

SCons

API Documentation

October 10, 2008

Contents

Contents	1
1 Package SCons	2
1.1 Modules	2
1.2 Variables	5
2 Module SCons.Action	6
2.1 Functions	7
2.2 Variables	7
2.3 Class ActionBase	8
2.3.1 Methods	8
2.4 Class CommandAction	8
2.4.1 Methods	8
2.5 Class CommandGeneratorAction	9
2.5.1 Methods	10
2.6 Class LazyAction	10
2.6.1 Methods	11
2.7 Class FunctionAction	12
2.7.1 Methods	12
2.8 Class ListAction	13
2.8.1 Methods	13
2.9 Class ActionCaller	14
2.9.1 Methods	14
2.10 Class ActionFactory	14
2.10.1 Methods	14
3 Module SCons.Builder	15
3.1 Functions	16
3.2 Variables	16
3.3 Class DictCmdGenerator	16
3.3.1 Methods	17
3.4 Class CallableSelector	18
3.4.1 Methods	18
3.5 Class DictEmitter	20
3.5.1 Methods	20
3.6 Class ListEmitter	21
3.6.1 Methods	21

3.7	Class OverrideWarner	23
3.7.1	Methods	23
3.8	Class EmitterProxy	24
3.8.1	Methods	24
3.9	Class BuilderBase	25
3.9.1	Methods	25
3.9.2	Class Variables	27
3.10	Class CompositeBuilder	27
3.10.1	Methods	27
4	Module SCons.CacheDir	28
4.1	Functions	28
4.2	Variables	28
4.3	Class CacheDir	28
4.3.1	Methods	28
5	Module SCons.Conftest	30
5.1	Functions	30
5.2	Variables	33
6	Module SCons.Debug	34
6.1	Functions	34
6.2	Variables	34
7	Module SCons.Defaults	36
7.1	Functions	36
7.2	Variables	37
7.3	Class NullCmdGenerator	37
7.3.1	Methods	38
7.4	Class Variable_Method_Caller	38
7.4.1	Methods	38
8	Module SCons.Environment	39
8.1	Functions	39
8.2	Variables	39
8.3	Class MethodWrapper	40
8.3.1	Methods	40
8.4	Class BuilderWrapper	40
8.4.1	Methods	41
8.5	Class BuilderDict	41
8.5.1	Methods	41
8.6	Class SubstitutionEnvironment	42
8.6.1	Methods	43
8.6.2	Class Variables	45
8.7	Class Base	45
8.7.1	Methods	46
8.7.2	Class Variables	54
8.8	Class OverrideEnvironment	54
8.8.1	Methods	55
8.8.2	Class Variables	64
8.9	Class Base	64
8.9.1	Methods	64

8.9.2	Class Variables	73
9	Module SCons.Errors	74
9.1	Variables	74
9.2	Class BuildError	74
9.2.1	Methods	74
9.2.2	Properties	75
9.3	Class InternalError	76
9.3.1	Methods	76
9.3.2	Properties	77
9.4	Class UserError	77
9.4.1	Methods	78
9.4.2	Properties	79
9.5	Class StopError	79
9.5.1	Methods	79
9.5.2	Properties	80
9.6	Class EnvironmentError	81
9.6.1	Methods	81
9.6.2	Properties	82
9.7	Class ExplicitExit	83
9.7.1	Methods	83
9.7.2	Properties	84
9.8	Class TaskmasterException	84
9.8.1	Methods	84
9.8.2	Properties	86
10	Module SCons.Executor	87
10.1	Functions	87
10.2	Variables	87
10.3	Class Executor	87
10.3.1	Methods	87
10.3.2	Class Variables	89
10.4	Class Null	89
10.4.1	Methods	89
11	Module SCons.Job	91
11.1	Variables	91
11.2	Class InterruptState	91
11.2.1	Methods	91
11.3	Class Jobs	91
11.3.1	Methods	91
11.4	Class Serial	92
11.4.1	Methods	92
11.5	Class Worker	93
11.5.1	Methods	93
11.5.2	Properties	94
11.6	Class ThreadPool	94
11.6.1	Methods	94
11.7	Class Parallel	95
11.7.1	Methods	95
12	Module SCons.Memoize	96

12.1	Functions	97
12.2	Variables	98
12.3	Class Counter	98
12.3.1	Methods	98
12.4	Class CountValue	98
12.4.1	Methods	99
12.5	Class CountDict	99
12.5.1	Methods	99
12.6	Class Memoizer	100
12.6.1	Methods	100
12.7	Class Memoized_Metaclass	100
12.7.1	Methods	100
12.7.2	Properties	101
13	Package SCons.Node	103
13.1	Modules	103
13.2	Functions	103
13.3	Variables	103
13.4	Class NodeInfoBase	104
13.4.1	Methods	104
13.4.2	Class Variables	104
13.5	Class BuildInfoBase	105
13.5.1	Methods	105
13.5.2	Class Variables	105
13.6	Class Node	105
13.6.1	Methods	105
13.6.2	Class Variables	115
13.7	Class NodeList	115
13.7.1	Methods	115
13.8	Class Walker	116
13.8.1	Methods	117
14	Module SCons.Node.Alias	118
14.1	Variables	118
14.2	Class AliasNameSpace	118
14.2.1	Methods	118
14.3	Class AliasNodeInfo	119
14.3.1	Methods	119
14.3.2	Class Variables	120
14.4	Class AliasBuildInfo	120
14.4.1	Methods	120
14.4.2	Class Variables	120
14.5	Class Alias	120
14.5.1	Methods	121
14.5.2	Class Variables	131
15	Module SCons.Node.FS	132
15.1	Functions	132
15.2	Variables	133
15.3	Class DiskChecker	134
15.3.1	Methods	134
15.4	Class EntryProxy	134

15.4.1	Methods	135
15.4.2	Class Variables	135
15.5	Class Base	135
15.5.1	Methods	136
15.5.2	Class Variables	147
15.6	Class Entry	147
15.6.1	Methods	147
15.6.2	Class Variables	160
15.7	Class LocalFS	160
15.7.1	Methods	160
15.7.2	Class Variables	161
15.8	Class FS	161
15.8.1	Methods	161
15.8.2	Class Variables	164
15.9	Class DirNodeInfo	164
15.9.1	Methods	164
15.9.2	Class Variables	165
15.10	Class DirBuildInfo	165
15.10.1	Methods	165
15.10.2	Class Variables	165
15.11	Class Dir	165
15.11.1	Methods	166
15.11.2	Class Variables	180
15.12	Class RootDir	180
15.12.1	Methods	181
15.12.2	Class Variables	196
15.13	Class FileInfo	196
15.13.1	Methods	196
15.13.2	Class Variables	197
15.14	Class FileBuildInfo	197
15.14.1	Methods	197
15.14.2	Class Variables	198
15.15	Class File	198
15.15.1	Methods	198
15.15.2	Class Variables	212
15.16	Class Finder	212
15.16.1	Methods	212
15.16.2	Class Variables	213
16	Module SCons.Node.Python	214
16.1	Variables	214
16.2	Class ValueNodeInfo	214
16.2.1	Methods	214
16.2.2	Class Variables	214
16.3	Class ValueBuildInfo	215
16.3.1	Methods	215
16.3.2	Class Variables	215
16.4	Class Value	215
16.4.1	Methods	215
16.4.2	Class Variables	226
17	Module SCons.PathList	227

17.1 Functions	227
17.2 Variables	227
18 Module SCons.SConf	228
18.1 Functions	228
18.2 Variables	229
18.3 Class SConfWarning	230
18.3.1 Methods	230
18.3.2 Properties	231
18.4 Class SConfError	231
18.4.1 Methods	232
18.4.2 Properties	233
18.5 Class ConfigureDryRunError	233
18.5.1 Methods	233
18.5.2 Properties	235
18.6 Class ConfigureCacheError	235
18.6.1 Methods	235
18.6.2 Properties	236
18.7 Class SConfBuildInfo	237
18.7.1 Methods	237
18.7.2 Class Variables	238
18.8 Class Streamer	238
18.8.1 Methods	238
18.9 Class SConfBuildTask	238
18.9.1 Methods	239
18.10 Class SConfBase	242
18.10.1 Methods	242
18.11 Class CheckContext	244
18.11.1 Methods	244
19 Module SCons.SConsign	246
19.1 Functions	246
19.2 Variables	246
19.3 Class SConsignEntry	246
19.3.1 Methods	247
19.3.2 Class Variables	247
19.4 Class Base	247
19.4.1 Methods	247
19.5 Class DB	248
19.5.1 Methods	248
19.6 Class Dir	248
19.6.1 Methods	249
19.7 Class DirFile	249
19.7.1 Methods	249
19.8 Class DB	250
19.8.1 Methods	250
20 Package SCons.Scanner	252
20.1 Modules	252
20.2 Functions	252
20.3 Variables	252
20.4 Class FindPathDirs	253

20.4.1	Methods	253
20.5	Class Base	253
20.5.1	Methods	253
20.6	Class Selector	255
20.6.1	Methods	257
20.7	Class Current	258
20.7.1	Methods	260
20.8	Class Classic	261
20.8.1	Methods	262
20.9	Class ClassicCPP	263
20.9.1	Methods	263
21	Module SCons.Scanner.C	265
21.1	Functions	265
21.2	Variables	265
21.3	Class SConsCPPScanner	265
21.3.1	Methods	265
21.4	Class SConsCPPScannerWrapper	269
21.4.1	Methods	269
22	Module SCons.Scanner.D	270
22.1	Functions	270
22.2	Variables	270
22.3	Class D	270
22.3.1	Methods	270
23	Module SCons.Scanner.Dir	272
23.1	Functions	272
23.2	Variables	272
24	Module SCons.Scanner.Fortran	273
24.1	Functions	273
24.2	Variables	273
24.3	Class F90Scanner	273
24.3.1	Methods	273
25	Module SCons.Scanner.IDL	275
25.1	Functions	275
25.2	Variables	275
26	Module SCons.Scanner.LaTeX	276
26.1	Functions	276
26.2	Variables	276
26.3	Class LaTeX	276
26.3.1	Methods	279
26.3.2	Class Variables	280
27	Module SCons.Scanner.Prog	281
27.1	Functions	281
27.2	Variables	281
28	Module SCons.Scanner.RC	282
28.1	Functions	282

