmp__(self, dict)</code>

<code>__contains__(self, key)</code>

<code>__getitem__(self, key)</code>

<code>__len__(self)</code>

<code>__repr__(self)</code>

<code>clear(self)</code>

<code>copy(self)</code>

<code>fromkeys(cls, iterable, value=False)</code>

`get(self, key, failobj=False)``has_key(self, key)``items(self)``iteritems(self)``iterkeys(self)``itervalues(self)``keys(self)``pop(self, key, *args)``popitem(self)``setdefault(self, key, failobj=False)``values(self)`

8.6 Class SubstitutionEnvironment

Known Subclasses: SCons.Environment.Base

Base class for different flavors of construction environments.

This class contains a minimal set of methods that handle construction 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.

`__cmp__(self, other)`

__delitem__(*self*, *key*)

__getitem__(*self*, *key*)

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

get(*self*, *key*, *default*=False)

Emulates the get() method of dictionaries.

has_key(*self*, *key*)

items(*self*)

arg2nodes(*self*, *args*, *node_factory*=<class SCons.Environment._Null at 0x13ff0b0>, *lookup_list*=<class SCons.Environment._Null at 0x13ff0b0>, ***kw*)

gvars(*self*)

lvars(*self*)

subst(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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*=False, *source*=False)

subst_list(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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

subst_path(*self*, *path*, *target*=False, *source*=False)

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

subst_target_source(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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=False*)

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 overridden 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=False*)

Merge the dict in args into the construction variables. 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.

8.6.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>

8.7 Class 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*)

AddMethod(*self*, *function*, *name*=False)

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.

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

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

Alias(*self*, *target*, *source*=[], *action*=False, ***kw*)

AlwaysBuild(*self*, **targets*)

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

Append values to existing construction variables in an Environment.

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

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.

AppendUnique(*self*, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there.

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

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

CacheDir(*self*, *path*)

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

Clone(*self*, *tools*=[], *toolpath*=False, *parse_flags*=False, ***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)

Explicitly 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=False)

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.

FindIndexes(*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*)

MergeFlags(*self*, *args*, *unique=False*)

Merge the dict in args into the construction variables. 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.

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

Override(*self*, *overrides*)

Produce a modified environment whose variables are overridden 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.

ParseConfig(*self*, *command*, *function=False*, *unique=False*)

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=False*, *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.

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.

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*=':')

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.

PrependUnique(*self*, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there.

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.

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*=False)

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 allowing files to be specified as a white-space separated list to be split.

The input rules are:

- A single string containing names separated by spaces. These will be split apart at the spaces.
- A single Node instance
- A list containing either strings or Node instances. 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*=False, ***kw*)

Value(*self*, *value*, *built_value*=False)

VariantDir(*self*, *variant_dir*, *src_dir*, *duplicate*=False)

WhereIs(*self*, *prog*, *path*=False, *pathtext*=False, *reject*=[])

Find prog in the path.

_cmp__(*self*, *other*)

_delitem__(*self*, *key*)

_getitem__(*self*, *key*)

```
__init__(self, platform=False, tools=False, toolpath=False, variables=False, parse_flags=False,
**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 (SubstitutionEnvironment) initialization, because we need to initialize things in a very specific order that doesn't work with the much simpler base class initialization.

Overrides: SCons.Environment.SubstitutionEnvironment.__init__

```
__setitem__(self, key, value)
```

```
arg2nodes(self, args, node_factory=<class SCons.Environment._Null at 0x13ff0b0>,
lookup_list=<class SCons.Environment._Null at 0x13ff0b0>, **kw)
```

```
backtick(self, command)
```

```
get(self, key, default=False)
```

Emulates the get() method of dictionaries.

```
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)
```

```
gvars(self)
```

```
has_key(self, key)
```

```
items(self)
```

```
lvars(self)
```

```
scanner_map_delete(self, kw=False)
```

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

```
subst(self, string, raw=0, target=False, source=False, conv=False)
```

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=False, source=False)
```

```
subst_list(self, string, raw=0, target=False, source=False, conv=False)
```

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

```
subst_path(self, path, target=False, source=False)
```

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

```
subst_target_source(self, string, raw=0, target=False, source=False, conv=False)
```

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.

8.7.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

8.8 Class OverrideEnvironment

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 (SubstitutionEnvironment) 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 (`SubstitutionEnvironment`) initialization, because we need to initialize things in a very specific order that doesn't work with the much simpler base class initialization.

Overrides: `SCons.Environment.Base.__init__` extit(inherited documentation)

`__getattr__(self, name)`

`__setattr__(self, name, value)`

`__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=False)`

Emulates the `get()` method of dictionaries.

Overrides: `SCons.Environment.SubstitutionEnvironment.get`

`has_key(self, key)`

Overrides: `SCons.Environment.SubstitutionEnvironment.has_key`

`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)

Action(*self*, **args*, ***kw*)**AddMethod**(*self*, *function*, *name*=`False`)

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.

AddPostAction(*self*, *files*, *action*)**AddPreAction**(*self*, *files*, *action*)**Alias**(*self*, *target*, *source*=[], *action*=`False`, ***kw*)**AlwaysBuild**(*self*, **targets*)**Append**(*self*, ***kw*)

Append values to existing construction variables in an Environment.

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

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.

AppendUnique(*self*, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there.

BuildDir(*self*, **args*, ***kw*)**Builder**(*self*, ***kw*)**CacheDir**(*self*, *path*)**Clean**(*self*, *targets*, *files*)**Clone**(*self*, *tools*=[], *toolpath*=`False`, *parse_flags*=`False`, ***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*)

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

Detect(*self*, *progs*)

Return the first available program in progs.

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

Dump(*self*, *key*=`False`)

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.

FindIndexes(*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*)

MergeFlags(*self*, *args*, *unique*=False)

Merge the dict in args into the construction variables. 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.

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

Override(*self*, *overrides*)

Produce a modified environment whose variables are overridden 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.

ParseConfig(*self*, *command*, *function*=False, *unique*=False)

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=False*, *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.

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.

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=':'*)

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.

PrependUnique(*self*, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there.

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.

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*=False)

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 allowing files to be specified as a white-space separated list to be split.

The input rules are:

- A single string containing names separated by spaces. These will be split apart at the spaces.
- A single Node instance
- A list containing either strings or Node instances. 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*=False, ***kw*)

Value(*self*, *value*, *built_value*=False)

VariantDir(*self*, *variant_dir*, *src_dir*, *duplicate*=False)

WhereIs(*self*, *prog*, *path*=False, *pathext*=False, *reject*=[])

Find prog in the path.

__cmp__(*self*, *other*)

arg2nodes(*self*, *args*, *node_factory*=<class SCons.Environment._Null at 0x13ff0b0>, *lookup_list*=<class SCons.Environment._Null at 0x13ff0b0>, ***kw*)

backtick(*self*, *command*)

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*=False)

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

subst(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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*=False, *source*=False)

subst_list(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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

subst_path(*self*, *path*, *target*=False, *source*=False)

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

subst_target_source(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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.

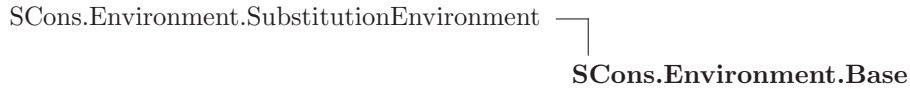
8.8.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>

continued on next page

Name	Description
memoizer_counters	Value: []

8.9 Class 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*)

AddMethod(*self*, *function*, *name*=False)

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.

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

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

Alias(*self*, *target*, *source*=[], *action*=False, ***kw*)

AlwaysBuild(*self*, **targets*)

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

Append values to existing construction variables in an Environment.

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

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.

AppendUnique(*self*, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there.

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

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

CacheDir(*self*, *path*)

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

Clone(*self*, *tools*=[], *toolpath*=False, *parse_flags*=False, ***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*)

Explicitly 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*=False)

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*)

MergeFlags(*self*, *args*, *unique*=False)

Merge the dict in args into the construction variables. 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.

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

Override(*self*, *overrides*)

Produce a modified environment whose variables are overridden 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.

ParseConfig(*self*, *command*, *function=False*, *unique=False*)

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=False*, *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.

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.

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=':'*)

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.

PrependUnique(*self*, ***kw*)

Append values to existing construction variables in an Environment, if they’re not already there.

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.

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*=False)

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 allowing files to be specified as a white-space separated list to be split.

The input rules are:

- A single string containing names separated by spaces. These will be split apart at the spaces.
- A single Node instance
- A list containing either strings or Node instances. 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*=False, ***kw*)

Value(*self*, *value*, *built_value*=False)

VariantDir(*self*, *variant_dir*, *src_dir*, *duplicate*=False)

WhereIs(*self*, *prog*, *path*=False, *pathext*=False, *reject*=[])

Find prog in the path.

__cmp__(*self*, *other*)

__delitem__(*self*, *key*)

__getitem__(*self*, *key*)

__init__(*self*, *platform*=False, *tools*=False, *toolpath*=False, *variables*=False, *parse_flags*=False, ***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 (SubstitutionEnvironment) initialization, because we need to initialize things in a very specific order that doesn't work with the much simpler base class initialization.

Overrides: SCons.Environment.SubstitutionEnvironment.__init__

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

arg2nodes(*self*, *args*, *node_factory*=<class SCons.Environment._Null at 0x13ff0b0>, *lookup_list*=<class SCons.Environment._Null at 0x13ff0b0>, ***kw*)

backtick(*self*, *command*)

get(*self*, *key*, *default=False*)

Emulates the get() method of dictionaries.

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*)

gvars(*self*)

has_key(*self*, *key*)

items(*self*)

lvars(*self*)

scanner_map_delete(*self*, *kw=False*)

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

subst(*self*, *string*, *raw=0*, *target=False*, *source=False*, *conv=False*)

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=False*, *source=False*)

subst_list(*self*, *string*, *raw=0*, *target=False*, *source=False*, *conv=False*)

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

subst_path(*self*, *path*, *target=False*, *source=False*)

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

subst_target_source(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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.

8.9.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

9 Module SCons.Errors

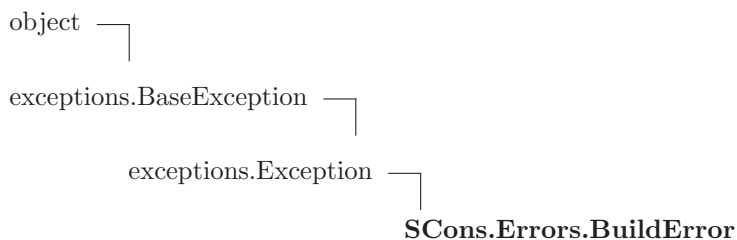
SCons.Errors

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

9.1 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Errors.py 3266 2008/08/12 07:31:01 knight'

9.2 Class BuildError



9.2.1 Methods

```
__init__(self, node=False, errstr='Unknown error', status=0, filename=False, executor=False,
action=False, command=False, *args)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.Exception.__init__ extit(inherited documentation)
```

```
__delattr__(...)
x.__delattr__('name') <==> del x.name
Overrides: object.__delattr__
```

```
__getattr__(...)
x.__getattr__('name') <==> x.name
Overrides: object.__getattr__
```

```
__getitem__(x, y)
x[y]
```

```
__getslice__(x, i, j)
x[i:j]
Use of negative indices is not supported.
```

__hash__(*x*)hash(*x*)**__new__**(*T*, *S*, ...)**Return Value**a new object with type *S*, a subtype of *T*

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ exitit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)repr(*x*)

Overrides: object.__repr__

__setattr__(...)*x*.__setattr__('name', value) <==> *x*.name = value

Overrides: object.__setattr__

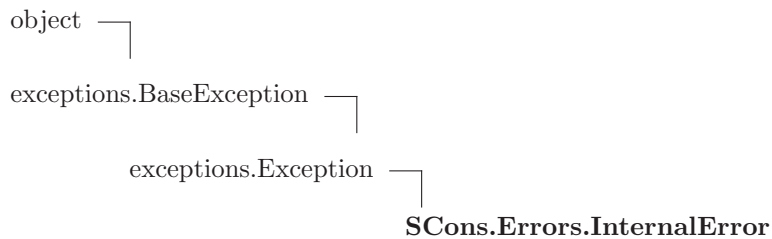
__setstate__(...)**__str__**(*x*)str(*x*)

Overrides: object.__str__

9.2.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

9.3 Class `InternalError`



9.3.1 Methods

`__delattr__`(...)

`x.__delattr__('name') <==> del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name') <==> x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

`__init__`(...)

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

`__new__`(*T*, *S*, ...)

Return Value

a new object with type *S*, a subtype of *T*

Overrides: `exceptions.BaseException.__new__`

`__reduce__`(...)

helper for pickle

Overrides: `object.__reduce__` extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)repr(*x*)

Overrides: object.__repr__

__setattr__(...)*x*.__setattr__('name', value) <==> *x*.name = value

Overrides: object.__setattr__

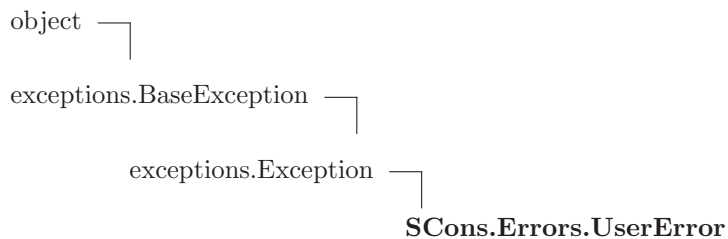
__setstate__(...)**__str__**(*x*)str(*x*)

Overrides: object.__str__

9.3.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

9.4 Class *UserError*

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

9.4.1 Methods

__delattr__(...)*x*.__delattr__('name') <==> del *x*.name

Overrides: object.__delattr__

__getattr__(...)

x.__getattr__('name') <==> x.name

Overrides: object.__getattr__

__getitem__(x, y)

x[y]

__getslice__(x, i, j)

x[i:j]

Use of negative indices is not supported.

__hash__(x)

hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(T, S, ...)

Return Value

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

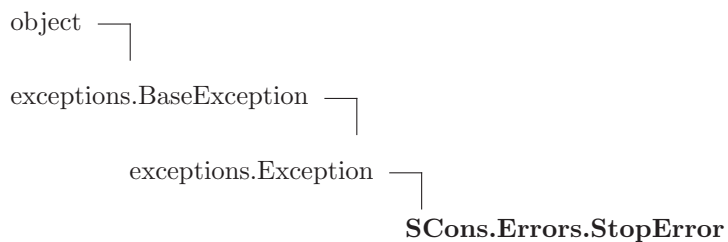
__setstate__(...)

<code>__str__(x)</code>
<code>str(x)</code>
Overrides: <code>object.__str__</code>

9.4.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

9.5 Class *StopError*



9.5.1 Methods

<code>__delattr__(...)</code>
<code>x.__delattr__('name') <==> del x.name</code>
Overrides: <code>object.__delattr__</code>

<code>__getattr__(...)</code>
<code>x.__getattr__('name') <==> x.name</code>
Overrides: <code>object.__getattr__</code>

<code>__getitem__(x, y)</code>
<code>x[y]</code>

<code>__getslice__(x, i, j)</code>
<code>x[i:j]</code>
Use of negative indices is not supported.

<code>__hash__(x)</code>
<code>hash(x)</code>

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(T, S, ...)**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)**__str__**(x)

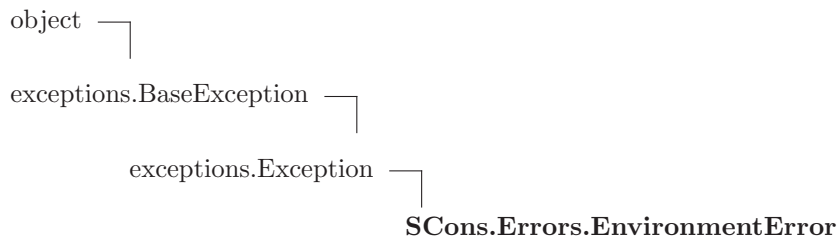
str(x)

Overrides: object.__str__

9.5.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

9.6 Class EnvironmentError



9.6.1 Methods

__delattr__(...)

`x.__delattr__('name') <==> del x.name`

Overrides: `object.__delattr__`

__getattr__(...)

`x.__getattr__('name') <==> x.name`

Overrides: `object.__getattr__`

__getitem__(*x*, *y*)

`x[y]`

__getslice__(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

__hash__(*x*)

`hash(x)`

__init__(...)

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

__new__(*T*, *S*, ...)

Return Value

a new object with type `S`, a subtype of `T`

Overrides: `exceptions.BaseException.__new__`

__reduce__(...)

helper for pickle

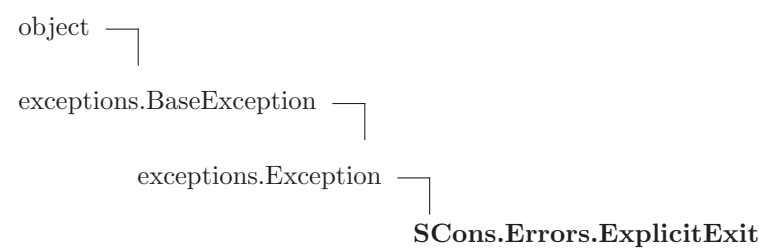
Overrides: `object.__reduce__` extit(inherited documentation)

<code>__reduce_ex__(...)</code> helper for pickle
<code>__repr__(x)</code> repr(x) Overrides: object.__repr__
<code>__setattr__(...)</code> x.__setattr__('name', value) <==> x.name = value Overrides: object.__setattr__
<code>__setstate__(...)</code>
<code>__str__(x)</code> str(x) Overrides: object.__str__

9.6.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

9.7 Class *ExplicitExit*



9.7.1 Methods

<code>__init__(self, node=False, status=False, *args)</code> x.__init__(...) initializes x; see x.__class__.__doc__ for signature Overrides: exceptions.Exception.__init__ exitit(inherited documentation)
--

__delattr__(...)

x.__delattr__('name') <==> del x.name

Overrides: object.__delattr__

__getattr__(...)

x.__getattr__('name') <==> x.name

Overrides: object.__getattr__

__getitem__(x, y)

x[y]

__getslice__(x, i, j)

x[i:j]

Use of negative indices is not supported.

__hash__(x)

hash(x)

__new__(T, S, ...)

Return Value

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

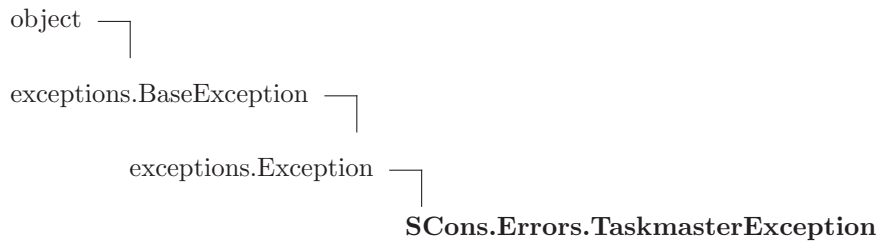
__setstate__(...)

<code>__str__(x)</code>
<code>str(x)</code>
Overrides: <code>object.__str__</code>

9.7.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

9.8 Class TaskmasterException



9.8.1 Methods

<code>__init__(self, node=False, exc_info=(None, None, None), *args)</code>
<code>x.__init__(...)</code> initializes <code>x</code> ; see <code>x.__class__.__doc__</code> for signature
Overrides: <code>exceptions.Exception.__init__</code> <code>exitit</code> (inherited documentation)

<code>__delattr__(...)</code>
<code>x.__delattr__('name')</code> <==> <code>del x.name</code>
Overrides: <code>object.__delattr__</code>

<code>__getattr__(...)</code>
<code>x.__getattr__('name')</code> <==> <code>x.name</code>
Overrides: <code>object.__getattr__</code>

<code>__getitem__(x, y)</code>
<code>x[y]</code>

__getslice__(*x, i, j*)

x[i:j]

Use of negative indices is not supported.

__hash__(*x*)

hash(x)

__new__(*T, S, ...*)

Return Value

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)

__str__(*x*)

str(x)

Overrides: object.__str__

9.8.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

10 Module SCons.Executor

SCons.Executor

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

10.1 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Executor.py 3266 2008/08/12 07:31:01 kn...

10.2 Class Executor

Known Subclasses: SCons.Executor.Null

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.

10.2.1 Methods

`__init__(self, action, env=False, overridelist=[{}], targets=[], sources=[], builder_kw={})`

`set_action_list(self, action)`

`get_action_list(self)`

`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*)

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*)

my_str(*self*)

__str__(*self*)

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_ignored_sources(*self*, *ignore*=())

process_sources(*self*, *func*, *ignore*=())

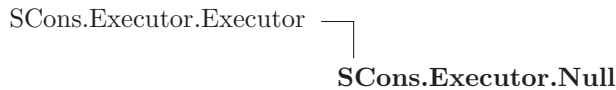
get_implicit_deps(*self*)

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

10.2.2 Class Variables

Name	Description
memoizer_counters	Value: []

10.3 Class 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.

10.3.1 Methods

__init__(*self*, **args*, ***kw*)
 Overrides: SCons.Executor.Executor.__init__

get_build_env(*self*)
 Fetch or create the appropriate build Environment for this Executor.
 Overrides: SCons.Executor.Executor.get_build_env extit(inherited documentation)

get_build_scanner_path(*self*)
 Fetch the scanner path for this executor's targets and sources.
 Overrides: SCons.Executor.Executor.get_build_scanner_path extit(inherited documentation)

cleanup(*self*)
 Overrides: SCons.Executor.Executor.cleanup

prepare(*self*)
 Preparatory checks for whether this Executor can go ahead and (try to) build its targets.
 Overrides: SCons.Executor.Executor.prepare extit(inherited documentation)

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

__str__(*self*)

add_post_action(*self*, *action*)

add_pre_action(*self*, *action*)

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.

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

Actually execute the action list.

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

get_action_list(*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_implicit_deps(*self*)

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

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

get_sources(*self*)

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.

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

my_str(*self*)

nullify(*self*)

process_sources(*self*, *func*, *ignore*=())

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.

scan_sources(*self*, *scanner*)

scan_targets(*self*, *scanner*)

set_action_list(*self*, *action*)

10.3.2 Class Variables

Name	Description
memoizer_counters	Value: []

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
<code>__revision__</code>	Value: 'src/engine/SCons/Job.py 3266 2008/08/12 07:31:01 knight'
<code>default_stack_size</code>	Value: 256
<code>interrupt_msg</code>	Value: 'Build interrupted.'

11.2 Class InterruptState

11.2.1 Methods

```
__init__(self)
```

```
set(self)
```

```
__call__(self)
```

11.3 Class 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.

```
run(self, postfunc=<function <lambda> at 0x14e2938>)
```

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.

11.4 Class 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*)

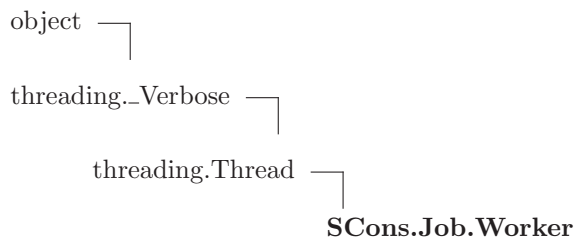
Create a new serial job given a taskmaster.

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).

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.

11.5 Class 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*)

Overrides: threading.Thread.__init__

run(*self*)

Overrides: threading.Thread.run

__delattr__(...)

x.__delattr__('name') <==> del x.name

__getattr__(...)

x.__getattr__('name') <==> x.name

__hash__(x)

hash(x)

__new__(T, S, ...)

Return Value

a new object with type S, a subtype of T

__reduce__(...)

helper for pickle

__reduce_ex__(...)

helper for pickle

__repr__(self)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

__str__(x)

str(x)

getName(self)

isAlive(self)

isDaemon(self)

join(self, timeout=False)

setDaemon(self, daemonic)

setName(self, name)

start(self)

11.5.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>

11.6 Class 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.

`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.

11.7 Class 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.

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.

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

<code>Dump(title=False)</code>

<code>EnableMemoization()</code>

12.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Memoize.py 3266 2008/08/12 07:31:01 kni...
<code>__doc__</code>	Value: ""Memoi...
<code>use_memoizer</code>	Value: False
<code>CounterList</code>	Value: []
<code>use_metaclass</code>	Value: False
<code>reason</code>	Value: 'new.instancemethod() bug'

12.3 Class 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)
```

```
display(self)
```

```
__cmp__(self, other)
```

12.4 Class CountValue

```

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)
```

```
__cmp__(self, other)
```

```
__init__(self, method_name)
```

```
display(self)
```

12.5 Class CountDict

```

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)
```

Overrides: SCons.Memoize.Counter.__init__

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

```
__cmp__(self, other)
```

```
display(self)
```

12.6 Class Memoizer

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

12.6.1 Methods

```
__init__(self)
```

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
 Overrides: object.__init__ extit(inherited documentation)

__call__(*x, ...*)

x(...)

__cmp__(*x, y*)

 cmp(*x*,*y*)

__delattr__(...)

x.__delattr__('name') <==> del *x*.name
 Overrides: object.__delattr__

__getattr__(...)

x.__getattr__('name') <==> *x*.name
 Overrides: object.__getattr__

__hash__(*x*)

 hash(*x*)
 Overrides: object.__hash__

__new__(*T, S, ...*)
Return Value
 a new object with type *S*, a subtype of *T*
 Overrides: object.__new__

__reduce__(...)

 helper for pickle

__reduce_ex__(...)

 helper for pickle

__repr__(*x*)

 repr(*x*)
 Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> *x*.name = value
 Overrides: object.__setattr__

<code>__str__(x)</code>
<code>str(x)</code>

<code>__subclasses__()</code>
Return Value list of immediate subclasses

<code>mro()</code>
return a type's method resolution order
Return Value list

12.7.2 Properties

Name	Description
<code>__base__</code>	Value: <member ' <code>__base__</code> ' of 'type' objects>
<code>__bases__</code>	Value: <attribute ' <code>__bases__</code> ' of 'type' objects>
<code>__basicsize__</code>	Value: <member ' <code>__basicsize__</code> ' of 'type' objects>
<code>__class__</code>	Value: <attribute ' <code>__class__</code> ' of 'object' objects>
<code>__dictoffset__</code>	Value: <member ' <code>__dictoffset__</code> ' of 'type' objects>
<code>__flags__</code>	Value: <member ' <code>__flags__</code> ' of 'type' objects>
<code>__itemsized__</code>	Value: <member ' <code>__itemsized__</code> ' of 'type' objects>
<code>__mro__</code>	Value: <member ' <code>__mro__</code> ' of 'type' objects>
<code>__name__</code>	Value: <attribute ' <code>__name__</code> ' of 'type' objects>
<code>__weakrefoffset__</code>	Value: <member ' <code>__weakrefoffset__</code> ' of 'type' objects>

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. 104)
- **FS:** `scons.Node.FS`
(Section 15, p. 115)
- **Python:** `scons.Node.Python`
(Section 16, p. 179)

13.2 Functions

<code>classname(obj)</code>

<code>Annotate(node)</code>

<code>get_children(node, parent)</code>

<code>ignore_cycle(node, stack)</code>
--

<code>do_nothing(node, parent)</code>

13.3 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Node/__init__.py 3266 2008/08/12 07:31:...
<code>no_state</code>	Value: 0
<code>pending</code>	Value: False
<code>executing</code>	Value: 2
<code>up_to_date</code>	Value: 3
<code>executed</code>	Value: 4

continued on next page

Name	Description
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 {}>]

13.4 Class 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)
```

```
convert(self, node, val)
```

```
format(self, field_list=False, names=0)
```

```
merge(self, other)
```

```
update(self, node)
```

13.4.2 Class Variables

Name	Description
current_version_id	Value: False

13.5 Class 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

<code>__init__(self, node)</code>

<code>merge(self, other)</code>

13.5.2 Class Variables

Name	Description
current_version_id	Value: False

13.6 Class 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

<code>Decider(self, function)</code>

<code>__init__(self)</code>

<code>add_dependency(self, depend)</code>

Adds dependencies.

<code>add_ignore(self, depend)</code>

Adds dependencies to ignore.

<code>add_prerequisite(self, prerequisite)</code>

Adds prerequisites

<code>add_source(self, source)</code>

Adds sources.

<code>add_to_implicit(self, deps)</code>
--

<code>add_to_waiting_parents(self, node)</code>

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...)
--

<code>add_to_waiting_s_e(self, node)</code>

add_wkid(*self*, *wkid*)

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

all_children(*self*, *scan=False*)

 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=False*)

 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=False*)

 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*=False)**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*=False)

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*=False)**

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.

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 repository?

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*=False)

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*=False)

Set the Node's nocache value.

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

Set the Node's noclean value.

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

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).

13.6.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

13.7 Class NodeList

UserList.UserList

```

  |
  +-- SCons.Node.NodeList

```

13.7.1 Methods

`__str__(self)``__add__(self, other)``__cmp__(self, other)``__contains__(self, item)``__delitem__(self, i)``__delslice__(self, i, j)``__eq__(self, other)``__ge__(self, other)``__getitem__(self, i)``__getslice__(self, i, j)``__gt__(self, other)``__iadd__(self, other)``__imul__(self, n)``__init__(self, initlist=False)``__le__(self, other)``__len__(self)``__lt__(self, other)``__mul__(self, n)``__ne__(self, other)``__radd__(self, other)``__repr__(self)``__rmul__(self, n)``__setitem__(self, i, item)`

`__setslice__(self, i, j, other)``append(self, item)``count(self, item)``extend(self, other)``index(self, item, *args)``insert(self, i, item)``pop(self, i=-1)``remove(self, item)``reverse(self)``sort(self, *args, **kwargs)`

13.8 Class 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 `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 0xdf9500>, cycle_func=<function ignore_cycle at 0xdfd1b8>, eval_func=<function do_nothing at 0xdfd230>)``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)`

14 Module *SCons.Node.Alias*

scons.Node.Alias

Alias nodes.

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

14.1 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Node/Alias.py 3266 2008/08/12 07:31:01 ...
<code>default_ans</code>	Value: {}

14.2 Class *AliasNameSpace*

UserDict.UserDict —
 SCons.Node.Alias.AliasNameSpace

14.2.1 Methods

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

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

__cmp__(*self*, *dict*)

__contains__(*self*, *key*)

__delitem__(*self*, *key*)

__getitem__(*self*, *key*)

__init__(*self*, *dict*=False, ****kwargs**)

__len__(*self*)

__repr__(*self*)

__setitem__(*self*, *key*, *item*)

clear(*self*)

copy(*self*)

`fromkeys(cls, iterable, value=False)``get(self, key, failobj=False)``has_key(self, key)``items(self)``iteritems(self)``iterkeys(self)``itervalues(self)``keys(self)``pop(self, key, *args)``popitem(self)``setdefault(self, key, failobj=False)``update(self, dict=False, **kwargs)``values(self)`

14.3 Class *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)``__init__(self, node)``convert(self, node, val)``format(self, field_list=False, names=0)``merge(self, other)`

update(*self*, *node*)

14.3.2 Class Variables

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

14.4 Class 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

__init__(*self*, *node*)

merge(*self*, *other*)

14.4.2 Class Variables

Name	Description
current_version_id	Value: 1

14.5 Class Alias



14.5.1 Methods

__init__(*self*, *name*)
 Overrides: SCons.Node.Node.__init__

str_for_display(*self*)

__str__(*self*)

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

Decider(*self*, *function*)

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=False*)

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

alter_targets(*self*)

Return a list of alternate targets for this Node.

builder_set(*self*, *builder*)

built(*self*)

Called just after this node is successfully built.

changed(*self*, *node=False*)

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.

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

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=False*)

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*=False)

Return the set builder, or a specified default value

get_cachedir_csig(*self*)**get_env(*self*)****get_env_scanner(*self*, *env*, *kw*={})****get_executor(*self*, *create*=False)**

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.

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.

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 repository?

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 (<i>self</i> , <i>always_build=False</i>)

Set the Node's always_build value.

set_executor (<i>self</i> , <i>executor</i>)

Set the action executor for this node.
--

set_explicit (<i>self</i> , <i>is_explicit</i>)
--

set_nocache (<i>self</i> , <i>nocache=False</i>)

Set the Node's nocache value.

set_noclean (<i>self</i> , <i>noclean=False</i>)

Set the Node's noclean value.

set_precious (<i>self</i> , <i>precious=False</i>)

Set the Node's precious value.

set_specific_source (<i>self</i> , <i>source</i>)
--

set_state (<i>self</i> , <i>state</i>)

state_has_changed (<i>self</i> , <i>target</i> , <i>prev_ni</i>)

store_info (<i>self</i>)

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

visited (<i>self</i>)

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

14.5.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

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=False)
```

```
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.

15.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Node/FS.py 3266 2008/08/12 07:31:01 kni...
<code>default_max_drift</code>	Value: 172800
<code>Save_Strings</code>	Value: False
<code>do_splitdrive</code>	Value: False
<code>needs_normpath_check</code>	Value: False
<code>Valid_Duplicates</code>	Value: ['hard-soft-copy', 'soft-hard-copy', 'hard-copy', 'soft-c...
<code>Link_Funcs</code>	Value: []
<code>Link</code>	Value: SCons.Action.Action(LinkFunc, None)
<code>LocalCopy</code>	Value: SCons.Action.Action(LinkFunc, LocalString)
<code>Unlink</code>	Value: SCons.Action.Action(UnlinkFunc, None)
<code>Mkdir</code>	Value: SCons.Action.Action(MkdirFunc, None, presub=None)
<code>MkdirBuilder</code>	Value: False
<code>DefaultSCCSBuilder</code>	Value: False

continued on next page

Name	Description
DefaultRCSBuilder	Value: False
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...
diskcheckers	Value: [diskcheck_match, diskcheck_rcs, diskcheck_sccs,]
glob_magic_check	Value: re.compile(r'[*\?\[\]]')
default_fs	Value: False

15.3 Class *DiskChecker*

15.3.1 Methods

__init__(*self*, *type*, *do*, *ignore*)

set_do(*self*)

set_ignore(*self*)

set(*self*, *list*)

15.4 Class *EntryProxy*

SCons.Util.Proxy —
SCons.Node.FS.EntryProxy

15.4.1 Methods

__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)

__cmp__(*self*, *other*)

__init__(*self*, *subject*)

Wrap an object as a Proxy object

get(*self*)

Retrieve the entire wrapped object

15.4.2 Class Variables

Name	Description
dictSpecialAttrs	Value: {'abspath': <function __get_abspath at 0xe68578>, 'base':...

15.5 Class 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.

15.5.1 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: SCons.Node.Node.__init__

`str_for_display(self)`

`must_be_same(self, klass)`

This node, which already existed, is being looked up as the specified class. 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.

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 repository?

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*=False)

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 0xd295f0>)

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.

reentry(*self*)**Decider(*self*, *function*)****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=False*)

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*=False)

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*=False)

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*=False)

do_not_store_info(*self*)

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

executor_cleanup(*self*)

Let the executor clean up any cached information.

explain(*self*)

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=False*)

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=False*)

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_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.

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.

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*=False)

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=False)
```

Set the Node's nocache value.

```
set_noclean(self, noclean=False)
```

Set the Node's noclean value.