28.2 Variables	282
29 Package SCons.Script	283
29.1 Modules	283
29.2 Functions	283
29.3 Variables	283
29.4 Class TargetList	287
29.4.1 Methods	287
30 Module SCons.Script.Interactive	290
30.1 Functions	290
30.2 Variables	290
30.3 Class SConsInteractiveCmd	290
30.3.1 Methods	290
30.3.2 Class Variables	293
31 Module SCons.Script.Main	294
31.1 Functions	294
31.2 Variables	294
31.3 Class SConsPrintHelpException	295
31.3.1 Methods	295
31.3.2 Properties	296
31.4 Class Progressor	297
31.4.1 Methods	297
31.4.2 Class Variables	297
31.5 Class BuildTask	297
31.5.1 Methods	298
31.5.2 Class Variables	301
31.6 Class CleanTask	301
31.6.1 Methods	301
31.7 Class QuestionTask	305
31.7.1 Methods	305
31.8 Class TreePrinter	308
31.8.1 Methods	308
31.9 Class FakeOptionParser	308
31.9.1 Methods	309
31.9.2 Class Variables	309
31.10 Class Stats	309
31.10.1 Methods	309
31.11 Class CountStats	309
31.11.1 Methods	309
31.12 Class MemStats	310
31.12.1 Methods	310
32 Module SCons.Script.SConscript'	311
32.1 Functions	311
32.2 Variables	311
32.3 Class SConscriptReturn	312
32.3.1 Methods	312
32.3.2 Properties	313
32.4 Class Frame	313
32.4.1 Methods	314

32.5	Class SConsEnvironment	314
32.5.1	Methods	314
32.5.2	Class Variables	323
32.6	Class DefaultEnvironmentCall	323
32.6.1	Methods	323
33	Module SCons.Sig	324
33.1	Variables	324
33.2	Class MD5Null	324
33.2.1	Methods	324
33.3	Class TimeStampNull	325
33.3.1	Methods	325
34	Module SCons.Subst	326
34.1	Functions	326
34.2	Variables	327
34.3	Class Literal	327
34.3.1	Methods	327
34.4	Class SpecialAttrWrapper	328
34.4.1	Methods	328
34.5	Class CmdStringHolder	328
34.5.1	Methods	328
34.6	Class NLWrapper	331
34.6.1	Methods	331
34.7	Class Targets_or_Sources	332
34.7.1	Methods	332
34.8	Class Target_or_Source	333
34.8.1	Methods	334
35	Module SCons.Taskmaster	335
35.1	Functions	335
35.2	Variables	335
35.3	Class Stats	336
35.3.1	Methods	336
35.4	Class Task	336
35.4.1	Methods	336
35.5	Class Taskmaster	340
35.5.1	Methods	340
36	Module SCons.Util	342
36.1	Functions	342
36.2	Variables	347
36.3	Class CallableComposite	348
36.3.1	Methods	348
36.4	Class NodeList	350
36.4.1	Methods	350
36.5	Class DisplayEngine	352
36.5.1	Methods	352
36.6	Class mystr	352
36.6.1	Methods	352
36.6.2	Properties	360
36.7	Class Proxy	361

36.7.1	Methods	361
36.8	Class <code>_NoError</code>	362
36.8.1	Methods	362
36.8.2	Properties	363
36.9	Class <code>CLVar</code>	363
36.9.1	Methods	364
36.10	Class <code>OrderedDict</code>	365
36.10.1	Methods	365
36.11	Class <code>Selector</code>	367
36.11.1	Methods	367
36.12	Class <code>LogicalLines</code>	368
36.12.1	Methods	368
36.13	Class <code>UniqueList</code>	368
36.13.1	Methods	368
36.14	Class <code>Unbuffered</code>	370
36.14.1	Methods	370
36.15	Class <code>Null</code>	371
36.15.1	Methods	371
37	Package <code>SCons.Variables</code>	372
37.1	Modules	372
37.2	Variables	372
37.3	Class Variables	372
37.3.1	Methods	372
37.3.2	Class Variables	374
38	Module <code>SCons.Variables.BoolVariable</code>	375
38.1	Functions	375
39	Module <code>SCons.Variables.EnumVariable</code>	376
39.1	Functions	376
40	Module <code>SCons.Variables.ListVariable</code>	377
40.1	Functions	377
41	Module <code>SCons.Variables.PackageVariable</code>	378
41.1	Functions	378
42	Module <code>SCons.Variables.PathVariable</code>	379
42.1	Variables	379
43	Module <code>SCons.Warnings</code>	381
43.1	Functions	381
43.2	Variables	381
43.3	Class <code>Warning</code>	382
43.3.1	Methods	382
43.3.2	Properties	383
43.4	Class <code>CacheWriteErrorWarning</code>	384
43.4.1	Methods	384
43.4.2	Properties	385
43.5	Class <code>CorruptSConsignWarning</code>	385
43.5.1	Methods	386
43.5.2	Properties	387

43.6 Class DependencyWarning	387
43.6.1 Methods	387
43.6.2 Properties	389
43.7 Class DeprecatedWarning	389
43.7.1 Methods	389
43.7.2 Properties	390
43.8 Class DeprecatedCopyWarning	391
43.8.1 Methods	391
43.8.2 Properties	392
43.9 Class DeprecatedSourceSignaturesWarning	393
43.9.1 Methods	393
43.9.2 Properties	394
43.10 Class DeprecatedTargetSignaturesWarning	395
43.10.1 Methods	395
43.10.2 Properties	396
43.11 Class DuplicateEnvironmentWarning	397
43.11.1 Methods	397
43.11.2 Properties	398
43.12 Class LinkWarning	398
43.12.1 Methods	399
43.12.2 Properties	400
43.13 Class MisleadingKeywordsWarning	400
43.13.1 Methods	400
43.13.2 Properties	402
43.14 Class MissingSConscriptWarning	402
43.14.1 Methods	402
43.14.2 Properties	403
43.15 Class NoMD5ModuleWarning	404
43.15.1 Methods	404
43.15.2 Properties	405
43.16 Class NoMetaclassSupportWarning	405
43.16.1 Methods	406
43.16.2 Properties	407
43.17 Class NoObjectCountWarning	407
43.17.1 Methods	407
43.17.2 Properties	409
43.18 Class NoParallelSupportWarning	409
43.18.1 Methods	409
43.18.2 Properties	410
43.19 Class PythonVersionWarning	411
43.19.1 Methods	411
43.19.2 Properties	412
43.20 Class ReservedVariableWarning	413
43.20.1 Methods	413
43.20.2 Properties	414
43.21 Class StackSizeWarning	414
43.21.1 Methods	415
43.21.2 Properties	416
43.22 Class FortranCxxMixWarning	416
43.22.1 Methods	416
43.22.2 Properties	418

44 Package SCons.compat	419
44.1 Modules	419
44.2 Functions	420
44.3 Variables	420
45 Module SCons.compat._scons_UserString	421
45.1 Functions	421
45.2 Variables	421
45.3 Class UserString	421
45.3.1 Methods	421
46 Module SCons.compat._scons_hashlib	423
46.1 Functions	423
46.2 Variables	423
46.3 Class md5obj	423
46.3.1 Methods	423
46.3.2 Class Variables	423
46.4 Class md5obj	424
46.4.1 Methods	424
46.4.2 Class Variables	424
47 Module SCons.compat._scons_itertools	425
47.1 Functions	425
47.2 Variables	426
48 Module SCons.compat._scons_optparse	427
48.1 Variables	427
48.2 Class OptParseError	428
48.2.1 Methods	428
48.2.2 Properties	429
48.3 Class OptionError	429
48.3.1 Methods	430
48.3.2 Properties	431
48.4 Class OptionConflictError	431
48.4.1 Methods	431
48.4.2 Properties	433
48.5 Class OptionValueError	433
48.5.1 Methods	433
48.5.2 Properties	434
48.6 Class BadOptionError	435
48.6.1 Methods	435
48.6.2 Properties	436
48.7 Class HelpFormatter	436
48.7.1 Methods	437
48.7.2 Class Variables	438
48.8 Class IndentedHelpFormatter	438
48.8.1 Methods	438
48.8.2 Class Variables	439
48.9 Class TitledHelpFormatter	439
48.9.1 Methods	439
48.9.2 Class Variables	440
48.10Class Option	440

48.10.1 Methods	441
48.10.2 Class Variables	441
48.11 Class Values	442
48.11.1 Methods	442
48.12 Class OptionContainer	442
48.12.1 Methods	443
48.13 Class OptionGroup	443
48.13.1 Methods	443
48.14 Class OptionParser	444
48.14.1 Methods	446
48.14.2 Class Variables	448
49 Module SCons.compat._scons_sets	449
49.1 Class BaseSet	449
49.1.1 Methods	450
49.1.2 Properties	453
49.2 Class ImmutableSet	453
49.2.1 Methods	453
49.2.2 Properties	456
49.3 Class Set	456
49.3.1 Methods	457
49.3.2 Properties	461
50 Module SCons.compat._scons_sets15	462
50.1 Class Set	462
50.1.1 Methods	462
51 Module SCons.compat._scons_shlex	464
51.1 Functions	464
51.2 Class shlex	464
51.2.1 Methods	464
52 Module SCons.compat._scons_subprocess	465
52.1 Functions	471
52.2 Variables	472
52.3 Class CalledProcessError	472
52.3.1 Methods	472
52.3.2 Properties	473
52.4 Class Popen	474
52.4.1 Methods	474
52.4.2 Properties	475
53 Module SCons.compat._scons_textwrap	476
53.1 Functions	476
53.2 Class TextWrapper	476
53.2.1 Methods	477
53.2.2 Class Variables	477
54 Module SCons.compat.builtins	479
54.1 Functions	479
54.2 Variables	480
55 Module SCons.cpp	481

55.1 Functions	481
55.2 Variables	481
55.3 Class FunctionEvaluator	481
55.3.1 Methods	482
55.4 Class PreProcessor	482
55.4.1 Methods	482
55.5 Class DumbPreProcessor	485
55.5.1 Methods	485
56 Module SCons.dblite	490
56.1 Functions	490
56.2 Variables	490
56.3 Class dblite	490
56.3.1 Methods	490
57 Module SCons.exitfuncs	491
57.1 Functions	491
57.2 Variables	491
58 Module md5	492
58.1 Variables	492

1 Package SCons

SCons

The main package for the SCons software construction utility.

Version: 1.1.0

Date: 2008/10/10 05:46:45

1.1 Modules

- **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.
(Section 2, p. 6)
- **Builder:** SCons.Builder
Builder object subsystem.
(Section 3, p. 15)
- **CacheDir:** CacheDir support
(Section 4, p. 28)
- **Conftest:** SCons.Conftest
Autoconf-like configuration support; low level implementation of tests.
(Section 5, p. 30)
- **Debug:** SCons.Debug
Code for debugging SCons internal things.
(Section 6, p. 34)
- **Defaults:** SCons.Defaults
Builders and other things for the local site.
(Section 7, p. 36)
- **Environment:** SCons.Environment
Base class for construction Environments.
(Section 8, p. 39)
- **Errors:** SCons.Errors
This file contains the exception classes used to handle internal and user errors in SCons.
(Section 9, p. 74)
- **Executor:** SCons.Executor
A module for executing actions with specific lists of target and source Nodes.
(Section 10, p. 87)
- **Job:** SCons.Job
This module defines the Serial and Parallel classes that execute tasks to complete a build.
(Section 11, p. 91)
- **Memoize:** Memoizer
A metaclass implementation to count hits and misses of the computed values that various methods cache in memory.
(Section 12, p. 96)
- **Node:** SCons.Node
The Node package for the SCons software construction utility.
(Section 13, p. 103)
 - **Alias:** scons.Node.Alias
Alias nodes.
(Section 14, p. 118)

- **FS**: `scons.Node.FS`
File system nodes.
(Section 15, p. 132)
- **Python**: `scons.Node.Python`
Python nodes.
(Section 16, p. 214)
- **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.
(Section 17, p. 227)
- **SConf**: `SCons.SConf`
Autoconf-like configuration support.
(Section 18, p. 228)
- **SConsign**: `SCons.SConsign`
Writing and reading information to the `.sconsign` file or files.
(Section 19, p. 246)
- **Scanner**: `SCons.Scanner`
The Scanner package for the SCons software construction utility.
(Section 20, p. 252)
 - **C**: `SCons.Scanner.C`
This module implements the dependency scanner for C/C++ code.
(Section 21, p. 265)
 - **D**: `SCons.Scanner.D`
Scanner for the Digital Mars "D" programming language.
(Section 22, p. 270)
 - **Dir** (Section 23, p. 272)
 - **Fortran**: `SCons.Scanner.Fortran`
This module implements the dependency scanner for Fortran code.
(Section 24, p. 273)
 - **IDL**: `SCons.Scanner.IDL`
This module implements the dependency scanner for IDL (Interface Definition Language) files.
(Section 25, p. 275)
 - **LaTeX**: `SCons.Scanner.LaTeX`
This module implements the dependency scanner for LaTeX code.
(Section 26, p. 276)
 - **Prog** (Section 27, p. 281)
 - **RC**: `SCons.Scanner.RC`
This module implements the dependency scanner for RC (Interface Definition Language) files.
(Section 28, p. 282)
- **Script**: `SCons.Script`
This file implements the `main()` function used by the `scons` script.
(Section 29, p. 283)
 - **Interactive**: SCons interactive mode
(Section 30, p. 290)
 - **Main**: `SCons.Script`
This file implements the `main()` function used by the `scons` script.
(Section 31, p. 294)
 - **SConscript'**: `SCons.Script.SConscript`
This module defines the Python API provided to `SConscript` and `SConstruct` files.
(Section 32, p. 311)

- **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.
(Section 33, p. 324)
- **Subst:** SCons.Subst
SCons string substitution.
(Section 34, p. 326)
- **Taskmaster:** Generic Taskmaster module for the SCons build engine.
(Section 35, p. 335)
- **Util:** SCons.Util
Various utility functions go here.
(Section 36, p. 342)
- **Variables:** engine.SCons.Variables
This file defines the Variables class that is used to add user-friendly customizable variables to an SCons build.
(Section 37, p. 372)
 - **BoolVariable**[?]: engine.SCons.Variables.BoolVariable
This file defines the option type for SCons implementing true/false values.
(Section 38, p. 375)
 - **EnumVariable**[?]: engine.SCons.Variables.EnumVariable
This file defines the option type for SCons allowing only specified input-values.
(Section 39, p. 376)
 - **ListVariable**[?]: engine.SCons.Variables.ListVariable
This file defines the option type for SCons implementing 'lists'.
(Section 40, p. 377)
 - **PackageVariable**[?]: engine.SCons.Variables.PackageVariable
This file defines the option type for SCons implementing 'package activation'.
(Section 41, p. 378)
 - **PathVariable**[?]: SCons.Variables.PathVariable
This file defines an option type for SCons implementing path settings.
(Section 42, p. 379)
- **Warnings:** SCons.Warnings
This file implements the warnings framework for SCons.
(Section 43, p. 381)
- **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.
(Section 44, p. 419)
 - **_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.
(Section 45, p. 421)
 - **_scons_hashlib:** hashlib backwards-compatibility module for older (pre-2.5) Python versions
This does not not NOT (repeat, *NOT*) provide complete hashlib functionality.
(Section 46, p. 423)
 - **_scons_itertools:** Implementations of itertools functions for Python versions that don't have iterators.
(Section 47, p. 425)
 - **_scons_optparse:** optparse - a powerful, extensible, and easy-to-use option parser.
(Section 48, p. 427)
 - **_scons_sets:** Classes to represent arbitrary sets (including sets of sets).

- (Section 49, p. 449)
- **scons_sets15** (Section 50, p. 462)
 - **scons_shlex**: A lexical analyzer class for simple shell-like syntaxes.
(Section 51, p. 464)
 - **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 52, p. 465)
 - **scons_textwrap**: Text wrapping and filling.
(Section 53, p. 476)
 - **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.
(Section 54, p. 479)
 - **cpp**: SCons C Pre-Processor module
(Section 55, p. 481)
 - **dblite** (Section 56, p. 490)
 - **exitfuncs**: SCons.exitfuncs
Register functions which are executed when SCons exits for any reason.
(Section 57, p. 491)

1.2 Variables

Name	Description
<code>__build__</code>	Value: 'r3603'
<code>__buildsys__</code>	Value: 'scons-dev'
<code>__developer__</code>	Value: 'scons'
<code>__revision__</code>	Value: 'src/engine/SCons/__init__.py 3603 2008/10/10 05:46:45 sc...

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 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

<code>rfile(<i>n</i>)</code>

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

<code>remove_set_lineno_codes(<i>x</i>)</code>
--

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

A factory for action objects.

2.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Action.py 3603 2008/10/10 05:46:45 scons'
<code>print_actions</code>	Value: False
<code>execute_actions</code>	Value: False
<code>print_actions_presub</code>	Value: 0
<code>SET_LINENO</code>	Value: <code>dis.SET_LINENO</code>
<code>HAVE_ARGUMENT</code>	Value: <code>dis.HAVE_ARGUMENT</code>
<code>default_ENV</code>	Value: False

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__(self, other)
```

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

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

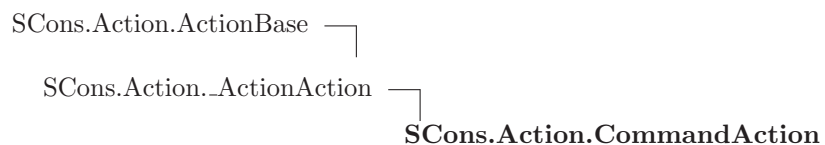
```
__radd__(self, other)
```

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

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

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 0x8391e9c>,
presub=<class SCons.Action._Null at 0x8391e9c>, show=<class SCons.Action._Null at 0x8391e9c>,
execute=<class SCons.Action._Null at 0x8391e9c>, chdir=<class SCons.Action._Null at 0x8391e9c>)
```

```
__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 0x8391e9c>,
presub=<class SCons.Action.Null at 0x8391e9c>, show=<class SCons.Action.Null at
0x8391e9c>, execute=<class SCons.Action.Null at 0x8391e9c>, chdir=<class
SCons.Action.Null at 0x8391e9c>)
```

```
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)
```

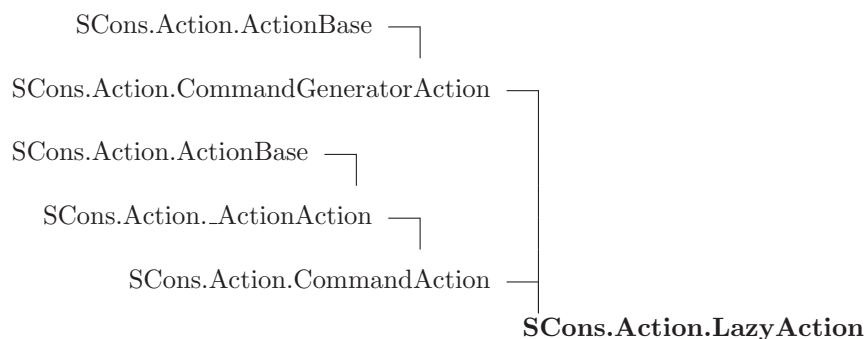
```
__radd__(self, other)
```

```
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.

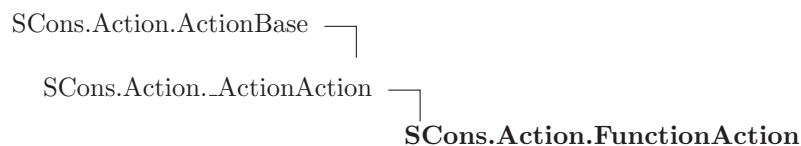
`genstring(self, target, source, env)`Overrides: *SCons.Action.ActionBase.genstring***`get_executor(self, env, overrides, tlist, slist, executor_kw)`**

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



Class for Python function actions.

2.7.1 Methods

```
__init__(self, execfunction, cmdstr=<class SCons.Action._Null at 0x8391e9c>, *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 0x8391e9c>,
presub=<class SCons.Action._Null at 0x8391e9c>, show=<class SCons.Action._Null at
0x8391e9c>, execute=<class SCons.Action._Null at 0x8391e9c>, chdir=<class
SCons.Action._Null at 0x8391e9c>)
```

```
__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

```
SCons.Action.ActionBase └─ SCons.Action.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 0x8391e9c>,
presub=<class SCons.Action.Null at 0x8391e9c>, show=<class SCons.Action.Null at 0x8391e9c>,
execute=<class SCons.Action.Null at 0x8391e9c>, chdir=<class SCons.Action.Null at 0x8391e9c>)
```

```
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

```
__init__(self, actfunc, strfunc, convert=<function <lambda> at 0x83b95a4>)
```

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

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 3603 2008/10/10 05:46:45 scons'
<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(<i>self</i>) Overrides: UserDict.UserDict.items

iteritems(<i>self</i>)

iterkeys(<i>self</i>)

itervalues(<i>self</i>)

keys(<i>self</i>) Overrides: UserDict.UserDict.keys

pop(<i>self</i>, <i>key</i>, *<i>args</i>)

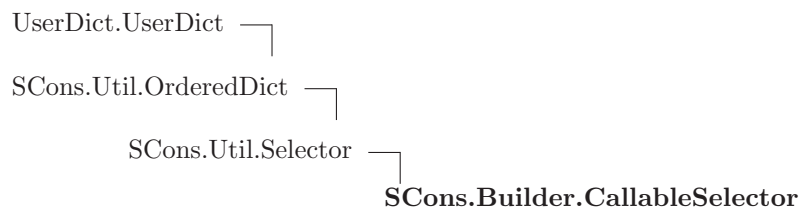
popitem(<i>self</i>) Overrides: UserDict.UserDict.popitem

setdefault(<i>self</i>, <i>key</i>, <i>failobj</i>=False) Overrides: UserDict.UserDict.setdefault

update(<i>self</i>, <i>dict</i>) Overrides: UserDict.UserDict.update
--

values(<i>self</i>) Overrides: UserDict.UserDict.values

3.4 Class CallableSelector



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

3.4.1 Methods

__call__(<i>self</i>, <i>env</i>, <i>source</i>) Overrides: SCons.Util.Selector.__call__
--

__cmp__(<i>self</i>, <i>dict</i>)
--

__contains__(<i>self</i>, <i>key</i>)
--

__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*)

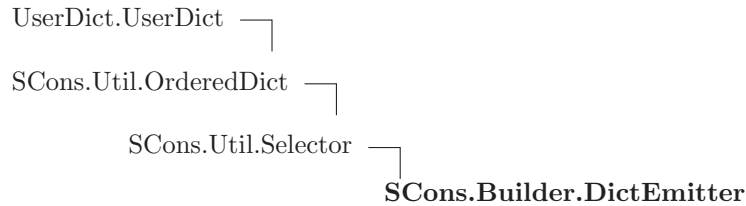
popitem(*self*)
 Overrides: UserDict.UserDict.popitem

setdefault(*self*, *key*, *failobj*=False)
 Overrides: UserDict.UserDict.setdefault

update (<i>self</i> , <i>dict</i>) Overrides: UserDict.UserDict.update
--

values (<i>self</i>) 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__ (<i>self</i> , <i>target</i> , <i>source</i> , <i>env</i>) Overrides: SCons.Util.Selector.__call__

__cmp__ (<i>self</i> , <i>dict</i>)
--

__contains__ (<i>self</i> , <i>key</i>)
--

__delitem__ (<i>self</i> , <i>key</i>) Overrides: UserDict.UserDict.__delitem__

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

__init__ (<i>self</i> , <i>dict</i> =False) Overrides: UserDict.UserDict.__init__
--

__len__ (<i>self</i>)

__repr__ (<i>self</i>)

__setitem__ (<i>self</i> , <i>key</i> , <i>item</i>) Overrides: UserDict.UserDict.__setitem__

clear (<i>self</i>) 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`