```
set_precious(self, precious=False)
```

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).

15.5.2 Class Variables

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

15.6 Class Entry

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.

15.6.1 Methods

diskcheck_match(*self*)

disambiguate(*self*, *must_exist=False*)

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.

Since this should return the real 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.FS.Base.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)

Decider(*self*, *function*)

RDirs(*self*, *pathlist*)

Search for a list of directories in the Repository list.

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.

__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: SCons.Node.Node.__init__

__str__(*self*)

A Node.FS.Base object’s string representation is its path name.

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=False*)

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 build().

builder_set(*self*, *builder*)

built(*self*)

Called just after this node is successfully built.

changed(*self*, *node=False*)

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.

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

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.

do_not_store_info(*self*)

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

executor_cleanup(*self*)

Let the executor clean up any cached information.

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.

Overrides: SCons.Node.Node.for_signature extit(inherited documentation)

get_abspath(*self*)

Get the absolute path of the file.

Overrides: SCons.Node.Node.get_abspath

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*=False)

Return the set builder, or a specified default value

get_cachedir_csig(*self*)

get_csig(*self*)

get_dir(*self*)

get_env(*self*)

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

get_executor(*self*, *create*=False)

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_path(*self*, *dir*=False)

Return path relative to the current working directory of the Node.FS.Base object that owns us.

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.

Overrides: `SCons.Node.Node.get_subst_proxy` extit(inherited documentation)

get_suffix(*self*)

Overrides: `SCons.Node.Node.get_suffix`

get_target_scanner(*self*)**getmtime**(*self*)**getsize**(*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_under(*self*, *dir*)**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.

isdir(*self*)**isfile**(*self*)**islink**(*self*)**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*)**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.

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.

reentry(*self*)**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 repository?

Overrides: SCons.Node.Node.rexists extit(inherited documentation)

rstr(*self*)

A Node.FS.Base object's string representation is its path name.

scan(*self*)

Scan this node's dependents for implicit dependencies.

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=False*)

Set the Node's always_build value.

set_executor(*self*, *executor*)

Set the action executor for this node.

set_explicit(*self*, *is_explicit*)

set_local(*self*)

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

Set the Node's nocache value.

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

Set the Node's noclean value.

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

Set the Node's precious value.

set_specific_source(*self*, *source*)

set_src_builder(*self*, *builder*)

Set the source code builder for this node.

set_state(*self*, *state*)

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).

srcnode(*self*)

If this node is in a build path, return the node corresponding to its source file. Otherwise, return ourself.

stat(*self*)

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).

str_for_display(*self*)

target_from_source(*self*, *prefix*, *suffix*, *splitext*=<function splitext at 0xd295f0>)

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.

visited(*self*)

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

15.6.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

15.7 Class LocalFS

Known Subclasses: `SCons.Node.FS.FS`

15.7.1 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*)

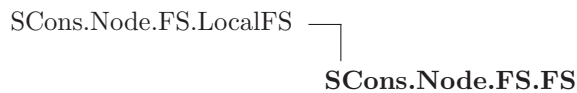
isfile(*self*, *path*)

`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)`

15.7.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>

15.8 Class FS



15.8.1 Methods

`__init__(self, path=False)`

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.

`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*=False, *create*=False)

Lookup 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*=False, *create*=False)

Lookup 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*=False, *create*=True)

Lookup 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*=False)

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=False*)

Globs

This is mainly a shim layer

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*)

islink(*self*, *path*)

link(*self*, *src*, *dst*)

listdir(*self*, *path*)

lstat(*self*, *path*)

makedirs(*self*, *path*)

mkdir(*self*, *path*)

open(*self*, *path*)

readlink(*self*, *file*)

rename(*self*, *old*, *new*)

stat(*self*, *path*)

symlink(*self*, *src*, *dst*)

unlink(*self*, *path*)

15.8.2 Class Variables

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

15.9 Class 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.

15.9.1 Methods

str_to_node(*self*, *s*)

__init__(*self*, *node*)

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

format(*self*, *field_list*=False, *names*=0)

merge(*self*, *other*)

update(*self*, *node*)

15.9.2 Class Variables

Name	Description
current_version_id	Value: 1
fs	Value: False

15.10 Class 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.

15.10.1 Methods

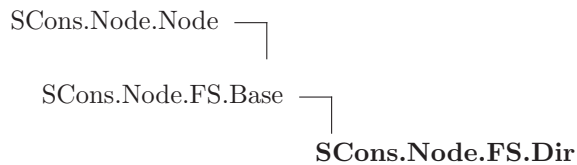
```
__init__(self, node)
```

```
merge(self, other)
```

15.10.2 Class Variables

Name	Description
current_version_id	Value: 1

15.11 Class Dir



Known Subclasses: SCons.Node.FS.RootDir

A class for directories in a file system.

15.11.1 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: SCons.Node.FS.Base.__init__ exitit(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 file 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` `exitit`(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_contents(*self*)

Return aggregate contents of all our children.

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` `exitit`(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 default, 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.

Decider(*self*, *function*)

RDirs(*self*, *pathlist*)

Search for a list of directories in the Repository list.

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.

__str__(*self*)

A Node.FS.Base object’s string representation is its path name.

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*=False)

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

builder_set(*self*, *builder*)

built(*self*)

Called just after this node is successfully built.

changed(*self*, *node*=False)

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.

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

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=False*)

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?

Overrides: SCons.Node.Node.exists exitit(inherited documentation)

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.

Overrides: SCons.Node.Node.for_signature exitit(inherited documentation)

get_abspath(*self*)

Get the absolute path of the file.

Overrides: SCons.Node.Node.get_abspath

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=False*)

Return the set builder, or a specified default value

get_cachedir_csig(*self*)

get_csig(*self*)

get_dir(*self*)

get_env(*self*)

get_executor(*self*, *create=False*)

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

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_path(*self*, *dir=False*)

Return path relative to the current working directory of the Node.FS.Base object that owns us.

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.

Overrides: `SCons.Node.Node.get_subst_proxy` extit(inherited documentation)

get_suffix(*self*)

Overrides: `SCons.Node.Node.get_suffix`

getmtime(*self*)**getsize**(*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_under(*self*, *dir*)

isdir(*self*)

isfile(*self*)

islink(*self*)

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*)

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.

new_binfo(*self*)

new_ninfo(*self*)

postprocess(*self*)

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

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.

reentry(*self*)

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 repository?

Overrides: SCons.Node.Node.rexists exit(inherited documentation)

rfile(*self*)**rstr(*self*)**

A Node.FS.Base object's string representation is its path name.

scan(*self*)

Scan this node's dependents for implicit dependencies.

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=False*)**

Set the Node's always_build value.

set_executor(*self*, *executor*)

Set the action executor for this node.

set_explicit(*self*, *is_explicit*)**set_local(*self*)****set_nocache(*self*, *nocache=False*)**

Set the Node's nocache value.

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

Set the Node's noclean value.

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

Set the Node's precious value.

set_specific_source(*self*, *source*)**set_src_builder(*self*, *builder*)**

Set the source code builder for this node.

set_state(*self*, *state*)

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).

stat(*self*)**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).

str_for_display(*self*)**target_from_source**(*self*, *prefix*, *suffix*, *splitext*=<function splitext at 0xd295f0>)

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.

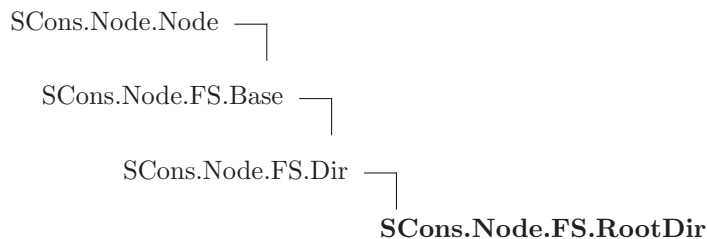
visited(*self*)

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

15.11.2 Class Variables

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

15.12 Class 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.

15.12.1 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: SCons.Node.FS.Dir.__init__

must_be_same(self, klass)

This node, which already existed, is being looked up as the specified class. 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: SCons.Node.FS.Base.__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)

Decider(self, function)**Dir(self, name, create=True)**

Looks up or creates a directory node named 'name' relative to this directory.

Entry(*self*, *name*)

Looks up or creates an entry node named 'name' relative to this directory.

File(*self*, *name*)

Looks up or creates a file node named 'name' relative to this directory.

RDirs(*self*, *pathlist*)

Search for a list of directories in the Repository list.

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.

addRepository(*self*, *dir*)**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=False*)

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

alter_targets(*self*)

Return any corresponding targets in a variant directory.

Overrides: SCons.Node.Node.alter_targets

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

A null "builder" for directories.

Overrides: SCons.Node.Node.build

builder_set(*self*, *builder*)

built(*self*)

Called just after this node is successfully built.

changed(*self*, *node=False*)

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.

Overrides: SCons.Node.Node.changed_since_last_build extit(inherited documentation)

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

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.

dir_on_disk(*self*, *name*)**disambiguate**(*self*, *must_exist=False*)**diskcheck_match**(*self*)**do_duplicate**(*self*, *src*)**do_not_store_info**(*self*)**entry_exists_on_disk**(*self*, *name*)**env_set**(*self*, *env*, *safe=0*)**executor_cleanup**(*self*)

Let the executor clean up any cached information.

exists(*self*)

Does this node exists?

Overrides: *SCons.Node.Node.exists* extit(inherited documentation)

explain(*self*)**file_on_disk**(*self*, *name*)**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)

getRepositories(*self*)

Returns a list of repositories for this directory.

get_abspath(*self*)

Get the absolute path of the file.

Overrides: *SCons.Node.Node.get_abspath*

get_all_rdirs(*self*)**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*=False)

Return the set builder, or a specified default value

get_cachedir_csig(*self*)**get_contents(*self*)**

Return aggregate contents of all our children.

get_csig(*self*)**get_env(*self*)****get_env_scanner(*self*, *env*, *kw*={})**

Overrides: *SCons.Node.Node.get_env_scanner*

get_executor(*self*, *create*=False)

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 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*

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_path(*self*, *dir*=False)

Return path relative to the current working directory of the Node.FS.Base object that owns us.

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.

Overrides: SCons.Node.Node.get_subst_proxy extit(inherited documentation)

get_suffix(*self*)Overrides: *SCons.Node.Node.get_suffix***get_target_scanner**(*self*)Overrides: *SCons.Node.Node.get_target_scanner***get_timestamp**(*self*)

Return the latest timestamp from among our children

getmtime(*self*)**getsize**(*self*)**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 default, 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.

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*)

If any child is not up-to-date, then this directory isn't, either.

Overrides: `SCons.Node.Node.is_up_to_date`

isdir(*self*)**isfile(*self*)****islink(*self*)****link(*self*, *srcdir*, *duplicate*)**

Set this directory as the variant directory for the supplied source directory.

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.

Overrides: `SCons.Node.Node.multiple_side_effect_has_builder` extit(inherited documentation)

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.

Overrides: *SCons.Node.Node.prepare* extit(inherited documentation)

rdir(*self*)**rel_path(*self*, *other*)**

Return a path to “other” relative to this directory.

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.

rentry(*self*)**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 repository?

Overrides: *SCons.Node.Node.rexists* extit(inherited documentation)

rfile(*self*)**rstr(*self*)**

A Node.FS.Base object’s string representation is its path name.

scan(*self*)

Scan this node's dependents for implicit dependencies.

scanner_key(*self*)

A directory does not get scanned.

Overrides: SCons.Node.Node.scanner_key

sconsign(*self*)

Return the .sconsign file info for this directory, creating it first if necessary.

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=False*)

Set the Node's always_build value.

set_executor(*self*, *executor*)

Set the action executor for this node.

set_explicit(*self*, *is_explicit*)**set_local**(*self*)**set_nocache**(*self*, *nocache=False*)

Set the Node's nocache value.

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

Set the Node's noclean value.

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

Set the Node's precious value.

set_specific_source(*self*, *source*)**set_src_builder**(*self*, *builder*)

Set the source code builder for this node.

set_state(*self*, *state*)

srcdir_duplicate (<i>self</i> , <i>name</i>)

srcdir_find_file (<i>self</i> , <i>filename</i>)

srcdir_list (<i>self</i>)

srcnode (<i>self</i>)

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

stat (<i>self</i>)

state_has_changed (<i>self</i> , <i>target</i> , <i>prev_ni</i>)

store_info (<i>self</i>)

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

str_for_display (<i>self</i>)
--

target_from_source (<i>self</i> , <i>prefix</i> , <i>suffix</i> , <i>splitext</i> =<function splitext at 0xd295f0>)

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.

visited (<i>self</i>)

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

walk (<i>self</i> , <i>func</i> , <i>arg</i>)
--

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).

15.12.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

15.13 Class 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.

15.13.1 Methods

str_to_node (<i>self</i> , <i>s</i>)

__init__ (<i>self</i> , <i>node</i>)

convert (<i>self</i> , <i>node</i> , <i>val</i>)

format (<i>self</i> , <i>field_list</i> =False, <i>names</i> =0)
--

merge (<i>self</i> , <i>other</i>)

update (<i>self</i> , <i>node</i>)

15.13.2 Class Variables

Name	Description
current_version_id	Value: 1
field_list	Value: ['csig', 'timestamp', 'size']
fs	Value: False

15.14 Class 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.

15.14.1 Methods

__init__ (<i>self</i> , <i>node</i>)

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)

merge(*self*, *other*)

prepare_dependencies(*self*)

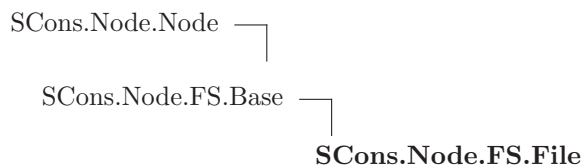
Prepares a FileBuildInfo object for explaining what changed

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).

15.14.2 Class Variables

Name	Description
current_version_id	Value: 1

15.15 Class File



A class for files in a file system.

15.15.1 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: SCons.Node.FS.Base.__init__ extit(inherited documentation)

Entry(*self*, *name*)

Create an entry node named 'name' relative to the SConscript directory of this file.

Dir(*self*, *name*, *create=True*)

Create a directory node named 'name' relative to the SConscript 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 SConscript directory of this file.

scanner_key(*self*)

Overrides: SCons.Node.Node.scanner_key

get_contents(*self*)

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

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

built(*self*)

Called just after this node is successfully built.

Overrides: SCons.Node.Node.built

visited(*self*)

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

Overrides: SCons.Node.Node.visited `exitit`(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 `exitit`(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.FS.Base.exists

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 exitit(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 exitit(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 exitit(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*)**Decider**(*self*, *function*)**RDirs**(*self*, *pathlist*)

Search for a list of directories in the Repository list.

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.

__str__(self)

A Node.FS.Base object's string representation is its path name.

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=False)

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

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().

changed(*self*, *node=False*)

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.

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

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=False*)**do_not_store_info**(*self*)**env_set**(*self*, *env*, *safe=0*)**executor_cleanup**(*self*)

Let the executor clean up any cached information.

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.

Overrides: SCons.Node.Node.for_signature extit(inherited documentation)

get_abspath(*self*)

Get the absolute path of the file.

Overrides: SCons.Node.Node.get_abspath

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*=False)

Return the set builder, or a specified default value

get_dir(*self*)**get_env(*self*)****get_env_scanner(*self*, *env*, *kw*={})****get_executor(*self*, *create*=False)**

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

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_path(*self*, *dir*=False)

Return path relative to the current working directory of the Node.FS.Base object that owns us.

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_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.

Overrides: SCons.Node.Node.get_subst_proxy extit(inherited documentation)

get_suffix(*self*)

Overrides: SCons.Node.Node.get_suffix

get_target_scanner(*self*)

getmtime(*self*)

getsize(*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_under(*self*, *dir*)**isdir(*self*)****isfile(*self*)****islink(*self*)****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.

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.

new_bininfo(*self*)

new_ninfo(*self*)

postprocess(*self*)

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

render_include_tree(*self*)

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

reentry(*self*)

reset_executor(*self*)

Remove cached executor; forces recompute when needed.

rexists(*self*)

Does this node exist locally or in a repository?

Overrides: SCons.Node.Node.rexists exitit(inherited documentation)

scan(*self*)

Scan this node's dependents for implicit dependencies.

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=False*)

Set the Node's always_build value.

set_executor(*self*, *executor*)

Set the action executor for this node.

set_explicit(*self*, *is_explicit*)

set_local(*self*)

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

Set the Node's nocache value.

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

Set the Node's noclean value.

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

Set the Node's precious value.

set_specific_source(*self*, *source*)

set_src_builder(*self*, *builder*)

Set the source code builder for this node.

set_state(*self*, *state*)

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).

srcnode(*self*)

If this node is in a build path, return the node corresponding to its source file. Otherwise, return *ourselves*.

stat(*self*)

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

str_for_display(*self*)

target_from_source(*self*, *prefix*, *suffix*, *splitext*=<function splitext at 0xd295f0>)

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.

15.15.2 Class Variables

Name	Description
memoizer_counters	Value: []
convert_copy_attrs	Value: ['bsources', 'bimplicit', 'bdepends', 'bact', 'bactsig', ...]
convert_sig_attrs	Value: ['bsourcesigs', 'bimplicitsigs', 'bdependsigs']
__metaclass__	Value: SCons.Memoize.Memoized_Metaclass

15.16 Class FileFinder

15.16.1 Methods

`__init__(self)`

`filedir_lookup(self, p, fd=False)`

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=False)`

`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.

15.16.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

16 Module SCons.Node.Python

scons.Node.Python

Python nodes.

16.1 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Node/Python.py 3266 2008/08/12 07:31:01...

16.2 Class 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*)

__init__(*self*, *node*)

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

format(*self*, *field_list*=False, *names*=0)

merge(*self*, *other*)

update(*self*, *node*)

16.2.2 Class Variables

Name	Description
<code>current_version_id</code>	Value: 1
<code>field_list</code>	Value: ['csig']

16.3 Class 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

<code>__init__(self, node)</code>

<code>merge(self, other)</code>

16.3.2 Class Variables

Name	Description
current_version_id	Value: 1

16.4 Class 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

<code>__init__(self, value, built_value=False)</code> Overrides: SCons.Node.Node.__init__
--

<code>str_for_display(self)</code>

<code>__str__(self)</code>

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_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=False*)

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

Decider(*self*, *function*)

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=False*)

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

alter_targets(*self*)

Return a list of alternate targets for this Node.

builder_set(*self*, *builder*)**built**(*self*)

Called just after this node is successfully built.

changed(*self*, *node*=False)

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.

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

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*=False)

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=False*)

Return the set builder, or a specified default value

get_cachedir_csig(*self*)**get_env(*self*)****get_env_scanner(*self*, *env*, *kw={}*)****get_executor(*self*, *create=False*)**

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.

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.

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 repository?

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 (<i>self</i> , <i>always_build=False</i>)

Set the Node's always_build value.

set_executor (<i>self</i> , <i>executor</i>)

Set the action executor for this node.
--

set_explicit (<i>self</i> , <i>is_explicit</i>)
--

set_nocache (<i>self</i> , <i>nocache=False</i>)

Set the Node's nocache value.

set_noclean (<i>self</i> , <i>noclean=False</i>)

Set the Node's noclean value.

set_precious (<i>self</i> , <i>precious=False</i>)

Set the Node's precious value.

set_specific_source (<i>self</i> , <i>source</i>)
--

set_state (<i>self</i> , <i>state</i>)

state_has_changed (<i>self</i> , <i>target</i> , <i>prev_ni</i>)

store_info (<i>self</i>)

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

visited (<i>self</i>)

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

16.4.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

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
<code>__revision__</code>	Value: 'src/engine/SCons/PathList.py 3266 2008/08/12 07:31:01 kn...
<code>__doc__</code>	Value: """SCons.PathL...
<code>TYPE_STRING_NO_SUBST</code>	Value: 0
<code>TYPE_STRING_SUBST</code>	Value: False
<code>TYPE_OBJECT</code>	Value: 2

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=False*, *language=False*)

CheckType(*context*, *type_name*, *includes=''*, *language=False*)

CheckTypeSize(*context*, *type_name*, *includes=''*, *language=False*, *expect=False*)

CheckDeclaration(*context*, *declaration*, *includes=''*, *language=False*)

createIncludesFromHeaders(*headers*, *leaveLast*, *include_quotes='\"'\"'*)

CheckHeader(*context*, *header*, *include_quotes='<>'*, *language=False*)

A test for a C or C++ header file.

CheckCHheader(*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=False*, *symbol='main'*, *header=False*, *language=False*, *autoadd=False*)

A test for a library. See also CheckLibWithHeader. Note that library may also be None to test whether the given symbol compiles without flags.

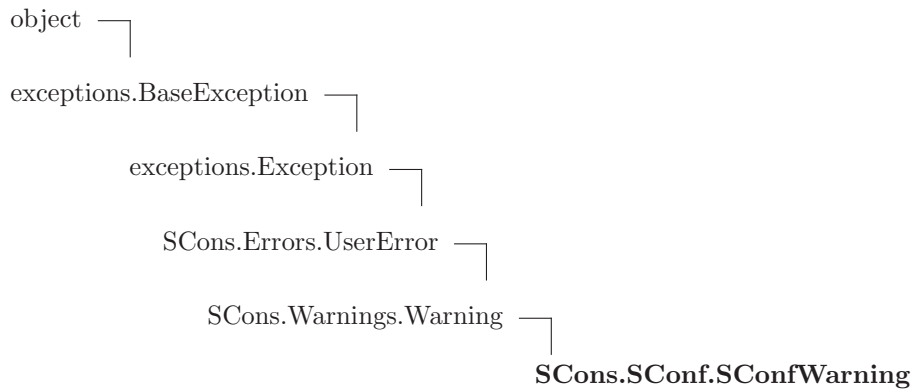
CheckLibWithHeader(*context, libs, header, language, call=False, autoadd=False*)

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
<code>__revision__</code>	Value: 'src/engine/SCons/SConf.py 3266 2008/08/12 07:31:01 knight'
<code>build_type</code>	Value: False
<code>build_types</code>	Value: ['clean', 'help']
<code>dryrun</code>	Value: 0
<code>AUTO</code>	Value: 0
<code>FORCE</code>	Value: False
<code>CACHE</code>	Value: 2
<code>cache_mode</code>	Value: 0
<code>progress_display</code>	Value: SCons.Util.display
<code>SConfFS</code>	Value: False
<code>sconf_global</code>	Value: False
<code>BooleanTypes</code>	Value: [<type 'int'>, <type 'bool'>]

18.3 Class SConfWarning



18.3.1 Methods

`__delattr__`(...)

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

__getattrute__(...)

`x.__getattrute__('name') <==> x.name`

Overrides: `object.__getattrute__`

__getitem__(*x*, *y*)

`x[y]`

__getslice__(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

__hash__(*x*)

`hash(x)`

__init__(...)

`x.__init__()` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

__new__(*T*, *S*, ...)

Return Value

a new object with type `S`, a subtype of `T`

Overrides: `exceptions.BaseException.__new__`

__reduce__(...)

helper for pickle

Overrides: `object.__reduce__` `exitit`(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

`repr(x)`

Overrides: `object.__repr__`

__setattr__(...)

`x.__setattr__('name', value) <==> x.name = value`

Overrides: `object.__setattr__`

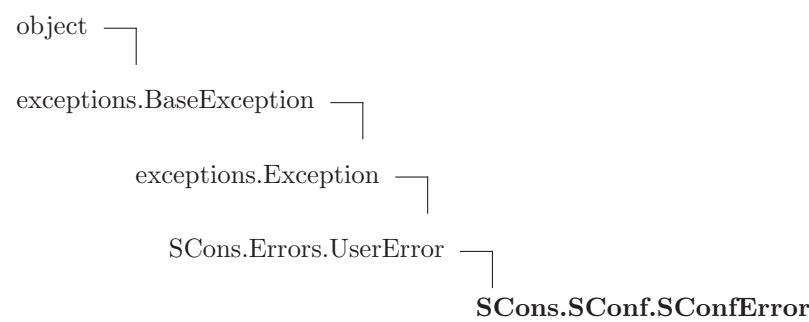
__setstate__(...)

<code>__str__(x)</code>
<code>str(x)</code>
Overrides: <code>object.__str__</code>

18.3.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

18.4 Class *SConfError*



Known Subclasses: *SCons.SConf.ConfigureCacheError*, *SCons.SConf.ConfigureDryRunError*

18.4.1 Methods

<code>__init__(self, msg)</code>
<code>x.__init__(...)</code> initializes <code>x</code> ; see <code>x.__class__.__doc__</code> for signature
Overrides: <code>exceptions.Exception.__init__</code> <code>exitit</code> (inherited documentation)

<code>__delattr__(...)</code>
<code>x.__delattr__('name')</code> <==> <code>del x.name</code>
Overrides: <code>object.__delattr__</code>

<code>__getattr__(...)</code>
<code>x.__getattr__('name')</code> <==> <code>x.name</code>
Overrides: <code>object.__getattr__</code>

<code>__getitem__(x, y)</code>
<code>x[y]</code>

__getslice__(*x*, *i*, *j*)

x[i:j]

Use of negative indices is not supported.

__hash__(*x*)

hash(x)

__new__(*T*, *S*, ...)

Return Valuea new object with type *S*, a subtype of *T*

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)

__str__(*x*)

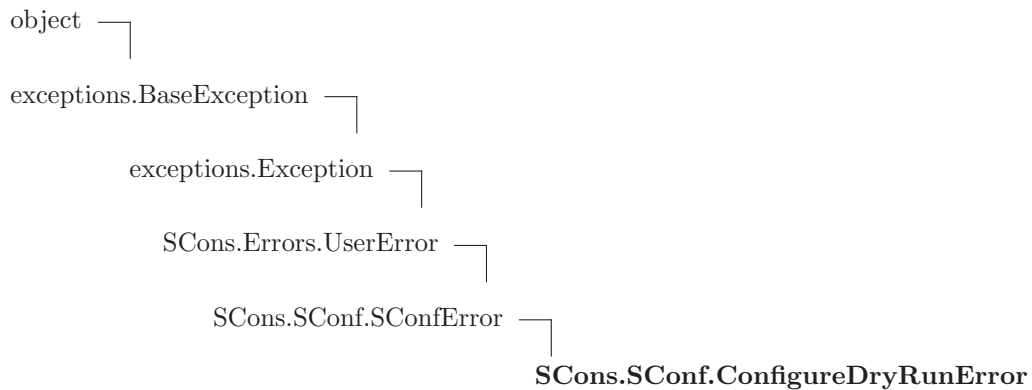
str(x)

Overrides: object.__str__

18.4.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

18.5 Class `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

<code>__init__(self, target)</code> Overrides: <code>SCons.SConf.SConfError.__init__</code>

<code>__delattr__(...)</code> <hr/> <code>x.__delattr__('name') <==> del x.name</code> Overrides: <code>object.__delattr__</code>
--

<code>__getattr__(...)</code> <hr/> <code>x.__getattr__('name') <==> x.name</code> Overrides: <code>object.__getattr__</code>
--

<code>__getitem__(x, y)</code> <hr/> <code>x[y]</code>
--

<code>__getslice__(x, i, j)</code> <hr/> <code>x[i:j]</code> Use of negative indices is not supported.

<code>__hash__(x)</code> <hr/> <code>hash(x)</code>

<code>__new__(T, S, ...)</code> Return Value a new object with type <code>S</code> , a subtype of <code>T</code> Overrides: <code>exceptions.BaseException.__new__</code>
--

__reduce__(...)
 helper for pickle
 Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)
 helper for pickle

__repr__(*x*)
 repr(*x*)
 Overrides: object.__repr__

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

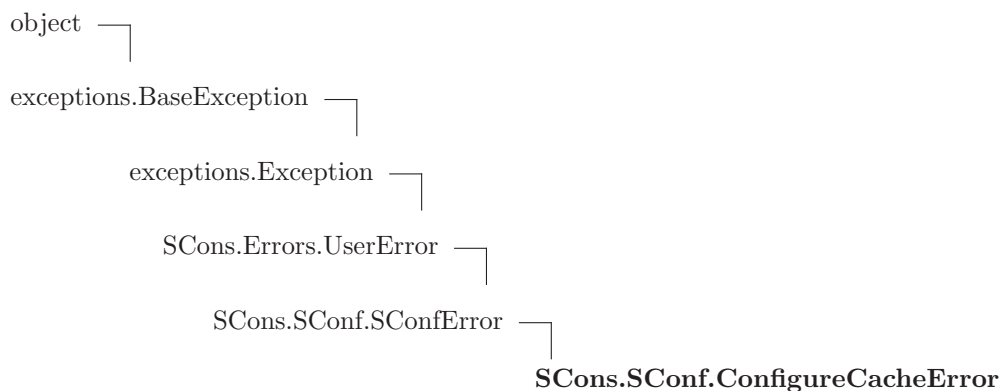
__setstate__(...)

__str__(*x*)
 str(*x*)
 Overrides: object.__str__

18.5.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

18.6 Class *ConfigureCacheError*



Raised when a use explicitly requested the cache feature, but the test is run the first time.

18.6.1 Methods

`__init__(self, target)`

Overrides: *SCons.SConf.SConfError.__init__*

`__delattr__(...)`

`x.__delattr__('name') <==> del x.name`

Overrides: *object.__delattr__*

`__getattr__(...)`

`x.__getattr__('name') <==> x.name`

Overrides: *object.__getattr__*

`__getitem__(x, y)`

`x[y]`

`__getslice__(x, i, j)`

`x[i:j]`

Use of negative indices is not supported.

`__hash__(x)`

`hash(x)`

`__new__(T, S, ...)`

Return Value

a new object with type *S*, a subtype of *T*

Overrides: *exceptions.BaseException.__new__*

`__reduce__(...)`

helper for pickle

Overrides: *object.__reduce__* *exitit*(inherited documentation)

`__reduce_ex__(...)`

helper for pickle

`__repr__(x)`

`repr(x)`

Overrides: *object.__repr__*

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)**__str__**(x)

str(x)

Overrides: object.__str__

18.6.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

18.7 Class *SConfBuildInfo*

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)**__init__**(self, node)**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)**merge(*self*, *other*)****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).

18.7.2 Class Variables

Name	Description
result	Value: False
string	Value: False
current_version_id	Value: 1

18.8 Class Streamer

'Sniffer' for a file-like writable object. Similar to the unix tool tee.

18.8.1 Methods**__init__(*self*, *orig*)****write(*self*, *str*)****writelines(*self*, *lines*)****getvalue(*self*)**

Return everything written to orig since the Streamer was created.

flush(*self*)**18.9 Class SConfBuildTask**

SCons.Taskmaster.Task

```

  |
  +-- 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` `exitit`(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.

Overrides: `SCons.Taskmaster.Task.failed` `exitit`(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` `exitit`(inherited documentation)

__init__(*self*, *tm*, *targets*, *top*, *node*)

exc_clear(*self*)

Clears any recorded exception.

This also changes the “`exception_raise`” attribute to point to the appropriate do-nothing method.

exc_info(*self*)

Returns info about a recorded exception.

exception_set(*self*, *exception*=`False`)

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

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.

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_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.

fail_continue(*self*)

Explicit continue-the-build failure.

This sets failure status on the target nodes and all of their dependent parent nodes.

fail_stop(*self*)

Explicit stop-the-build failure.

get_target(*self*)

Fetch the target being built or updated by this task.

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.

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.

needs_execute(*self*)

Called to determine whether the task's execute() method should be run. This method allows one to skip the somewhat costly execution of the execute() method in a separate thread. For example, that would be unnecessary for up-to-date targets.

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.

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.

18.10 Class 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 explicitly. 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 explicitly cache this error.

18.10.1 Methods

__init__(*self*, *env*, *custom_tests*={}, *conf_dir*='\$CONFIGUREDIR', *log_file*='\$CONFIGURELOG', *config_h*=False, *_depth*=0)

Constructor. Pass additional tests in the custom_tests-dictionary, 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)

Finish(*self*)

Call this method after finished with your tests: env = sconf.Finish()

Define(*self*, *name*, *value*=False, *comment*=False)

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=False*, *extension=''*)

Low level TryBuild implementation. Normally you don’t need to call that - you can use TryCompile / TryLink / TryRun instead

TryAction(*self*, *action*, *text=False*, *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

18.11 Class 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.

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 'ok' or 'failed'. 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*)

RunProg(*self*, *text*, *ext*)

AppendLIBS(*self*, *lib_name_list*)

SetLIBS(*self*, *val*)

Display(*self*, *msg*)

Log(*self*, *msg*)

19 Module SCons.SConsign

SCons.SConsign

Writing and reading information to the .sconsign file or files.

19.1 Functions

corrupt_dblite_warning (<i>filename</i>)

Get_DataBase (<i>dir</i>)

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 (<i>name</i> , <i>dbm_module=False</i>)

Arrange for all signatures to be stored in a global .sconsign.db* file.

19.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/SConsign.py 3266 2008/08/12 07:31:01 kn...
<code>sig_files</code>	Value: []
<code>DataBase</code>	Value: {}
<code>DB_Name</code>	Value: '.sconsign'
<code>DB_sync_list</code>	Value: []

19.3 Class 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__ (<i>self</i>)

convert_to_sconsign (<i>self</i>)
--

```
convert_from_sconsign(self, dir, name)
```

19.3.2 Class Variables

Name	Description
current_version_id	Value: False

19.4 Class 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)
```

```
get_entry(self, filename)
```

Fetch the specified entry attribute.