```
UserList.UserList └─ SCons.Builder.ListEmitter
```

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

3.6.1 Methods

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

`--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 0x840f65c>,
is_explicit=False, src_builder=False, 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 0x840f65c>,
**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



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 3603 2008/10/10 05:46:45 sc...
<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

__init__ (<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.

CheckCC(*context*)

Configure check for a working C compiler.

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

CheckSHCC(*context*)

Configure check for a working shared C compiler.

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

CheckCXX(*context*)

Configure check for a working CXX compiler.

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

CheckSHCXX(*context*)

Configure check for a working shared CXX compiler.

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

CheckFunc(*context*, *function_name*, *header=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 3603 2008/10/10 05:46:45 scons'
<code>tracked_classes</code>	Value: {'Action.CommandAction': [<weakref at 0x87abf54; to 'inst...
<code>caller_bases</code>	Value: {}
<code>caller_dicts</code>	Value: {}

continued on next page

Name	Description
shorten_list	Value: [('scons/SCons/', 1), ('src/engine/SCons/', 1), ('usr/...
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 `qconf`.

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 `_fetchDefaultEnvironment()` function, which more efficiently returns the initialized default construction environment without checking for its existence.

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

StaticObjectEmitter(*target, source, env*)

SharedObjectEmitter(*target, source, env*)

SharedFlagChecker(*source, target, env*)

get_paths_str(*dest*)

chmod_func(*dest, mode*)

chmod_strfunc(*dest, mode*)

copy_func(*dest, src*)

delete_func(*dest, must_exist=0*)

delete_strfunc(*dest, must_exist=0*)

mkdir_func(*dest*)

move_func(*dest*, *src*)

touch_func(*dest*)

7.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Defaults.py 3603 2008/10/10 05:46:45 sc...
<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(move_func, lambda dest, src: 'Move("%s", "%...</code>
<code>Touch</code>	Value: <code>ActionFactory(touch_func, lambda file: 'Touch("%s)' % get_...</code>
<code>ConstructionEnvironment</code>	Value: <code>{'BUILDERS': {}, 'CONFIGUREDIRENTRY': '#/.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 3603 2008/10/10 05:46:45...'
<code>CleanTargets</code>	Value: {}
<code>CalculatorArgs</code>	Value: {}
<code>AliasBuilder</code>	Value: <code>SCons.Builder.Builder(action= alias_builder, target_facto...</code>
<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 0x87a886c>, *args, **kw)
Overrides: SCons.Environment.MethodWrapper.__call__
```

```
__repr__(self)
```

```
__str__(self)
```

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

```
__setattr__(self, name, value)
```

```
__init__(self, object, method, name=False)
```

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

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

```
__init__(self, dict, env)
Overrides: UserDict.UserDict.__init__
```

```
__semi_deepcopy__(self)
```

```
__setitem__(self, item, val)
Overrides: UserDict.UserDict.__setitem__
```

```
__delitem__(self, item)
Overrides: UserDict.UserDict.__delitem__
```

update(*self*, *dict*)
 Overrides: UserDict.UserDict.update

__cmp__(*self*, *dict*)

__contains__(*self*, *key*)

__getitem__(*self*, *key*)

__len__(*self*)

__repr__(*self*)

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*)

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 0x87a886c>,
lookup_list=<class SCons.Environment.Null at 0x87a886c>, **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, *dict*=False)

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

8.6.2 Class Variables

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

8.7 Class Base

`SCons.Environment.SubstitutionEnvironment`

SCons.Environment.Base

Known Subclasses: `SCons.Environment.OverrideEnvironment`, `SCons.Script.SConsScript.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*=':', *delete_existing*=False)

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

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

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

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

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

Builder(*self*, **kw)

CacheDir(*self*, path)

Clean(*self*, targets, files)

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

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

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=':'*, *delete_existing=False*)

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

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

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

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

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 0x87a886c>,
lookup_list=<class SCons.Environment._Null at 0x87a886c>, **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.MemoizedMetaclass</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__` `exitit`(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* `exitit`(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*=':', *delete_existing*=False)

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

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

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

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

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

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

CacheDir(*self*, *path*)

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

Clone(*self*, *tools*=[], *toolpath*=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.

FindIdxes(*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, *dict*=False)

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

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*=':', *delete_existing*=False)

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

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

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

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

If delete_existing is 1, removes existing values first, so values move to front.

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 0x87a886c>, *lookup_list*=<class SCons.Environment.Null at 0x87a886c>, ***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.MemoizedMetaclass</code>
<code>memoizer_counters</code>	Value: <code>[]</code>

8.9 Class Base

`SCons.Environment.SubstitutionEnvironment`  **`SCons.Environment.Base`**

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

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

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

8.9.1 Methods

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

```
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*=':', *delete_existing*=False)

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

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

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

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

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

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

CacheDir(*self*, *path*)

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

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

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

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*=':', *delete_existing*=False)

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

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

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

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

If delete_existing is 1, removes existing values first, so values move to front.

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 0x87a886c>, *lookup_list*=<class SCons.Environment.Null at 0x87a886c>, ****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

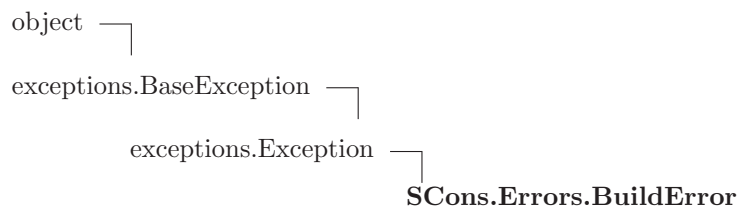
SCons.Errors

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 3603 2008/10/10 05:46:45 scons'

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__ 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__(...)
```

```
__str__(x)
```

```
str(x)
```

Overrides: object.__str__

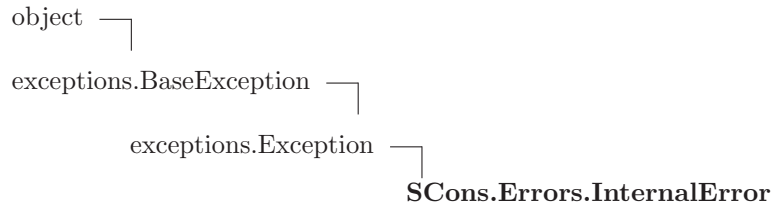
9.2.2 Properties

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

continued on next page

Name	Description
message	Value: <member 'message' of 'exceptions.BaseException' objects>

9.3 Class *InternalError*



9.3.1 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 <code>x</code> ; 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__
```

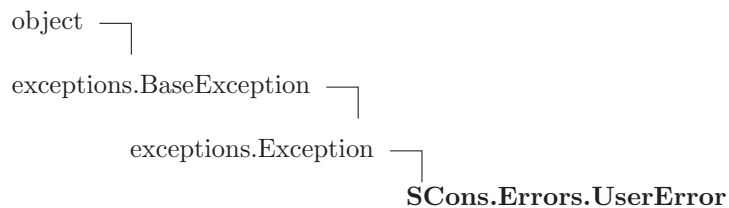
```
__setstate__(...)
```

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

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 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__
```

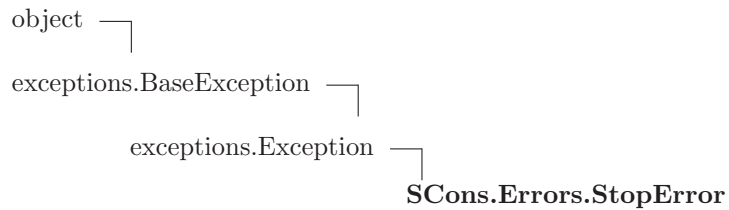
```
__setstate__(...)
```

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

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

```
__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 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__`

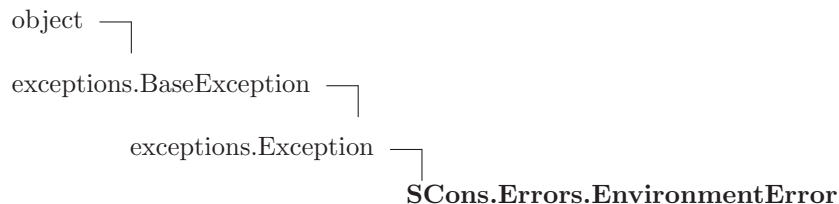
`__setstate__(...)`

`__str__(x)`

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

<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>

<code>__init__(...)</code>
<code>x.__init__(...)</code> initializes <code>x</code> ; see <code>x.__class__.__doc__</code> for signature
Overrides: <code>exceptions.BaseException.__init__</code>

```
__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)
```

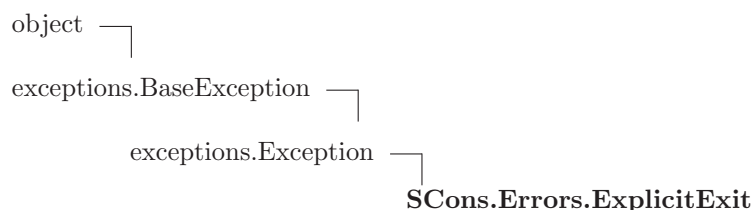
```
str(x)
```

Overrides: object.__str__

9.6.2 Properties

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

9.7 Class *ExplicitExit*



9.7.1 Methods

```
__init__(self, node=False, status=False, *args)
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__ 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)`

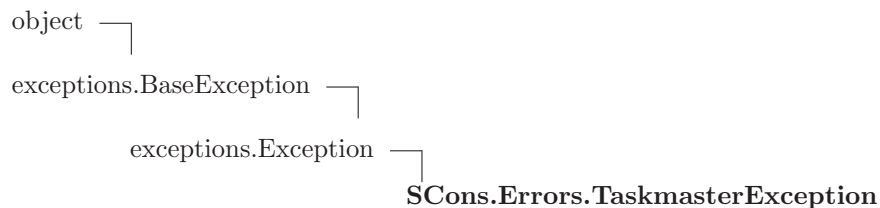
`str(x)`

Overrides: `object.__str__`

9.7.2 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

`__init__(self, node=False, exc_info=(None, None, None), *args)`

`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>

9.8.2 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 Functions

`get_NullEnvironment()`

Use singleton pattern for Null Environments.