```
set_entry(self, filename, obj)
```

Set the entry.

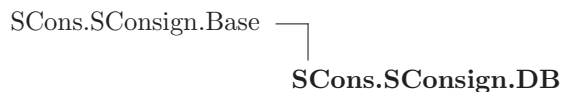
```
do_not_set_entry(self, filename, obj)
```

```
store_info(self, filename, node)
```

```
do_not_store_info(self, filename, node)
```

```
merge(self)
```

19.5 Class 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)`

Overrides: SCons.SConsign.Base.`__init__`

`write(self, sync=False)`

`do_not_set_entry(self, filename, obj)`

`do_not_store_info(self, filename, node)`

`get_entry(self, filename)`

Fetch the specified entry attribute.

`merge(self)`

`set_entry(self, filename, obj)`

Set the entry.

`store_info(self, filename, node)`

19.6 Class Dir



Known Subclasses: SCons.SConsign.DirFile

19.6.1 Methods

`__init__(self, fp=False, dir=False)`

fp - file pointer to read entries from

Overrides: SCons.SConsign.Base.`__init__`

`do_not_set_entry(self, filename, obj)`

`do_not_store_info(self, filename, node)`

`get_entry(self, filename)`

Fetch the specified entry attribute.

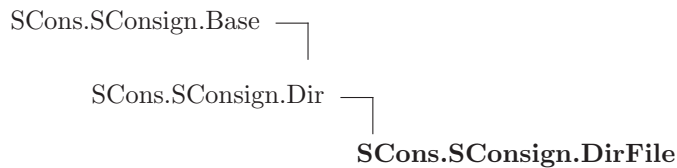
`merge(self)`

set_entry (<i>self</i> , <i>filename</i> , <i>obj</i>)

Set the entry.

store_info (<i>self</i> , <i>filename</i> , <i>node</i>)

19.7 Class *DirFile*



Encapsulates reading and writing a per-directory .sconsign file.

19.7.1 Methods

__init__ (<i>self</i> , <i>dir</i>)
--

<i>dir</i> - the directory for the file

Overrides: <i>SCons.SConsign.Dir.__init__</i>

write (<i>self</i> , <i>sync=False</i>)
--

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.
--

do_not_set_entry (<i>self</i> , <i>filename</i> , <i>obj</i>)
--

do_not_store_info (<i>self</i> , <i>filename</i> , <i>node</i>)
--

get_entry (<i>self</i> , <i>filename</i>)
--

Fetch the specified entry attribute.

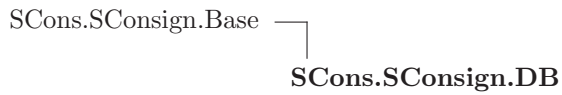
merge (<i>self</i>)

set_entry (<i>self</i> , <i>filename</i> , <i>obj</i>)

Set the entry.

store_info (<i>self</i> , <i>filename</i> , <i>node</i>)

19.8 Class 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

<code>__init__(self, dir)</code> Overrides: <code>SCons.SConsign.Base.__init__</code>

<code>write(self, sync=False)</code>

<code>do_not_set_entry(self, filename, obj)</code>

<code>do_not_store_info(self, filename, node)</code>

<code>get_entry(self, filename)</code> Fetch the specified entry attribute.

<code>merge(self)</code>

<code>set_entry(self, filename, obj)</code> Set the entry.
--

<code>store_info(self, filename, node)</code>
--

20 Package SCons.Scanner

SCons.Scanner

The Scanner package for the SCons software construction utility.

20.1 Modules

- **C**: SCons.Scanner.C
(Section 21, p. 219)
- **D**: SCons.Scanner.D
(Section 22, p. 223)
- **Dir** (Section 23, p. 225)
- **Fortran**: SCons.Scanner.Fortran
(Section 24, p. 226)
- **IDL**: SCons.Scanner.IDL
(Section 25, p. 228)
- **LaTeX**: SCons.Scanner.LaTeX
(Section 26, p. 229)
- **Prog** (Section 27, p. 231)

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
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/__init__.py 3266 2008/08/12 07:...

20.4 Class 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)`

`__call__(self, env, dir=False, target=False, source=False, argument=False)`

20.5 Class Base

Known Subclasses: SCons.Scanner.Current, SCons.Scanner.Selector

The base class for dependency scanners. This implements straightforward, single-pass scanning of a single file.

20.5.1 Methods

```
__init__(self, function, name='NONE', argument=<class SCons.Scanner._Null at 0x13f02f0>,
keys=<class SCons.Scanner._Null at 0x13f02f0>, path_function=False, node_class=<class
SCons.Node.FS.Entry at 0xe57710>, node_factory=False, scan_check=False, recursive=False)
```

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')
```

```
path(self, env, dir=False, target=False, source=False)
```

```
__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)``__str__(self)``add_skey(self, skey)`

Add a skey to the list of skeys

`get_skeys(self, env=False)``select(self, node)``recurse_nodes(self, nodes)``add_scanner(self, skey, scanner)`

20.6 Class 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.

'keys' - 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 'keys' 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: SCons.Scanner.Base.__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

`__cmp__(self, other)`

`__hash__(self)`

`__str__(self)``add_skey(self, skey)`

Add a skey to the list of skeys

`get_skeys(self, env=False)``path(self, env, dir=False, target=False, source=False)``recurse_nodes(self, nodes)`

20.7 Class *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.

'keys' - 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 'keys' 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: SCons.Scanner.Base.__init__ `exitit`(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.

`__cmp__(self, other)`

`__hash__(self)`

`__str__(self)`

`add_scanner(self, skey, scanner)`

```
add_skey(self, skey)
```

Add a skey to the list of skeys

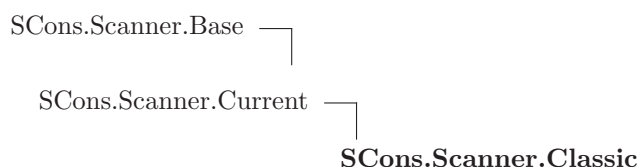
```
get_skeys(self, env=False)
```

```
path(self, env, dir=False, target=False, source=False)
```

```
recurse_nodes(self, nodes)
```

```
select(self, node)
```

20.8 Class Classic



Known Subclasses: SCons.Scanner.ClassicCPP, SCons.Scanner.D.D, SCons.Scanner.Fortran.F90Scanner, SCons.Scanner.LaTeX.LaTeX

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

```
__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)
```

```
__init__(self, name, suffixes, path_variable, regex, *args, **kw)
```

Overrides: SCons.Scanner.Current.__init__

```
__str__(self)
```

```
add_scanner(self, skey, scanner)
```

```
add_skey(self, skey)
```

Add a skey to the list of skeys

```
find_include(self, include, source_dir, path)
```

```
find_include_names(self, node)
```

```
get_skeys(self, env=False)
```

```
path(self, env, dir=False, target=False, source=False)
```

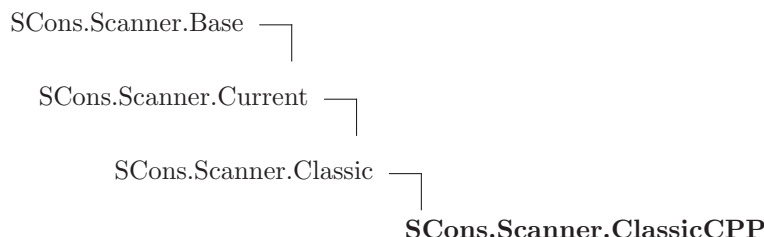
```
recurse_nodes(self, nodes)
```

```
scan(self, node, path=())
```

```
select(self, node)
```

```
sort_key(self, include)
```

20.9 Class 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

```
__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)
```

```
__init__(self, name, suffixes, path_variable, regex, *args, **kw)  
Overrides: SCons.Scanner.Current.__init__
```

```
__str__(self)
```

```
add_scanner(self, skey, scanner)
```

```
add_skey(self, skey)
```

Add a skey to the list of skeys

```
find_include_names(self, node)
```

```
get_skeys(self, env=False)
```

```
path(self, env, dir=False, target=False, source=False)
```

```
recurse_nodes(self, nodes)
```

```
scan(self, node, path=())
```

```
select(self, node)
```

21 Module SCons.Scanner.C

SCons.Scanner.C

This module implements the dependency scanner for C/C++ code.

21.1 Functions

dictify_CPPDEFINES (<i>env</i>)
--

CScanner ()

Return a prototype Scanner instance for scanning source files that use the C pre-processor
--

21.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/C.py 3266 2008/08/12 07:31:01 k...

21.3 Class SConsCPPScanner

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__ (<i>self</i> , * <i>args</i> , ** <i>kw</i>)
--

Overrides: SCons.cpp.PreProcessor.__init__
--

initialize_result (<i>self</i> , <i>fname</i>)

Overrides: SCons.cpp.PreProcessor.initialize_result

finalize_result (<i>self</i> , <i>fname</i>)

Overrides: SCons.cpp.PreProcessor.finalize_result

find_include_file (<i>self</i> , <i>t</i>)

Finds the #include file for a given preprocessor tuple.

Overrides: SCons.cpp.PreProcessor.find_include_file <i>exitit</i> (inherited documentation)

read_file (<i>self</i> , <i>file</i>)
--

Overrides: SCons.cpp.PreProcessor.read_file

__call__(*self*, *file*)

Pre-processes a file.

This is the main public entry point.

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.

process_contents(*self*, *contents*, *fname*=False)

Pre-processes a file contents.

This is the main internal entry point.

resolve_include(*self*, *t*)

Resolve a tuple-sized `#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*=False)

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*=False)

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.

21.4 Class *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

<code>__init__(self, name, variable)</code>

<code>__call__(self, node, env, path=())</code>

<code>recurse_nodes(self, nodes)</code>

<code>select(self, node)</code>

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

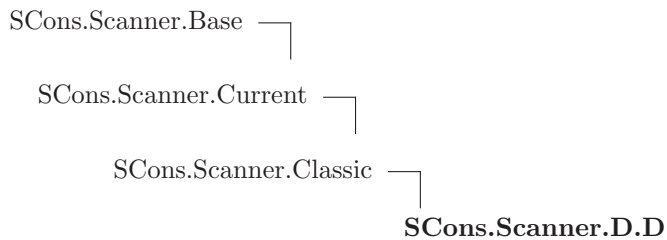
DScanner()

Return a prototype Scanner instance for scanning D source files

22.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/D.py 3266 2008/08/12 07:31:01 k...

22.3 Class D



22.3.1 Methods

__init__(self)

Overrides: SCons.Scanner.Classic.__init__

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

__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)
```

```
__str__(self)
```

```
add_scanner(self, skey, scanner)
```

```
add_skey(self, skey)
```

Add a skey to the list of skeys

```
get_skeys(self, env=False)
```

```
path(self, env, dir=False, target=False, source=False)
```

```
recurse_nodes(self, nodes)
```

```
scan(self, node, path=())
```

```
select(self, node)
```

```
sort_key(self, include)
```

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
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/Dir.py 3266 2008/08/12 07:31:01...
<code>skip_entry</code>	Value: {'.': 1, '..': 1, '.sconsign': 1, .sconsign.bak': 1, '.s...
<code>skip_entry_list</code>	Value: ['.', '..', '.sconsign', '.sconsign.dblite', .sconsign.d...
<code>skip</code>	Value: '.sconsign.db'

24 Module SCons.Scanner.Fortran

SCons.Scanner.Fortran

This module implements the dependency scanner for Fortran code.

24.1 Functions

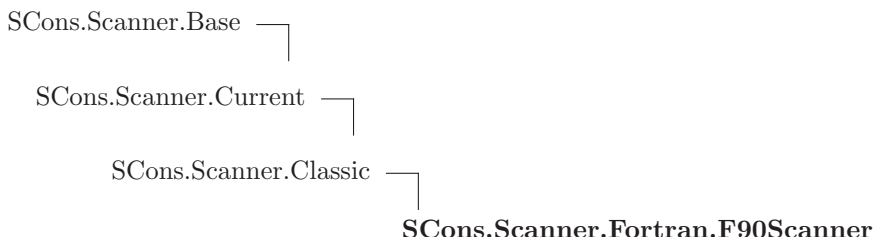
FortranScan(*path_variable*='FORTRANPATH')

Return a prototype Scanner instance for scanning source files for Fortran USE & INCLUDE statements

24.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/Fortran.py 3266 2008/08/12 07:3...

24.3 Class 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*)

Overrides: SCons.Scanner.Classic.__init__

scan(*self, node, env, path=()*)

Overrides: SCons.Scanner.Classic.scan

__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*)

__str__(*self*)

add_scanner(*self*, *key*, *scanner*)

add_key(*self*, *key*)

Add a key to the list of keys

find_include(*self*, *include*, *source_dir*, *path*)

find_include_names(*self*, *node*)

get_keys(*self*, *env*=False)

path(*self*, *env*, *dir*=False, *target*=False, *source*=False)

recurse_nodes(*self*, *nodes*)

select(*self*, *node*)

sort_key(*self*, *include*)

25 Module SCons.Scanner.IDL

SCons.Scanner.IDL

This module implements the dependency 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
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/IDL.py 3266 2008/08/12 07:31:01...'

26 Module SCons.Scanner.LaTeX

SCons.Scanner.LaTeX

This module implements the dependency scanner for LaTeX code.

26.1 Functions

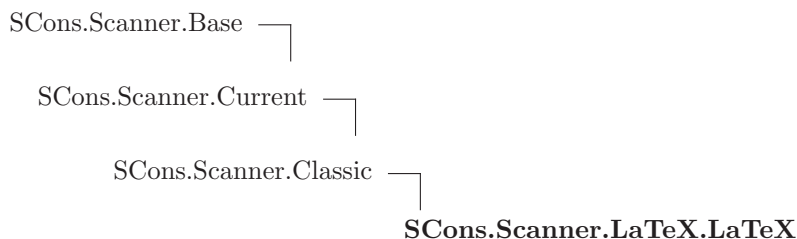
LaTeXScanner()

Return a prototype Scanner instance for scanning LaTeX source files

26.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/LaTeX.py 3266 2008/08/12 07:31:...

26.3 Class 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 need a should append .tex suffix for the “include” keywords, append .tex if there is no extension for the “input” keyword, but leave the file name untouched for “includegraphics.” For the “bibliography” keyword we need to add .bib if there is no extension. (This need to be revisited since if there is no extension for an “includegraphics” keyword latex will append .ps or .eps to find the file; while pdftex will use other extensions.)

26.3.1 Methods

latex_name(self, include)

sort_key(self, include)

Overrides: SCons.Scanner.Classic.sort_key

find_include(self, include, source_dir, path)

Overrides: SCons.Scanner.Classic.find_include

scan(*self*, *node*, *path*=())

Overrides: *SCons.Scanner.Classic.scan*

__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*)

__init__(*self*, *name*, *suffices*, *path_variable*, *regex*, **args*, ***kw*)

Overrides: *SCons.Scanner.Current.__init__*

__str__(*self*)

add_scanner(*self*, *key*, *scanner*)

add_skey(*self*, *skey*)

Add a skey to the list of skeys

find_include_names(*self*, *node*)

get_skeys(*self*, *env*=False)

path(*self*, *env*, *dir*=False, *target*=False, *source*=False)

recurse_nodes(*self*, *nodes*)

select(*self*, *node*)

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
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/Prog.py 3266 2008/08/12 07:31:0...
<code>print_find_libs</code>	Value: False

28 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.

28.1 Modules

- **Interactive:** SCons interactive mode
(Section 29, p. 240)
- **Main:** SCons.Script
(Section 30, p. 244)
- **SConscript’:** SCons.Script.SConscript
(Section 31, p. 257)

28.2 Functions

HelpFunction (<i>text</i>)

Variables (<i>files</i> =[], <i>args</i> ={})

Options (<i>files</i> =[], <i>args</i> ={})

28.3 Variables

Name	Description
<code>__revision__</code>	Value: <code>'src/engine/SCons/Script/__init__.py 3266 2008/08/12 07:3...</code>
<code>start_time</code>	Value: <code>1218551580.18</code>
<code>call_stack</code>	Value: <code>[]</code>
<code>PathVariable</code>	Value: <code><SCons.Variables.PathVariable._PathVariableClass instance...</code>
<code>PathOption</code>	Value: <code><SCons.Variables.PathVariable._PathVariableClass instance...</code>
<code>Chmod</code>	Value: <code>ActionFactory(chmod_func, chmod_strfunc)</code>
<code>Copy</code>	Value: <code>ActionFactory(copy_func, lambda dest, src: 'Copy("%s", "%...</code>
<code>Delete</code>	Value: <code>ActionFactory(delete_func, delete_strfunc)</code>
<code>Mkdir</code>	Value: <code>ActionFactory(mkdir_func, lambda dir: 'Mkdir("%s)' % get_p...</code>

continued on next page

Name	Description
Move	Value: ActionFactory(lambda dest, src: os.rename(src, dest), lam...
Touch	Value: ActionFactory(touch_func, lambda file: 'Touch(%s)' % get_...
CScanner	Value: SCons.Tool.CScanner
DScanner	Value: SCons.Tool.DScanner
DirScanner	Value: SCons.Scanner.Dir.DirScanner()
ProgramScanner	Value: SCons.Tool.ProgramScanner
SourceFileScanner	Value: SCons.Tool.SourceFileScanner
CScan	Value: SCons.Tool.CScanner
ARGUMENTS	Value: {}
ARGLIST	Value: []
BUILD_TARGETS	Value: []
COMMAND_LINE_TARGETS	Value: []
DEFAULT_TARGETS	Value: []
help_text	Value: False
sconsript_reading	Value: 0
GlobalDefaultEnvironmentFunctions	Value: ['Default', 'EnsurePythonVersion', 'EnsureSConsVersion', ...]
GlobalDefaultBuilders	Value: ['CFile', 'CXXFile', 'DVI', 'Jar', 'Java', 'JavaH', 'Libr...
SConsript	Value: <SCons.Script.SConsript.DefaultEnvironmentCall instance ...
Command	Value: <SCons.Script.SConsript.DefaultEnvironmentCall instance ...
AddPostAction	Value: <SCons.Script.SConsript.DefaultEnvironmentCall instance ...
AddPreAction	Value: <SCons.Script.SConsript.DefaultEnvironmentCall instance ...
Alias	Value: <SCons.Script.SConsript.DefaultEnvironmentCall instance ...
AlwaysBuild	Value: <SCons.Script.SConsript.DefaultEnvironmentCall instance ...
BuildDir	Value: <SCons.Script.SConsript.DefaultEnvironmentCall instance ...
CFile	Value: <SCons.Script.SConsript.DefaultEnvironmentCall instance ...

continued on next page

Name	Description
CXXFile	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
CacheDir	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Clean	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
DVI	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Decider	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Default	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Depends	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Dir	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
EnsurePythonVersion	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
EnsureSConsVersion	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Entry	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Execute	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Exit	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Export	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
File	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
FindFile	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...

continued on next page

Name	Description
FindInstalledFiles	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
FindSourceFiles	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Flatten	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
GetBuildPath	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
GetLaunchDir	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Glob	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Help	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Ignore	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Import	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Install	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
InstallAs	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Jar	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Java	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
JavaH	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Library	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Literal	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...

continued on next page

Name	Description
Local	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
M4	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
MSVSPProject	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
NoCache	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
NoClean	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Object	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
PCH	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
PDF	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Package	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
ParseDepends	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
PostScript	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Precious	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Program	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
RES	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
RMIC	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Repository	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...

continued on next page

Name	Description
Requires	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
SConscriptChdir	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
SConsignFile	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
SharedLibrary	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
SharedObject	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
SideEffect	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
SourceCode	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
SourceSignatures	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Split	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
StaticLibrary	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
StaticObject	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Tag	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Tar	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
TargetSignatures	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
TypeLibrary	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Value	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...

continued on next page

Name	Description
VariantDir	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Zip	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...

28.4 Class *TargetList*

UserList.UserList —
SCons.Script.TargetList

28.4.1 Methods

`__add__(self, other)`

`__cmp__(self, other)`

`__contains__(self, item)`

`__delitem__(self, i)`

`__delslice__(self, i, j)`

`__eq__(self, other)`

`__ge__(self, other)`

`__getitem__(self, i)`

`__getslice__(self, i, j)`

`__gt__(self, other)`

`__iadd__(self, other)`

`__imul__(self, n)`

`__init__(self, initlist=False)`

`__le__(self, other)`

`__len__(self)`

`__lt__(self, other)`

`__mul__(self, n)``__ne__(self, other)``__radd__(self, other)``__repr__(self)``__rmul__(self, n)``__setitem__(self, i, item)``__setslice__(self, i, j, other)``append(self, item)``count(self, item)``extend(self, other)``index(self, item, *args)``insert(self, i, item)``pop(self, i=-1)``remove(self, item)``reverse(self)``sort(self, *args, **kws)`

29 Module *SCons.Script.Interactive*

SCons interactive mode

29.1 Functions

interact (<i>fs, parser, options, targets, target_top</i>)

29.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Script/Interactive.py 3266 2008/08/12 0...
<code>__doc__</code>	Value: ...

29.3 Class *SConsInteractiveCmd*

```
cmd.Cmd └─ SCons.Script.Interactive.SConsInteractiveCmd
```

<code>build [TARGETS]</code>	Build the specified TARGETS and their dependencies. 'b' is a synonym.
<code>clean [TARGETS]</code>	Clean (remove) the specified TARGETS and their dependencies. 'c' is a synonym.
<code>exit</code>	Exit SCons interactive mode.
<code>help [COMMAND]</code>	Prints help for the specified COMMAND. 'h' and '?' are synonyms.
<code>shell [COMMANDLINE]</code>	Execute COMMANDLINE in a subshell. 'sh' and '!' are synonyms.
<code>version</code>	Prints SCons version information.

29.3.1 Methods

__init__ (<i>self, **kw</i>) 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__ exitit(inherited documentation)
--

default (<i>self, argv</i>) 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 exitit(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` `exitit`(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.

cmdloop(*self*, *intro=False*)

Repeatedly issue a prompt, accept input, parse an initial prefix off the received input, and dispatch to action methods, passing them the remainder of the line as argument.

columnize(*self*, *list*, *displaywidth=80*)

Display a list of strings as a compact set of columns.

Each column is only as wide as necessary. Columns are separated by two spaces (one was not legible enough).

complete(*self*, *text*, *state*)

Return the next possible completion for 'text'.

If a command has not been entered, then complete against command list. Otherwise try to call `complete_<command>` to get list of completions.

complete_help(*self*, **args*)

completedefault(*self*, **ignored*)

Method called to complete an input line when no command-specific `complete_*`() method is available. By default, it returns an empty list.

completenames(*self*, *text*, **ignored*)

emptyline(*self*)

Called when an empty line is entered in response to the prompt. If this method is not overridden, it repeats the last nonempty command entered.

get_names(*self*)

parseline(*self*, *line*)

Parse the line into a command name and a string containing the arguments. Returns a tuple containing (command, args, line). 'command' and 'args' may be None if the line couldn't be parsed.

postcmd(*self*, *stop*, *line*)

Hook method executed just after a command dispatch is finished.

postloop(*self*)

Hook method executed once when the `cmdloop()` method is about to return.

precmd(*self*, *line*)

Hook method executed just before the command line is interpreted, but after the input prompt is generated and issued.

preloop(*self*)

Hook method executed once when the `cmdloop()` method is called.

print_topics(*self*, *header*, *cmds*, *cmdlen*, *maxcol*)

29.3.2 Class Variables

Name	Description
synonyms	Value: {'b': 'build', 'c': 'clean', 'h': 'help', 'scons': 'build...'}
doc_header	Value: 'Documented commands (type help <topic>):'
doc_leader	Value: ''
identchars	Value: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_.'...

continued on next page

Name	Description
intro	Value: False
lastcmd	Value: ''
misc_header	Value: 'Miscellaneous help topics:'
nohelp	Value: '*** No help on %s'
prompt	Value: '(Cmd) '
ruler	Value: '='
undoc_header	Value: 'Undocumented commands:'
use_rawinput	Value: False

30 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.

30.1 Functions

<code>fetch_win32_parallel_msg()</code>

<code>Progress(*args, **kw)</code>

<code>GetBuildFailures()</code>

<code>python_version_string()</code>

<code>python_version_unsupported(version=(2, 5, 1, 'final', 0))</code>
--

<code>python_version_deprecated(version=(2, 5, 1, 'final', 0))</code>

<code>AddOption(*args, **kw)</code>

<code>GetOption(name)</code>

<code>SetOption(name, value)</code>

<code>find_deepest_user_frame(tb)</code>
--

Find the deepest stack frame that is not part of SCons. Input is a “pre-processed” stack trace in the form returned by <code>traceback.extract_tb()</code> or <code>traceback.extract_stack()</code>

<code>version_string(label, module)</code>
--

<code>main()</code>

30.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Script/Main.py 3266 2008/08/12 07:31:01...'
<code>display</code>	Value: SCons.Util.display

continued on next page

Name	Description
progress_display	Value: <code>SCons.Util.DisplayEngine()</code>
first_command_start	Value: <code>False</code>
last_command_end	Value: <code>False</code>
ProgressObject	Value: <code>Null()</code>
print_objects	Value: <code>0</code>
print_memoizer	Value: <code>0</code>
print_stacktrace	Value: <code>0</code>
print_time	Value: <code>0</code>
sconscript_time	Value: <code>0</code>
cumulative_command_time	Value: <code>0</code>
exit_status	Value: <code>0</code>
this_build_status	Value: <code>0</code>
num_jobs	Value: <code>False</code>
delayed_warnings	Value: <code>[]</code>
OptionsParser	Value: <code>FakeOptionParser()</code>
count_stats	Value: <code>CountStats()</code>
memory_stats	Value: <code>MemStats()</code>

30.3 Class *SConsPrintHelpException*



30.3.1 Methods

`__delattr__(...)`

`x.__delattr__('name') <==> del x.name`

Overrides: `object.__delattr__`

`__getattr__(...)`

`x.__getattr__('name') <==> x.name`

Overrides: `object.__getattr__`

`__getitem__(x, y)`

`x[y]`

__getslice__(*x, i, j*)

x[i:j]

Use of negative indices is not supported.

__hash__(*x*)

hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(*T, S, ...*)

Return Valuea new object with type *S*, a subtype of *T*

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)

__str__(*x*)

str(x)

Overrides: object.__str__

30.3.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>

continued on next page

Name	Description
args	Value: <attribute 'args' of 'exceptions.BaseException' objects>
message	Value: <member 'message' of 'exceptions.BaseException' objects>

30.4 Class Progressor

30.4.1 Methods

```
__init__(self, obj, interval=False, file=False, overwrite=False)
```

```
write(self, s)
```

```
erase_previous(self)
```

```
spinner(self, node)
```

```
string(self, node)
```

```
replace_string(self, node)
```

```
__call__(self, node)
```

30.4.2 Class Variables

Name	Description
prev	Value: ''
count	Value: 0
target_string	Value: '\$TARGET'

30.5 Class BuildTask

```
SCons.Taskmaster.Task └─ SCons.Script.Main.BuildTask
```

An SCons build task.

30.5.1 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*)

Called to determine whether the task's execute() method should be run.

This method allows one to skip the somewhat costly execution of the execute() method in a separate thread. For example, that would be unnecessary for up-to-date targets.

Overrides: SCons.Taskmaster.Task.needs_execute 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)

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.

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 exitit(inherited documentation)

make_ready(*self*)

Make a task ready for execution

Overrides: SCons.Taskmaster.Task.make_ready

__init__(*self*, *tm*, *targets*, *top*, *node*)**exc_clear(*self*)**

Clears any recorded exception.

This also changes the "exception_raise" attribute to point to the appropriate do-nothing method.

exc_info(*self*)

Returns info about a recorded exception.

exception_set(*self*, *exception*=False)

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

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_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.

fail_continue(*self*)

Explicit continue-the-build failure.

This sets failure status on the target nodes and all of their dependent parent nodes.

fail_stop(*self*)

Explicit stop-the-build failure.

get_target(*self*)

Fetch the target being built or updated by this task.

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.

30.5.2 Class Variables

Name	Description
progress	Value: Null()

30.6 Class CleanTask



An SCons clean task.

30.6.1 Methods

fs_delete(*self*, *path*, *pathstr*, *remove=False*)**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` `exitit`(inherited documentation)

__init__(*self*, *tm*, *targets*, *top*, *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.

exc_clear(*self*)

Clears any recorded exception.

This also changes the "exception_raise" attribute to point to the appropriate do-nothing method.

exc_info(*self*)

Returns info about a recorded exception.

exception_set(*self*, *exception*=False)

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

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_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.

fail_continue(*self*)

Explicit continue-the-build failure.

This sets failure status on the target nodes and all of their dependent parent nodes.

fail_stop(*self*)

Explicit stop-the-build failure.

failed(*self*)

Default action when a task fails: stop the build.

get_target(*self*)

Fetch the target being built or updated by this task.

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.

needs_execute(*self*)

Called to determine whether the task's execute() method should be run.

This method allows one to skip the somewhat costly execution of the execute() method in a separate thread. For example, that would be unnecessary for up-to-date targets.

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.

30.7 Class QuestionTask

SCons.Taskmaster.Task

SCons.Script.Main.QuestionTask

An SCons task for the -q (question) option.

30.7.1 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)

__init__(*self*, *tm*, *targets*, *top*, *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.

exc_clear(*self*)

Clears any recorded exception.

This also changes the "exception_raise" attribute to point to the appropriate do-nothing method.

exc_info(*self*)

Returns info about a recorded exception.

exception_set(*self*, *exception*=False)

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

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_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.

fail_continue(*self*)

Explicit continue-the-build failure.

This sets failure status on the target nodes and all of their dependent parent nodes.

fail_stop(*self*)

Explicit stop-the-build failure.

failed(*self*)

Default action when a task fails: stop the build.

get_target(*self*)

Fetch the target being built or updated by this task.

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.

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.

needs_execute(*self*)

Called to determine whether the task's execute() method should be run.

This method allows one to skip the somewhat costly execution of the execute() method in a separate thread. For example, that would be unnecessary for up-to-date targets.

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.

30.8 Class TreePrinter

30.8.1 Methods

__init__(*self*, *derived*=False, *prune*=False, *status*=False)

get_all_children(*self*, *node*)

get_derived_children(*self*, *node*)

display(*self*, *t*)

30.9 Class 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.

30.9.1 Methods

add_local_option(*self*, **args*, ***kw*)

30.9.2 Class Variables

Name	Description
values	Value: FakeOptionValues()

30.10 Class Stats

Known Subclasses: SCons.Script.Main.CountStats, SCons.Script.Main.MemStats

30.10.1 Methods

__init__(*self*)

enable(*self*, *outfp*)

do_nothing(*self*, **args*, ***kw*)

30.11 Class CountStats

SCons.Script.Main.Stats —
SCons.Script.Main.CountStats

30.11.1 Methods

do_append(*self*, *label*)

do_print(*self*)

__init__(*self*)

do_nothing(*self*, **args*, ***kw*)

enable(*self*, *outfp*)

30.12 Class MemStats

SCons.Script.Main.Stats —
SCons.Script.Main.MemStats

30.12.1 Methods

do_append(*self*, *label*)

do_print(*self*)

__init__(*self*)

do_nothing(*self*, **args*, ***kw*)

enable(*self*, *outfp*)

31 Module *SCons.Script.SConscript*

SCons.Script.SConscript

This module defines the Python API provided to *SConscript* and *SConstruct* files.

31.1 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.stdout*)

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.

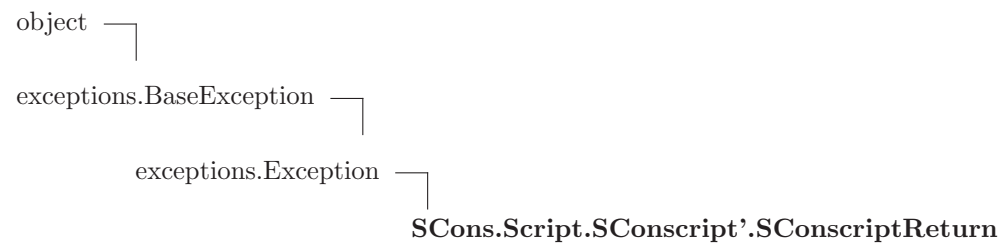
31.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Script/SConscript.py 3266 2008/08/12 07...
<code>launch_dir</code>	Value: '/home/knight/SCons/tigris.org/trunk'
<code>GlobalDict</code>	Value: False
<code>global_exports</code>	Value: {}
<code>sconscript_chdir</code>	Value: False
<code>call_stack</code>	Value: []

continued on next page

Name	Description
stack_bottom	Value: '% Stack boTTom %'

31.3 Class *SConscriptReturn*



31.3.1 Methods

<code>__delattr__</code> (...)
<code>x.__delattr__('name') <==> del x.name</code>
Overrides: <code>object.__delattr__</code>

<code>__getattr__</code> (...)
<code>x.__getattr__('name') <==> x.name</code>
Overrides: <code>object.__getattr__</code>

<code>__getitem__</code> (<i>x</i> , <i>y</i>)
<code>x[y]</code>

<code>__getslice__</code> (<i>x</i> , <i>i</i> , <i>j</i>)
<code>x[i:j]</code>
Use of negative indices is not supported.

<code>__hash__</code> (<i>x</i>)
<code>hash(x)</code>

<code>__init__</code> (...)
<code>x.__init__</code> (...) initializes x; see <code>x.__class__.__doc__</code> for signature
Overrides: <code>exceptions.BaseException.__init__</code>

<code>__new__</code> (<i>T</i> , <i>S</i> , ...)
Return Value
a new object with type <i>S</i> , a subtype of <i>T</i>
Overrides: <code>exceptions.BaseException.__new__</code>

```
__reduce__(...)
helper for pickle
Overrides: object.__reduce__ extit(inherited documentation)
```

```
__reduce_ex__(...)
helper for pickle
```

```
__repr__(x)
repr(x)
Overrides: object.__repr__
```

```
__setattr__(...)
x.__setattr__('name', value) <==> x.name = value
Overrides: object.__setattr__
```

```
__setstate__(...)
```

```
__str__(x)
str(x)
Overrides: object.__str__
```

31.3.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

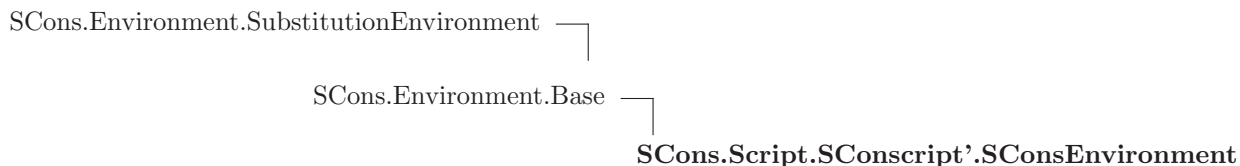
31.4 Class Frame

A frame on the SConstruct/SConscript call stack

31.4.1 Methods

```
__init__(self, fs, exports, sconscript)
```


31.5 Class *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.

31.5.1 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*)

GetLaunchDir(*self*)

GetOption(*self*, *name*)

Help(*self*, *text*)

Import(*self*, **vars*)

SConscript(*self*, **ls*, ***kw*)

SConscriptChdir(*self*, *flag*)

SetOption(*self*, *name*, *value*)

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

AddMethod(*self*, *function*, *name*=False)

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.

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

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

Alias(*self*, *target*, *source*=[], *action*=False, ***kw*)

AlwaysBuild(*self*, **targets*)

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

Append values to existing construction variables in an Environment.

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

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.

AppendUnique(*self*, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there.

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

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

CacheDir(*self*, *path*)

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

Clone(*self*, *tools*=[], *toolpath*=False, *parse_flags*=False, ***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.

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

Decider(*self*, *function*)

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

Explicitly 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*=False)

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*)

MergeFlags(*self*, *args*, *unique=False*)

Merge the dict in args into the construction variables. 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.

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

Override(*self*, *overrides*)

Produce a modified environment whose variables are overridden 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.

ParseConfig(*self*, *command*, *function=False*, *unique=False*)

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=False*, *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.

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.

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*=':')

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.

PrependUnique(*self*, ***kw*)

Append values to existing construction variables in an Environment, if they're not already there.

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.

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*=False)

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 allowing files to be specified as a white-space separated list to be split.

The input rules are:

- A single string containing names separated by spaces. These will be split apart at the spaces.
- A single Node instance
- A list containing either strings or Node instances. 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*=False, ***kw*)

Value(*self*, *value*, *built_value*=False)

VariantDir(*self*, *variant_dir*, *src_dir*, *duplicate*=False)

WhereIs(*self*, *prog*, *path*=False, *pathext*=False, *reject*=[])

Find prog in the path.

__cmp__(*self*, *other*)

__delitem__(*self*, *key*)

__getitem__(*self*, *key*)

__init__(*self*, *platform*=False, *tools*=False, *toolpath*=False, *variables*=False, *parse_flags*=False, ***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 (SubstitutionEnvironment) initialization, because we need to initialize things in a very specific order that doesn't work with the much simpler base class initialization.

Overrides: SCons.Environment.SubstitutionEnvironment.__init__

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

arg2nodes(*self*, *args*, *node_factory*=<class *SCons.Environment._Null* at 0x13ff0b0>, *lookup_list*=<class *SCons.Environment._Null* at 0x13ff0b0>, ***kw*)

backtick(*self*, *command*)

get(*self*, *key*, *default*=False)

Emulates the get() method of dictionaries.

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*)

gvars(*self*)

has_key(*self*, *key*)

items(*self*)

lvars(*self*)

scanner_map_delete(*self*, *kw*=False)

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

subst(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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*=False, *source*=False)

subst_list(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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

subst_path(*self*, *path*, *target*=False, *source*=False)

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

subst_target_source(*self*, *string*, *raw*=0, *target*=False, *source*=False, *conv*=False)

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.

31.5.2 Class Variables

Name	Description
<code>__metaclass__</code>	Value: <code>SCons.Memoize.Memoized_Metaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

31.6 Class `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).

31.6.1 Methods

__init__(*self*, *method_name*, *subst*=0)

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

32 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.

32.1 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Sig.py 3266 2008/08/12 07:31:01 knight'
<code>__doc__</code>	Value: ""Place-holder for the old SCons.Sig module hierar...
<code>msg</code>	Value: 'The SCons.Sig module no longer exists.\nRemove the f...
<code>default_calc</code>	Value: False
<code>default_module</code>	Value: False
<code>MD5</code>	Value: MD5Null()
<code>TimeStamp</code>	Value: TimeStampNull()

32.2 Class MD5Null

```
SCons.Util.Null └─ SCons.Sig.MD5Null
```

32.2.1 Methods

```
__repr__(self)  
Overrides: SCons.Util.Null.__repr__
```

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

```
__delattr__(self, name)
```

```
__getattr__(self, mname)
```

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

```
__new__(cls, *args, **kwargs)
```

```
__nonzero__(self)
```

<code>__setattr__(self, name, value)</code>

32.3 Class `TimeStampNull`



32.3.1 Methods

<code>__repr__(self)</code> Overrides: <code>SCons.Util.Null.__repr__</code>

<code>__call__(self, *args, **kwargs)</code>
--

<code>__delattr__(self, name)</code>

<code>__getattr__(self, mname)</code>

<code>__init__(self, *args, **kwargs)</code>
--

<code>__new__(cls, *args, **kwargs)</code>
--

<code>__nonzero__(self)</code>

<code>__setattr__(self, name, value)</code>

33 Module SCons.Subst

SCons.Subst

SCons string substitution.

33.1 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(*list*, *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=False*, *target=False*, *source=False*, *gvars={}*, *lvars={}*, *conv=False*)

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=False*, *target=False*, *source=False*, *gvars={}*, *lvars={}*, *conv=False*)

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...

33.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Subst.py 3266 2008/08/12 07:31:01 knight'
<code>AllowableExceptions</code>	Value: (<type 'exceptions.IndexError'>, <type 'exceptions.NameEr...>)
<code>SUBST_CMD</code>	Value: 0
<code>SUBST_RAW</code>	Value: False
<code>SUBST_SIG</code>	Value: 2

33.3 Class 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.

33.3.1 Methods

```
__init__(self, lstr)
```

```
__str__(self)
```

```
escape(self, escape_func)
```

```
for_signature(self)
```

```
is_literal(self)
```

33.4 Class 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.

33.4.1 Methods

`__init__(self, lstr, for_signature=False)`

The `for_signature` parameter, if supplied, will be the canonical string we return from `for_signature()`. Else we will simply return `lstr`.

`__str__(self)`

`escape(self, escape_func)`

`for_signature(self)`

`is_literal(self)`

33.5 Class 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.

33.5.1 Methods

`__init__(self, cmd, literal=False)`

Overrides: `UserString.UserString.__init__`

`is_literal(self)`

`escape(self, escape_func, quote_func=<function quote_spaces at 0xe5f410>)`

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.