10.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Executor.py 3603 2008/10/10 05:46:45 sc...
<code>nullenv</code>	Value: False

10.3 Class Executor

A class for controlling instances of executing an action.

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

10.3.1 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_unignored_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.

10.3.2 Class Variables

Name	Description
memoizer_counters	Value: []

10.4 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 disappear 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.

10.4.1 Methods

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

```
get_build_env(self)
```

```
get_build_scanner_path(self)
```

```
cleanup(self)
```

```
prepare(self)
```

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

```
get_action_list(self)
```

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

```
get_contents(self)
```

```
add_pre_action(self, action)
```

```
add_post_action(self, action)
```

```
set_action_list(self, action)
```

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 3603 2008/10/10 05:46:45 scons'
<code>explicit_stack_size</code>	Value: False
<code>default_stack_size</code>	Value: 256
<code>interrupt_msg</code>	Value: 'Build interrupted.'

11.2 Class InterruptState

11.2.1 Methods

<code>__init__(self)</code>
<code>set(self)</code>
<code>__call__(self)</code>

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

<p><code>__init__(self, num, taskmaster)</code></p> <hr/> <p>create 'num' jobs using the given taskmaster.</p> <p>If 'num' is 1 or less, then a serial job will be used, otherwise a parallel job with 'num' worker threads will be used.</p> <p>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.</p>
--


```
run(self, postfunc=<function <lambda> at 0x889c41c>)
```

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)
```

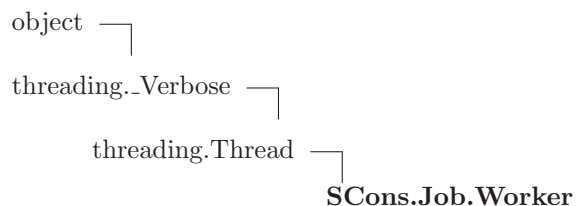
Create a new serial job given a taskmaster.

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

<code>__init__(self, requestQueue, resultsQueue, interrupted)</code> Overrides: <code>threading.Thread.__init__</code>
--

<code>run(self)</code> Overrides: <code>threading.Thread.run</code>

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

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

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

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

<code>__reduce__(...)</code> <hr/> helper for pickle
--

<code>__reduce_ex__(...)</code> <hr/> helper for pickle

<code>__repr__(self)</code> <code>repr(x)</code> Overrides: <code>object.__repr__</code> <code>exitit</code> (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>

EnableMemoization()

12.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Memoize.py 3603 2008/10/10 05:46:45 scons'
<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

<code>__init__(self, method_name)</code>
--

<code>display(self)</code>

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

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



12.7.1 Methods

```
__init__(cls, name, bases, cls_dict)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__ exitit(inherited documentation)
```

```
__call__(x, ...)
```

```
x(...)
```

```
__cmp__(x, y)
```

```
cmp(x,y)
```

```
__delattr__(...)
```

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

Overrides: object.__delattr__

```
__getattribute__(...)
```

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

Overrides: object.__getattribute__

```
__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__

`--str--(x)`

`str(x)`

`--subclasses--()`

Return Value

list of immediate subclasses

`mro()`

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>

continued on next page

Name	Description
<code>__itemsize__</code>	Value: <member ' <code>__itemsize__</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`
Alias nodes.
(Section 14, p. 118)
- **FS:** `scons.Node.FS`
File system nodes.
(Section 15, p. 132)
- **Python:** `scons.Node.Python`
Python nodes.
(Section 16, p. 214)

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 3603 2008/10/10 05:46:...
<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
<code>failed</code>	Value: 5
<code>StateString</code>	Value: {0: 'no_state', 1: 'pending', 2: 'executing', 3: 'up_to_d...
<code>implicit_cache</code>	Value: 0
<code>implicit_deps_unchanged</code>	Value: 0
<code>implicit_deps_changed</code>	Value: 0
<code>arg2nodes_lookups</code>	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
<code>current_version_id</code>	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.

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 rebound 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_bininfo(*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, **kws)`

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 0x83f987c>, cycle_func=<function  
ignore_cycle at 0x840bcd14>, eval_func=<function do_nothing at 0x840bd14>)
```

```
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 3603 2008/10/10 05:46:45 ...'
<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 `exitit`(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 *exitit*(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(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).

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

<code>save_strings(<i>val</i>)</code>

<code>initialize_do_splitdrive()</code>

<code>initialize_normpath_check()</code>
--

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`.

<code>set_duplicate(<i>duplicate</i>)</code>
--

<code>LinkFunc(<i>target</i>, <i>source</i>, <i>env</i>)</code>

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

<code>UnlinkFunc(<i>target</i>, <i>source</i>, <i>env</i>)</code>

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

<code>get_MkdirBuilder()</code>

<code>get_DefaultSCCSBuilder()</code>

<code>get_DefaultRCSBuilder()</code>

<code>do_diskcheck_match(<i>node</i>, <i>predicate</i>, <i>errorfmt</i>)</code>

<code>ignore_diskcheck_match(<i>node</i>, <i>predicate</i>, <i>errorfmt</i>)</code>

<code>do_diskcheck_rcs(<i>node</i>, <i>name</i>)</code>

```
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.

```
invalidate_node_memos(targets)
```

Invalidate the memoized values of all Nodes (files or directories) that are associated with the given entries. Has been added to clear the cache of nodes affected by a direct execution of an action (e.g. Delete/Copy/Chmod). Existing Node caches become inconsistent if the action is run through Execute(). The argument 'targets' can be a single Node object or filename, or a sequence of Nodes/filenames.

15.2 Variables

Name	Description
__revision__	Value: 'src/engine/SCons/Node/FS.py 3603 2008/10/10 05:46:45 scons'

continued on next page

Name	Description
do_store_info	Value: True
default_max_drift	Value: 172800
Save_Strings	Value: False
do_splitdrive	Value: False
needs_normpath_check	Value: False
Valid_Duplicates	Value: ['hard-soft-copy', 'soft-hard-copy', 'hard-copy', 'soft-c...]
Link_Funcs	Value: []
Link	Value: SCons.Action.Action(LinkFunc, None)
LocalCopy	Value: SCons.Action.Action(LinkFunc, LocalString)
Unlink	Value: SCons.Action.Action(UnlinkFunc, None)
Mkdir	Value: SCons.Action.Action(MkdirFunc, None, presub=None)
MkdirBuilder	Value: False
DefaultSCCSBuilder	Value: False
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

<code>__init__(self, type, do, ignore)</code>
<code>set_do(self)</code>
<code>set_ignore(self)</code>
<code>set(self, list)</code>

15.4 Class *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
<code>dictSpecialAttrs</code>	Value: {'abspath': <function __get_abspath at 0x8475764>, 'base'...

15.5 Class Base

```

SCons.Node.Node └─ SCons.Node.FS.Base

```

Known Subclasses: `SCons.Node.FS.Dir`, `SCons.Node.FS.Entry`, `SCons.Node.FS.File`

A generic class for file system entries. This class is for when we don't know yet whether the entry being looked up is a file or a directory. Instances of this class can morph into either `Dir` or `File` objects by a later, more precise lookup.

Note: this class does not define `__cmp__` and `__hash__` for efficiency reasons. `SCons` does a lot of comparing of `Node.FS.{Base,Entry,File,Dir}` objects, so those operations must be as fast as possible, which means we want to use Python's built-in object identity comparisons.

15.5.1 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 klass. Raise an exception if it isn't.

`get_dir(self)`

`get_suffix(self)`

Overrides: SCons.Node.Node.get_suffix

`rfile(self)`

`__str__(self)`

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

`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 0x83b2f44>)

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).

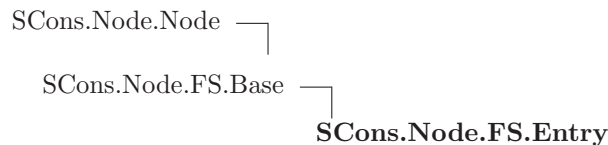
```
visited(self)
```

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



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 *exitit*(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 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.

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 0x83b2f44>)

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*)

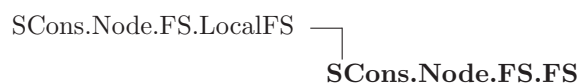
link(*self*, *src*, *dst*)

`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)`

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

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

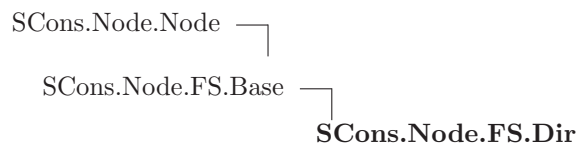
<code>__init__(self, node)</code>

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

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__` extit(inherited documentation)

`diskcheck_match(self)`**`Entry(self, name)`**

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

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

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

`File(self, name)`

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

`link(self, srcdir, duplicate)`

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

`getRepositories(self)`

Returns a list of repositories for this directory.

`get_all_rdirs(self)`**`addRepository(self, dir)`****`up(self)`**

rel_path(*self*, *other*)

 Return a path to "other" relative to this directory.

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

 Overrides: SCons.Node.Node.get_env_scanner

get_target_scanner(*self*)

 Overrides: SCons.Node.Node.get_target_scanner

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

 Return this directory's implicit dependencies.

We don't bother caching the results because the scan typically shouldn't be requested more than once (as opposed to scanning .h file contents, which can be requested as many times as the files is #included by other files).

 Overrides: SCons.Node.Node.get_found_includes

prepare(*self*)

 Prepare for this Node to be built.

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

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

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

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

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

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

 A null "builder" for directories.

 Overrides: SCons.Node.Node.build

multiple_side_effect_has_builder(*self*)

Return whether this Node has a builder or not.

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

Overrides: SCons.Node.Node.multiple_side_effect_has_builder 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 content signatures and names of all our children separated by new-lines. Ensure that the nodes are sorted.

get_csig(*self*)

Compute the content signature for Directory nodes. In general, this is not needed and the content signature is not stored in the DirNodeInfo. However, if `get_contents` on a Dir node is called which has a child directory, the child directory should return the hash of its contents.

Overrides: SCons.Node.Node.get_csig

do_duplicate(*self*, *src*)

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

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

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

Overrides: SCons.Node.Node.changed_since_last_build *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_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 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.

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 0x83b2f44>)
```

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.