`__add__(self, other)`

`__cmp__(self, string)`

`__complex__(self)`

`__contains__(self, char)`

`__float__(self)`

`__getitem__(self, index)``__getslice__(self, start, end)``__hash__(self)``__int__(self)``__len__(self)``__long__(self)``__mod__(self, args)``__mul__(self, n)``__radd__(self, other)``__repr__(self)``__rmul__(self, n)``__str__(self)``capitalize(self)``center(self, width, *args)``count(self, sub, start=0, end=9223372036854775807)``decode(self, encoding=False, errors=False)``encode(self, encoding=False, errors=False)``endswith(self, suffix, start=0, end=9223372036854775807)``expandtabs(self, tabsize=8)``find(self, sub, start=0, end=9223372036854775807)``index(self, sub, start=0, end=9223372036854775807)``isalnum(self)``isalpha(self)``isdecimal(self)`

`isdigit(self)``islower(self)``isnumeric(self)``isspace(self)``istitle(self)``isupper(self)``join(self, seq)``ljust(self, width, *args)``lower(self)``lstrip(self, chars=False)``partition(self, sep)``replace(self, old, new, maxsplit=-1)``rfind(self, sub, start=0, end=9223372036854775807)``rindex(self, sub, start=0, end=9223372036854775807)``rjust(self, width, *args)``rpartition(self, sep)``rsplit(self, sep=False, maxsplit=-1)``rstrip(self, chars=False)``split(self, sep=False, maxsplit=-1)``splitlines(self, keepends=0)``startswith(self, prefix, start=0, end=9223372036854775807)``strip(self, chars=False)``swapcase(self)``title(self)`

```
translate(self, *args)
```

```
upper(self)
```

```
zfill(self, width)
```

33.6 Class 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...

33.6.1 Methods

```
__init__(self, list, func)
```

33.7 Class Targets_or_Sources

```
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 `UserList.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 `UserList.UserList` methods in practice.

33.7.1 Methods

```
__init__(self, nl)
```

Overrides: `UserList.UserList.__init__`

```
__getattr__(self, attr)
```

```
__getitem__(self, i)
```

Overrides: `UserList.UserList.__getitem__`

```
__getslice__(self, i, j)
```

Overrides: `UserList.UserList.__getslice__`

```
__str__(self)
```


`__repr__(self)`
Overrides: `UserList.UserList.__repr__`

`__add__(self, other)`

`__cmp__(self, other)`

`__contains__(self, item)`

`__delitem__(self, i)`

`__delslice__(self, i, j)`

`__eq__(self, other)`

`__ge__(self, other)`

`__gt__(self, other)`

`__iadd__(self, other)`

`__imul__(self, n)`

`__le__(self, other)`

`__len__(self)`

`__lt__(self, other)`

`__mul__(self, n)`

`__ne__(self, other)`

`__radd__(self, other)`

`__rmul__(self, n)`

`__setitem__(self, i, item)`

`__setslice__(self, i, j, other)`

`append(self, item)`

`count(self, item)`

`extend(self, other)`

```
index(self, item, *args)
```

```
insert(self, i, item)
```

```
pop(self, i=-1)
```

```
remove(self, item)
```

```
reverse(self)
```

```
sort(self, *args, **kws)
```

33.8 Class *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.

33.8.1 Methods

```
__init__(self, nl)
```

```
__getattr__(self, attr)
```

```
__str__(self)
```

```
__repr__(self)
```

34 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.

34.1 Functions

dump_stats()

find_cycle(<i>stack</i>, <i>visited</i>)

34.2 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>__revision__</code>	Value: 'src/engine/SCons/Taskmaster.py 3266 2008/08/12 07:31:01 ...
<code>StateString</code>	Value: {0: 'no_state', 1: 'pending', 2: 'executing', 3: 'up_to_d...
<code>NODE_NO_STATE</code>	Value: 0
<code>NODE_PENDING</code>	Value: False
<code>NODE_EXECUTING</code>	Value: 2
<code>NODE_UP_TO_DATE</code>	Value: 3
<code>NODE_EXECUTED</code>	Value: 4
<code>NODE_FAILED</code>	Value: 5
<code>CollectStats</code>	Value: False
<code>StatsNodes</code>	Value: []
<code>fmt</code>	Value: '%(considered)3d %(already_handled)3d %(problem)3d %(chil...

34.3 Class 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.)

34.3.1 Methods

`__init__(self)`

Instantiates a Taskmaster.Stats object, initializing all appropriate counters to zero.

34.4 Class Task

Known Subclasses: SCons.SConf.SConfBuildTask, SCons.Script.Main.BuildTask, SCons.Script.Main.CleanTask, SCons.Script.Main.QuestionTask

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 customize 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.

34.4.1 Methods

`__init__(self, tm, targets, top, 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 actual 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*)

Called to determine whether the task's `execute()` method should be run.
 This method allows one to skip the somewhat costly execution of the `execute()` method in a separate thread. For example, that would be unnecessary for up-to-date targets.

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.

fail_stop(*self*)

Explicit stop-the-build failure.

fail_continue(*self*)

Explicit continue-the-build failure.

This sets failure status on the target nodes and all of their dependent parent nodes.

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*=False)

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

34.5 Class Taskmaster

The Taskmaster for walking the dependency DAG.

34.5.1 Methods

__init__(*self*, *targets*=[], *tasker*=<class SCons.Taskmaster.Task at 0x1a09890>, *order*=False, *trace*=False)

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.

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*, *mark_fail*=<function <lambda> at 0x1a16e60>)

Perform clean-up about nodes that will never be built.

stop(*self*)

Stops the current build completely.

cleanup(*self*)

Check for dependency cycles.

35 Module *SCons.Util*

SCons.Util

Various utility functions go here.

35.1 Functions

dictify(*keys*, *values*, *result*={})

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 inconsistent 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*=[], *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 children of *root*.

1 results in a pipe, and 0 results in no pipe.

visited - a dictionary of visited nodes in the current 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`, except it prints lines directly instead of creating a string representation 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 children of root.
 1 results in a pipe, and 0 results in no pipe.
`visited` - a dictionary of visited nodes in the current branch if not prune,
 or in the whole tree if prune.

```
is_Dict(obj, instance=<built-in function isinstance>, DictTypes=dict, UserDict)
```

```
is_List(obj, instance=<built-in function isinstance>, ListTypes=list, UserList)
```

```
is_Sequence(obj, instance=<built-in function isinstance>, SequenceTypes=(<type  
'list'>, <type 'tuple'>, <class UserList.UserList ...>))
```

```
is_Tuple(obj, instance=<built-in function isinstance>, tuple=<type 'tuple'>)
```

```
is_String(obj, instance=<built-in function isinstance>, StringTypes=(<type 'str'>,  
<type 'unicode'>, <class UserString.UserSt...>))
```

```
is_Scalar(obj, instance=<built-in function isinstance>, StringTypes=(<type 'str'>,  
<type 'unicode'>, <class UserString.UserSt..., SequenceTypes=(<type 'list'>, <type  
'tuple'>, <class UserList.UserList ...>))
```

```
do_flatten(sequence, result, instance=<built-in function isinstance>, StringTypes=(<type  
'str'>, <type 'unicode'>, <class UserString.UserSt..., SequenceTypes=(<type 'list'>,  
<type 'tuple'>, <class UserList.UserList ...>))
```

```
flatten(obj, instance=<built-in function isinstance>, StringTypes=(<type 'str'>, <type  
'unicode'>, <class UserString.UserSt..., SequenceTypes=(<type 'list'>, <type  
'tuple'>, <class UserList.UserList ..., do_flatten=<function do_flatten at 0xd29f50>))
```

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, instance=<built-in function isinstance>, StringTypes=(<type 'str'>, <type 'unicode'>, <class UserString.UserSt...), SequenceTypes=(<type 'list'>, <type 'tuple'>, <class UserList.UserList ...), do_flatten=<function do_flatten at 0xd29f50>)
```

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, instance=<built-in function isinstance>, str=<type 'str'>, UserString=<class UserString.UserString at 0x2b53a6a07fb0>, BaseStringTypes=(<type 'str'>, <type 'unicode'>))
```

```
to_String_for_subst(s, instance=<built-in function isinstance>, join=<function join at 0x2b53a7f1cc80>, str=<type 'str'>, to_String=<function to_String at 0xd2b140>, BaseStringTypes=(<type 'str'>, <type 'unicode'>), SequenceTypes=(<type 'list'>, <type 'tuple'>, <class UserList.UserList ...), UserString=<class UserString.UserString at 0x2b53a6a07fb0>)
```

```
to_String_for_signature(obj, to_String_for_subst=<function to_String_for_subst at 0xd2b1b8>, AttributeError=<type 'exceptions.AttributeError'>)
```

```
semi_deepcopy(x)
```

```
RegGetValue(root, key)
```

This utility function returns a value in the registry without having to open the key first. Only available on Windows platforms with a version of Python that can read the registry. Returns the same thing as `SCons.Util.RegQueryValueEx`, except you just specify the entire path to the value, and don't have to bother opening the key first. So:

Instead of: `k = SCons.Util.RegOpenKeyEx(SCons.Util.HKEY_LOCAL_MACHINE, r'SOFTWARE\Microsoft\Windows\CurrentVersion')`
`out = SCons.Util.RegQueryValueEx(k, 'ProgramFilesDir')`

You can write: `out = SCons.Util.RegGetValue(SCons.Util.HKEY_LOCAL_MACHINE, r'SOFTWARE\Microsoft\Windows\CurrentVersion\ProgramFilesDir')`

```
WhereIs(file, path=False, pathtext=False, reject=[])
```

```
PrependPath(oldpath, newpath, sep=':')
```

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"`

AppendPath(*oldpath*, *newpath*, *sep*=':')

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 use `os.path.normpath` and `os.path.normcase` on 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”

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 $O(N \cdot \log^2(N))$ 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*=False)

uniquer_hashables(*seq*)

make_path_relative(*path*)

makes an absolute path name to a relative pathname.

AddMethod(*object, function, name=False*)

Adds either a bound method to an instance or an unbound method to a class. If name is omitted 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*)**MD5collect**(*signatures*)

Collects a list of signatures into an aggregate signature.
signatures - a list of signatures returns - the aggregate signature

35.2 Variables

Name	Description
UnicodeType	Value: types.UnicodeType
DictTypes	Value: dict, UserDict
ListTypes	Value: list, UserList
SequenceTypes	Value: (<type 'list'>, <type 'tuple'>, <class UserList.UserList ...
StringTypes	Value: (<type 'str'>, <type 'unicode'>, <class UserString.UserSt...
BaseStringTypes	Value: (<type 'str'>, <type 'unicode'>)
d	Value: {<type 'instance'>: <function _semi_deepcopy_inst at 0xd2...
can_read_reg	Value: 0
hkey_mod	Value: win32con
RegOpenKeyEx	Value: win32api.RegOpenKeyEx
RegEnumKey	Value: win32api.RegEnumKey
RegEnumValue	Value: win32api.RegEnumValue
RegQueryValueEx	Value: win32api.RegQueryValueEx
HKEY_CLASSES_ROOT	Value: hkey_mod.HKEY_CLASSES_ROOT
HKEY_LOCAL_MACHINE	Value: hkey_mod.HKEY_LOCAL_MACHINE

continued on next page

Name	Description
HKEY_CURRENT_USER	Value: <code>hkey_mod.HKEY_CURRENT_USER</code>
HKEY_USERS	Value: <code>hkey_mod.HKEY_USERS</code>
display	Value: <code>SCons.Util.display</code>
md5	Value: <code>True</code>

35.3 Class CallableComposite

UserList.UserList —
SCons.Util.CallableComposite

A simple composite callable class that, when called, will invoke all of its contained callables with the same arguments.

35.3.1 Methods

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

`__add__(self, other)`

`__cmp__(self, other)`

`__contains__(self, item)`

`__delitem__(self, i)`

`__delslice__(self, i, j)`

`__eq__(self, other)`

`__ge__(self, other)`

`__getitem__(self, i)`

`__getslice__(self, i, j)`

`__gt__(self, other)`

`__iadd__(self, other)`

`__imul__(self, n)`

`__init__(self, initlist=False)`

`__le__(self, other)`

`__len__(self)``__lt__(self, other)``__mul__(self, n)``__ne__(self, other)``__radd__(self, other)``__repr__(self)``__rmul__(self, n)``__setitem__(self, i, item)``__setslice__(self, i, j, other)``append(self, item)``count(self, item)``extend(self, other)``index(self, item, *args)``insert(self, i, item)``pop(self, i=-1)``remove(self, item)``reverse(self)``sort(self, *args, **kws)`

35.4 Class NodeList

```
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' ]
```

35.4.1 Methods

```
__nonzero__(self)
```

```
__str__(self)
```

```
__getattr__(self, name)
```

```
__add__(self, other)
```

```
__cmp__(self, other)
```

```
__contains__(self, item)
```

```
__delitem__(self, i)
```

```
__delslice__(self, i, j)
```

```
__eq__(self, other)
```

```
__ge__(self, other)
```

```
__getitem__(self, i)
```

```
__getslice__(self, i, j)
```

```
__gt__(self, other)
```

```
__iadd__(self, other)
```

```
__imul__(self, n)
```

```
__init__(self, initlist=False)
```

```
__le__(self, other)
```

```
__len__(self)
```

```
__lt__(self, other)
```

```
__mul__(self, n)
```

```
__ne__(self, other)
```

```
__radd__(self, other)
```

```
__repr__(self)
```

```
__rmul__(self, n)
```

```
__setitem__(self, i, item)
```

```
__setslice__(self, i, j, other)
```

```
append(self, item)
```

```
count(self, item)
```

```
extend(self, other)
```

```
index(self, item, *args)
```

```
insert(self, i, item)
```

```
pop(self, i=-1)
```

```
remove(self, item)
```

```
reverse(self)
```

```
sort(self, *args, **kws)
```

35.5 Class DisplayEngine

35.5.1 Methods

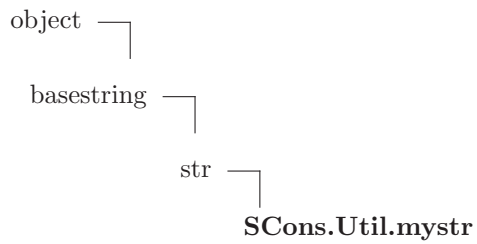
```
__init__(self)
```

```
print_it(self, text, append_newline=False)
```

```
dont_print(self, text, append_newline=False)
```

```
set_mode(self, mode)
```


35.6 Class *mystr*



35.6.1 Methods

`__add__`(*x*, *y*)

`x+y`

`__contains__`(*x*, *y*)

`y in x`

`__delattr__`(...)

`x.__delattr__('name') <==> del x.name`

`__eq__`(*x*, *y*)

`x==y`

`__ge__`(*x*, *y*)

`x>=y`

`__getattr__`(...)

`x.__getattr__('name') <==> x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getnewargs__`(...)

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

__gt__(*x*, *y*)

x>*y*

__hash__(*x*)

hash(*x*)

Overrides: object.__hash__

__init__(...)

x.__init__(...) initializes *x*; see *x*.__class__.__doc__ for signature

__le__(*x*, *y*)

x<=*y*

__len__(*x*)

len(*x*)

__lt__(*x*, *y*)

x<*y*

__mod__(*x*, *y*)

x%*y*

__mul__(*x*, *n*)

*x***n*

__ne__(*x*, *y*)

x!=*y*

__new__(*T*, *S*, ...)

Return Value

a new object with type *S*, a subtype of *T*

Overrides: basestring.__new__

__reduce__(...)

helper for pickle

__reduce_ex__(...)

helper for pickle

__repr__(*x*)repr(*x*)

Overrides: object.__repr__

__rmod__(*x*, *y*)*y*%*x***__rmul__**(*x*, *n*)*n***x***__setattr__**(...)*x*.__setattr__('name', value) <==> *x*.name = value**__str__**(*x*)str(*x*)

Overrides: object.__str__

capitalize(*S*)Return a copy of the string *S* with only its first character capitalized.**Return Value**

string

center(*S*, *width*, *fillchar*=...)Return *S* centered in a string of length *width*. Padding is done using the specified fill character (default is a space)**Return Value**

string

count(*S*, *sub*, *start*=..., *end*=...)Return the number of non-overlapping occurrences of substring *sub* in string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.**Return Value**

int

decode(*S*, *encoding*=..., *errors*=...)Decodes *S* using the codec registered for encoding. *encoding* defaults to the default encoding. *errors* may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeDecodeError. Other possible values are 'ignore' and 'replace' as well as any other name registered with codecs.register_error that is able to handle UnicodeDecodeErrors.**Return Value**

object

encode(*S*, *encoding*=..., *errors*=...)

Encodes *S* using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that is able to handle UnicodeEncodeErrors.

Return Value

object

endswith(*S*, *suffix*, *start*=..., *end*=...)

Return True if *S* ends with the specified suffix, False otherwise. With optional start, test *S* beginning at that position. With optional end, stop comparing *S* at that position. suffix can also be a tuple of strings to try.

Return Value

bool

expandtabs(*S*, *tabsize*=...)

Return a copy of *S* where all tab characters are expanded using spaces. If tabsize is not given, a tab size of 8 characters is assumed.

Return Value

string

find(*S*, *sub*, *start*=... , *end*=...)

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *s*[start:end]. Optional arguments start and end are interpreted as in slice notation. Return -1 on failure.

Return Value

int

index(*S*, *sub*, *start*=... , *end*=...)

Like *S*.find() but raise ValueError when the substring is not found.

Return Value

int

isalnum(*S*)

Return True if all characters in *S* are alphanumeric and there is at least one character in *S*, False otherwise.

Return Value

bool

isalpha(*S*)

Return True if all characters in *S* are alphabetic and there is at least one character in *S*, False otherwise.

Return Value

bool

isdigit(*S*)

Return True if all characters in *S* are digits and there is at least one character in *S*, False otherwise.

Return Value

bool

islower(*S*)

Return True if all cased characters in *S* are lowercase and there is at least one cased character in *S*, False otherwise.

Return Value

bool

isspace(*S*)

Return True if all characters in *S* are whitespace and there is at least one character in *S*, False otherwise.

Return Value

bool

istitle(*S*)

Return True if *S* is a titlecased string and there is at least one character in *S*, i.e. uppercase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

Return Value

bool

isupper(*S*)

Return True if all cased characters in *S* are uppercase and there is at least one cased character in *S*, False otherwise.

Return Value

bool

join(*S*, *sequence*)

Return a string which is the concatenation of the strings in the sequence. The separator between elements is *S*.

Return Value

string

ljust(*S*, *width*, *fillchar*=...)

Return *S* left justified in a string of length *width*. Padding is done using the specified fill character (default is a space).

Return Value

string

lower(*S*)

Return a copy of the string *S* converted to lowercase.

Return Value

string

rstrip(*S*, *chars*=...)

Return a copy of the string *S* with leading whitespace removed. If *chars* is given and not *None*, remove characters in *chars* instead. If *chars* is unicode, *S* will be converted to unicode before stripping

Return Value

string or unicode

partition(*S*, *sep*)

Searches for the separator *sep* in *S*, and returns the part before it, the separator itself, and the part after it. If the separator is not found, returns *S* and two empty strings.

Return Value

(head, sep, tail)

replace(...)

S.replace(*old*, *new*[, *count*]) -> string

Return a copy of string *S* with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind(*S*, *sub*, *start*=... , *end*=...)

Return the highest index in *S* where substring *sub* is found, such that *sub* is contained within *s*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation. Return -1 on failure.

Return Value

int

rindex(*S*, *sub*, *start*=... , *end*=...)

Like *S*.rfind() but raise *ValueError* when the substring is not found.

Return Value

int

rjust(*S*, *width*, *fillchar*=...)

Return *S* right justified in a string of length *width*. Padding is done using the specified fill character (default is a space)

Return Value

string

rpartition(*S*, *sep*)

Searches for the separator *sep* in *S*, starting at the end of *S*, and returns the part before it, the separator itself, and the part after it. If the separator is not found, returns two empty strings and *S*.

Return Value

(tail, sep, head)

rsplit(*S*, *sep*=... , *maxsplit*=...)

Return a list of the words in the string *S*, using *sep* as the delimiter string, starting at the end of the string and working to the front. If *maxsplit* is given, at most *maxsplit* splits are done. If *sep* is not specified or is None, any whitespace string is a separator.

Return Value

list of strings

rstrip(*S*, *chars*=...)

Return a copy of the string *S* with trailing whitespace removed. If *chars* is given and not None, remove characters in *chars* instead. If *chars* is unicode, *S* will be converted to unicode before stripping

Return Value

string or unicode

split(*S*, *sep*=... , *maxsplit*=...)

Return a list of the words in the string *S*, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done. If *sep* is not specified or is None, any whitespace string is a separator.

Return Value

list of strings

splitlines(*S*, *keepends*=...)

Return a list of the lines in *S*, breaking at line boundaries. Line breaks are not included in the resulting list unless *keepends* is given and true.

Return Value

list of strings

startswith(*S*, *prefix*, *start*=..., *end*=...)

Return True if *S* starts with the specified prefix, False otherwise. With optional *start*, test *S* beginning at that position. With optional *end*, stop comparing *S* at that position. *prefix* can also be a tuple of strings to try.

Return Value

bool

strip(*S*, *chars*=...)

Return a copy of the string *S* with leading and trailing whitespace removed. If *chars* is given and not None, remove characters in *chars* instead. If *chars* is unicode, *S* will be converted to unicode before stripping

Return Value

string or unicode

swapcase(*S*)

Return a copy of the string *S* with uppercase characters converted to lowercase and vice versa.

Return Value

string

title(*S*)

Return a titlecased version of *S*, i.e. words start with uppercase characters, all remaining cased characters have lowercase.

Return Value

string

translate(*S*, *table*, *deletechars*=...)

Return a copy of the string *S*, where all characters occurring in the optional argument *deletechars* are removed, and the remaining characters have been mapped through the given translation table, which must be a string of length 256.

Return Value

string

upper(*S*)

Return a copy of the string *S* converted to uppercase.

Return Value

string

zfill(*S*, *width*)

Pad a numeric string *S* with zeros on the left, to fill a field of the specified width. The string *S* is never truncated.

Return Value

string

35.6.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>

35.7 Class 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

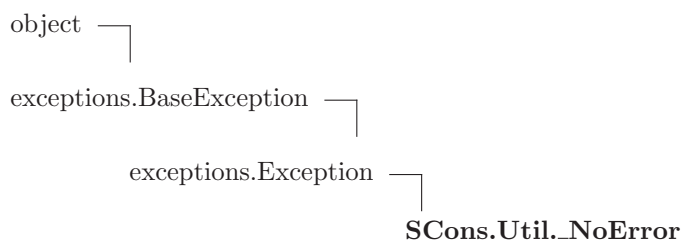
```
x = objA.var1
```

Inherit from this class to create a Proxy.

35.7.1 Methods

<code>__init__(self, subject)</code>
Wrap an object as a Proxy object
<code>__getattr__(self, name)</code>
Retrieve an attribute from the wrapped object. If the named attribute doesn't exist, <code>AttributeError</code> is raised
<code>get(self)</code>
Retrieve the entire wrapped object
<code>__cmp__(self, other)</code>

35.8 Class _NoError



35.8.1 Methods**`__delattr__(...)`**`x.__delattr__('name') <==> del x.name`Overrides: `object.__delattr__`**`__getattribute__(...)`**`x.__getattribute__('name') <==> x.name`Overrides: `object.__getattribute__`**`__getitem__(x, y)`**`x[y]`**`__getslice__(x, i, j)`**`x[i:j]`

Use of negative indices is not supported.

`__hash__(x)``hash(x)`**`__init__(...)`**`x.__init__(...)` initializes x; see `x.__class__.__doc__` for signatureOverrides: `exceptions.BaseException.__init__`**`__new__(T, S, ...)`****Return Value**a new object with type `S`, a subtype of `T`Overrides: `exceptions.BaseException.__new__`**`__reduce__(...)`**

helper for pickle

Overrides: `object.__reduce__` `exitit(inherited documentation)`**`__reduce_ex__(...)`**

helper for pickle

`__repr__(x)``repr(x)`Overrides: `object.__repr__`

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)

__str__(x)

str(x)

Overrides: object.__str__

35.8.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

35.9 Class CLVar

```
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.

35.9.1 Methods

__init__(self, seq=[])

Overrides: UserList.UserList.__init__

__coerce__(self, other)

__str__(self)

__add__(self, other)

__cmp__(self, other)

__contains__(self, item)

`__delitem__(self, i)``__delslice__(self, i, j)``__eq__(self, other)``__ge__(self, other)``__getitem__(self, i)``__getslice__(self, i, j)``__gt__(self, other)``__iadd__(self, other)``__imul__(self, n)``__le__(self, other)``__len__(self)``__lt__(self, other)``__mul__(self, n)``__ne__(self, other)``__radd__(self, other)``__repr__(self)``__rmul__(self, n)``__setitem__(self, i, item)``__setslice__(self, i, j, other)``append(self, item)``count(self, item)``extend(self, other)``index(self, item, *args)``insert(self, i, item)`

pop(*self*, *i*=-1)**remove**(*self*, *item*)**reverse**(*self*)**sort**(*self*, **args*, ***kws*)

35.10 Class OrderedDict

**Known Subclasses:** SCons.Util.Selector

35.10.1 Methods

__init__(*self*, *dict*=False)

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*=False)

Overrides: UserDict.UserDict.setdefault

update(*self*, *dict*)

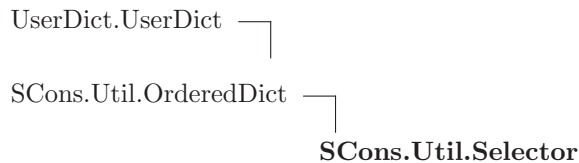
Overrides: UserDict.UserDict.update

values(*self*)

Overrides: UserDict.UserDict.values

`__cmp__(self, dict)``__contains__(self, key)``__getitem__(self, key)``__len__(self)``__repr__(self)``fromkeys(cls, iterable, value=False)``get(self, key, failobj=False)``has_key(self, key)``iteritems(self)``iterkeys(self)``itervalues(self)``pop(self, key, *args)`

35.11 Class 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.

35.11.1 Methods

`__call__(self, env, source)``__cmp__(self, dict)``__contains__(self, key)``__delitem__(self, key)`

Overrides: UserDict.UserDict.__delitem__

__getitem__(*self*, *key*)

__init__(*self*, *dict*=False)

Overrides: UserDict.UserDict.__init__

__len__(*self*)

__repr__(*self*)

__setitem__(*self*, *key*, *item*)

Overrides: UserDict.UserDict.__setitem__

clear(*self*)

Overrides: UserDict.UserDict.clear

copy(*self*)

Overrides: UserDict.UserDict.copy

fromkeys(*cls*, *iterable*, *value*=False)

get(*self*, *key*, *failobj*=False)

has_key(*self*, *key*)

items(*self*)

Overrides: UserDict.UserDict.items

iteritems(*self*)

iterkeys(*self*)

itervalues(*self*)

keys(*self*)

Overrides: UserDict.UserDict.keys

pop(*self*, *key*, **args*)

popitem(*self*)

Overrides: UserDict.UserDict.popitem

setdefault(*self*, *key*, *failobj*=False)

Overrides: UserDict.UserDict.setdefault

update(*self*, *dict*)

Overrides: UserDict.UserDict.update

values(*self*)

Overrides: UserDict.UserDict.values

35.12 Class LogicalLines

35.12.1 Methods

<code>__init__(self, fileobj)</code>

<code>readline(self)</code>

<code>readlines(self)</code>

35.13 Class UniqueList



35.13.1 Methods

<code>__init__(self, seq=[])</code> Overrides: UserList.UserList.__init__
--

<code>__lt__(self, other)</code> Overrides: UserList.UserList.__lt__

<code>__le__(self, other)</code> Overrides: UserList.UserList.__le__

<code>__eq__(self, other)</code> Overrides: UserList.UserList.__eq__

<code>__ne__(self, other)</code> Overrides: UserList.UserList.__ne__

<code>__gt__(self, other)</code> Overrides: UserList.UserList.__gt__

<code>__ge__(self, other)</code> Overrides: UserList.UserList.__ge__

<code>__cmp__(self, other)</code> Overrides: UserList.UserList.__cmp__

<code>__len__(self)</code> Overrides: UserList.UserList.__len__
--

<code>__getitem__(self, i)</code> Overrides: UserList.UserList.__getitem__

__setitem__(*self*, *i*, *item*)Overrides: `UserList.UserList.__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: `UserList.UserList.__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: `UserList.UserList.append`**insert**(*self*, *i*)Overrides: `UserList.UserList.insert`**count**(*self*, *item*)Overrides: `UserList.UserList.count`**index**(*self*, *item*)Overrides: `UserList.UserList.index`**reverse**(*self*)Overrides: `UserList.UserList.reverse`**sort**(*self*, **args*, ***kws*)Overrides: `UserList.UserList.sort`**extend**(*self*, *other*)Overrides: `UserList.UserList.extend`**__contains__**(*self*, *item*)**__delitem__**(*self*, *i*)

```
__delslice__(self, i, j)
```

```
__repr__(self)
```

```
pop(self, i=-1)
```

```
remove(self, item)
```

35.14 Class Unbuffered

A proxy class that wraps a file object, flushing after every write, and delegating everything else to the wrapped object.

35.14.1 Methods

```
__init__(self, file)
```

```
write(self, arg)
```

```
__getattr__(self, attr)
```

35.15 Class Null

Known Subclasses: SCons.Sig.MD5Null, SCons.Sig.TimeStampNull

Null objects always and reliably “do nothing.”

35.15.1 Methods

```
__new__(cls, *args, **kwargs)
```

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

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

```
__repr__(self)
```

```
__nonzero__(self)
```

```
__getattr__(self, mname)
```

```
__setattr__(self, name, value)
```

```
__delattr__(self, name)
```

36 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.

36.1 Modules

- **BoolVariable'**: engine.SCons.Variables.BoolVariable
This file defines the option type for SCons implementing true/false values.
(Section 37, p. 313)
- **EnumVariable'**: engine.SCons.Variables.EnumVariable
This file defines the option type for SCons allowing only specified input-values.
(Section 38, p. 314)
- **ListVariable'**: engine.SCons.Variables.ListVariable
This file defines the option type for SCons implementing 'lists'.
(Section 39, p. 315)
- **PackageVariable'**: engine.SCons.Variables.PackageVariable
This file defines the option type for SCons implementing 'package activation'.
(Section 40, p. 316)
- **PathVariable'**: SCons.Variables.PathVariable
This file defines an option type for SCons implementing path settings.
(Section 41, p. 317)

36.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Variables/__init__.py 3266 2008/08/12 0...

36.3 Class Variables

36.3.1 Methods

`__init__(self, files=[], args={}, is_global=False)`

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

`keys(self)`

Returns the keywords for the options

Add(*self*, *key*, *help*='', *default*=False, *validator*=False, *converter*=False, ***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 option's value
 Called with (*key*, *value*, *environment*)

converter - optional function that is called to convert the option'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*=False)

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*=False)

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*=[])

36.3.2 Class Variables

Name	Description
instance	Holds all the options, updates the environment with the variables, and renders the help text. Value: <code>False</code>
format	Value: <code>'\n%s: %s\n default: %s\n actual: %s\n'</code>
format_	Value: <code>'\n%s: %s\n default: %s\n actual: %s\n aliases: ...'</code>

37 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:
    ...
```

37.1 Functions

<code>BoolVariable</code> <i>(key, help, default)</i>
The input parameters describe a boolean option, thus they are returned with the correct converter and validator appended. The 'help' text will be appended by '(yes no)' to show the valid values. The result is usable for input to <code>opts.Add()</code> .

38 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':
    ...
```

38.1 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 allowed 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).

39 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 elements 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(...)
```

39.1 Functions

ListVariable (<i>key, help, default, names, map={}</i>)
--

<p>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 <code>opts.Add()</code> .</p> <p>A 'package list' option may either be 'all', 'none' or a list of package names (separated by space).</p>

40 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 existence)
```

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 ...
```

40.1 Functions

<code>PackageVariable</code> <i>(key, help, default, searchfunc=False)</i>
<p>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 <code>opts.Add()</code> .</p> <p>A 'package list' option may either be 'all', 'none' or a list of package names (seperated by space).</p>

41 Module *SCons.Variables.PathVariable*

SCons.Variables.PathVariable

This file defines an option type for SCons implementing path settings.

To be used whenever 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 Options 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'))
```

41.1 Variables

Name	Description
PathVariable	Value: <SCons.Variables.PathVariable._PathVariableClass instance...

42 Module SCons.Warnings

SCons.Warnings

This file implements the warnings framework for SCons.

42.1 Functions

suppressWarningClass(*clazz*)

Suppresses all warnings that are of type *clazz* or derived from *clazz*.

enableWarningClass(*clazz*)

Suppresses all warnings that are of type *clazz* or derived from *clazz*.

warningAsException(*flag=False*)

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.

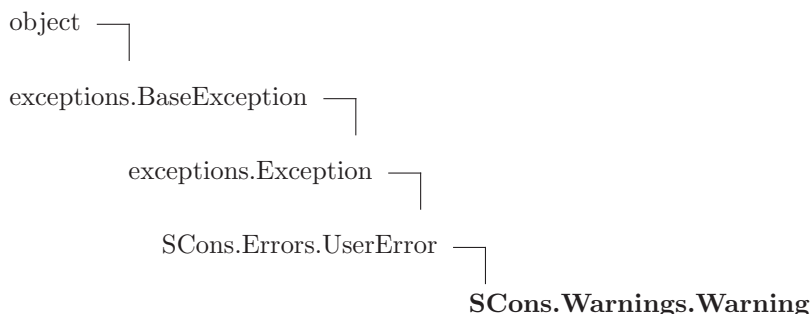
For example, 'deprecated' will enable the DeprecatedWarning class. 'no-dependency' will disable the .DependencyWarning class.

As a special case, --warn=all and --warn=no-all will enable or disable (respectively) the base Warning class of all warnings.

42.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Warnings.py 3266 2008/08/12 07:31:01 kn...'

42.3 Class Warning



Known Subclasses: SCons.SConf.SConfWarning, SCons.Warnings.CacheWriteErrorWarning, SCons.Warnings.CorruptSCacheWarning, SCons.Warnings.DependencyWarning, SCons.Warnings.DeprecatedWarning, SCons.Warnings.DuplicateEnvironmentWarning, SCons.Warnings.LinkWarning, SCons.Warnings.MisleadingKeywordsWarning, SCons.Warnings.MissingSConscriptWarning, SCons.Warnings.NoMD5ModuleWarning, SCons.Warnings.NoMetaclassSupportWarning, SCons.Warnings.NoObjectCountWarning, SCons.Warnings.NoParallelSupportWarning, SCons.Warnings.ReservedVariableWarning, SCons.Warnings.StackSizeWarning

42.3.1 Methods

`__delattr__`(...)

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name')` <==> `x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

`__init__`(...)

`x.__init__`(...) initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

__new__(*T, S, ...*)

Return Value

a new object with type *S*, a subtype of *T*

Overrides: `exceptions.BaseException.__new__`

__reduce__(...)

helper for pickle

Overrides: `object.__reduce__` `exitit`(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

`repr(x)`

Overrides: `object.__repr__`

__setattr__(...)

`x.__setattr__('name', value) <==> x.name = value`

Overrides: `object.__setattr__`

__setstate__(...)

__str__(*x*)

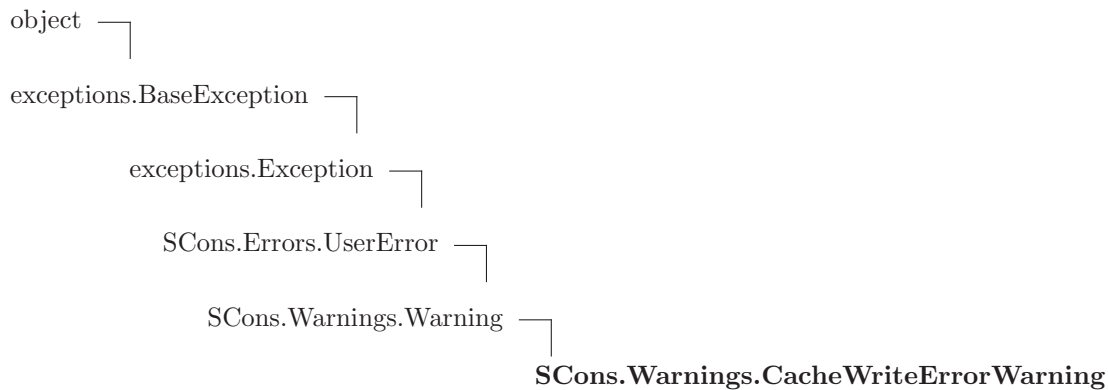
`str(x)`

Overrides: `object.__str__`

42.3.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.4 Class `CacheWriteErrorWarning`



42.4.1 Methods

`__delattr__`(...)

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name')` <==> `x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

`__init__`(...)

`x.__init__()` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

`__new__`(*T*, *S*, ...)