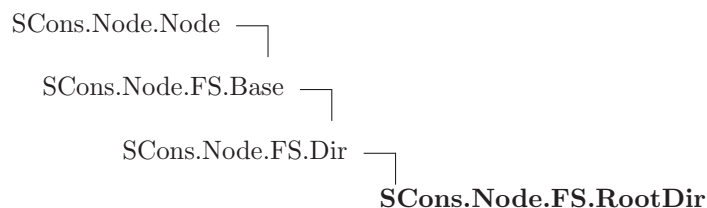
```
visited(self)
```

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 klass. Raise an exception if it isn't.

Overrides: SCons.Node.FS.Base.must_be_same exitit(inherited documentation)

__str__(*self*)

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

Overrides: SCons.Node.FS.Base.__str__ exitit(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 rebound 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 content signatures and names of all our children separated by new-lines. Ensure that the nodes are sorted.

get_csig(*self*)

Compute the content signature for Directory nodes. In general, this is not needed and the content signature is not stored in the DirNodeInfo. However, if get_contents on a Dir node is called which has a child directory, the child directory should return the hash of its contents.

Overrides: SCons.Node.Node.get_csig

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(*self*, *name*)

srcdir_find_file(*self*, *filename*)

srcdir_list(*self*)

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

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 0x83b2f44>)

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).


```
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).

15.12.2 Class Variables

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

15.13 Class *FileInfo*

```
SCons.Node.NodeInfoBase └─ SCons.Node.FS.FileNodeInfo
```

The generic base class for signature information for a *Node*.

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

15.13.1 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)
```

15.13.2 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__(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 (<i>self</i> , <i>other</i>)

prepare_dependencies (<i>self</i>)

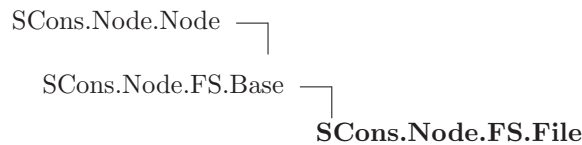
Prepares a FileBuildInfo object for explaining what changed

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

__init__ (<i>self</i> , <i>name</i> , <i>directory</i> , <i>fs</i>)
--

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

Create an entry node named 'name' relative to the SConstruct 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_content_hash**(*self*)

Compute and return the MD5 hash for this file.

get_size(*self*)**get_timestamp**(*self*)**store_info**(*self*)

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

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

convert_old_entry(*self*, *old_entry*)**get_stored_info**(*self*)

Overrides: SCons.Node.Node.get_stored_info

get_stored_implicit(*self*)

Fetch the stored implicit dependencies

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

rel_path(*self*, *other*)

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

Return the included implicit dependencies in this file.
Cache results so we only scan the file once per path
regardless of how many times this information is requested.

Overrides: SCons.Node.Node.get_found_includes

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 extit(inherited documentation)

```
is_up_to_date(self)
```

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

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

rfile(*self*)
 Overrides: SCons.Node.FS.Base.rfile

rstr(*self*)
 A Node.FS.Base object's string representation is its path name.
 Overrides: SCons.Node.FS.Base.rstr extit(inherited documentation)

get_cachedir_csig(*self*)

Fetch a Node's content signature for purposes of computing another Node's cachesig.

This is a wrapper around the normal get_csig() method that handles the somewhat obscure case of using CacheDir with the -n option. Any files that don't exist would normally be "built" by fetching them from the cache, but the normal get_csig() method will try to open up the local file, which doesn't exist because the -n option meant we didn't actually pull the file from cachedir. But since the file *does* actually exist in the cachedir, we can use its contents for the csig.

Overrides: SCons.Node.Node.get_cachedir_csig

get_cachedir_bsig(*self*)

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_binfo(*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.

rentry(*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 extit(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 0x83b2f44>)
```

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: []
md5_chunksize	Value: 64
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 findfile() 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: <code>'src/engine/SCons/Node/Python.py 3603 2008/10/10 05:46:45...'</code>

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

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 *exitit*(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(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).

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
__revision__	Value: 'src/engine/SCons/PathList.py 3603 2008/10/10 05:46:45 sc...
__doc__	Value: ""SCons.PathL...
TYPE_STRING_NO_SUBST	Value: 0
TYPE_STRING_SUBST	Value: False
TYPE_OBJECT	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.

CheckCC(*context*)

CheckCXX(*context*)

CheckSHCC(*context*)

CheckSHCXX(*context*)

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.

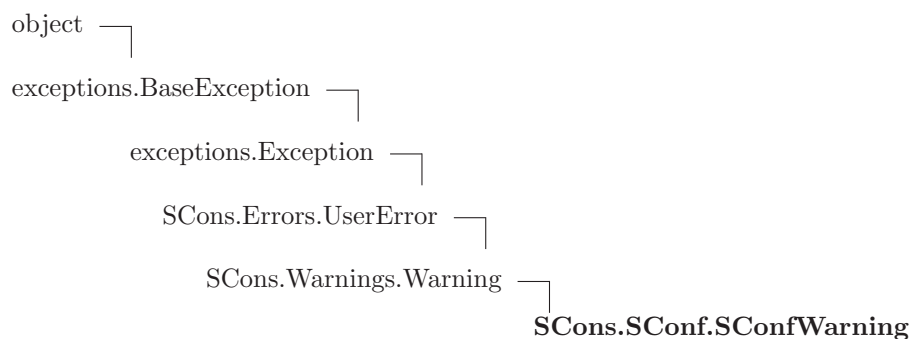
CheckLibWithHeader(*context*, *libs*, *header*, *language*, *call*=False, *autoadd*=False)

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
__revision__	Value: 'src/engine/SCons/SConf.py 3603 2008/10/10 05:46:45 scons'
build_type	Value: False
build_types	Value: ['clean', 'help']
dryrun	Value: 0
AUTO	Value: 0
FORCE	Value: False
CACHE	Value: 2
cache_mode	Value: 0
progress_display	Value: SCons.Util.display
SConfFS	Value: False
sconf_global	Value: False
BooleanTypes	Value: [<type 'int'>, <type 'bool'>]

18.3 Class *SConfWarning*



18.3.1 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__
```

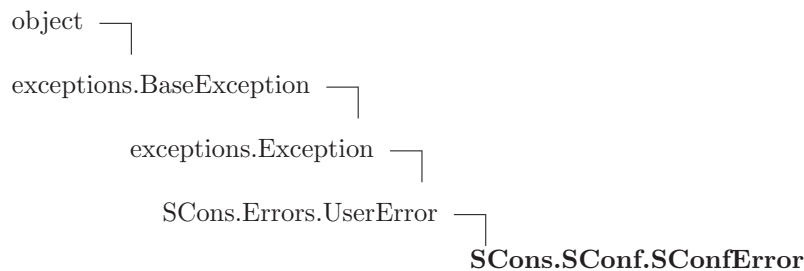
```
__setstate__(...)
```

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

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

__init__(*self*, *msg*)
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__` `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)
```

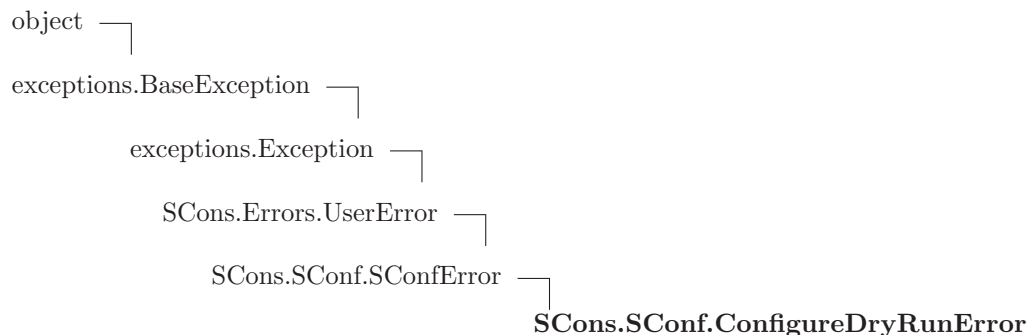
```
str(x)
```

Overrides: `object.__str__`

18.4.2 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

```
__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__ 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__(...)`

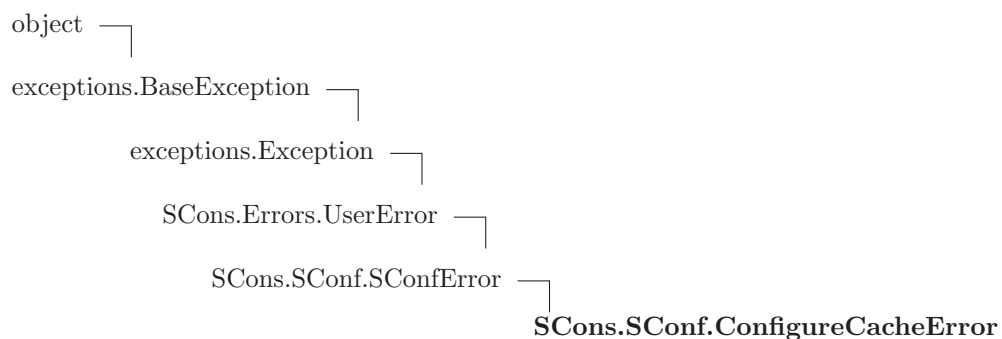
`__str__(x)`

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

18.5.2 Properties

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

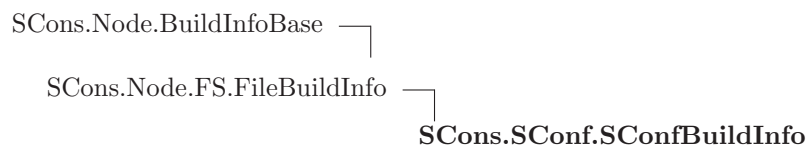
Overrides: `object.__reduce__` `exitit`(inherited documentation)`__reduce_ex__()`

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

18.7 Class 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

<code>set_build_result(<i>self</i>, <i>result</i>, <i>string</i>)</code>
--

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

<code>convert_from_sconsign(<i>self</i>, <i>dir</i>, <i>name</i>)</code>
--

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.

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

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.

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

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

<code>prepare_dependencies(<i>self</i>)</code>
--

Prepares a FileBuildInfo object for explaining what changed

The `bources`, `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 extit(inherited documentation)

display_cached_string(*self*, *bi*)

Logs the original builder messages, given the SConfBuildInfo instance *bi*.

failed(*self*)

Default action when a task fails: stop the build.

Overrides: SCons.Taskmaster.Task.failed extit(inherited documentation)

collect_node_states(*self*)

execute(*self*)

Called to execute the task.