Return Value

a new object with type `S`, a subtype of `T`

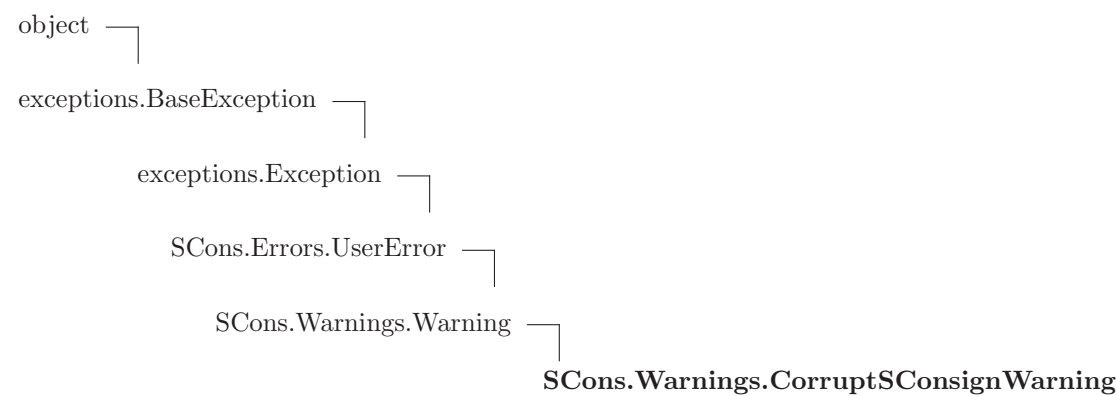
Overrides: `exceptions.BaseException.__new__`

<code>__reduce__(...)</code> helper for pickle Overrides: <code>object.__reduce__</code> <code>__init__(inherited documentation)</code>
<code>__reduce_ex__(...)</code> helper for pickle
<code>__repr__(x)</code> repr(x) Overrides: <code>object.__repr__</code>
<code>__setattr__(...)</code> x.__setattr__('name', value) <==> x.name = value Overrides: <code>object.__setattr__</code>
<code>__setstate__(...)</code>
<code>__str__(x)</code> str(x) Overrides: <code>object.__str__</code>

42.4.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.5 Class *CorruptSConsignWarning*



42.5.1 Methods**__delattr__(...)**`x.__delattr__('name') <==> del x.name`Overrides: `object.__delattr__`**__getattribute__(...)**`x.__getattribute__('name') <==> x.name`Overrides: `object.__getattribute__`**__getitem__(*x*, *y*)**`x[y]`**__getslice__(*x*, *i*, *j*)**`x[i:j]`

Use of negative indices is not supported.

__hash__(*x*)`hash(x)`**__init__(...)**`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signatureOverrides: `exceptions.BaseException.__init__`**__new__(*T*, *S*, ...)****Return Value**a new object with type `S`, a subtype of `T`Overrides: `exceptions.BaseException.__new__`**__reduce__(...)**

helper for pickle

Overrides: `object.__reduce__` `exitit(inherited documentation)`**__reduce_ex__(...)**

helper for pickle

__repr__(*x*)`repr(x)`Overrides: `object.__repr__`

<code>__setattr__(...)</code>
<code>x.__setattr__('name', value) <==> x.name = value</code>
Overrides: <code>object.__setattr__</code>

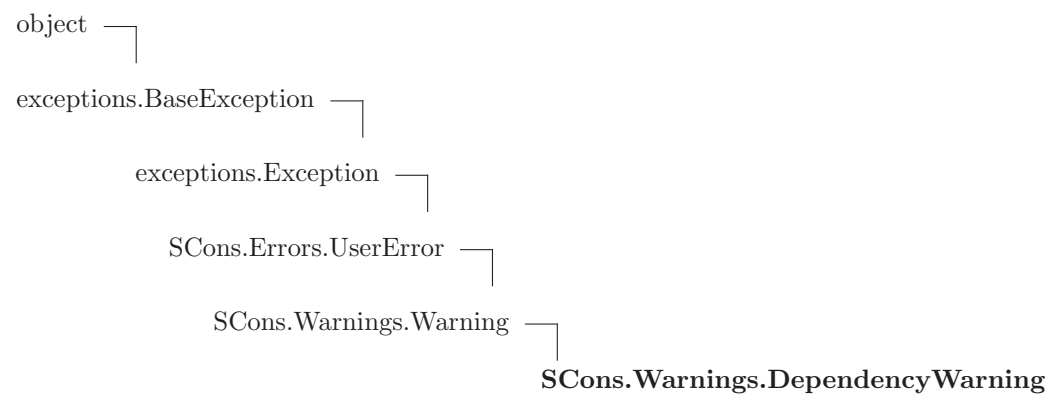
<code>__setstate__(...)</code>

<code>__str__(x)</code>
<code>str(x)</code>
Overrides: <code>object.__str__</code>

42.5.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.6 Class DependencyWarning



42.6.1 Methods

<code>__delattr__(...)</code>
<code>x.__delattr__('name') <==> del x.name</code>
Overrides: <code>object.__delattr__</code>

<code>__getattr__(...)</code>
<code>x.__getattr__('name') <==> x.name</code>
Overrides: <code>object.__getattr__</code>

__getitem__(*x*, *y*)

x[y]

__getslice__(*x*, *i*, *j*)

x[i:j]

Use of negative indices is not supported.

__hash__(*x*)

hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(*T*, *S*, ...)

Return Valuea new object with type *S*, a subtype of *T*

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)

__str__(*x*)

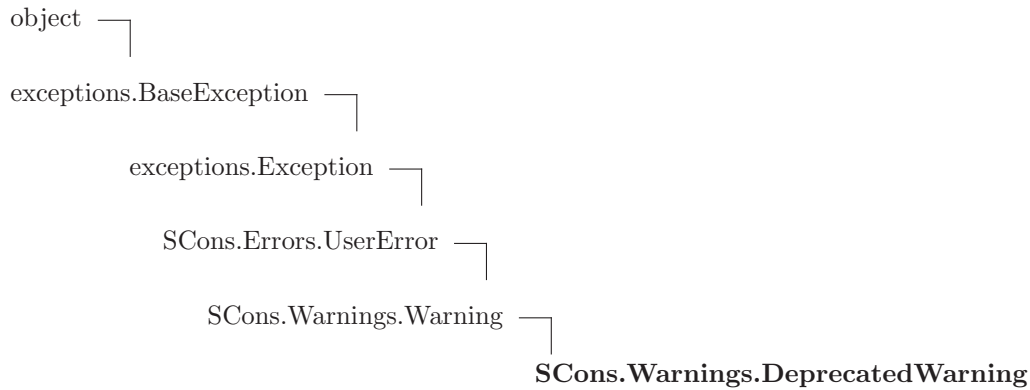
str(x)

Overrides: object.__str__

42.6.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.7 Class *DeprecatedWarning*



Known Subclasses: *SCons.Warnings.DeprecatedCopyWarning*, *SCons.Warnings.DeprecatedSourceSignaturesWarning*, *SCons.Warnings.DeprecatedTargetSignaturesWarning*, *SCons.Warnings.PythonVersionWarning*

42.7.1 Methods

`__delattr__`(...)

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name')` <==> `x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(T, S, ...)

Return Value

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)

__str__(x)

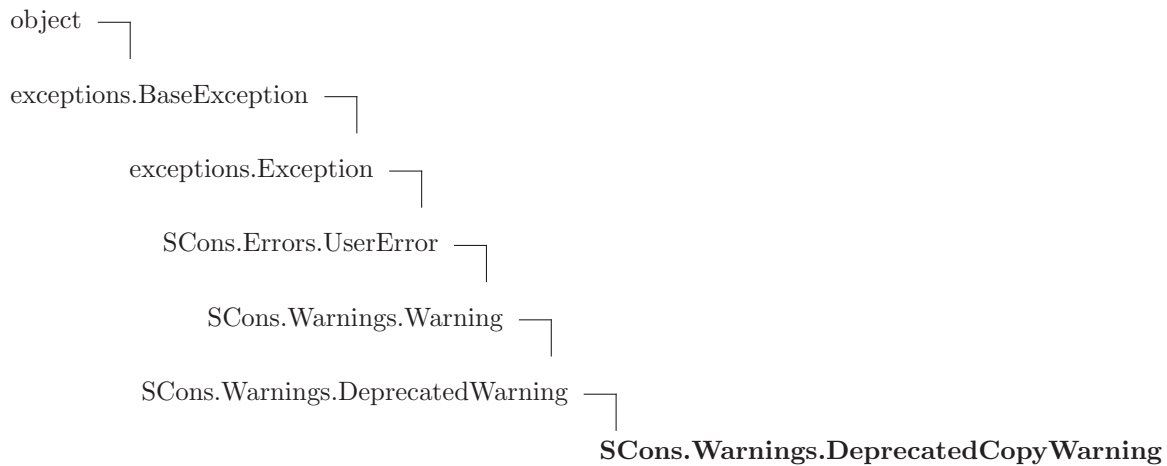
str(x)

Overrides: object.__str__

42.7.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.8 Class `DeprecatedCopyWarning`



42.8.1 Methods

`__delattr__(...)`

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

`__getattr__(...)`

`x.__getattr__('name')` <==> `x.name`

Overrides: `object.__getattr__`

`__getitem__(x, y)`

`x[y]`

`__getslice__(x, i, j)`

`x[i:j]`

Use of negative indices is not supported.

`__hash__(x)`

`hash(x)`

`__init__(...)`

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

__new__(*T, S, ...*)

Return Value

a new object with type *S*, a subtype of *T*

Overrides: `exceptions.BaseException.__new__`

__reduce__(...)

helper for pickle

Overrides: `object.__reduce__` `exitit`(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

`repr(x)`

Overrides: `object.__repr__`

__setattr__(...)

`x.__setattr__('name', value) <==> x.name = value`

Overrides: `object.__setattr__`

__setstate__(...)

__str__(*x*)

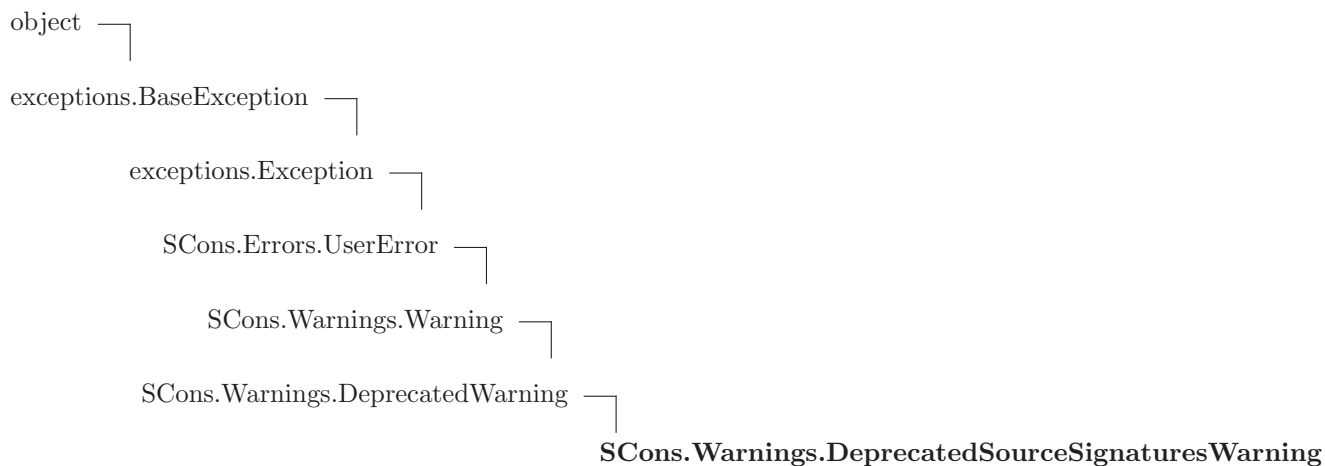
`str(x)`

Overrides: `object.__str__`

42.8.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.9 Class `DeprecatedSourceSignaturesWarning`



42.9.1 Methods

`__delattr__`(...)

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name')` <==> `x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

`__init__`(...)

`x.__init__`(...) initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

__new__(*T, S, ...*)

Return Value

a new object with type *S*, a subtype of *T*

Overrides: `exceptions.BaseException.__new__`

__reduce__(...)

helper for pickle

Overrides: `object.__reduce__` `exitit`(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

`repr(x)`

Overrides: `object.__repr__`

__setattr__(...)

`x.__setattr__('name', value) <==> x.name = value`

Overrides: `object.__setattr__`

__setstate__(...)

__str__(*x*)

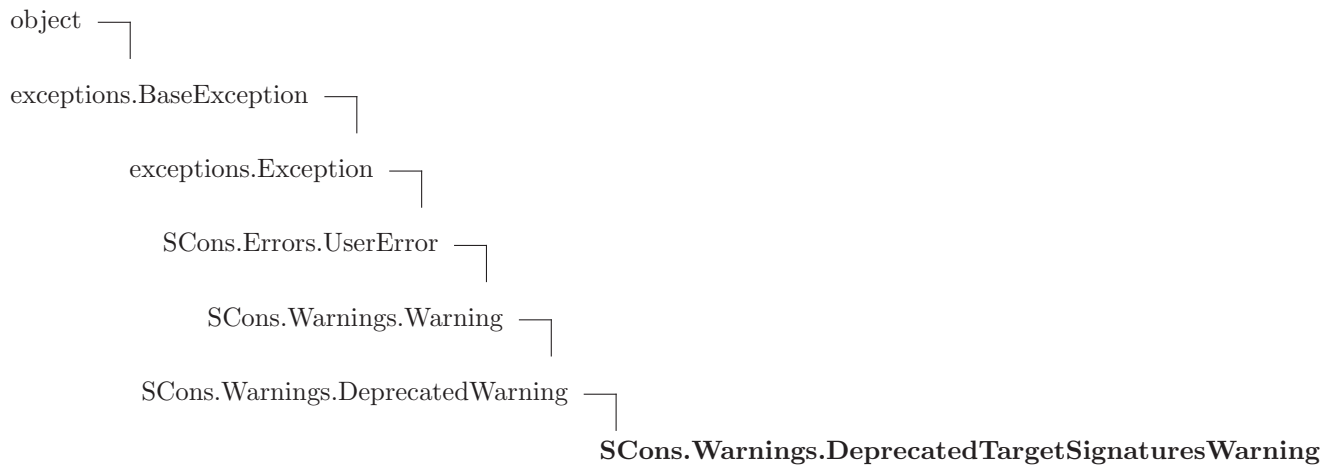
`str(x)`

Overrides: `object.__str__`

42.9.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.10 Class `DeprecatedTargetSignaturesWarning`



42.10.1 Methods

`__delattr__`(...)

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name')` <==> `x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

`__init__`(...)

`x.__init__`(...) initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

__new__(*T, S, ...*)**Return Value**a new object with type *S*, a subtype of *T*Overrides: `exceptions.BaseException.__new__`**__reduce__**(...)

helper for pickle

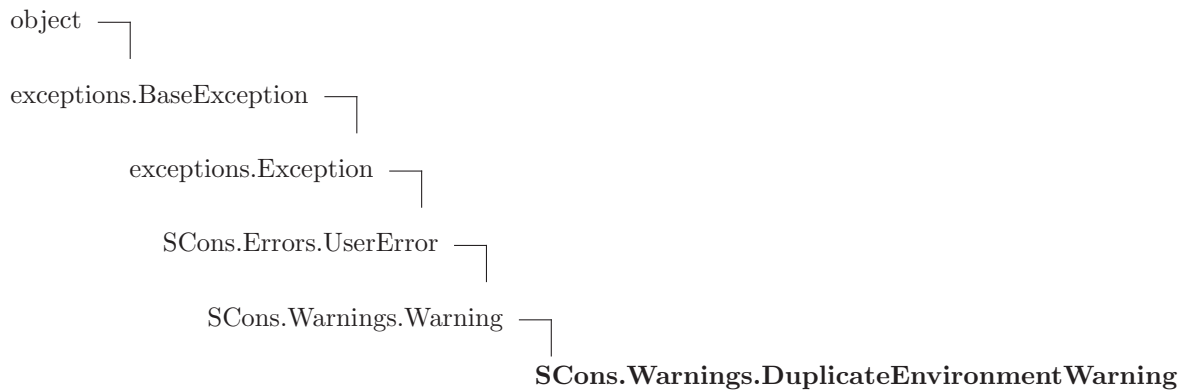
Overrides: `object.__reduce__` `exitit`(inherited documentation)**__reduce_ex__**(...)

helper for pickle

__repr__(*x*)`repr(x)`Overrides: `object.__repr__`**__setattr__**(...)`x.__setattr__('name', value) <==> x.name = value`Overrides: `object.__setattr__`**__setstate__**(...)**__str__**(*x*)`str(x)`Overrides: `object.__str__`**42.10.2 Properties**

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.11 Class DuplicateEnvironmentWarning



42.11.1 Methods

`__delattr__(...)`

`x.__delattr__('name') <==> del x.name`

Overrides: `object.__delattr__`

`__getattr__(...)`

`x.__getattr__('name') <==> x.name`

Overrides: `object.__getattr__`

`__getitem__(x, y)`

`x[y]`

`__getslice__(x, i, j)`

`x[i:j]`

Use of negative indices is not supported.

`__hash__(x)`

`hash(x)`

`__init__(...)`

`x.__init__(...)` initializes x; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

`__new__(T, S, ...)`

Return Value

a new object with type `S`, a subtype of `T`

Overrides: `exceptions.BaseException.__new__`

`__reduce__(...)`
helper for pickle
Overrides: object.__reduce__ extit(inherited documentation)

`__reduce_ex__(...)`
helper for pickle

`__repr__(x)`
repr(x)
Overrides: object.__repr__

`__setattr__(...)`
x.__setattr__('name', value) <==> x.name = value
Overrides: object.__setattr__

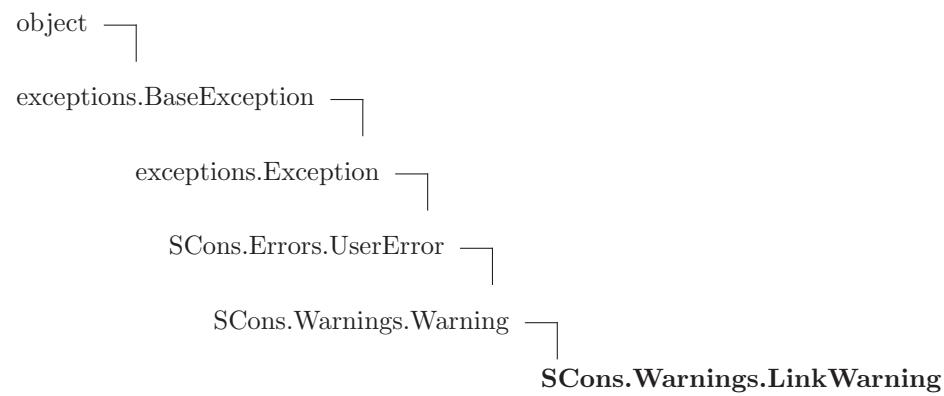
`__setstate__(...)`

`__str__(x)`
str(x)
Overrides: object.__str__

42.11.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.12 Class LinkWarning



Known Subclasses: `SCons.Warnings.FortranCxxMixWarning`

42.12.1 Methods

`__delattr__`(...)

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name')` <==> `x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

`__init__`(...)

`x.__init__`(...) initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

`__new__`(*T*, *S*, ...)

Return Value

a new object with type `S`, a subtype of `T`

Overrides: `exceptions.BaseException.__new__`

`__reduce__`(...)

helper for pickle

Overrides: `object.__reduce__` `exitit`(inherited documentation)

`__reduce_ex__`(...)

helper for pickle

`__repr__`(*x*)

`repr(x)`

Overrides: `object.__repr__`

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

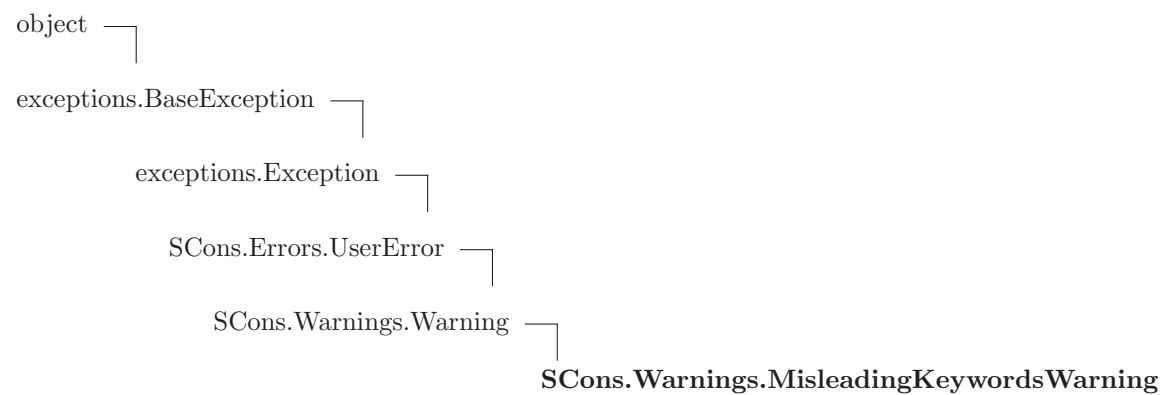
__setstate__(...)

__str__(x)
str(x)
Overrides: object.__str__

42.12.2 Properties

Name	Description
__class__	Value: <attribute '__class__' of 'object' objects>
args	Value: <attribute 'args' of 'exceptions.BaseException' objects>
message	Value: <member 'message' of 'exceptions.BaseException' objects>

42.13 Class MisleadingKeywordsWarning



42.13.1 Methods

__delattr__(...)
x.__delattr__('name') <==> del x.name
Overrides: object.__delattr__

__getattr__(...)
x.__getattr__('name') <==> x.name
Overrides: object.__getattr__

__getitem__(*x*, *y*)

x[y]

__getslice__(*x*, *i*, *j*)

x[i:j]

Use of negative indices is not supported.

__hash__(*x*)

hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(*T*, *S*, ...)**Return Value**a new object with type *S*, a subtype of *T*

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)

__str__(*x*)

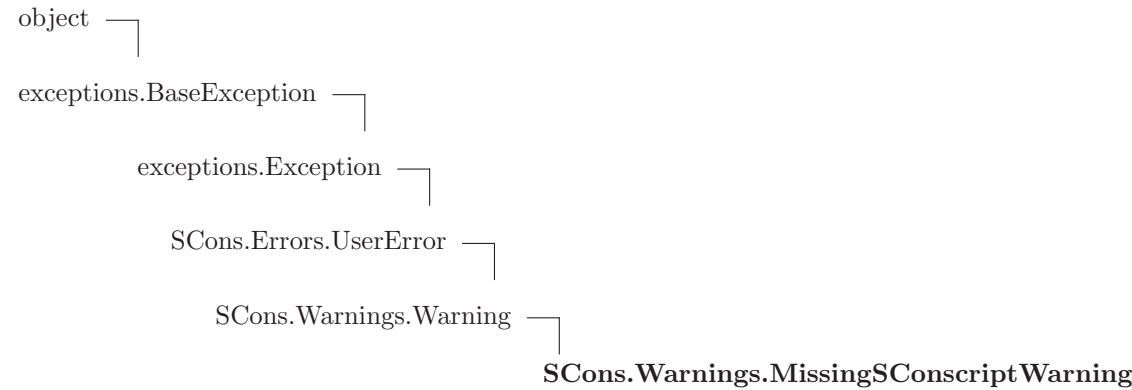
str(x)

Overrides: object.__str__

42.13.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.14 Class MissingSConscriptWarning



42.14.1 Methods

<code>__delattr__</code> (...) x. <code>__delattr__</code> ('name') <==> del x.name Overrides: object. <code>__delattr__</code>
<code>__getattr__</code> (...) x. <code>__getattr__</code> ('name') <==> x.name Overrides: object. <code>__getattr__</code>
<code>__getitem__</code> (x, y) x[y]
<code>__getslice__</code> (x, i, j) x[i:j] Use of negative indices is not supported.
<code>__hash__</code> (x) hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(T, S, ...)**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)**__str__**(x)

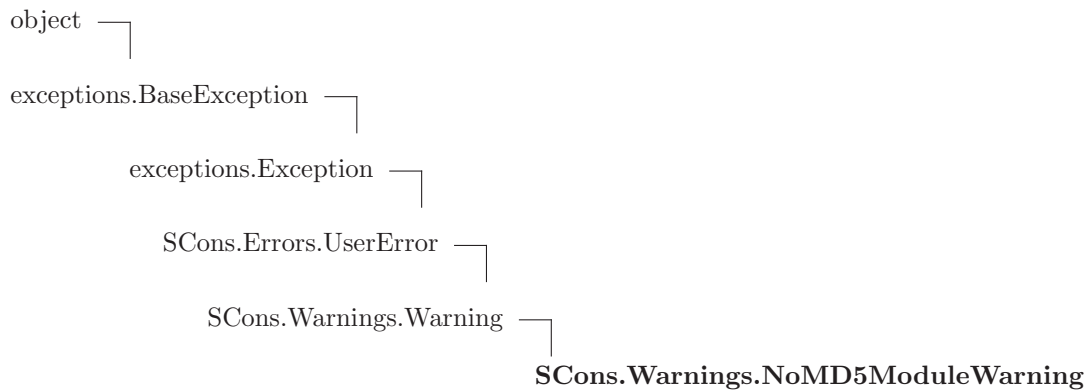
str(x)

Overrides: object.__str__

42.14.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.15 Class NoMD5ModuleWarning



42.15.1 Methods

`__delattr__(...)`

`x.__delattr__('name') <==> del x.name`

Overrides: `object.__delattr__`

`__getattr__(...)`

`x.__getattr__('name') <==> x.name`

Overrides: `object.__getattr__`

`__getitem__(x, y)`

`x[y]`

`__getslice__(x, i, j)`

`x[i:j]`

Use of negative indices is not supported.

`__hash__(x)`

`hash(x)`

`__init__(...)`

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

`__new__(T, S, ...)`

Return Value

a new object with type `S`, a subtype of `T`

Overrides: `exceptions.BaseException.__new__`

`__reduce__(...)`
helper for pickle
Overrides: object.__reduce__ extit(inherited documentation)

`__reduce_ex__(...)`

helper for pickle

`__repr__(x)`

repr(x)
Overrides: object.__repr__

`__setattr__(...)`

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

`__setstate__(...)`

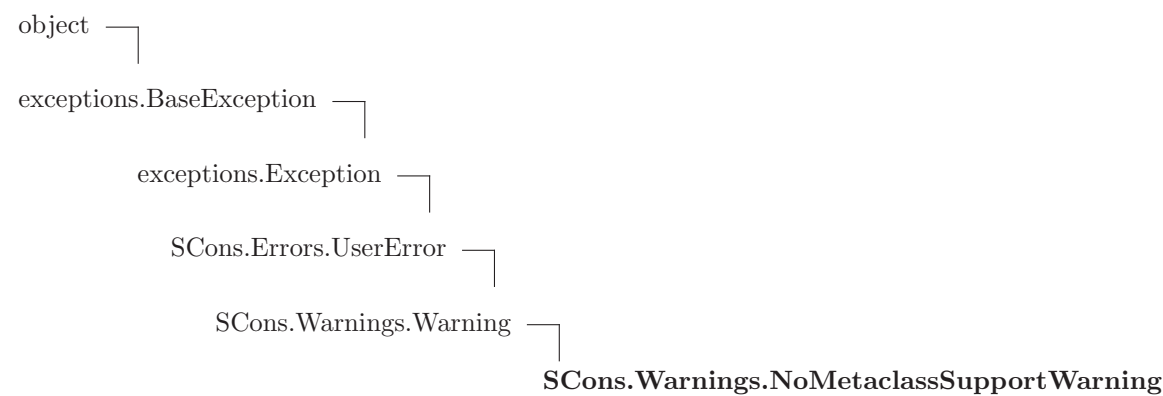
`__str__(x)`

str(x)
Overrides: object.__str__

42.15.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.16 Class NoMetaclassSupportWarning



42.16.1 Methods

__delattr__(...)

x.__delattr__('name') <==> del x.name

Overrides: object.__delattr__

__getattribute__(...)

x.__getattribute__('name') <==> x.name

Overrides: object.__getattribute__

__getitem__(x, y)

x[y]

__getslice__(x, i, j)

x[i:j]

Use of negative indices is not supported.

__hash__(x)

hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(T, S, ...)**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

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

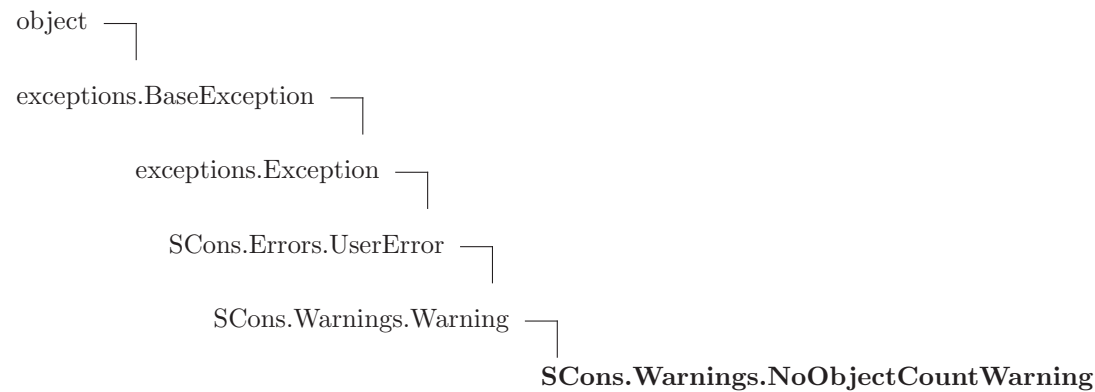
__setstate__(...)

__str__(x)
str(x)
Overrides: object.__str__

42.16.2 Properties

Name	Description
__class__	Value: <attribute '__class__' of 'object' objects>
args	Value: <attribute 'args' of 'exceptions.BaseException' objects>
message	Value: <member 'message' of 'exceptions.BaseException' objects>

42.17 Class NoObjectCountWarning



42.17.1 Methods

__delattr__(...)
x.__delattr__('name') <==> del x.name
Overrides: object.__delattr__

__getattr__(...)
x.__getattr__('name') <==> x.name
Overrides: object.__getattr__

`__getitem__`(*x*, *y*)

x[y]

`__getslice__`(*x*, *i*, *j*)

x[i:j]

Use of negative indices is not supported.

`__hash__`(*x*)

hash(x)

`__init__`(...)

x.`__init__`(...) initializes x; see x.`__class__.__doc__` for signatureOverrides: `exceptions.BaseException.__init__`

`__new__`(*T*, *S*, ...)**Return Value**a new object with type *S*, a subtype of *T*Overrides: `exceptions.BaseException.__new__`

`__reduce__`(...)

helper for pickle

Overrides: `object.__reduce__` `exitit`(inherited documentation)

`__reduce_ex__`(...)

helper for pickle

`__repr__`(*x*)

repr(x)Overrides: `object.__repr__`

`__setattr__`(...)

x.`__setattr__`('name', value) <==> x.name = valueOverrides: `object.__setattr__`

`__setstate__`(...)

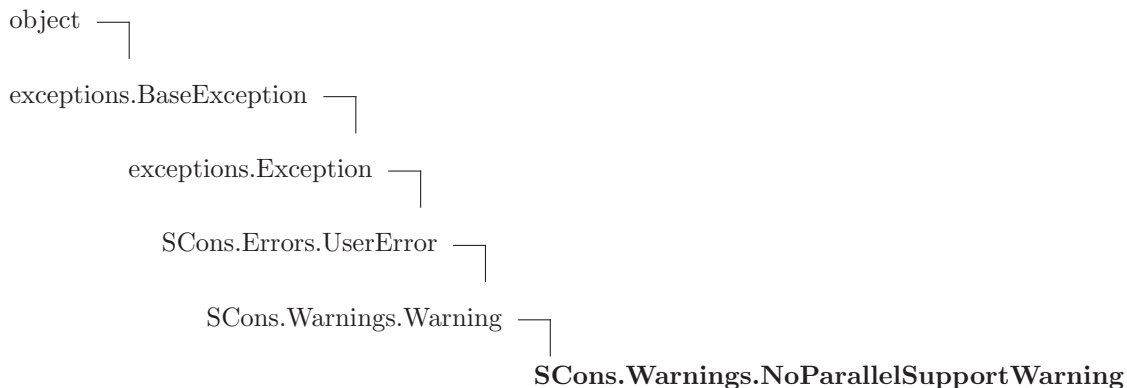
`__str__`(*x*)

str(x)Overrides: `object.__str__`

42.17.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.18 Class *NoParallelSupportWarning*



42.18.1 Methods

`__delattr__`(...)

`x.__delattr__('name')` <==> `del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name')` <==> `x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(T, S, ...)**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)**__str__**(x)

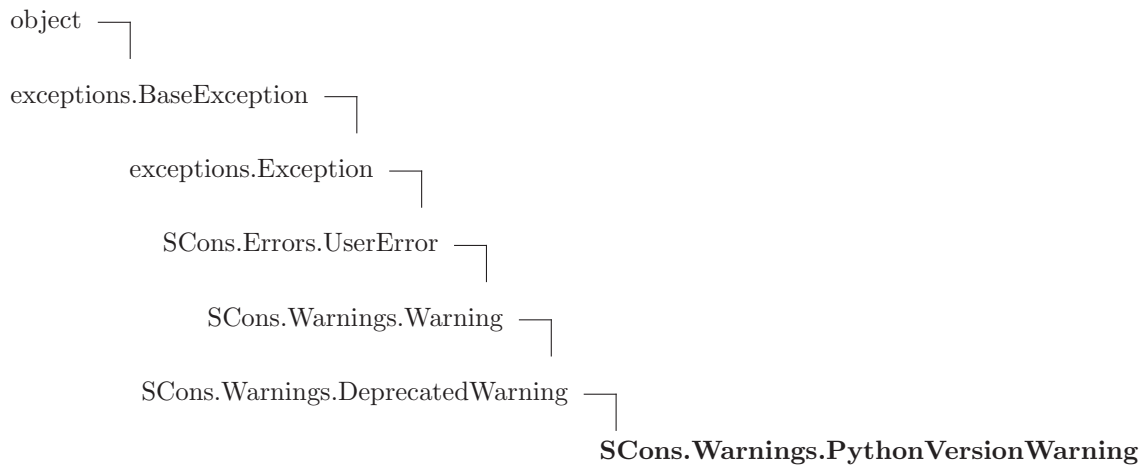
str(x)

Overrides: object.__str__

42.18.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.19 Class `PythonVersionWarning`



42.19.1 Methods

`__delattr__``(...)``x.__delattr__('name') <==> del x.name`Overrides: `object.__delattr__`**`__getattr__`**`(...)``x.__getattr__('name') <==> x.name`Overrides: `object.__getattr__`**`__getitem__`**`(x, y)``x[y]`**`__getslice__`**`(x, i, j)``x[i:j]`

Use of negative indices is not supported.

`__hash__``(x)``hash(x)`**`__init__`**`(...)``x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signatureOverrides: `exceptions.BaseException.__init__`

__new__(*T, S, ...*)

Return Value

a new object with type *S*, a subtype of *T*

Overrides: `exceptions.BaseException.__new__`

__reduce__(...)

helper for pickle

Overrides: `object.__reduce__` `exitit`(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

`repr(x)`

Overrides: `object.__repr__`

__setattr__(...)

`x.__setattr__('name', value) <==> x.name = value`

Overrides: `object.__setattr__`

__setstate__(...)

__str__(*x*)

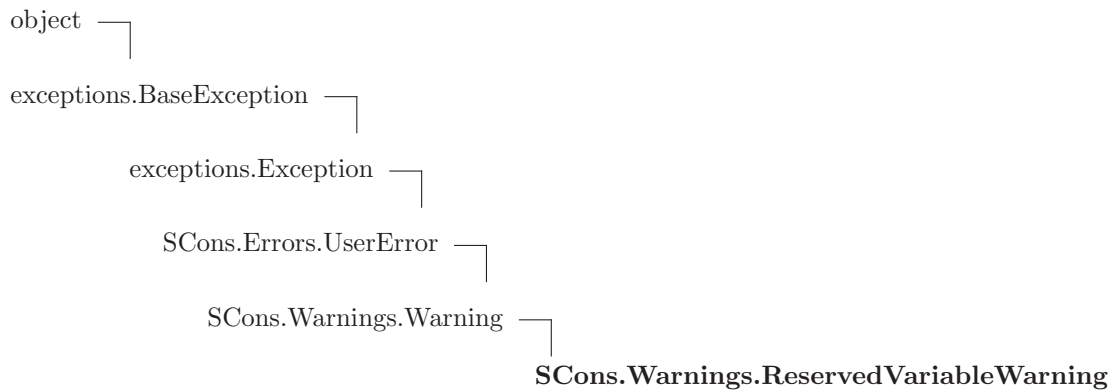
`str(x)`

Overrides: `object.__str__`

42.19.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

42.20 Class `ReservedVariableWarning`



42.20.1 Methods

`__delattr__`(...)

`x.__delattr__('name') <==> del x.name`

Overrides: `object.__delattr__`

`__getattr__`(...)

`x.__getattr__('name') <==> x.name`

Overrides: `object.__getattr__`

`__getitem__`(*x*, *y*)

`x[y]`

`__getslice__`(*x*, *i*, *j*)

`x[i:j]`

Use of negative indices is not supported.