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

Overrides: SCons.Taskmaster.Task.execute extit(inherited documentation)

__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*='\$CONFIGUREDİR', *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*)

CompileSharedObject(*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(*filename*)

Get_DataBase(*dir*)

Reset()

Reset global state. Used by unit tests that end up using SConsign multiple times to get a clean slate for each test.

write()

File(*name*, *dbm_module*=False)

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

19.2 Variables

Name	Description
__revision__	Value: 'src/engine/SCons/SConsign.py 3603 2008/10/10 05:46:45 sc...'
sig_files	Value: []
DataBase	Value: {}
DB_Name	Value: '.sconsign'
DB_sync_list	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__(self)
```

```
convert_to_sconsign(self)
```

```
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.

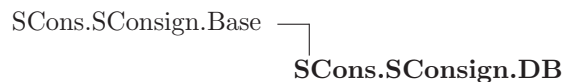
```
do_not_set_entry(self, filename, obj)
```

```
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

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

<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.

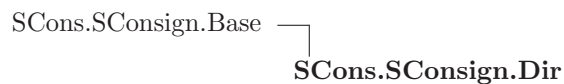
<code>merge(self)</code>

<code>set_entry(self, filename, obj)</code>
--

Set the entry.

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

19.6 Class Dir



Known Subclasses: SCons.SConsign.DirFile

19.6.1 Methods

<code>__init__(self, fp=False, dir=False)</code>
--

<code>fp</code> - file pointer to read entries from

Overrides: <code>SCons.SConsign.Base.__init__</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.

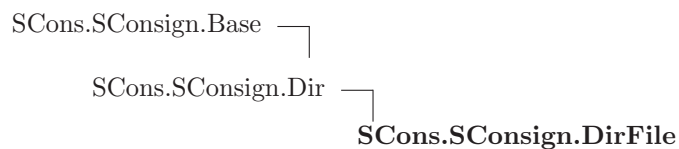
<code>merge(self)</code>

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

Set the entry.

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

19.7 Class *DirFile*



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

19.7.1 Methods

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

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

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

```
write(self, sync=False)
```

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(self, filename, obj)
```

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

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

Fetch the specified entry attribute.

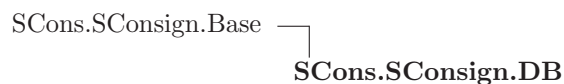
```
merge(self)
```

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

Set the entry.

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

19.8 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

```
__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)
```


20 Package SCons.Scanner

SCons.Scanner

The Scanner package for the SCons software construction utility.

20.1 Modules

- **C:** SCons.Scanner.C
This module implements the dependency scanner for C/C++ code.
(Section 21, p. 265)
- **D:** SCons.Scanner.D
Scanner for the Digital Mars "D" programming language.
(Section 22, p. 270)
- **Dir** (Section 23, p. 272)
- **Fortran:** SCons.Scanner.Fortran
This module implements the dependency scanner for Fortran code.
(Section 24, p. 273)
- **IDL:** SCons.Scanner.IDL
This module implements the dependency scanner for IDL (Interface Definition Language) files.
(Section 25, p. 275)
- **LaTeX:** SCons.Scanner.LaTeX
This module implements the dependency scanner for LaTeX code.
(Section 26, p. 276)
- **Prog** (Section 27, p. 281)
- **RC:** SCons.Scanner.RC
This module implements the dependency scanner for RC (Interface Definition Language) files.
(Section 28, p. 282)

20.2 Functions

Scanner(*function*, **args*, ***kw*)

Public interface factory function for creating different types of Scanners based on the different types of "functions" that may be supplied.

TODO: Deprecate this some day. We've moved the functionality inside the Base class and really don't need this factory function any more. It was, however, used by some of our Tool modules, so the call probably ended up in various people's custom modules patterned on SCons code.

20.3 Variables

Name	Description
__revision__	Value: 'src/engine/SCons/Scanner/__init__.py 3603 2008/10/10 05:...

continued on next page

Name	Description
------	-------------

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, SCons.Scanner.LaTeX.LaTeX

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

20.5.1 Methods

```
__call__(self, node, env, path=())
```

This method scans a single object. 'node' is the node that will be passed to the scanner function, and 'env' is the environment that will be passed to the scanner function. A list of direct dependency nodes for the specified node will be returned.

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

```
__hash__(self)
```

```
__init__(self, function, name='NONE', argument=<class SCons.Scanner.Null at 0x879ad4c>,
keys=<class SCons.Scanner.Null at 0x879ad4c>, path_function=False, node_class=<class
SCons.Node.FS.Entry at 0x846fb6c>, 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.

'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)
```

`__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)`

20.6 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.

'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.

270

Examples:

```
s = Scanner(my_scanner_function)
```

```
s = Scanner(function = my_scanner_function)
```

```
__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__* *exitit*(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.

'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.

273

Examples:

```
s = Scanner(my_scanner_function)
```

```
s = Scanner(function = my_scanner_function)
```

```
--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

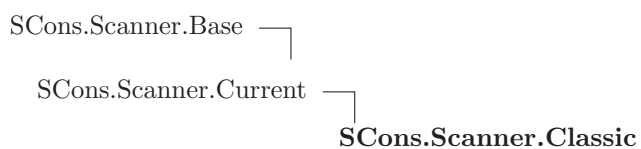
```
get_skeys(self, env=False)
```

```
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

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)`

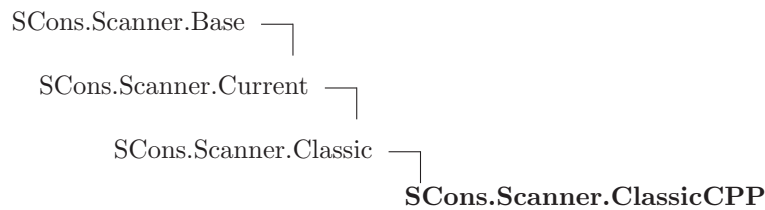
`recurse_nodes(self, nodes)`

`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 (<i>self</i> , <i>include</i> , <i>source_dir</i> , <i>path</i>) Overrides: SCons.Scanner.Classic.find_include

sort_key (<i>self</i> , <i>include</i>) Overrides: SCons.Scanner.Classic.sort_key

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

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

__hash__ (<i>self</i>)

__init__ (<i>self</i> , <i>name</i> , <i>suffixes</i> , <i>path_variable</i> , <i>reger</i> , <i>*args</i> , <i>**kw</i>) Overrides: SCons.Scanner.Current.__init__

__str__ (<i>self</i>)

add_scanner (<i>self</i> , <i>skey</i> , <i>scanner</i>)

add_skey (<i>self</i> , <i>skey</i>)

Add a skey to the list of skeys

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

```
get_keys(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 3603 2008/10/10 05:46:45 s...

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: <i>SCons.cpp.PreProcessor.__init__</i>

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

Overrides: <i>SCons.cpp.PreProcessor.initialize_result</i>
--

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

Overrides: <i>SCons.cpp.PreProcessor.finalize_result</i>
--

find_include_file(*self*, *t*)

Finds the #include file for a given preprocessor tuple.

Overrides: SCons.cpp.PreProcessor.find_include_file extit(inherited documentation)

read_file(*self*, *file*)

Overrides: SCons.cpp.PreProcessor.read_file

__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

`__init__(self, name, variable)`

`__call__(self, node, env, path=())`

`recurse_nodes(self, nodes)`

`select(self, node)`

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

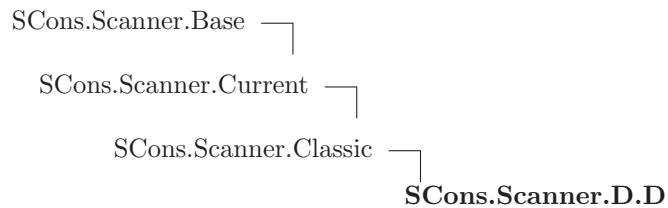
DScanner()

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 3603 2008/10/10 05:46:45 s...

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 3603 2008/10/10 05:46:45...'
<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

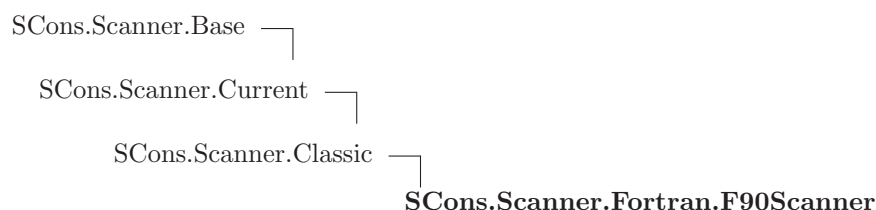
FortranScan(*path_variable*='FORTRANPATH')

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 3603 2008/10/10 05:4...'

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, 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)
```

```
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
__revision__	Value: 'src/engine/SCons/Scanner/IDL.py 3603 2008/10/10 05:46:45...'

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 when built with latex.

PDFLaTeXScanner()

Return a prototype Scanner instance for scanning LaTeX source files when built with pdflatex.

26.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/LaTeX.py 3603 2008/10/10 05:46:...

26.3 Class LaTeX

```

SCons.Scanner.Base └─ SCons.Scanner.LaTeX.LaTeX

```

Class for scanning LaTeX files for included files.

Unlike most scanners, which use regular expressions that just return the included file name, this returns a tuple consisting of the keyword for the inclusion ("include", "includegraphics", "input", or "bibliography"), and then the file name itself. Based on a quick look at LaTeX documentation, it seems that we should append .tex suffix for the "include" keywords, append .tex if there is no extension for the "input" keyword, and need to add .bib for the "bibliography" keyword that does not accept extensions by itself.

Finally, if there is no extension for an "includegraphics" keyword latex will append .ps or .eps to find the file, while pdftex may use .pdf, .jpg, .tif, .mps, or .png.

The actual subset and search order may be altered by `DeclareGraphicsExtensions` command. This complication is ignored. The default order corresponds to experimentation with teTeX

```
$ latex --version
```

```
pdfTeX 3.141592-1.21a-2.2 (Web2C 7.5.4)
kpathsea version 3.5.4
```

The order is:

```
['.eps', '.ps'] for latex
['.png', '.pdf', '.jpg', '.tif'].
```

Another difference is that the search path is determined by the type of the file being searched:

```
env['TEXINPUTS'] for "input" and "include" keywords
env['TEXINPUTS'] for "includegraphics" keyword
env['BIBINPUTS'] for "bibliography" keyword
env['BSTINPUTS'] for "bibliographystyle" keyword
```

FIXME: also look for the class or style in `document[class|style]{}`

FIXME: also look for the argument of `bibliographystyle{}`

26.3.1 Methods

```
__init__(self, name, suffixes, graphics_extensions, *args, **kw)
```

Construct a new scanner object given a scanner function.

'function' - a scanner function taking two or three arguments and returning a list of strings.

'name' - a name for identifying this scanner object.

'argument' - an optional argument that, if specified, will be passed to both the scanner function and the path_function.

'skeys' - an optional list argument that can be used to determine which scanner should be used for a given Node. In the case of File nodes, for example, the 'skeys' would be file suffixes.