`__hash__`(*x*)

`hash(x)`

`__init__`(...)

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature

Overrides: `exceptions.BaseException.__init__`

`__new__`(*T*, *S*, ...)

Return Value

a new object with type `S`, a subtype of `T`

Overrides: `exceptions.BaseException.__new__`

__reduce__(...)
helper for pickle
Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex(...)
helper for pickle

__repr__(*x*)
repr(*x*)
Overrides: object.__repr__

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

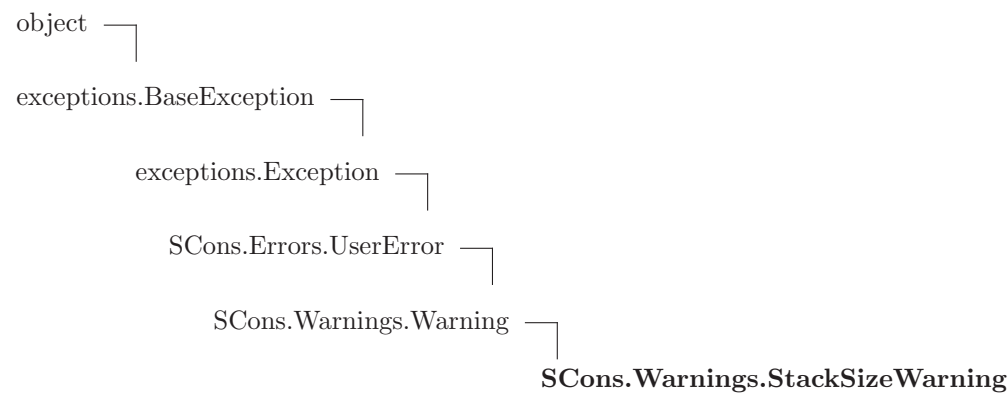
__setstate(...)

__str__(*x*)
str(*x*)
Overrides: object.__str__

42.20.2 Properties

Name	Description
__class__	Value: <attribute '__class__' of 'object' objects>
args	Value: <attribute 'args' of 'exceptions.BaseException' objects>
message	Value: <member 'message' of 'exceptions.BaseException' objects>

42.21 Class StackSizeWarning



42.21.1 Methods

__delattr__(...)

x.__delattr__('name') <==> del x.name

Overrides: object.__delattr__

__getattribute__(...)

x.__getattribute__('name') <==> x.name

Overrides: object.__getattribute__

__getitem__(x, y)

x[y]

__getslice__(x, i, j)

x[i:j]

Use of negative indices is not supported.

__hash__(x)

hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(T, S, ...)**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

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

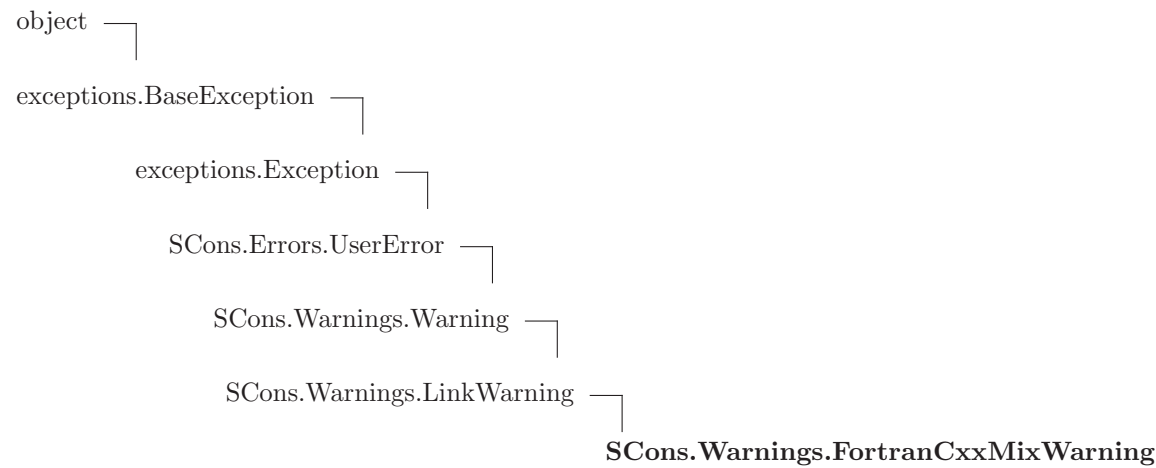
__setstate__(...)

__str__(x)
str(x)
Overrides: object.__str__

42.21.2 Properties

Name	Description
__class__	Value: <attribute '.__class__' of 'object' objects>
args	Value: <attribute 'args' of 'exceptions.BaseException' objects>
message	Value: <member 'message' of 'exceptions.BaseException' objects>

42.22 Class FortranCxxMixWarning



42.22.1 Methods

__delattr__(...)
x.__delattr__('name') <==> del x.name
Overrides: object.__delattr__

__getattr__(...)

x.__getattr__('name') <==> x.name

Overrides: object.__getattr__

__getitem__(x, y)

x[y]

__getslice__(x, i, j)

x[i:j]

Use of negative indices is not supported.

__hash__(x)

hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__

__new__(T, S, ...)

Return Value

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)

<code>__str__(x)</code>
<code>str(x)</code>
Overrides: <code>object.__str__</code>

42.22.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

43 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.

This package will be imported by other code:

```
import SCons.compat
```

But other code will not generally reference things in this package through the SCons.compat namespace. The modules included here add things to the `__builtin__` 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 `__builtin__` name space come from our 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.

43.1 Modules

- **`_scons_UserString`**: A user-defined wrapper around string objects
(Section 44, p. 359)
- **`_scons_hashlib`**: hashlib backwards-compatibility module for older (pre-2.5) Python versions
(Section 45, p. 361)
- **`_scons_itertools`**: Implementations of itertools functions for Python versions that don't have iterators.
(Section 46, p. 363)
- **`_scons_optparse`**: optparse - a powerful, extensible, and easy-to-use option parser.
(Section 47, p. 365)
- **`_scons_sets`**: Classes to represent arbitrary sets (including sets of sets).
(Section 48, p. 386)
- **`_scons_sets15`** (Section 49, p. 398)
- **`_scons_shlex`**: A lexical analyzer class for simple shell-like syntaxes.
(Section 50, p. 400)
- **`_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.
(Section 51, p. 401)
- **`_scons_textwrap`**: Text wrapping and filling.
(Section 52, p. 412)

- **builtins:** Compatibility idioms for `__builtin__` names
(Section 53, p. 414)

43.2 Functions

import_as(*module*, *name*)

Imports the specified module (from our local directory) as the specified name.

43.3 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>__revision__</code>	Value: 'src/engine/SCons/compat/__init__.py 3266 2008/08/12 07:3...
<code>version_string</code>	Value: <code>string.split(sys.version)</code> [0]
<code>version_ints</code>	Value: <code>map(int, string.split(version_string, '.'))</code>

44 Module *SCons.compat._scons_UserString*

A user-defined wrapper around string objects

This class is “borrowed” from the Python 2.2 *UserString* and modified slightly for use with *SCons*. It is *NOT* guaranteed to be fully compliant with the standard *UserString* class from all later versions of Python. In particular, it does not necessarily contain all of the methods found in later versions.

44.1 Functions

is_String (<i>obj</i>)

44.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/compat/_scons_UserString.py 3266 2008/0...
<code>__doc__</code>	Value: ...

44.3 Class *UserString*

44.3.1 Methods

__init__ (<i>self</i> , <i>seq</i>)
--

__str__ (<i>self</i>)

__repr__ (<i>self</i>)

__int__ (<i>self</i>)

__long__ (<i>self</i>)

__float__ (<i>self</i>)

__complex__ (<i>self</i>)

__hash__ (<i>self</i>)

__cmp__ (<i>self</i> , <i>string</i>)
--

__contains__ (<i>self</i> , <i>char</i>)

__len__ (<i>self</i>)

__getitem__ (<i>self</i> , <i>index</i>)

```
__getslice__(self, start, end)
```

```
__add__(self, other)
```

```
__radd__(self, other)
```

```
__mul__(self, n)
```

```
__rmul__(self, n)
```

45 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.

45.1 Functions

<code>md5(string='')</code>

45.2 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>__revision__</code>	Value: 'src/engine/SCons/compat/_scons_hashlib.py 3266 2008/08/1...

45.3 Class `md5obj`

45.3.1 Methods

<code>__init__(self, name, string='')</code>
--

<code>__repr__(self)</code>

<code>copy(self)</code>

<code>digest(self)</code>

<code>update(self, arg)</code>

<code>hexdigest(self)</code>

45.3.2 Class Variables

Name	Description
<code>md5_module</code>	Value: <module 'md5' from '/usr/lib/python2.5/md5.pyc'>

45.4 Class `md5obj`

45.4.1 Methods

<code>__init__(self, name, string='')</code>
--

<code>__repr__(self)</code>

<code>copy(self)</code>

<code>digest(self)</code>

<code>update(self, arg)</code>

<code>hexdigest(self)</code>

45.4.2 Class Variables

Name	Description
<code>md5_module</code>	Value: <module 'md5' from '/usr/lib/python2.5/md5.pyc'>

46 Module `SCons.compat._scons_itertools`

Implementations of `itertools` functions for Python versions that don't have iterators.

These implement the functions by creating the entire list, not returning it element-by-element as the real `itertools` functions do. This means that early Python versions won't get the performance benefit of using the `itertools`, but we can still use them so the later Python versions do get the advantages of using iterators.

Because we return the entire list, we intentionally do not implement the `itertools` functions that “return” infinitely-long lists: the `count()`, `cycle()` and `repeat()` functions. Other functions below have remained unimplemented simply because they aren't being used (yet) and it wasn't obvious how to do it. Or, conversely, we only implemented those functions that *were* easy to implement (mostly because the Python documentation contained examples of equivalent code).

Note that these do not have independent unit tests, so it's possible that there are bugs.

46.1 Functions

```
chain(*iterables)
```

```
count(n=0)
```

```
cycle(iterable)
```

```
dropwhile(predicate, iterable)
```

```
groupby(iterable, *args)
```

```
ifilter(predicate, iterable)
```

```
ifilterfalse(predicate, iterable)
```

```
imap(function, *iterables)
```

```
islice(*args, **kw)
```

```
izip(*iterables)
```

```
repeat(*args, **kw)
```

```
starmap(*args, **kw)
```

```
takewhile(predicate, iterable)
```

```
tee(*args, **kw)
```

46.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/compat/_scons_itertools.py 3266 2008/08...
<code>__doc__</code>	Value: ...

47 Module `SCons.compat._scons_optparse`

`optparse` - a powerful, extensible, and easy-to-use option parser.

By Greg Ward <gward@python.net>

Originally distributed as `Optik`; see <http://optik.sourceforge.net/> .

If you have problems with this module, please do not file bugs, patches, or feature requests with Python; instead, use `Optik`'s SourceForge project page:
<http://sourceforge.net/projects/optik>

For support, use the `optik-users@lists.sourceforge.net` mailing list (<http://lists.sourceforge.net/lists/listinfo/optik-users>).

Version: 1.5.3

Copyright: Copyright (c) 2001-2006 Gregory P. Ward. All rights reserved. Copyright (c) 2002-2006 Python Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

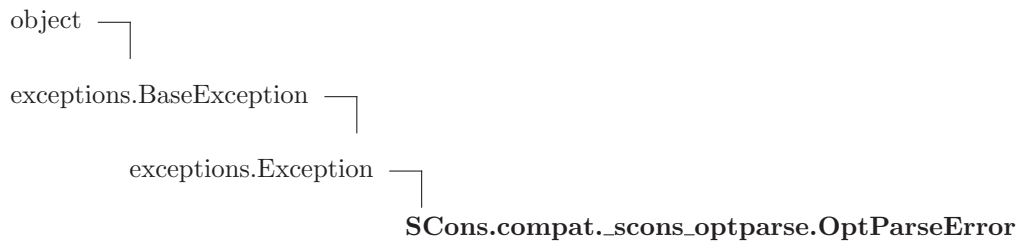
- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the author nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

47.1 Variables

Name	Description
<code>SUPPRESS_HELP</code>	Value: <code>'SUPPRESSHELP'</code>
<code>SUPPRESS_USAGE</code>	Value: <code>'SUPPRESSUSAGE'</code>

47.2 Class `OptParseError`



Known Subclasses: `SCons.compat._scons_optparse.BadOptionError`, `SCons.compat._scons_optparse.OptionError`, `SCons.compat._scons_optparse.OptionValueError`

47.2.1 Methods

`__init__(self, msg)`
`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature
 Overrides: `exceptions.Exception.__init__` `exitit`(inherited documentation)

`__str__(self)`
`str(x)`
 Overrides: `exceptions.BaseException.__str__` `exitit`(inherited documentation)

`__delattr__(...)`
`x.__delattr__('name') <==> del x.name`
 Overrides: `object.__delattr__`

`__getattr__(...)`
`x.__getattr__('name') <==> x.name`
 Overrides: `object.__getattr__`

`__getitem__(x, y)`
`x[y]`

`__getslice__(x, i, j)`
`x[i:j]`
 Use of negative indices is not supported.

`__hash__(x)`
`hash(x)`

`__new__(T, S, ...)`
Return Value
a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__

`__reduce__(...)`
helper for pickle
Overrides: object.__reduce__ extit(inherited documentation)

`__reduce_ex__(...)`
helper for pickle

`__repr__(x)`
repr(x)
Overrides: object.__repr__

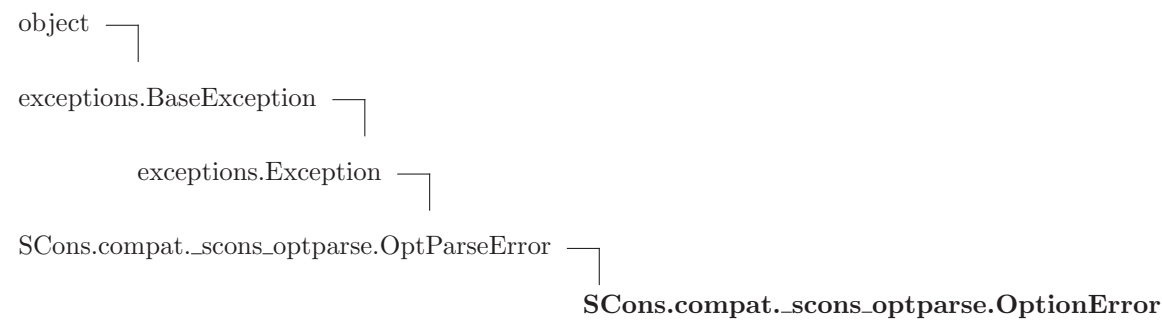
`__setattr__(...)`
x.__setattr__('name', value) <==> x.name = value
Overrides: object.__setattr__

`__setstate__(...)`

47.2.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

47.3 Class *OptionError*



Known Subclasses: *SCons.compat._scons_optparse.OptionConflictError*

Raised if an *Option* instance is created with invalid or inconsistent arguments.

47.3.1 Methods

`__init__(self, msg, option)`
`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature
 Overrides: *SCons.compat._scons_optparse*.*OptParseError*.`__init__`

`__str__(self)`
`str(x)`
 Overrides: *SCons.compat._scons_optparse*.*OptParseError*.`__str__`

`__delattr__(...)`
`x.__delattr__('name')` <==> `del x.name`
 Overrides: *object*.`__delattr__`

`__getattr__(...)`
`x.__getattr__('name')` <==> `x.name`
 Overrides: *object*.`__getattr__`

`__getitem__(x, y)`
`x[y]`

`__getslice__(x, i, j)`
`x[i:j]`
 Use of negative indices is not supported.

`__hash__(x)`
`hash(x)`

`__new__(T, S, ...)`
Return Value
 a new object with type `S`, a subtype of `T`
 Overrides: *exceptions.BaseException*.`__new__`

`__reduce__(...)`
 helper for pickle
 Overrides: *object*.`__reduce__` `exitit`(inherited documentation)

`__reduce_ex__(...)`
 helper for pickle

<code>__repr__(x)</code>
<code>repr(x)</code>
Overrides: <code>object.__repr__</code>

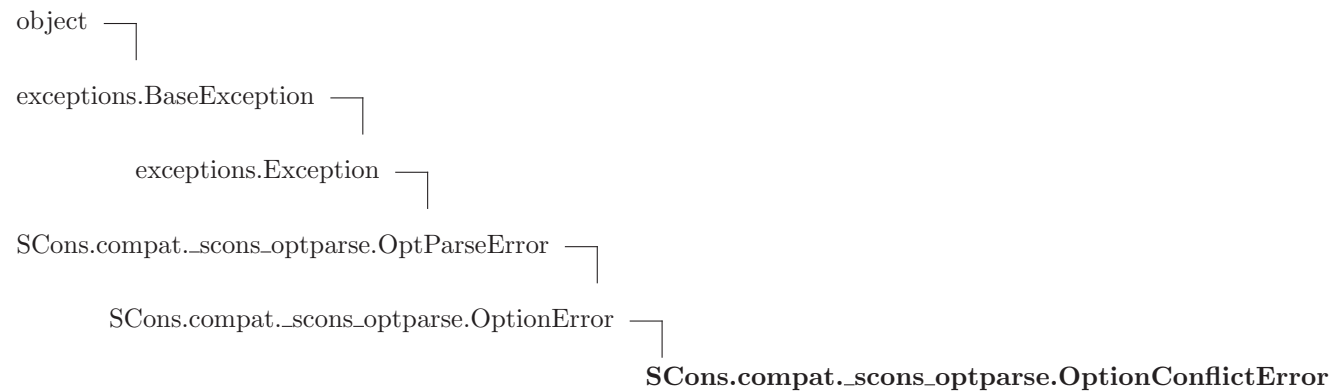
<code>__setattr__(...)</code>
<code>x.__setattr__('name', value) <==> x.name = value</code>
Overrides: <code>object.__setattr__</code>

<code>__setstate__(...)</code>

47.3.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

47.4 Class OptionConflictError



Raised if conflicting options are added to an OptionParser.

47.4.1 Methods

<code>__delattr__(...)</code>
<code>x.__delattr__('name') <==> del x.name</code>
Overrides: <code>object.__delattr__</code>

__getattribute__(...)

x.__getattribute__('name') <==> x.name

Overrides: object.__getattribute__

__getitem__(x, y)

x[y]

__getslice__(x, i, j)

x[i:j]

Use of negative indices is not supported.

__hash__(x)

hash(x)

__init__(self, msg, option)

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

Overrides: SCons.compat._scons_optparse.OptParseError.__init__

__new__(T, S, ...)**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(...)

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

Overrides: object.__setattr__

__setstate__(...)**__str__**(self)

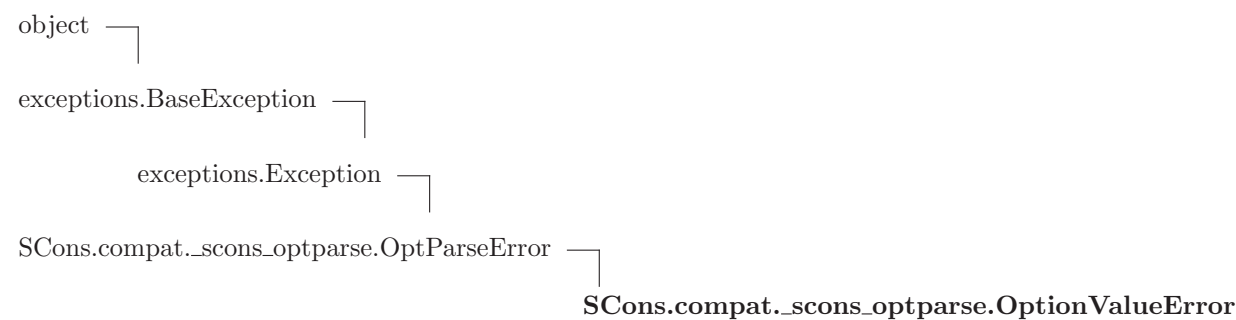
str(x)

Overrides: SCons.compat._scons_optparse.OptParseError.__str__

47.4.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

47.5 Class OptionValueError



Raised if an invalid option value is encountered on the command line.

47.5.1 Methods

<code>__delattr__</code> (...) x. <code>__delattr__</code> ('name') <==> del x.name Overrides: object. <code>__delattr__</code>
<code>__getattr__</code> (...) x. <code>__getattr__</code> ('name') <==> x.name Overrides: object. <code>__getattr__</code>
<code>__getitem__</code> (x, y) x[y]
<code>__getslice__</code> (x, i, j) x[i:j] Use of negative indices is not supported.
<code>__hash__</code> (x) hash(x)

__init__(self, msg)
 x.__init__(...) initializes x; see x.__class__.__doc__ for signature
 Overrides: exceptions.Exception.__init__ extit(inherited documentation)

__new__(T, S, ...)
Return Value
 a new object with type S, a subtype of T
 Overrides: exceptions.BaseException.__new__

__reduce__(...)
 helper for pickle
 Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(...)
 helper for pickle

__repr__(x)
 repr(x)
 Overrides: object.__repr__

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

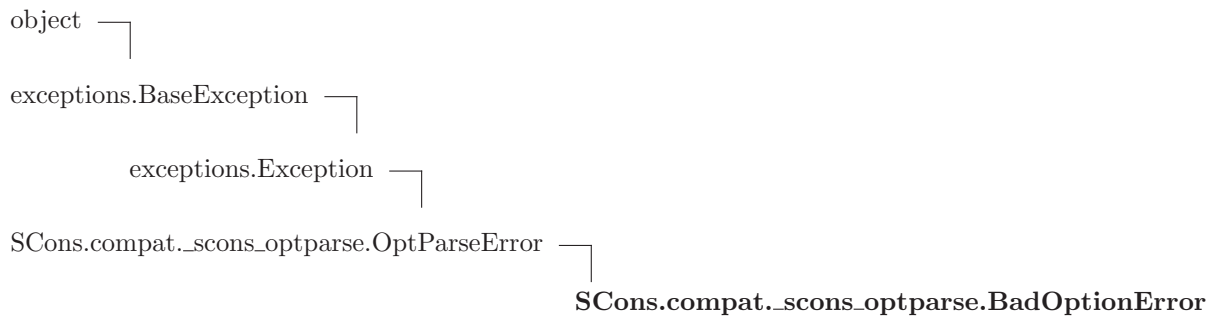
__setstate__(...)

__str__(self)
 str(x)
 Overrides: exceptions.BaseException.__str__ extit(inherited documentation)

47.5.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

47.6 Class *BadOptionError*



Known Subclasses: *SCons.compat._scons_optparse.AmbiguousOptionError*

Raised if an invalid option is seen on the command line.

47.6.1 Methods

`__init__(self, opt_str)`
 Overrides: *SCons.compat._scons_optparse.OptParseError.__init__*

`__str__(self)`
 Overrides: *SCons.compat._scons_optparse.OptParseError.__str__*

`__delattr__(...)`
`x.__delattr__('name') <==> del x.name`
 Overrides: *object.__delattr__*

`__getattr__(...)`
`x.__getattr__('name') <==> x.name`
 Overrides: *object.__getattr__*

`__getitem__(x, y)`
`x[y]`

`__getslice__(x, i, j)`
`x[i:j]`
 Use of negative indices is not supported.

`__hash__(x)`
`hash(x)`

__new__(*T, S, ...*)

Return Value

a new object with type *S*, a subtype of *T*

Overrides: `exceptions.BaseException.__new__`

__reduce__(...)

helper for pickle

Overrides: `object.__reduce__` `exitit`(inherited documentation)

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

`repr(x)`

Overrides: `object.__repr__`

__setattr__(...)

`x.__setattr__('name', value) <==> x.name = value`

Overrides: `object.__setattr__`

__setstate__(...)

47.6.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>
<code>args</code>	Value: <attribute <code>'args'</code> of <code>'exceptions.BaseException'</code> objects>
<code>message</code>	Value: <member <code>'message'</code> of <code>'exceptions.BaseException'</code> objects>

47.7 Class HelpFormatter

Known Subclasses: `SCons.compat._scons_optparse.IndentedHelpFormatter`, `SCons.compat._scons_optparse.TitledHelpFor`

Abstract base class for formatting option help. `OptionParser` instances should use one of the `HelpFormatter` subclasses for formatting help; by default `IndentedHelpFormatter` is used.

Instance attributes: `parser` (`OptionParser`)

the controlling `OptionParser` instance

indent_increment (`int`)

the number of columns to indent per nesting level

max_help_position (`int`)

the maximum starting column for option help text

help_position (int)

the calculated starting column for option help text; initially the same as the maximum

width (int)

total number of columns for output (pass `None` to constructor for this value to be taken from the `$COLUMNS` environment variable)

level (int)

current indentation level

current_indent (int)

current indentation level (in columns)

help_width (int)

number of columns available for option help text (calculated)

default_tag (str)

text to replace with each option's default value, `"%default"` by default. Set to false value to disable default value expansion.

option_strings ({ Option } (str))

maps `Option` instances to the snippet of help text explaining the syntax of that option, e.g. `"-h, --help"` or `"-fFILE, --file=FILE"`

_short_opt_fmt (str)

format string controlling how short options with values are printed in help text. Must be either `"%s%s"` (`"-fFILE"`) or `"%s %s"` (`"-f FILE"`), because those are the two syntaxes that Optik supports.

_long_opt_fmt (str)

similar but for long options; must be either `"%s %s"` (`"--file FILE"`) or `"%s=%s"` (`"--file=FILE"`).

47.7.1 Methods

<code>__init__(self, indent_increment, max_help_position, width, short_first)</code>
--

<code>set_parser(self, parser)</code>

<code>set_short_opt_delimiter(self, delim)</code>

<code>set_long_opt_delimiter(self, delim)</code>
--

<code>indent(self)</code>

<code>dedent(self)</code>

<code>format_usage(self, usage)</code>
--

<code>format_heading(self, heading)</code>
--

<code>format_description(self, description)</code>
--

<code>format_epilog(self, epilog)</code>
--

<code>expand_default(self, option)</code>

<code>format_option(self, option)</code>
--

<code>store_option_strings(self, parser)</code>

<code>format_option_strings(self, option)</code>
--

Return a comma-separated list of option strings & metavariables.
--

47.7.2 Class Variables

Name	Description
NO_DEFAULT_VALUE	Value: 'none'

47.8 Class IndentedHelpFormatter

SCons.compat._scons_optparse.HelpFormatter

SCons.compat._scons_optparse.IndentedHelpFormatter

Format help with indented section bodies.

47.8.1 Methods

<code>__init__(self, indent_increment=2, max_help_position=24, width=False, short_first=False)</code>

Overrides: SCons.compat._scons_optparse.HelpFormatter.__init__
--

<code>format_usage(self, usage)</code>
--

Overrides: SCons.compat._scons_optparse.HelpFormatter.format_usage
--

<code>format_heading(self, heading)</code>
--

Overrides: SCons.compat._scons_optparse.HelpFormatter.format_heading
--

<code>dedent(self)</code>

<code>expand_default(self, option)</code>

<code>format_description(self, description)</code>
--

<code>format_epilog(self, epilog)</code>
--

<code>format_option(self, option)</code>
--

format_option_strings (<i>self</i> , <i>option</i>)
--

Return a comma-separated list of option strings & metavariables.
--

indent (<i>self</i>)

set_long_opt_delimiter (<i>self</i> , <i>delim</i>)
--

set_parser (<i>self</i> , <i>parser</i>)

set_short_opt_delimiter (<i>self</i> , <i>delim</i>)

store_option_strings (<i>self</i> , <i>parser</i>)

47.8.2 Class Variables

Name	Description
NO_DEFAULT_VALUE	Value: 'none'

47.9 Class *TitledHelpFormatter*

SCons.compat._scons_optparse.HelpFormatter

SCons.compat._scons_optparse.TitledHelpFormatter

Format help with underlined section headers.

47.9.1 Methods

__init__ (<i>self</i> , <i>indent_increment</i> =0, <i>max_help_position</i> =24, <i>width</i> =False, <i>short_first</i> =0)

Overrides: <i>SCons.compat._scons_optparse.HelpFormatter.__init__</i>

format_usage (<i>self</i> , <i>usage</i>)
--

Overrides: <i>SCons.compat._scons_optparse.HelpFormatter.format_usage</i>

format_heading (<i>self</i> , <i>heading</i>)
--

Overrides: <i>SCons.compat._scons_optparse.HelpFormatter.format_heading</i>

dedent (<i>self</i>)

expand_default (<i>self</i> , <i>option</i>)

format_description (<i>self</i> , <i>description</i>)
--

format_epilog (<i>self</i> , <i>epilog</i>)
--

format_option (<i>self</i> , <i>option</i>)
--

format_option_strings (<i>self</i> , <i>option</i>)
--

Return a comma-separated list of option strings & metavariables.
--

indent (<i>self</i>)

set_long_opt_delimiter (<i>self</i> , <i>delim</i>)
--

set_parser (<i>self</i> , <i>parser</i>)

set_short_opt_delimiter (<i>self</i> , <i>delim</i>)

store_option_strings (<i>self</i> , <i>parser</i>)

47.9.2 Class Variables

Name	Description
NO_DEFAULT_VALUE	Value: 'none'

47.10 Class Option

Instance attributes: `_short_opts` : [string] `_long_opts` : [string]

`action` : string type : string dest : string default : any nargs : int const : any choices : [string]
`callback` : function callback_args : (any*) callback_kwargs : { string : any } help : string metavar : string

47.10.1 Methods

__init__ (<i>self</i> , * <i>opts</i> , ** <i>attrs</i>)

__str__ (<i>self</i>)

__repr__ (<i>self</i>)

takes_value (<i>self</i>)

get_opt_string (<i>self</i>)

check_value (<i>self</i> , <i>opt</i> , <i>value</i>)
--

convert_value (<i>self</i> , <i>opt</i> , <i>value</i>)
--

process (<i>self</i> , <i>opt</i> , <i>value</i> , <i>values</i> , <i>parser</i>)
--

take_action (<i>self</i> , <i>action</i> , <i>dest</i> , <i>opt</i> , <i>value</i> , <i>values</i> , <i>parser</i>)
--

47.10.2 Class Variables

Name	Description
ATTRS	Value: ['action', 'type', 'dest', 'default', 'nargs', 'const', '...']
ACTIONS	Value: ('store', 'store_const', 'store_true', 'store_false', 'ap...')
STORE_ACTIONS	Value: ('store', 'store_const', 'store_true', 'store_false', 'ap...')
TYPED_ACTIONS	Value: ('store', 'append', 'callback')
ALWAYS_TYPED_ACTIONS	Value: ('store', 'append')
CONST_ACTIONS	Value: ('store_const', 'append_const')
TYPES	Value: ('string', 'int', 'long', 'float', 'complex', 'choice')
TYPE_CHECKER	Value: {'choice': <function check_choice at 0xf2f2a8>, 'complex'...}
CHECK_METHODS	Value: [_check_action, _check_type, _check_choice, _check_dest, ...]

47.11 Class Values

47.11.1 Methods

```
__init__(self, defaults=False)
```

```
__str__(self)
```

```
__repr__(self)
```

```
__cmp__(self, other)
```

```
read_module(self, modname, mode='careful')
```

```
read_file(self, filename, mode='careful')
```

```
ensure_value(self, attr, value)
```

47.12 Class OptionContainer

Known Subclasses: `SCons.compat._scons_optparse.OptionGroup`, `SCons.compat._scons_optparse.OptionParser`

Abstract base class.

Class attributes: `standard_option_list` ([`Option`])

list of standard options that will be accepted by all instances of this parser class (intended to be overridden by subclasses).

Instance attributes: option_list ([Option])

the list of Option objects contained by this OptionContainer

_short_opt ({ string} (Option))

dictionary mapping short option strings, eg. “-f” or “-X”, to the Option instances that implement them. If an Option has multiple short option strings, it will appear in this dictionary multiple times. [1]

_long_opt ({ string} (Option))

dictionary mapping long option strings, eg. “--file” or “--exclude”, to the Option instances that implement them. Again, a given Option can occur multiple times in this dictionary. [1]

defaults ({ string} (any))

dictionary mapping option destination names to default values for each destination [1]

[1] These mappings are common to (shared by) all components of the controlling OptionParser, where they are initially created.

47.12.1 Methods

__init__ (<i>self</i> , <i>option_class</i> , <i>conflict_handler</i> , <i>description</i>)
--

set_conflict_handler (<i>self</i> , <i>handler</i>)
--

set_description (<i>self</i> , <i>description</i>)

get_description (<i>self</i>)
--

destroy (<i>self</i>)

see OptionParser.destroy().

add_option (<i>Option</i>)

add_option(opt_str, ..., kwarg=val, ...)
--

add_options (<i>self</i> , <i>option_list</i>)

get_option (<i>self</i> , <i>opt_str</i>)
--

has_option (<i>self</i> , <i>opt_str</i>)
--

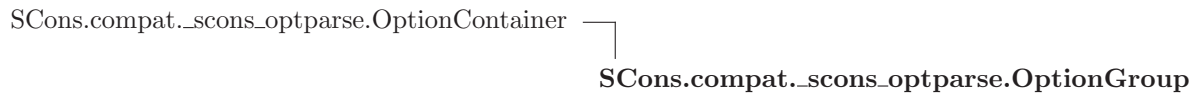
remove_option (<i>self</i> , <i>opt_str</i>)

format_option_help (<i>self</i> , <i>formatter</i>)
--

format_description (<i>self</i> , <i>formatter</i>)
--

format_help (<i>self</i> , <i>formatter</i>)

47.13 Class *OptionGroup*



47.13.1 Methods

`__init__(self, parser, title, description=False)`
 Overrides: *SCons.compat._scons_optparse.OptionContainer.__init__*

`set_title(self, title)`

`destroy(self)`
 see *OptionParser.destroy()*.
 Overrides: *SCons.compat._scons_optparse.OptionContainer.destroy*

`format_help(self, formatter)`
 Overrides: *SCons.compat._scons_optparse.OptionContainer.format_help*

`add_option(Option)`
`add_option(opt_str, ..., kwarg=val, ...)`

`add_options(self, option_list)`

`format_description(self, formatter)`

`format_option_help(self, formatter)`

`get_description(self)`

`get_option(self, opt_str)`

`has_option(self, opt_str)`

`remove_option(self, opt_str)`

`set_conflict_handler(self, handler)`

`set_description(self, description)`

47.14 Class *OptionParser*



Class attributes:

`standard_option_list` : [Option]
 list of standard options that will be accepted by all instances of this parser class (intended to be overridden by subclasses).

Instance attributes:

`usage` : string
 a usage string for your program. Before it is displayed to the user, "%prog" will be expanded to the name of your program (self.prog or os.path.basename(sys.argv[0])).

`prog` : string
 the name of the current program (to override os.path.basename(sys.argv[0])).

`epilog` : string
 paragraph of help text to print after option help

`option_groups` : [OptionGroup]
 list of option groups in this parser (option groups are irrelevant for parsing the command-line, but very useful for generating help)

`allow_interspersed_args` : bool = true
 if true, positional arguments may be interspersed with options. Assuming -a and -b each take a single argument, the command-line
 -ablah foo bar -bboo baz
 will be interpreted the same as
 -ablah -bboo -- foo bar baz
 If this flag were false, that command line would be interpreted as
 -ablah -- foo bar -bboo baz
 -- ie. we stop processing options as soon as we see the first non-option argument. (This is the tradition followed by Python's getopt module, Perl's Getopt::Std, and other argument-parsing libraries, but it is generally annoying to users.)

`process_default_values` : bool = true
 if true, option default values are processed similarly to option values from the command line: that is, they are passed to the type-checking function for the option's type (as long as the default value is a string). (This really only matters if you have defined custom types; see SF bug #955889.) Set it to false to restore the behaviour of Optik 1.4.1 and earlier.

`rargs` : [string]
 the argument list currently being parsed. Only set when parse_args() is active, and continually trimmed down as we consume arguments. Mainly there for the benefit of callback options.

`largs` : [string]
 the list of leftover arguments that we have skipped while parsing options. If allow_interspersed_args is false, this list is always empty.

`values` : Values

the set of option values currently being accumulated. Only set when `parse_args()` is active. Also mainly for callbacks.

Because of the `'rargs'`, `'larges'`, and `'values'` attributes, `OptionParser` is not thread-safe. If, for some perverse reason, you need to parse command-line arguments simultaneously in different threads, use different `OptionParser` instances.

47.14.1 Methods

```
__init__(self, usage=False, option_list=False, option_class=<class
SCons.compat._scons_optparse.Option at 0xf2b410>, version=False,
conflict_handler='error', description=False, formatter=False, add_help_option=True,
prog=False, epilog=False)
Overrides: SCons.compat._scons_optparse.OptionContainer.__init__
```

```
destroy(self)
```

Declare that you are done with this `OptionParser`. This cleans up reference cycles so the `OptionParser` (and all objects referenced by it) can be garbage-collected promptly. After calling `destroy()`, the `OptionParser` is unusable.

Overrides: `SCons.compat._scons_optparse.OptionContainer.destroy`

```
set_usage(self, usage)
```

```
enable_interspersed_args(self)
```

```
disable_interspersed_args(self)
```

```
set_process_default_values(self, process)
```

```
set_default(self, dest, value)
```

```
set_defaults(self, **kwargs)
```

```
get_default_values(self)
```

```
add_option_group(self, *args, **kwargs)
```

```
get_option_group(self, opt_str)
```

```
parse_args(self, args=False, values=False)
```

```
parse_args(args ([string] = sys.argv[1:]),
values : Values = None)
-> (values : Values, args : [string])
```

Parse the command-line options found in `'args'` (default: `sys.argv[1:]`). Any errors result in a call to `'error()'`, which by default prints the usage message to `stderr` and calls `sys.exit()` with an error message. On success returns a pair (values, args) where `'values'` is an `Values` instance (with all your option values) and `'args'` is the list of arguments left over after parsing options.

check_values(*self*, *values*, *args*)

check_values(values : Values, args : [string]) -> (values : Values, args : [string])
 Check that the supplied option values and leftover arguments are valid. Returns the option values and leftover arguments (possibly adjusted, possibly completely new -- whatever you like). Default implementation just returns the passed-in values; subclasses may override as desired.

get_prog_name(*self*)

expand_prog_name(*self*, *s*)

get_description(*self*)

Overrides: *SCons.compat._scons_optparse*.OptionContainer.get_description

exit(*self*, *status*=0, *msg*=False)

error(*self*, *msg*)

error(msg : string)
 Print a usage message incorporating 'msg' to stderr and exit. If you override this in a subclass, it should not return -- it should either exit or raise an exception.

get_usage(*self*)

print_usage(*self*, *file*=False)

print_usage(file : file = stdout)
 Print the usage message for the current program (self.usage) to 'file' (default stdout). Any occurrence of the string "%prog" in self.usage is replaced with the name of the current program (basename of sys.argv[0]). Does nothing if self.usage is empty or not defined.

get_version(*self*)

print_version(*self*, *file*=False)

print_version(file : file = stdout)
 Print the version message for this program (self.version) to 'file' (default stdout). As with print_usage(), any occurrence of "%prog" in self.version is replaced by the current program's name. Does nothing if self.version is empty or undefined.

format_option_help(*self*, *formatter*=False)

Overrides: *SCons.compat._scons_optparse*.OptionContainer.format_option_help

format_epilog(*self*, *formatter*)

format_help(*self*, *formatter*=False)

Overrides: *SCons.compat._scons_optparse*.OptionContainer.format_help

print_help(*self*, *file*=False)

 print_help(*file* : *file* = stdout)

Print an extended help message, listing all options and any help text provided with them, to 'file' (default stdout).

add_option(*Option*)

 add_option(*opt_str*, ..., *kwarg*=val, ...)

add_options(*self*, *option_list*)

format_description(*self*, *formatter*)

get_option(*self*, *opt_str*)

has_option(*self*, *opt_str*)

remove_option(*self*, *opt_str*)

set_conflict_handler(*self*, *handler*)

set_description(*self*, *description*)

47.14.2 Class Variables

Name	Description
standard_option_list	Value: []

48 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'.

48.1 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.

48.1.1 Methods

<code>__init__(self)</code>
This is an abstract class.
Overrides: object.__init__

<code>__len__(self)</code>
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.

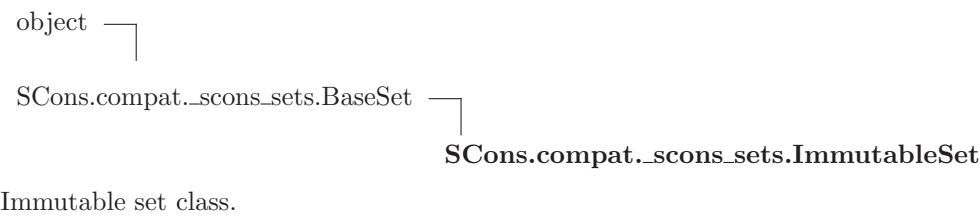
__lt__(*self*, *other*)**__gt__**(*self*, *other*)

__delattr__ (...) x.__delattr__('name') <==> del x.name
__getattr__ (...) x.__getattr__('name') <==> x.name
__hash__ (x) hash(x)
__new__ (T, S, ...) Return Value a new object with type S, a subtype of T
__reduce__ (...) helper for pickle
__reduce_ex__ (...) helper for pickle
__setattr__ (...) x.__setattr__('name', value) <==> x.name = value

48.1.2 Properties

Name	Description
__class__	Value: <attribute '.__class__' of 'object' objects>

48.2 Class ImmutableSet



48.2.1 Methods**`__init__(self, iterable=False)`**

Construct an immutable set from an optional iterable.

Overrides: *SCons.compat._scons_sets.BaseSet.__init__***`__hash__(self)`**`hash(x)`Overrides: `object.__hash__` `exitit`(inherited documentation)**`__getstate__(self)`****`__setstate__(self, state)`****`__and__(self, other)`**

Return the intersection of two sets as a new set.

(I.e. all elements that are in both sets.)

`__cmp__(self, other)`**`__contains__(self, element)`**

Report whether an element is a member of a set.

(Called in response to the expression `'element in self'`.)**`__copy__(self)`**

Return a shallow copy of a set.

`__deepcopy__(self, memo)`Return a deep copy of a set; used by `copy` module.**`__delattr__(...)`**`x.__delattr__('name') <==> del x.name`**`__eq__(self, other)`****`__ge__(self, other)`**

Report whether this set contains another set.

`__getattr__(...)``x.__getattr__('name') <==> x.name`**`__gt__(self, other)`**

__iter__(self)

Return an iterator over the elements or a set.
 This is the keys iterator for the underlying dict.

__le__(self, other)

Report whether another set contains this set.

__len__(self)

Return the number of elements of a set.

__lt__(self, other)**__ne__(self, other)****__new__(T, S, ...)****Return Value**

a new object with type S, a subtype of T

__or__(self, other)

Return the union of two sets as a new set.
 (I.e. all elements that are in either set.)

__reduce__(...)

helper for pickle

__reduce_ex__(...)

helper for pickle

__repr__(self)

Return string representation of a set.
 This looks like 'Set([<list of elements>])'.
 Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

__str__(self)

Return string representation of a set.
 This looks like 'Set([<list of elements>])'.
 Overrides: object.__str__

`__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.)

`__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.)

`copy(self)`

Return a shallow copy of a set.

`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.)

`intersection(self, other)`

Return the intersection of two sets as a new set.
(I.e. all elements that are in both sets.)

`issubset(self, other)`

Report whether another set contains this set.

`issuperset(self, other)`

Report whether this set contains another set.

`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.)

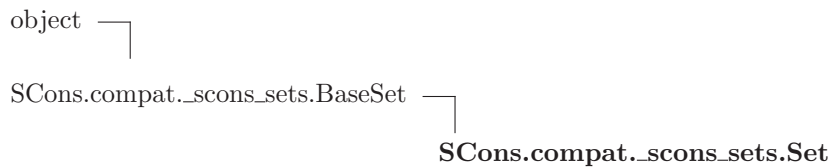
`union(self, other)`

Return the union of two sets as a new set.
(I.e. all elements that are in either set.)

48.2.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>

48.3 Class Set



Mutable set class.

48.3.1 Methods

`__init__(self, iterable=False)`

Construct a set from an optional iterable.

Overrides: `SCons.compat._scons_sets.BaseSet.__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*)

__and__(*self*, *other*)

Return the intersection of two sets as a new set.

(I.e. all elements that are in both sets.)

__cmp__(*self*, *other*)

__contains__(*self*, *element*)

Report whether an element is a member of a set.

(Called in response to the expression `'element in self'`.)

__copy__(*self*)

Return a shallow copy of a set.

__deepcopy__(*self*, *memo*)

Return a deep copy of a set; used by copy module.

__delattr__(...)

x.__delattr__('name') <==> del x.name

__eq__(*self*, *other*)

__ge__(*self*, *other*)

Report whether this set contains another set.

__getattr__(...)

x.__getattr__('name') <==> x.name

__gt__(*self*, *other*)

__iter__(*self*)

Return an iterator over the elements or a set.
This is the keys iterator for the underlying dict.

__le__(*self*, *other*)

Report whether another set contains this set.

__len__(*self*)

Return the number of elements of a set.

__lt__(*self*, *other*)

__ne__(*self*, *other*)

__new__(*T*, *S*, ...)

Return Value

a new object with type *S*, a subtype of *T*

__or__(*self*, *other*)

Return the union of two sets as a new set.
(I.e. all elements that are in either set.)

__reduce__(...)

helper for pickle

__reduce_ex__(...)

helper for pickle

__repr__(self)

Return string representation of a set.
This looks like 'Set([<list of elements>])'.

Overrides: object.__repr__

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

__str__(self)

Return string representation of a set.
This looks like 'Set([<list of elements>])'.

Overrides: object.__str__

__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.)

__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.)

copy(self)

Return a shallow copy of a set.

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.)

intersection(self, other)

Return the intersection of two sets as a new set.
(I.e. all elements that are in both sets.)

issubset(self, other)

Report whether another set contains this set.

issuperset(*self*, *other*)

Report whether this set contains another set.

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.)

union(*self*, *other*)

Return the union of two sets as a new set.
(I.e. all elements that are in either set.)

48.3.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>

49 Module *SCons.compat._scons_sets15*

49.1 Class Set

The set class. It can contain mutable objects.

49.1.1 Methods

`__init__(self, seq=False)`

The constructor. It can take any object giving an iterator as an optional argument to populate the new set.

`__str__(self)`

`copy(self)`

Shallow copy of a set object.

`__contains__(self, elem)`

`__len__(self)`

`__getitem__(self, index)`

`items(self)`

Returns a list of the elements in the set.

`add(self, elem)`

Add one element to the set.

`remove(self, elem)`

Remove an element from the set. Return an error if elem is not in the set.

`discard(self, elem)`

Remove an element from the set. Do nothing if elem is not in the set.

`sort(self, func=<built-in function cmp>)`

`__iter__(self)`

`__or__(self, other)`

Union of two sets.

__sub__(*self*, *other*)

Difference of two sets.

__and__(*self*, *other*)

Intersection of two sets.

__add__(*self*, *other*)

Symmetric difference of two sets.

__mul__(*self*, *other*)

Cartesian product of two sets.

__lt__(*self*, *other*)

Returns 1 if the lhs set is contained but not equal to the rhs set.

__le__(*self*, *other*)

Returns 1 if the lhs set is contained in the rhs set.

__eq__(*self*, *other*)

Returns 1 if the sets are equal.

__cmp__(*self*, *other*)

Returns 1 if the sets are equal.

50 Module *SCons.compat._scons_shlex*

A lexical analyzer class for simple shell-like syntaxes.

50.1 Functions

split (<i>s</i> , <i>comments</i> =False)

50.2 Class *shlex*

A lexical analyzer class for simple shell-like syntaxes.

50.2.1 Methods

__init__ (<i>self</i> , <i>instream</i> =False, <i>infile</i> =False, <i>posix</i> =False)
--

push_token (<i>self</i> , <i>tok</i>)
--

Push a token onto the stack popped by the <code>get_token</code> method

push_source (<i>self</i> , <i>newstream</i> , <i>newfile</i> =False)
--

Push an input source onto the lexer's input source stack.

pop_source (<i>self</i>)

Pop the input source stack.

get_token (<i>self</i>)

Get a token from the input stream (or from stack if it's nonempty)
--

read_token (<i>self</i>)

sourcehook (<i>self</i> , <i>newfile</i>)
--

Hook called on a filename to be sourced.
--

error_leader (<i>self</i> , <i>infile</i> =False, <i>lineno</i> =False)

Emit a C-compiler-like, Emacs-friendly error-message leader.
--

__iter__ (<i>self</i>)

next (<i>self</i>)

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 child's 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)`

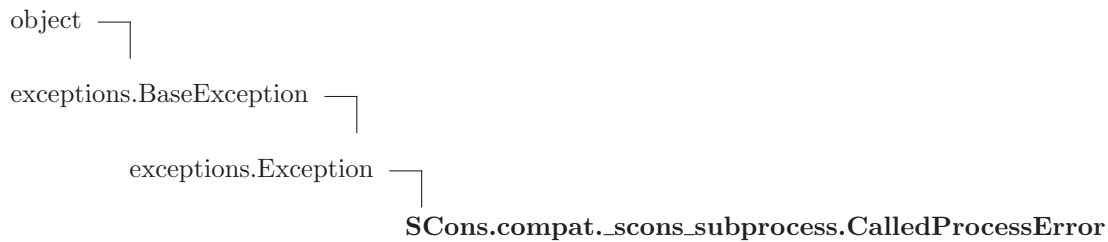
Run command with arguments. Wait for command to complete. If the exit code was zero then return, otherwise raise `CalledProcessError`. The `CalledProcessError` object will have the return code in the returncode attribute.

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



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: `exceptions.Exception.__init__` `exitit`(inherited documentation)

`__str__(self)`
`str(x)`
 Overrides: `exceptions.BaseException.__str__` `exitit`(inherited documentation)

`__delattr__(...)`
`x.__delattr__('name') <==> del x.name`
 Overrides: `object.__delattr__`

`__getattr__(...)`
`x.__getattr__('name') <==> x.name`
 Overrides: `object.__getattr__`

`__getitem__(x, y)`
`x[y]`

`__getslice__(x, i, j)`
`x[i:j]`
 Use of negative indices is not supported.

`__hash__(x)`
`hash(x)`

`__new__(T, S, ...)`
Return Value
 a new object with type `S`, a subtype of `T`
 Overrides: `exceptions.BaseException.__new__`

`__reduce__(...)`
helper for pickle
Overrides: object.__reduce__ extit(inherited documentation)

`__reduce_ex__(...)`
helper for pickle

`__repr__(x)`
repr(x)
Overrides: object.__repr__

`__setattr__(...)`
x.__setattr__('name', value) <==> x.name = value
Overrides: object.__setattr__

`__setstate__(...)`

51.3.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute '.__class__' of 'object' objects>
<code>args</code>	Value: <attribute 'args' of 'exceptions.BaseException' objects>
<code>message</code>	Value: <member 'message' of 'exceptions.BaseException' objects>

51.4 Class Popen



51.4.1 Methods

`__init__(self, args, bufsize=0, executable=False, stdin=False, stdout=False, stderr=False, preexec_fn=False, close_fds=False, shell=False, cwd=False, env=False, universal_newlines=False, startupinfo=False, creationflags=0)`
Create new Popen instance.
Overrides: object.__init__

`__del__(self)`

communicate(*self*, *input*=False)

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*=False)

Check if child process has terminated. Returns returncode attribute.

wait(*self*)

Wait for child process to terminate. Returns returncode attribute.

__delattr__(...)

x.__delattr__('name') <==> del x.name

__getattr__(...)

x.__getattr__('name') <==> x.name

__hash__(*x*)

hash(x)

__new__(*T*, *S*, ...)

Return Value

a new object with type *S*, a subtype of *T*

__reduce__(...)

helper for pickle

__reduce_ex__(...)

helper for pickle

__repr__(*x*)

repr(x)

__setattr__(...)

x.__setattr__('name', value) <==> x.name = value

__str__(*x*)

str(x)

51.4.2 Properties

Name	Description
<code>__class__</code>	Value: <attribute <code>'__class__'</code> of <code>'object'</code> objects>

52 Module *SCons.compat._scons_textwrap*

Text wrapping and filling.

52.1 Functions

wrap(*text*, *width*=70, ***kwargs*)

Wrap a single paragraph of text, returning a list of wrapped lines.
Reformat the single paragraph in 'text' so it fits in lines of no more than 'width' columns, and return a list of wrapped lines. By default, tabs in 'text' are expanded with `string.expandtabs()`, and all other whitespace characters (including newline) are converted to space. See `TextWrapper` class for available keyword args to customize wrapping behaviour.

fill(*text*, *width*=70, ***kwargs*)

Fill a single paragraph of text, returning a new string.
Reformat the single paragraph in 'text' to fit in lines of no more than 'width' columns, and return a new string containing the entire wrapped paragraph. As with `wrap()`, tabs are expanded and other whitespace characters converted to space. See `TextWrapper` class for available keyword args to customize wrapping behaviour.

52.2 Class `TextWrapper`

Object for wrapping/filling text. The public interface consists of the `wrap()` and `fill()` methods; the other methods are just there for subclasses to override in order to tweak the default behaviour. If you want to completely replace the main wrapping algorithm, you'll probably have to override `_wrap_chunks()`.

Several instance attributes control various aspects of wrapping: **width** (default: 70) the maximum width of wrapped lines (unless `break_long_words` is false)

initial_indent (default: "") string that will be prepended to the first line of wrapped output. Counts towards the line's width.

subsequent_indent (default: "") string that will be prepended to all lines save the first of wrapped output; also counts towards each line's width.

expand_tabs (default: true) Expand tabs in input text to spaces before further processing. Each tab will become 1 .. 8 spaces, depending on its position in its line. If false, each tab is treated as a single character.

replace_whitespace (default: true) Replace all whitespace characters in the input text by spaces after tab expansion. Note that if `expand_tabs` is false and `replace_whitespace` is true, every tab will be converted to a single space!

fix_sentence_endings (default: false) Ensure that sentence-ending punctuation is always followed by two spaces. Off by default because the algorithm is (unavoidably) imperfect.

break_long_words (default: true) Break words longer than 'width'. If false, those words will not be broken, and some lines might be longer than 'width'.

52.2.1 Methods

__init__(*self*, *width*=70, *initial_indent*='', *subsequent_indent*='', *expand_tabs*=True, *replace_whitespace*=True, *fix_sentence_endings*=False, *break_long_words*=True)

wrap(*self*, *text*)

wrap(*text* : string) -> [string]

Reformat the single paragraph in 'text' so it fits in lines of no more than 'self.width' columns, and return a list of wrapped lines. Tabs in 'text' are expanded with string.expandtabs(), and all other whitespace characters (including newline) are converted to space.

fill(*self*, *text*)

fill(*text* : string) -> string

Reformat the single paragraph in 'text' to fit in lines of no more than 'self.width' columns, and return a new string containing the entire wrapped paragraph.

52.2.2 Class Variables

Name	Description
whitespace_trans	Value: '\x00\x01\x02\x03\x04\x05\x06\x07\x08\x0e\x0f\x10\x11...'
unicode_whitespace_trans	Value: {9: 32, 10: 32, 11: 32, 12: 32, 13: 32, 32: 32}
uspace	Value: 32
wordsep_re	Value: re.compile(r'(\s+ [\^\s\w]*\w{2,}-(?=\w{2,}) (?<=[\w!"'\&\...'
sentence_end_re	Value: re.compile(r'[abcdefghijklmnopqrstuvwxy] [\.\!\\?] ["'\']?')
x	Value: 32

53 Module SCons.compat.builtins

Compatibility idioms for `__builtin__` names

This module adds names to the `__builtin__` 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 `__builtin__` names:

`all()` `any()` `bool()` `dict()` `True` `False` `zip()`

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.

53.1 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.

bool(*value*)

Demote a value to 0 or 1, depending on its truth value.

This is not to be confused with `types.BooleanType`, which is way too hard to duplicate in early Python versions to be worth the trouble.

dict(*seq*=[], ***kwargs*)

New dictionary initialization.

zip(**lists*)

Emulates the behavior we need from the built-in `zip()` function added in Python 2.2.

Returns a list of tuples, where each tuple contains the *i*-th element from each of the argument sequences. The returned list is truncated in length to the length of the shortest argument sequence.

53.2 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>__revision__</code>	Value: 'src/engine/SCons/compat/builtins.py 3266 2008/08/12 07:3...

continued on next page

Name	Description
False	Value: False
True	Value: True

54 Module SCons.cpp

SCons C Pre-Processor module

54.1 Functions

CPP_to_Python_Ops_Sub(*m*, *d*={'\r': ' ', '!': ' ' not ' ', '!=': ' != ', '&&': ' and ', ':'...})

CPP_to_Python(*s*)

Converts a C pre-processor expression into an equivalent Python expression that can be evaluated.

54.2 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>cpp_lines_dict</code>	Value: {'define': '\\s+([_A-Za-z][_A-Za-z0-9_]+)(\\([^\)]*\))?....
<code>Table</code>	Value: {'define': <code>re.compile(r'\\s+([_A-Za-z][_A-Za-z0-9_]+)(\\([^\]</code>
<code>e</code>	Value: '^\\s*#\\s*(elif undef include_next endif else include if...
<code>CPP_Expression</code>	Value: <code>re.compile(r'(?m)^\\s*#\\s*(elif undef include_next endif e...</code>
<code>CPP_to_Python_Ops_Dict</code>	Value: {'\r': ' ', '!': ' ' not ' ', '!=': ' != ', '&&': ' and ', ':'...
<code>CPP_to_Python_Ops_Expression</code>	Value: <code>re.compile(r'\\ \\ && != ! \\r :\\ \\?')</code>
<code>CPP_to_Python_Eval_List</code>	Value: [[<code>re.compile(r'defined\\s+(\\w+)')</code> , '__dict__.has_key("\\1"...
<code>line_continuations</code>	Value: <code>re.compile(r'\\\\r?\\n')</code>
<code>function_name</code>	Value: <code>re.compile(r'\\S+\\((\\[^\)]*\\)\\)')</code>
<code>function_arg_separator</code>	Value: <code>re.compile(r',\\s*')</code>

54.3 Class FunctionEvaluator

Handles delayed evaluation of a #define function call.

54.3.1 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.

__call__(self, *values)

Evaluates the expansion of a `#define` macro function called with the specified values.

54.4 Class PreProcessor

Known Subclasses: `SCons.cpp.DumbPreProcessor`, `SCons.Scanner.C.SConsCPPScanner`

The main workhorse class for handling C pre-processing.

54.4.1 Methods

__call__(self, file)

Pre-processes a file.

This is the main public entry point.

__init__(self, current='.', cpppath=(), dict={}, all=0)

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.**evalExpression**(*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*=False)

Pre-processes a file contents.

This is the main internal entry point.

read_file(*self*, *file*)**resolve_include**(*self*, *t*)Resolve a tuple-sized `#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*=False)**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*=False)**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.

54.5 Class DumbPreProcessor

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.

54.5.1 Methods

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

Overrides: SCons.cpp.PreProcessor.__init__

__call__(*self*, *file*)

Pre-processes a file.

This is the main public entry point.

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*=False)

Pre-processes a file contents.

This is the main internal entry point.

read_file(*self*, *file*)**resolve_include**(*self*, *t*)

Resolve a tuple-sized #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*=False)

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=False*)

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.

55 Module SCons.dblite

55.1 Functions

<code>corruption_warning(filename)</code>

<code>is_string(s)</code>

<code>unicode(s)</code>

<code>open(file, flag=False, mode=438)</code>

55.2 Variables

Name	Description
keep_all_files	Value: 0
ignore_corrupt_dbfiles	Value: 0
dblite_suffix	Value: '.dblite'
tmp_suffix	Value: '.tmp'

55.3 Class dblite

55.3.1 Methods

<code>__init__(self, file_base_name, flag, mode)</code>

<code>__del__(self)</code>

<code>sync(self)</code>

<code>__getitem__(self, key)</code>

<code>__setitem__(self, key, value)</code>
--

<code>keys(self)</code>

<code>has_key(self, key)</code>

<code>__contains__(self, key)</code>

<code>iterkeys(self)</code>

<code>__iter__(self)</code>

<code>__len__(self)</code>

56 Module *SCons.exitfuncs*

SCons.exitfuncs

Register functions which are executed when *SCons* exits for any reason.

56.1 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*

56.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/exitfuncs.py 3266 2008/08/12 07:31:01 k...
<code>x</code>	Value: <code>sys.exitfunc</code>

57 Module md5

57.1 Variables

Name	Description
blocksize	Value: False
digest_size	Value: 16

Index

- cmd.Cmd.cmdloop (*function*), 241
- cmd.Cmd.columnize (*function*), 241
- cmd.Cmd.complete (*function*), 241
- cmd.Cmd.complete_help (*function*), 241
- cmd.Cmd.completedefault (*function*), 242
- cmd.Cmd.completenames (*function*), 242
- cmd.Cmd.emptyline (*function*), 242
- cmd.Cmd.get_names (*function*), 242
- cmd.Cmd.parseline (*function*), 242
- cmd.Cmd.postcmd (*function*), 242
- cmd.Cmd.postloop (*function*), 242
- cmd.Cmd.precmd (*function*), 242
- cmd.Cmd.preloop (*function*), 242
- cmd.Cmd.print_topics (*function*), 242
- exceptions.BaseException.__getitem__ (*function*), 64, 66, 68, 69, 71, 73, 74, 192, 193, 195, 197, 245, 258, 301, 320, 322, 324, 325, 327, 329, 331, 333, 335, 337, 338, 340, 342, 344, 345, 347, 349, 351, 353, 355, 366, 368, 370, 371, 373, 408
- exceptions.BaseException.__getslice__ (*function*), 64, 66, 68, 69, 71, 73, 74, 192, 193, 195, 197, 245, 258, 301, 320, 322, 324, 326, 327, 329, 331, 333, 335, 337, 339, 340, 342, 344, 346, 347, 349, 351, 353, 355, 366, 368, 370, 371, 373, 408
- exceptions.BaseException.__setstate__ (*function*), 65, 67, 68, 70, 72, 73, 75, 192, 194, 196, 198, 246, 259, 302, 321, 323, 325, 326, 328, 330, 332, 334, 336, 338, 339, 341, 343, 345, 346, 348, 350, 352, 354, 355, 367, 369, 370, 372, 374, 409
- md5 (*module*), 425
- object.__delattr__ (*function*), 82, 292, 388, 390, 395, 410
- object.__getattr__ (*function*), 83, 389, 390, 395, 410
- object.__hash__ (*function*), 64, 66, 68, 69, 71, 73, 75, 83, 192, 194, 195, 197, 246, 258, 301, 320, 322, 324, 326, 327, 329, 331, 333, 335, 337, 339, 340, 342, 344, 346, 347, 349, 351, 353, 355, 366, 368, 370, 371, 373, 389, 408, 410
- object.__init__ (*function*), 293
- object.__new__ (*function*), 83, 389, 391, 395, 410
- object.__reduce__ (*function*), 83, 90, 293, 389, 391, 395, 410
- object.__reduce_ex__ (*function*), 65, 66, 68, 70, 71, 73, 75, 83, 90, 192, 194, 196, 197, 246, 259, 293, 301, 321, 323, 324, 326, 328, 330, 332, 334, 336, 337, 339, 341, 343, 344, 346, 348, 350, 352, 353, 355, 367, 368, 370, 372, 374, 389, 391, 396, 409, 410
- object.__repr__ (*function*), 410
- object.__setattr__ (*function*), 83, 294, 389, 391, 396, 410
- object.__str__ (*function*), 83, 90, 410
- SCons (*package*), 2–4
 - SCons.Action (*module*), 5–13
 - SCons.Builder (*module*), 14–26
 - SCons.Builder.Builder (*function*), 15
 - SCons.Builder.BuilderBase (*class*), 23–25
 - SCons.Builder.CallableSelector (*class*), 17–18
 - SCons.Builder.CompositeBuilder (*class*), 25–26
 - SCons.Builder.DictCmdGenerator (*class*), 15–17
 - SCons.Builder.DictEmitter (*class*), 18–20
 - SCons.Builder.EmitterProxy (*class*), 23
 - SCons.Builder.ListEmitter (*class*), 20–22
 - SCons.Builder.OverrideWarner (*class*), 22–23
 - SCons.CacheDir (*module*), 27–28
 - SCons.CacheDir.CacheDir (*class*), 27–28
 - SCons.CacheDir.CachePushFunc (*function*), 27
 - SCons.CacheDir.CacheRetrieveFunc (*function*), 27
 - SCons.CacheDir.CacheRetrieveString (*function*), 27
- SCons.compat (*package*), 357–358
 - SCons.compat._scons_hashlib (*module*), 361–362
 - SCons.compat._scons_itertools (*module*), 363–364
 - SCons.compat._scons_optparse (*module*), 365–385
 - SCons.compat._scons_sets (*module*), 386–397
 - SCons.compat._scons_sets15 (*module*), 398–399
 - SCons.compat._scons_shlex (*module*), 400
 - SCons.compat._scons_subprocess (*module*), 401–411
 - SCons.compat._scons_textwrap (*module*), 412–413
 - SCons.compat._scons_UserString (*module*), 359–360

- SCons.compat.builtins (*module*), 414–415
- SCons.Conftest (*module*), 29–30
- SCons.cpp (*module*), 416–422
- SCons.dblite (*module*), 423
 - SCons.dblite.corruption_warning (*function*), 423
 - SCons.dblite.dblite (*class*), 423
 - SCons.dblite.is_string (*function*), 423
 - SCons.dblite.open (*function*), 423
 - SCons.dblite.unicode (*function*), 423
- SCons.Debug (*module*), 31–32
 - SCons.Debug.caller_stack (*function*), 31
 - SCons.Debug.caller_trace (*function*), 31
 - SCons.Debug.countLoggedInstances (*function*), 31
 - SCons.Debug.dump_caller_counts (*function*), 31
 - SCons.Debug.dumpLoggedInstances (*function*), 31
 - SCons.Debug.fetchLoggedInstances (*function*), 31
 - SCons.Debug.func_shorten (*function*), 31
 - SCons.Debug.listLoggedInstances (*function*), 31
 - SCons.Debug.logInstanceCreation (*function*), 31
 - SCons.Debug.memory (*function*), 31
 - SCons.Debug.string_to_classes (*function*), 31
 - SCons.Debug.Trace (*function*), 31
- SCons.Defaults (*module*), 33–35
- SCons.Environment (*module*), 36–63
 - SCons.Environment.alias_builder (*function*), 36
 - SCons.Environment.apply_tools (*function*), 36
 - SCons.Environment.Base (*class*), 41–48, 56–63
 - SCons.Environment.build_source (*function*), 36
 - SCons.Environment.BuilderDict (*class*), 38–39
 - SCons.Environment.BuilderWrapper (*class*), 37–38
 - SCons.Environment.copy_non_reserved_keywords (*function*), 36
 - SCons.Environment.default_copy_from_cache (*function*), 36
 - SCons.Environment.default_decide_source (*function*), 36
 - SCons.Environment.default_decide_target (*function*), 36
 - SCons.Environment.is_valid_construction_var (*function*), 36
 - SCons.Environment.MethodWrapper (*class*), 36–37
 - SCons.Environment.NoSubstitutionProxy (*function*), 36
 - SCons.Environment.OverrideEnvironment (*class*), 48–56
 - SCons.Environment.SubstitutionEnvironment (*class*), 39–41
- SCons.Errors (*module*), 64–75
- SCons.Executor (*module*), 76–80
 - SCons.Executor.Executor (*class*), 76–78
 - SCons.Executor.Null (*class*), 78–80
- SCons.exitfuncs (*module*), 424
 - SCons.exitfuncs.register (*function*), 424
- SCons.Job (*module*), 81–85
 - SCons.Job.InterruptState (*class*), 81
 - SCons.Job.Jobs (*class*), 81–82
 - SCons.Job.Parallel (*class*), 84–85
 - SCons.Job.Serial (*class*), 82
 - SCons.Job.ThreadPool (*class*), 84
 - SCons.Job.Worker (*class*), 82–84
- SCons.Memoize (*module*), 86–91
- SCons.Node (*package*), 92–103
 - SCons.Node.Alias (*module*), 104–114
 - SCons.Node.FS (*module*), 115–178
 - SCons.Node.Python (*module*), 179–188
- SCons.PathList (*module*), 189
 - SCons.PathList.node_conv (*function*), 189
 - SCons.PathList.PathList (*function*), 189
- SCons.Scanner (*module*)
 - SCons.Scanner.Base (*class*), 210–212
 - SCons.Scanner.Classic (*class*), 216–217
 - SCons.Scanner.ClassicCPP (*class*), 217–218
 - SCons.Scanner.Current (*class*), 214–216
 - SCons.Scanner.FindPathDirs (*class*), 210
 - SCons.Scanner.Scanner (*function*), 210
 - SCons.Scanner.Selector (*class*), 212–214
- SCons.Scanner (*package*), 210–218
 - SCons.Scanner.C (*module*), 219–222
 - SCons.Scanner.D (*module*), 223–224
 - SCons.Scanner.Dir (*module*), 225
 - SCons.Scanner.Fortran (*module*), 226–227
 - SCons.Scanner.IDL (*module*), 228
 - SCons.Scanner.LaTeX (*module*), 229–230
 - SCons.Scanner.Prog (*module*), 231
- SCons.SConf (*module*), 190–204
 - SCons.SConf.CheckCHheader (*function*), 190
 - SCons.SConf.CheckContext (*class*), 203–204
 - SCons.SConf.CheckCXXHeader (*function*), 190
 - SCons.SConf.CheckDeclaration (*function*), 190
 - SCons.SConf.CheckFunc (*function*), 190
 - SCons.SConf.CheckHeader (*function*), 190

- SCons.SConf.CheckLib (*function*), 190
- SCons.SConf.CheckLibWithHeader (*function*), 190
- SCons.SConf.CheckType (*function*), 190
- SCons.SConf.CheckTypeSize (*function*), 190
- SCons.SConf.ConfigureCacheError (*class*), 196–198
- SCons.SConf.ConfigureDryRunError (*class*), 195–196
- SCons.SConf.CreateConfigHBuilder (*function*), 190
- SCons.SConf.createIncludesFromHeaders (*function*), 190
- SCons.SConf.SConf (*function*), 190
- SCons.SConf.SConfBase (*class*), 202–203
- SCons.SConf.SConfBuildInfo (*class*), 198–199
- SCons.SConf.SConfBuildTask (*class*), 199–202
- SCons.SConf.SConfError (*class*), 193–195
- SCons.SConf.SConfWarning (*class*), 191–193
- SCons.SConf.SetBuildType (*function*), 190
- SCons.SConf.SetCacheMode (*function*), 190
- SCons.SConf.SetProgressDisplay (*function*), 190
- SCons.SConf.Streamer (*class*), 199
- SCons.SConsign (*module*), 205–209
 - SCons.SConsign.Base (*class*), 206
 - SCons.SConsign.corrupt_dblite_warning (*function*), 205
 - SCons.SConsign.DB (*class*), 206–209
 - SCons.SConsign.Dir (*class*), 207–208
 - SCons.SConsign.DirFile (*class*), 208
 - SCons.SConsign.File (*function*), 205
 - SCons.SConsign.GetDataBase (*function*), 205
 - SCons.SConsign.Reset (*function*), 205
 - SCons.SConsign.SConsignEntry (*class*), 205–206
 - SCons.SConsign.write (*function*), 205
- SCons.Script (*module*)
 - SCons.Script.HelpFunction (*function*), 232
 - SCons.Script.Options (*function*), 232
 - SCons.Script.TargetList (*class*), 238–239
 - SCons.Script.Variables (*function*), 232
- SCons.Script (*package*), 232–239
 - SCons.Script.Interactive (*module*), 240–243
 - SCons.Script.Main (*module*), 244–256
 - SCons.Script.SConscript' (*module*), 257–267
- SCons.Sig (*module*), 268–269
 - SCons.Sig.MD5Null (*class*), 268–269
 - SCons.Sig.TimeStampNull (*class*), 269
- SCons.Subst (*module*), 270–277
 - SCons.Subst.CmdStringHolder (*class*), 272–275
 - SCons.Subst.escape_list (*function*), 270
 - SCons.Subst.Literal (*class*), 271
 - SCons.Subst.NLWrapper (*class*), 275
 - SCons.Subst.quote_spaces (*function*), 270
 - SCons.Subst.raise_exception (*function*), 270
 - SCons.Subst.scons_subst (*function*), 270
 - SCons.Subst.scons_subst_list (*function*), 270
 - SCons.Subst.scons_subst_once (*function*), 270
 - SCons.Subst.SetAllowableExceptions (*function*), 270
 - SCons.Subst.SpecialAttrWrapper (*class*), 271–272
 - SCons.Subst.subst_dict (*function*), 270
 - SCons.Subst.Target_or_Source (*class*), 277
 - SCons.Subst.Targets_or_Sources (*class*), 275–277
- SCons.Taskmaster (*module*), 278–282
 - SCons.Taskmaster.dump_stats (*function*), 278
 - SCons.Taskmaster.find_cycle (*function*), 278
 - SCons.Taskmaster.Stats (*class*), 279
 - SCons.Taskmaster.Task (*class*), 279–281
 - SCons.Taskmaster.Taskmaster (*class*), 281–282
- SCons.Util (*module*), 283–309
- SCons.Variables (*package*), 310–312
 - SCons.Variables.BoolVariable' (*module*), 313
 - SCons.Variables.EnumVariable' (*module*), 314
 - SCons.Variables.ListVariable' (*module*), 315
 - SCons.Variables.PackageVariable' (*module*), 316
 - SCons.Variables.PathVariable' (*module*), 317–318
 - SCons.Variables.Variables (*class*), 310–312
- SCons.Warnings (*module*), 319–356
- SCons.Action._ActionAction (*class*)
 - SCons.Action._ActionAction.__call__ (*method*), 7, 10
 - SCons.Action._ActionAction.print_cmd_line (*method*), 8, 10, 11
- SCons.Action.Action (*function*), 6
- SCons.Action.ActionBase (*class*), 6
 - SCons.Action.ActionBase.__add__ (*method*), 6–10, 12
 - SCons.Action.ActionBase.__cmp__ (*method*), 6–9, 11, 12
 - SCons.Action.ActionBase.__radd__ (*method*), 6–9, 11, 12
 - SCons.Action.ActionBase.genstring (*method*), 6, 7, 11
 - SCons.Action.ActionBase.get_executor (*method*), 6–9, 11, 12

- SCons.Action.ActionBase.presub_lines (*method*), 6, 8, 10, 11
- SCons.Action.ActionCaller (*class*), 12
 - SCons.Action.ActionCaller.__call__ (*method*), 12
 - SCons.Action.ActionCaller.__init__ (*method*), 12
 - SCons.Action.ActionCaller.__str__ (*method*), 12
 - SCons.Action.ActionCaller.get_contents (*method*), 12
 - SCons.Action.ActionCaller.strfunction (*method*), 12
 - SCons.Action.ActionCaller.subst (*method*), 12
 - SCons.Action.ActionCaller.subst_args (*method*), 12
 - SCons.Action.ActionCaller.subst_kw (*method*), 12
- SCons.Action.ActionFactory (*class*), 12–13
 - SCons.Action.ActionFactory.__call__ (*method*), 13
 - SCons.Action.ActionFactory.__init__ (*method*), 13
- SCons.Action.CommandAction (*class*), 6–8
 - SCons.Action.CommandAction.__str__ (*method*), 7
 - SCons.Action.CommandAction.execute (*method*), 7, 9
 - SCons.Action.CommandAction.get_contents (*method*), 7
 - SCons.Action.CommandAction.get_implicit_deps (*method*), 7
 - SCons.Action.CommandAction.process (*method*), 7, 10
 - SCons.Action.CommandAction.strfunction (*method*), 7, 10
- SCons.Action.CommandGeneratorAction (*class*), 8
 - SCons.Action.CommandGeneratorAction.__call__ (*method*), 8
 - SCons.Action.CommandGeneratorAction.__init__ (*method*), 8
 - SCons.Action.CommandGeneratorAction.__str__ (*method*), 8, 9
 - SCons.Action.CommandGeneratorAction.get_contents (*method*), 8
 - SCons.Action.CommandGeneratorAction.get_implicit_deps (*method*), 8, 10
- SCons.Action.default_exitstatfunc (*function*), 5
- SCons.Action.FunctionAction (*class*), 10–11
 - SCons.Action.FunctionAction.__str__ (*method*), 10
 - SCons.Action.FunctionAction.execute (*method*), 10
 - SCons.Action.FunctionAction.function_name (*method*), 10
 - SCons.Action.FunctionAction.get_contents (*method*), 10
 - SCons.Action.FunctionAction.get_implicit_deps (*method*), 10
 - SCons.Action.FunctionAction.strfunction (*method*), 10
- SCons.Action.LazyAction (*class*), 8–10
 - SCons.Action.LazyAction.get_parent_class (*method*), 9
- SCons.Action.ListAction (*class*), 11–12
 - SCons.Action.ListAction.__call__ (*method*), 11
 - SCons.Action.ListAction.__init__ (*method*), 11
 - SCons.Action.ListAction.__str__ (*method*), 11
 - SCons.Action.ListAction.get_contents (*method*), 11
 - SCons.Action.ListAction.get_implicit_deps (*method*), 11
- SCons.Action.remove_set_lineno_codes (*function*), 6
- SCons.Action.rfile (*function*), 5
- SCons.compat.import_as (*function*), 358
- SCons.Conftest.CheckBuilder (*function*), 29
- SCons.Conftest.CheckDeclaration (*function*), 30
- SCons.Conftest.CheckFunc (*function*), 29
- SCons.Conftest.CheckHeader (*function*), 29
- SCons.Conftest.CheckLib (*function*), 30
- SCons.Conftest.CheckType (*function*), 29
- SCons.Conftest.CheckTypeSize (*function*), 29
- SCons.cpp.CPP_to_Python (*function*), 416
- SCons.cpp.CPP_to_Python_Ops_Sub (*function*), 416
- SCons.cpp.DumbPreProcessor (*class*), 419–422
 - SCons.cpp.FunctionEvaluator (*class*), 416–417
 - SCons.cpp.FunctionEvaluator.__call__ (*method*), 416
 - SCons.cpp.FunctionEvaluator.__init__ (*method*), 416
- SCons.cpp.PreProcessor (*class*), 417–419
 - SCons.cpp.PreProcessor.__call__ (*method*), 219, 417, 419
 - SCons.cpp.PreProcessor.__init__ (*method*), 417
 - SCons.cpp.PreProcessor.all_include (*method*), 220, 417, 420
 - SCons.cpp.PreProcessor.do_define (*method*), 220, 417, 420
 - SCons.cpp.PreProcessor.do_elif (*method*), 220, 417, 420
 - SCons.cpp.PreProcessor.do_else (*method*), 220, 417, 420
 - SCons.cpp.PreProcessor.do_endif (*method*), 220, 417, 420
 - SCons.cpp.PreProcessor.do_if (*method*), 220, 417, 420
 - SCons.cpp.PreProcessor.do_ifdef (*method*), 220, 417, 420

- SCons.cpp.PreProcessor.do_ifndef (*method*), 220
 417, 420
 SCons.cpp.PreProcessor.do_import (*method*),
 220, 417, 420
 SCons.cpp.PreProcessor.do_include (*method*),
 220, 418, 420
 SCons.cpp.PreProcessor.do_nothing (*method*),
 220, 418, 420
 SCons.cpp.PreProcessor.do_undef (*method*), 220
 418, 420
 SCons.cpp.PreProcessor.eval_expression (*method*),
 221, 418, 421
 SCons.cpp.PreProcessor.finalize_result (*method*),
 418, 421
 SCons.cpp.PreProcessor.find_include_file (*method*),
 418, 421
 SCons.cpp.PreProcessor.initialize_result (*method*),
 418, 421
 SCons.cpp.PreProcessor.process_contents (*method*),
 221, 418, 421
 SCons.cpp.PreProcessor.read_file (*method*), 418,
 421
 SCons.cpp.PreProcessor.resolve_include (*method*),
 221, 418, 421
 SCons.cpp.PreProcessor.restore (*method*), 221,
 418, 421
 SCons.cpp.PreProcessor.save (*method*), 221, 419,
 421
 SCons.cpp.PreProcessor.scons_current_file (*method*),
 221, 419, 421
 SCons.cpp.PreProcessor.start_handling_includes
 (*method*), 221, 419, 421
 SCons.cpp.PreProcessor.stop_handling_includes
 (*method*), 221, 419, 421
 SCons.cpp.PreProcessor.tupleize (*method*), 221,
 419, 422
 SCons.Defaults.chmod_func (*function*), 33
 SCons.Defaults.chmod_strfunc (*function*), 33
 SCons.Defaults.copy_func (*function*), 33
 SCons.Defaults.DefaultEnvironment (*function*), 33
 SCons.Defaults.delete_func (*function*), 33
 SCons.Defaults.delete_strfunc (*function*), 33
 SCons.Defaults.get_paths_str (*function*), 33
 SCons.Defaults.mkdir_func (*function*), 33
 SCons.Defaults.NullCmdGenerator (*class*), 34–35
 SCons.Defaults.NullCmdGenerator.__call__ (*method*),
 35
 SCons.Defaults.NullCmdGenerator.__init__ (*method*),
 35
 SCons.Defaults.SharedFlagChecker (*function*), 33
 SCons.Defaults.SharedObjectEmitter (*function*), 33
 SCons.Defaults.StaticObjectEmitter (*function*), 33
 SCons.Defaults.touch_func (*function*), 33
 SCons.Defaults.Variable_Method_Caller (*class*), 35
 SCons.Defaults.Variable_Method_Caller.__call__
 (*method*), 35
 SCons.Defaults.Variable_Method_Caller.__init__
 (*method*), 35
 SCons.Errors.BuildError (*class*), 64–66
 SCons.Errors.EnvironmentError (*class*), 71–72
 SCons.Errors.ExplicitExit (*class*), 72–74
 SCons.Errors.InternalError (*class*), 66–67
 SCons.Errors.StopError (*class*), 69–71
 SCons.Errors.TaskmasterException (*class*), 74–75
 SCons.Errors.UserError (*class*), 67–69
 SCons.Memoize.CountDict (*class*), 89
 SCons.Memoize.CountDict.__call__ (*method*), 89
 SCons.Memoize.Counter (*class*), 88
 SCons.Memoize.Counter.__cmp__ (*method*), 88,
 89
 SCons.Memoize.Counter.__init__ (*method*), 88
 SCons.Memoize.Counter.display (*method*), 88,
 89
 SCons.Memoize.CountValue (*class*), 88–89
 SCons.Memoize.CountValue.__call__ (*method*),
 88
 SCons.Memoize.Dump (*function*), 87
 SCons.Memoize.EnableMemoization (*function*), 87
 SCons.Memoize.Memoized_Metaclass (*class*), 89–
 91
 SCons.Memoize.Memoizer (*class*), 89
 SCons.Memoize.Memoizer.__init__ (*method*), 89
 SCons.Node.Annotate (*function*), 92
 SCons.Node.BuildInfoBase (*class*), 93–94
 SCons.Node.BuildInfoBase.__init__ (*method*), 94,
 106, 142, 165, 180, 198
 SCons.Node.BuildInfoBase.merge (*method*), 94,
 106, 142, 166, 180, 199
 SCons.Node.classname (*function*), 92
 SCons.Node.do_nothing (*function*), 92
 SCons.Node.get_children (*function*), 92
 SCons.Node.ignore_cycle (*function*), 92
 SCons.Node.Node (*class*), 94–101
 SCons.Node.Node.__init__ (*method*), 94
 SCons.Node.Node.add_dependency (*method*),
 94, 108, 120, 129, 146, 155, 171, 181
 SCons.Node.Node.add_ignore (*method*), 94, 108,
 120, 129, 146, 155, 171, 182
 SCons.Node.Node.add_prerequisite (*method*),
 94, 108, 121, 129, 146, 155, 171, 182
 SCons.Node.Node.add_source (*method*), 94, 108,
 121, 129, 146, 155, 171, 182
 SCons.Node.Node.add_to_implicit (*method*), 94,
 108, 121, 129, 146, 155, 171, 182
 SCons.Node.Node.add_to_waiting_parents (*method*),
 94, 108, 121, 129, 147, 155, 171, 182

- SCons.Node.Node.add_to_waiting_set (*method*), 94, 108, 121, 129, 147, 155, 171, 182
- SCons.Node.Node.add_wkid (*method*), 94, 108, 121, 129, 147, 155, 171, 182
- SCons.Node.Node.all_children (*method*), 95, 108, 121, 129, 147, 155, 171, 182
- SCons.Node.Node.alter_targets (*method*), 95, 108, 121, 130, 182
- SCons.Node.Node.build (*method*), 95, 121, 130, 171
- SCons.Node.Node.builder_set (*method*), 95, 108, 121, 130, 147, 156, 182
- SCons.Node.Node.built (*method*), 95, 108, 121, 130, 147, 156, 182
- SCons.Node.Node.changed (*method*), 95, 108, 121, 130, 147, 156, 171, 182
- SCons.Node.Node.changed_since_last_build (*method*), 95, 122
- SCons.Node.Node.children (*method*), 95, 109, 122, 130, 147, 156, 172, 183
- SCons.Node.Node.children_are_up_to_date (*method*), 95, 109, 122, 130, 147, 156, 172, 183
- SCons.Node.Node.clear (*method*), 96, 109, 122, 130, 147, 156, 172, 183
- SCons.Node.Node.clear_memoized_values (*method*), 96, 109, 122, 130, 147, 157, 172, 183
- SCons.Node.Node.Decider (*method*), 94, 108, 120, 128, 146, 154, 170, 181
- SCons.Node.Node.del_binfo (*method*), 96, 109, 122, 130, 147, 157, 172, 183
- SCons.Node.Node.disambiguate (*method*), 96, 109, 122, 148, 157, 172, 183
- SCons.Node.Node.do_not_store_info (*method*), 96, 109, 122, 130, 148, 157, 172, 183
- SCons.Node.Node.env_set (*method*), 96, 109, 122, 131, 148, 157, 172, 183
- SCons.Node.Node.executor_cleanup (*method*), 96, 109, 122, 131, 148, 157, 172, 183
- SCons.Node.Node.exists (*method*), 96, 109, 183
- SCons.Node.Node.explain (*method*), 96, 109, 123, 131, 148, 157, 172, 183
- SCons.Node.Node.for_signature (*method*), 96, 109, 183
- SCons.Node.Node.get_abspath (*method*), 96, 110, 184
- SCons.Node.Node.get_binfo (*method*), 96, 110, 123, 131, 148, 158, 173, 184
- SCons.Node.Node.get_build_env (*method*), 97, 110, 123, 131, 148, 158, 173, 184
- SCons.Node.Node.get_build_scanner_path (*method*), 97, 110, 123, 131, 148, 158, 173, 184
- SCons.Node.Node.get_builder (*method*), 97, 110, 123, 131, 148, 158, 173, 184
- SCons.Node.Node.get_cachedir_csig (*method*), 97, 110, 123, 131, 149, 158, 184
- SCons.Node.Node.get_csig (*method*), 97, 123, 131, 149, 158
- SCons.Node.Node.get_env (*method*), 97, 110, 123, 131, 149, 158, 173, 184
- SCons.Node.Node.get_env_scanner (*method*), 97, 110, 123, 132, 173, 184
- SCons.Node.Node.get_executor (*method*), 97, 110, 123, 132, 149, 158, 173, 184
- SCons.Node.Node.get_found_includes (*method*), 97, 110, 123, 132, 184
- SCons.Node.Node.get_implicit_deps (*method*), 97, 110, 123, 132, 149, 158, 173, 184
- SCons.Node.Node.get_ninfo (*method*), 97, 111, 123, 132, 149, 159, 173, 185
- SCons.Node.Node.get_source_scanner (*method*), 97, 111, 124, 132, 149, 159, 174, 185
- SCons.Node.Node.get_state (*method*), 98, 111, 124, 132, 149, 159, 174, 185
- SCons.Node.Node.get_stored_implicit (*method*), 98, 111, 124, 132, 149, 159, 185
- SCons.Node.Node.get_stored_info (*method*), 98, 111, 124, 132, 149, 159, 185
- SCons.Node.Node.get_string (*method*), 98, 111, 124, 132, 149, 159, 174, 185
- SCons.Node.Node.get_subst_proxy (*method*), 98, 111, 185
- SCons.Node.Node.get_suffix (*method*), 98, 111, 185
- SCons.Node.Node.get_target_scanner (*method*), 98, 111, 124, 133, 174, 185
- SCons.Node.Node.has_builder (*method*), 98, 99, 111, 112, 124, 125, 133, 134, 150, 160, 174, 175, 185, 186
- SCons.Node.Node.has_explicit_builder (*method*), 98, 112, 124, 133, 150, 160, 175, 186
- SCons.Node.Node.is_derived (*method*), 99, 112, 124, 133, 150, 160, 175, 186
- SCons.Node.Node.is_literal (*method*), 99, 112, 125, 133, 150, 161, 175, 186
- SCons.Node.Node.is_up_to_date (*method*), 99, 125, 134
- SCons.Node.Node.make_ready (*method*), 99, 125, 134, 151, 161
- SCons.Node.Node.missing (*method*), 99, 112, 125, 134, 151, 161, 175, 186
- SCons.Node.Node.new_binfo (*method*), 99, 112, 125, 134, 151, 161, 175, 186
- SCons.Node.Node.new_ninfo (*method*), 99, 112, 125, 151, 161, 175, 186
- SCons.Node.Node.postprocess (*method*), 99, 112, 125, 134, 151, 161, 176, 186

- SCons.Node.Node.prepare (*method*), 99, 112, 125, 134, 186
- SCons.Node.Node.remove (*method*), 100, 113, 126, 135, 151, 162, 187
- SCons.Node.Node.render_include_tree (*method*), 100, 113, 126, 135, 151, 162, 176, 187
- SCons.Node.Node.reset_executor (*method*), 100, 113, 126, 135, 151, 162, 176, 187
- SCons.Node.Node.retrieve_from_cache (*method*), 100, 113, 126, 135, 151, 162, 187
- SCons.Node.Node.rexists (*method*), 100, 113, 187
- SCons.Node.Node.scan (*method*), 100, 113, 126, 135, 152, 162, 176, 187
- SCons.Node.Node.scanner_key (*method*), 100, 113, 126, 187
- SCons.Node.Node.select_scanner (*method*), 100, 113, 126, 135, 152, 163, 176, 187
- SCons.Node.Node.set_always_build (*method*), 100, 113, 126, 136, 152, 163, 176, 187
- SCons.Node.Node.set_executor (*method*), 101, 114, 126, 136, 152, 163, 176, 188
- SCons.Node.Node.set_explicit (*method*), 101, 114, 127, 136, 152, 163, 176, 188
- SCons.Node.Node.set_nocache (*method*), 101, 114, 127, 136, 152, 163, 176, 188
- SCons.Node.Node.set_noclean (*method*), 101, 114, 127, 136, 152, 163, 176, 188
- SCons.Node.Node.set_precious (*method*), 101, 114, 127, 136, 152, 163, 176, 188
- SCons.Node.Node.set_specific_source (*method*), 101, 114, 127, 136, 152, 163, 177, 188
- SCons.Node.Node.set_state (*method*), 101, 114, 127, 136, 152, 163, 177, 188
- SCons.Node.Node.state_has_changed (*method*), 101, 114, 127, 136, 153, 164, 177, 188
- SCons.Node.Node.store_info (*method*), 101, 114, 127, 137, 153, 164, 188
- SCons.Node.Node.visited (*method*), 101, 114, 127, 137, 153, 164, 188
- SCons.Node.NodeInfoBase (*class*), 93
 - SCons.Node.NodeInfoBase.__init__ (*method*), 93, 105, 141, 165, 179
 - SCons.Node.NodeInfoBase.convert (*method*), 93, 105, 141, 165, 179
 - SCons.Node.NodeInfoBase.format (*method*), 93, 105, 141, 165, 179
 - SCons.Node.NodeInfoBase.merge (*method*), 93, 105, 141, 165, 179
 - SCons.Node.NodeInfoBase.update (*method*), 93, 105, 141, 165, 179
- SCons.Node.NodeList (*class*), 101–103
 - SCons.Node.NodeList.__str__ (*method*), 102
- SCons.Node.Walker (*class*), 103
 - SCons.Node.Walker.__init__ (*method*), 103
 - SCons.Node.Walker.is_done (*method*), 103
 - SCons.Node.Walker.next (*method*), 103
- SCons.Scanner.Dir.DirEntryScanner (*function*), 225
- SCons.Scanner.Dir.DirScanner (*function*), 225
- SCons.Scanner.Dir.do_not_scan (*function*), 225
- SCons.Scanner.Dir.only_dirs (*function*), 225
- SCons.Scanner.Dir.scan_in_memory (*function*), 225
- SCons.Scanner.Dir.scan_on_disk (*function*), 225
- SCons.Script.Interactive.interact (*function*), 240
- SCons.Script.Interactive.SConsInteractiveCmd (*class*), 240–243
 - SCons.Script.Interactive.SConsInteractiveCmd.do_build (*method*), 241
 - SCons.Script.Interactive.SConsInteractiveCmd.do_clean (*method*), 241
 - SCons.Script.Interactive.SConsInteractiveCmd.do_EOF (*method*), 241
 - SCons.Script.Interactive.SConsInteractiveCmd.do_exit (*method*), 241
 - SCons.Script.Interactive.SConsInteractiveCmd.do_shell (*method*), 241
 - SCons.Script.Interactive.SConsInteractiveCmd.do_version (*method*), 241
- SCons.Util.NoError (*class*), 300–302
- SCons.Util.AddMethod (*function*), 286
- SCons.Util.adjustixes (*function*), 286
- SCons.Util.AppendPath (*function*), 285
- SCons.Util.CallableComposite (*class*), 288–289
 - SCons.Util.CallableComposite.__call__ (*method*), 288
- SCons.Util.case_sensitive_suffixes (*function*), 286
- SCons.Util.CLVar (*class*), 302–304
 - SCons.Util.CLVar.__coerce__ (*method*), 302
 - SCons.Util.CLVar.__str__ (*method*), 302
- SCons.Util.containsAll (*function*), 283
- SCons.Util.containsAny (*function*), 283
- SCons.Util.containsOnly (*function*), 283
- SCons.Util.dictify (*function*), 283
- SCons.Util.DisplayEngine (*class*), 291
 - SCons.Util.DisplayEngine.__init__ (*method*), 291
 - SCons.Util.DisplayEngine.dont_print (*method*), 291
 - SCons.Util.DisplayEngine.print_it (*method*), 291
 - SCons.Util.DisplayEngine.set_mode (*method*), 291
- SCons.Util.do_flatten (*function*), 284
- SCons.Util.flatten (*function*), 284
- SCons.Util.flatten_sequence (*function*), 284
- SCons.Util.get_environment_var (*function*), 283
- SCons.Util.get_native_path (*function*), 286
- SCons.Util.IDX (*function*), 283

- SCons.Util.is_Dict (*function*), 284
- SCons.Util.is_List (*function*), 284
- SCons.Util.is_Scalar (*function*), 284
- SCons.Util.is_Sequence (*function*), 284
- SCons.Util.is_String (*function*), 284
- SCons.Util.is_Tuple (*function*), 284
- SCons.Util.LogicalLines (*class*), 306–307
 - SCons.Util.LogicalLines.__init__ (*method*), 307
 - SCons.Util.LogicalLines.readline (*method*), 307
 - SCons.Util.LogicalLines.readlines (*method*), 307
- SCons.Util.make_path_relative (*function*), 286
- SCons.Util.MD5collect (*function*), 287
- SCons.Util.MD5signature (*function*), 287
- SCons.Util.mystr (*class*), 291–300
- SCons.Util.NodeList (*class*), 289–291
 - SCons.Util.NodeList.__getattr__ (*method*), 290
 - SCons.Util.NodeList.__nonzero__ (*method*), 290
 - SCons.Util.NodeList.__str__ (*method*), 290
- SCons.Util.Null (*class*), 309
 - SCons.Util.Null.__call__ (*method*), 268, 269, 309
 - SCons.Util.Null.__delattr__ (*method*), 268, 269, 309
 - SCons.Util.Null.__getattr__ (*method*), 268, 269, 309
 - SCons.Util.Null.__init__ (*method*), 268, 269, 309
 - SCons.Util.Null.__new__ (*method*), 268, 269, 309
 - SCons.Util.Null.__nonzero__ (*method*), 268, 269, 309
 - SCons.Util.Null.__repr__ (*method*), 309
 - SCons.Util.Null.__setattr__ (*method*), 268, 269, 309
- SCons.Util.OrderedDict (*class*), 304–305
- SCons.Util.PrependPath (*function*), 285
- SCons.Util.print_tree (*function*), 283
- SCons.Util.Proxy (*class*), 300
 - SCons.Util.Proxy.__cmp__ (*method*), 25, 117, 300
 - SCons.Util.Proxy.__getattr__ (*method*), 25, 300
 - SCons.Util.Proxy.__init__ (*method*), 117, 300
 - SCons.Util.Proxy.get (*method*), 25, 117, 300
- SCons.Util.RegGetValue (*function*), 285
- SCons.Util.RenameFunction (*function*), 287
- SCons.Util.render_tree (*function*), 283
- SCons.Util.Selector (*class*), 305–306
 - SCons.Util.Selector.__call__ (*method*), 305
- SCons.Util.semi_deepcopy (*function*), 285
- SCons.Util.Split (*function*), 286
- SCons.Util.splitext (*function*), 283
- SCons.Util.to_String (*function*), 285
- SCons.Util.to_String_for_signature (*function*), 285
- SCons.Util.to_String_for_subst (*function*), 285
- SCons.Util.Unbuffered (*class*), 309
 - SCons.Util.Unbuffered.__getattr__ (*method*), 309
 - SCons.Util.Unbuffered.__init__ (*method*), 309
 - SCons.Util.Unbuffered.write (*method*), 309
- SCons.Util.unique (*function*), 286
- SCons.Util.UniqueList (*class*), 307–309
- SCons.Util.uniquer (*function*), 286
- SCons.Util.uniquer_hashables (*function*), 286
- SCons.Util.updrive (*function*), 283
- SCons.Util.WhereIs (*function*), 285
- SCons.Warnings.CacheWriteErrorWarning (*class*), 322–323
- SCons.Warnings.CorruptSConsignWarning (*class*), 323–325
- SCons.Warnings.DependencyWarning (*class*), 325–327
- SCons.Warnings.DeprecatedCopyWarning (*class*), 329–331
- SCons.Warnings.DeprecatedSourceSignaturesWarning (*class*), 331–333
- SCons.Warnings.DeprecatedTargetSignaturesWarning (*class*), 333–335
- SCons.Warnings.DeprecatedWarning (*class*), 327–329
- SCons.Warnings.DuplicateEnvironmentWarning (*class*), 335–336
- SCons.Warnings.enableWarningClass (*function*), 319
- SCons.Warnings.FortranCxxMixWarning (*class*), 354–356
- SCons.Warnings.LinkWarning (*class*), 336–338
- SCons.Warnings.MisleadingKeywordsWarning (*class*), 338–340
- SCons.Warnings.MissingSConscriptWarning (*class*), 340–342
- SCons.Warnings.NoMD5ModuleWarning (*class*), 342–343
- SCons.Warnings.NoMetaclassSupportWarning (*class*), 343–345
- SCons.Warnings.NoObjectCountWarning (*class*), 345–347
- SCons.Warnings.NoParallelSupportWarning (*class*), 347–349
- SCons.Warnings.process_warn_strings (*function*), 319
- SCons.Warnings.PythonVersionWarning (*class*), 349–351
- SCons.Warnings.ReservedVariableWarning (*class*), 351–352
- SCons.Warnings.StackSizeWarning (*class*), 352–354
- SCons.Warnings.suppressWarningClass (*function*), 319
- SCons.Warnings.warn (*function*), 319
- SCons.Warnings.Warning (*class*), 320–322
- SCons.Warnings.warningAsException (*function*), 319
- str.__add__ (*function*), 292
- str.__contains__ (*function*), 292

- `str.__eq__` (*function*), 292
- `str.__ge__` (*function*), 292
- `str.__getitem__` (*function*), 292
- `str.__getnewargs__` (*function*), 292
- `str.__getslice__` (*function*), 292
- `str.__gt__` (*function*), 292
- `str.__le__` (*function*), 293
- `str.__len__` (*function*), 293
- `str.__lt__` (*function*), 293
- `str.__mod__` (*function*), 293
- `str.__mul__` (*function*), 293
- `str.__ne__` (*function*), 293
- `str.__rmod__` (*function*), 294
- `str.__rmul__` (*function*), 294
- `str.capitalize` (*function*), 294
- `str.center` (*function*), 294
- `str.count` (*function*), 294
- `str.decode` (*function*), 294
- `str.encode` (*function*), 294
- `str.endswith` (*function*), 295
- `str.expandtabs` (*function*), 295
- `str.find` (*function*), 295
- `str.index` (*function*), 295
- `str.isalnum` (*function*), 295
- `str.isalpha` (*function*), 295
- `str.isdigit` (*function*), 296
- `str.islower` (*function*), 296
- `str.isspace` (*function*), 296
- `str.istitle` (*function*), 296
- `str.isupper` (*function*), 296
- `str.join` (*function*), 296
- `str.ljust` (*function*), 296
- `str.lower` (*function*), 297
- `str.lstrip` (*function*), 297
- `str.partition` (*function*), 297
- `str.replace` (*function*), 297
- `str.rfind` (*function*), 297
- `str.rindex` (*function*), 297
- `str.rjust` (*function*), 297
- `str.rpartition` (*function*), 298
- `str.rsplit` (*function*), 298
- `str.rstrip` (*function*), 298
- `str.split` (*function*), 298
- `str.splitlines` (*function*), 298
- `str.startswith` (*function*), 298
- `str.strip` (*function*), 299
- `str.swapcase` (*function*), 299
- `str.title` (*function*), 299
- `str.translate` (*function*), 299
- `str.upper` (*function*), 299
- `str.zfill` (*function*), 299
- `threading.Thread.getName` (*function*), 83
- `threading.Thread.isAlive` (*function*), 83
- `threading.Thread.isDaemon` (*function*), 83
- `threading.Thread.join` (*function*), 83
- `threading.Thread.setDaemon` (*function*), 83
- `threading.Thread.setName` (*function*), 83
- `threading.Thread.start` (*function*), 83
- `type.__call__` (*function*), 90
- `type.__cmp__` (*function*), 90
- `type.__subclasses__` (*function*), 91
- `type.mro` (*function*), 91
- `UserDict.UserDict.__cmp__` (*function*), 16, 17, 19, 22, 38, 104, 304, 305
- `UserDict.UserDict.__contains__` (*function*), 16, 17, 19, 22, 38, 104, 305
- `UserDict.UserDict.__delitem__` (*function*), 22, 104
- `UserDict.UserDict.__getitem__` (*function*), 16, 17, 19, 22, 38, 104, 305
- `UserDict.UserDict.__init__` (*function*), 104
- `UserDict.UserDict.__len__` (*function*), 16, 17, 19, 22, 38, 104, 305, 306
- `UserDict.UserDict.__repr__` (*function*), 16, 18, 19, 22, 38, 104, 305, 306
- `UserDict.UserDict.__setitem__` (*function*), 22, 104
- `UserDict.UserDict.clear` (*function*), 22, 38, 104
- `UserDict.UserDict.copy` (*function*), 22, 38, 104
- `UserDict.UserDict.fromkeys` (*class method*), 16, 18, 19, 22, 38, 104, 305, 306
- `UserDict.UserDict.get` (*function*), 16, 18, 19, 22, 38, 105, 305, 306
- `UserDict.UserDict.has_key` (*function*), 16, 18, 19, 22, 39, 105, 305, 306
- `UserDict.UserDict.items` (*function*), 23, 39, 105
- `UserDict.UserDict.iteritems` (*function*), 16, 18, 20, 23, 39, 105, 305, 306
- `UserDict.UserDict.iterkeys` (*function*), 16, 18, 20, 23, 39, 105, 305, 306
- `UserDict.UserDict.itervalues` (*function*), 17, 18, 20, 23, 39, 105, 305, 306
- `UserDict.UserDict.keys` (*function*), 23, 39, 105
- `UserDict.UserDict.pop` (*function*), 17, 18, 20, 23, 39, 105, 305, 306
- `UserDict.UserDict.popitem` (*function*), 23, 39, 105
- `UserDict.UserDict.setdefault` (*function*), 23, 39, 105
- `UserDict.UserDict.update` (*function*), 23, 105
- `UserDict.UserDict.values` (*function*), 23, 39, 105
- `UserList.UserList.__add__` (*function*), 20, 102, 238, 276, 288, 290, 302
- `UserList.UserList.__cmp__` (*function*), 20, 102, 238, 276, 288, 290, 302
- `UserList.UserList.__contains__` (*function*), 20, 102, 238, 276, 288, 290, 302, 308

- UserList.UserList.__delitem__ (function), 20, 102, 238, 276, 288, 290, 302, 308
- UserList.UserList.__delslice__ (function), 20, 102, 238, 276, 288, 290, 303, 308
- UserList.UserList.__eq__ (function), 21, 102, 238, 276, 288, 290, 303
- UserList.UserList.__ge__ (function), 21, 102, 238, 276, 288, 290, 303
- UserList.UserList.__getitem__ (function), 21, 102, 238, 288, 290, 303
- UserList.UserList.__getslice__ (function), 21, 102, 238, 288, 290, 303
- UserList.UserList.__gt__ (function), 21, 102, 238, 276, 288, 290, 303
- UserList.UserList.__iadd__ (function), 21, 102, 238, 276, 288, 290, 303
- UserList.UserList.__imul__ (function), 21, 102, 238, 276, 288, 290, 303
- UserList.UserList.__init__ (function), 21, 102, 238, 288, 290
- UserList.UserList.__le__ (function), 21, 102, 238, 276, 288, 290, 303
- UserList.UserList.__len__ (function), 21, 102, 238, 276, 288, 290, 303
- UserList.UserList.__lt__ (function), 21, 102, 238, 276, 289, 290, 303
- UserList.UserList.__mul__ (function), 21, 102, 238, 239, 276, 289–291, 303
- UserList.UserList.__ne__ (function), 21, 102, 239, 276, 289, 290, 303
- UserList.UserList.__radd__ (function), 21, 102, 239, 276, 289, 290, 303
- UserList.UserList.__repr__ (function), 21, 102, 239, 289, 290, 303, 309
- UserList.UserList.__setitem__ (function), 21, 102, 239, 276, 289, 291, 303
- UserList.UserList.__setslice__ (function), 21, 102, 239, 276, 289, 291, 303
- UserList.UserList.append (function), 21, 103, 239, 276, 289, 291, 303
- UserList.UserList.count (function), 21, 103, 239, 276, 289, 291, 303
- UserList.UserList.extend (function), 21, 103, 239, 276, 289, 291, 303
- UserList.UserList.index (function), 21, 103, 239, 276, 289, 291, 303
- UserList.UserList.insert (function), 21, 103, 239, 277, 289, 291, 303
- UserList.UserList.pop (function), 21, 103, 239, 277, 289, 291, 303, 309
- UserList.UserList.remove (function), 22, 103, 239, 277, 289, 291, 304, 309
- UserList.UserList.reverse (function), 22, 103, 239, 277, 289, 291, 304
- UserList.UserList.sort (function), 22, 103, 239, 277, 289, 291, 304
- UserString.UserString.__add__ (function), 272
- UserString.UserString.__cmp__ (function), 272
- UserString.UserString.__complex__ (function), 272
- UserString.UserString.__contains__ (function), 272
- UserString.UserString.__float__ (function), 272
- UserString.UserString.__getitem__ (function), 272
- UserString.UserString.__getslice__ (function), 273
- UserString.UserString.__hash__ (function), 273
- UserString.UserString.__int__ (function), 273
- UserString.UserString.__len__ (function), 273
- UserString.UserString.__long__ (function), 273
- UserString.UserString.__mod__ (function), 273
- UserString.UserString.__mul__ (function), 273
- UserString.UserString.__radd__ (function), 273
- UserString.UserString.__repr__ (function), 273
- UserString.UserString.__str__ (function), 273
- UserString.UserString.capitalize (function), 273
- UserString.UserString.center (function), 273
- UserString.UserString.count (function), 273
- UserString.UserString.decode (function), 273
- UserString.UserString.encode (function), 273
- UserString.UserString.endswith (function), 273
- UserString.UserString.expandtabs (function), 273
- UserString.UserString.find (function), 273
- UserString.UserString.index (function), 273
- UserString.UserString.isalnum (function), 273
- UserString.UserString.isalpha (function), 273
- UserString.UserString.isdecimal (function), 273
- UserString.UserString.isdigit (function), 273
- UserString.UserString.islower (function), 274
- UserString.UserString.isnumeric (function), 274
- UserString.UserString.isspace (function), 274
- UserString.UserString.istitle (function), 274
- UserString.UserString.isupper (function), 274
- UserString.UserString.join (function), 274
- UserString.UserString.ljust (function), 274
- UserString.UserString.lower (function), 274
- UserString.UserString.lstrip (function), 274
- UserString.UserString.partition (function), 274
- UserString.UserString.replace (function), 274
- UserString.UserString.rfind (function), 274
- UserString.UserString.rindex (function), 274
- UserString.UserString.rjust (function), 274
- UserString.UserString.rpartition (function), 274
- UserString.UserString.rsplit (function), 274
- UserString.UserString.rstrip (function), 274
- UserString.UserString.split (function), 274
- UserString.UserString.splitlines (function), 274
- UserString.UserString.startswith (function), 274
- UserString.UserString.strip (function), 274

UserString.UserString.swapcase (*function*), 274
UserString.UserString.title (*function*), 274
UserString.UserString.translate (*function*), 274
UserString.UserString.upper (*function*), 275
UserString.UserString.zfill (*function*), 275