'path_function' - a function that takes four or five arguments (a construction environment, Node for the directory containing the SConscript file that defined the primary target, list of target nodes, list of source nodes, and optional argument for this instance) and returns a tuple of the directories that can be searched for implicit dependency files. May also return a callable() which is called with no args and returns the tuple (supporting Bindable class).

'node_class' - the class of Nodes which this scan will return. If node_class is None, then this scanner will not enforce any Node conversion and will return the raw results from the underlying scanner function.

'node_factory' - the factory function to be called to translate the raw results returned by the scanner function into the expected node_class objects.

'scan_check' - a function to be called to first check whether this node really needs to be scanned.

'recursive' - specifies that this scanner should be invoked recursively on all of the implicit dependencies it returns (the canonical example being #include lines in C source files). May be a callable, which will be called to filter the list of nodes found to select a subset for recursive scanning (the canonical example being only recursively scanning subdirectories within a directory).

The scanner function's first argument will be a Node that should be scanned for dependencies, the second argument will be an Environment object, the third argument will be the tuple of paths returned by the path_function, and the fourth argument will be the value passed into 'argument', and the returned list should contain the Nodes for all the direct dependencies of the file.

292

Examples:

```
s = Scanner(my_scanner_function)
```

```
s = Scanner(function = my_scanner_function)
```

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

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

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

```
__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)
```

26.3.2 Class Variables

Name	Description
keyword_paths	Value: {'bibliography': 'BIBINPUTS', 'bibliographystyle': 'BSTIN...
env_variables	Value: ['BIBINPUTS', 'TEXINPUTS', 'BSTINPUTS']

27 Module SCons.Scanner.Prog

27.1 Functions

ProgramScanner(***kw*)

Return a prototype Scanner instance for scanning executable files for static-lib dependencies

scan(*node*, *env*, *libpath*=())

This scanner scans program files for static-library dependencies. It will search the LIBPATH environment variable for libraries specified in the LIBS variable, returning any files it finds as dependencies.

27.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Scanner/Prog.py 3603 2008/10/10 05:46:4...
<code>print_find_libs</code>	Value: False

28 Module SCons.Scanner.RC

SCons.Scanner.RC

This module implements the dependency scanner for RC (Interface Definition Language) files.

28.1 Functions

RCSan()
Return a prototype Scanner instance for scanning RC source files

28.2 Variables

Name	Description
__revision__	Value: 'src/engine/SCons/Scanner/RC.py 3603 2008/10/10 05:46:45 ...

29 Package SCons.Script

SCons.Script

This file implements the `main()` function used by the `scons` script.

Architecturally, this *is* the `scons` script, and will likely only be called from the external `"scons"` wrapper. Consequently, anything here should not be, or be considered, part of the build engine. If it's something that we expect other software to want to use, it should go in some other module. If it's specific to the `"scons"` script invocation, it goes here.

29.1 Modules

- **Interactive:** SCons interactive mode
(Section 30, p. 290)
- **Main:** SCons.Script
This file implements the `main()` function used by the `scons` script.
(Section 31, p. 294)
- **SConscript':** SCons.Script.SConscript
This module defines the Python API provided to SConscript and SConstruct files.
(Section 32, p. 311)

29.2 Functions

HelpFunction (<i>text</i>)

Variables (<i>files</i> =[], <i>args</i> ={})

Options (<i>files</i> =[], <i>args</i> ={})

29.3 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Script/__init__.py 3603 2008/10/10 05:4...
<code>start_time</code>	Value: 1223643003.04
<code>call_stack</code>	Value: []
<code>PathVariable</code>	Value: <SCons.Variables.PathVariable.PathVariableClass instance...>
<code>PathOption</code>	Value: <SCons.Variables.PathVariable.PathVariableClass instance...>
<code>Chmod</code>	Value: ActionFactory(chmod_func, chmod_strfunc)
<code>Copy</code>	Value: ActionFactory(copy_func, lambda dest, src: 'Copy("%s", "%...'>
<code>Delete</code>	Value: ActionFactory(delete_func, delete_strfunc)

continued on next page

Name	Description
Mkdir	Value: ActionFactory(mkdir_func, lambda dir: 'Mkdir(%s)' % get_p...
Move	Value: ActionFactory(move_func, lambda dest, src: 'Move("%s", "%...
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
sconscript_reading	Value: 0
GlobalDefaultEnvironmentFunctions	Value: ['Default', 'EnsurePythonVersion', 'EnsureSConsVersion', ...]
GlobalDefaultBuilders	Value: ['CFile', 'CXXFile', 'DVI', 'Jar', 'Java', 'JavaH', 'Libr...
SConscript	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Command	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
AddPostAction	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
AddPreAction	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Alias	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
AlwaysBuild	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
BuildDir	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
CFile	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
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 ...

continued on next page

Name	Description
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 ...
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 ...

continued on next page

Name	Description
Library	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Literal	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
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 ...
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 ...

continued on next page

Name	Description
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 ...
VariantDir	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...
Zip	Value: <SCons.Script.SConscript.DefaultEnvironmentCall instance ...

29.4 Class TargetList

UserList.UserList —
SCons.Script.TargetList

29.4.1 Methods

<code>--add--(self, other)</code>
<code>--cmp--(self, other)</code>
<code>--contains--(self, item)</code>
<code>--delitem--(self, i)</code>
<code>--delslice--(self, i, j)</code>
<code>--eq--(self, other)</code>
<code>--ge--(self, other)</code>
<code>--getitem--(self, i)</code>

`--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)
```

30 Module SCons.Script.Interactive

SCons interactive mode

30.1 Functions

<code>interact(<i>fs, parser, options, targets, target_top</i>)</code>
--

30.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Script/Interactive.py 3603 2008/10/10 0...
<code>__doc__</code>	Value: ...

30.3 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.

30.3.1 Methods

<p><code>__init__(<i>self, **kw</i>)</code> Instantiate a line-oriented interpreter framework.</p> <p>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.</p> <p>Overrides: cmd.Cmd.__init__ exitit(inherited documentation)</p>
--

default(*self*, *argv*)

Called on an input line when the command prefix is not recognized.

If this method is not overridden, it prints an error message and returns.

Overrides: cmd.Cmd.default 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*)**30.3.2 Class Variables**

Name	Description
<code>synonyms</code>	Value: <code>{'b': 'build', 'c': 'clean', 'h': 'help', 'scons': 'build...'}</code>
<code>doc_header</code>	Value: <code>'Documented commands (type help <topic>):'</code>
<code>doc_leader</code>	Value: <code>''</code>
<code>identchars</code>	Value: <code>'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789_.'</code>
<code>intro</code>	Value: <code>False</code>
<code>lastcmd</code>	Value: <code>''</code>
<code>misc_header</code>	Value: <code>'Miscellaneous help topics:'</code>
<code>nohelp</code>	Value: <code>'*** No help on %s'</code>
<code>prompt</code>	Value: <code>'(Cmd) '</code>
<code>ruler</code>	Value: <code>'='</code>
<code>undoc_header</code>	Value: <code>'Undocumented commands:'</code>
<code>use_rawinput</code>	Value: <code>False</code>

31 Module SCons.Script.Main

SCons.Script

This file implements the `main()` function used by the `scons` script.

Architecturally, this *is* the `scons` script, and will likely only be called from the external "scons" wrapper. Consequently, anything here should not be, or be considered, part of the build engine. If it's something that we expect other software to want to use, it should go in some other module. If it's specific to the "scons" script invocation, it goes here.

31.1 Functions

<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, 2, 'final', 0))</code>
--

<code>python_version_deprecated(version=(2, 5, 2, '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 `traceback.extract_tb()` or `traceback.extract_stack()`

<code>version_string(label, module)</code>
--

<code>main()</code>

31.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Script/Main.py 3603 2008/10/10 05:46:45...'
<code>display</code>	Value: <code>SCons.Util.display</code>
<code>progress_display</code>	Value: <code>SCons.Util.DisplayEngine()</code>
<code>first_command_start</code>	Value: <code>False</code>
<code>last_command_end</code>	Value: <code>False</code>
<code>ProgressObject</code>	Value: <code>Null()</code>
<code>print_objects</code>	Value: <code>0</code>
<code>print_memoizer</code>	Value: <code>0</code>
<code>print_stacktrace</code>	Value: <code>0</code>
<code>print_time</code>	Value: <code>0</code>
<code>sconscript_time</code>	Value: <code>0</code>
<code>cumulative_command_time</code>	Value: <code>0</code>
<code>exit_status</code>	Value: <code>0</code>
<code>this_build_status</code>	Value: <code>0</code>
<code>num_jobs</code>	Value: <code>False</code>
<code>delayed_warnings</code>	Value: <code>[]</code>
<code>OptionsParser</code>	Value: <code>FakeOptionParser()</code>
<code>count_stats</code>	Value: <code>CountStats()</code>
<code>memory_stats</code>	Value: <code>MemStats()</code>

31.3 Class SConsPrintHelpException



31.3.1 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 signatureOverrides: `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__` `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__`**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 Progressor

31.4.1 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)
```

31.4.2 Class Variables

Name	Description
<code>prev</code>	Value: <code>''</code>
<code>count</code>	Value: <code>0</code>
<code>target_string</code>	Value: <code>'\$TARGET'</code>

31.5 Class BuildTask

```

SCons.Taskmaster.Task └─ SCons.Script.Main.BuildTask

```

An SCons build task.

31.5.1 Methods

display(*self*, *message*)

Hook to allow the calling interface to display a message.

This hook gets called as part of preparing a task for execution (that is, a Node to be built). As part of figuring out what Node should be built next, the actually target list may be altered, along with a message describing the alteration. The calling interface can subclass Task and provide a concrete implementation of this method to see those messages.

Overrides: SCons.Taskmaster.Task.display extit(inherited documentation)

prepare(*self*)

Called just before the task is executed.

This is mainly intended to give the target Nodes a chance to unlink underlying files and make all necessary directories before the Action is actually called to build the targets.

Overrides: SCons.Taskmaster.Task.prepare extit(inherited documentation)

needs_execute(*self*)

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 extit(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.

31.5.2 Class Variables

Name	Description
progress	Value: Null()

31.6 Class CleanTask

SCons.Taskmaster.Task  SCons.Script.Main.CleanTask

An SCons clean task.

31.6.1 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 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_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.

31.7 Class *QuestionTask*



An SCons task for the `-q` (question) option.

31.7.1 Methods

`prepare(self)`

Called just before the task is executed.

This is mainly intended to give the target Nodes a chance to unlink underlying files and make all necessary directories before the Action is actually called to build the targets.

Overrides: `SCons.Taskmaster.Task.prepare` extit(inherited documentation)

`execute(self)`

Called to execute the task.

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

Overrides: `SCons.Taskmaster.Task.execute` extit(inherited documentation)

`executed(self)`

Called when the task has been successfully executed and the Taskmaster instance wants to call the Node's callback methods.

This may have been a do-nothing operation (to preserve build order), so we must check the node's state before deciding whether it was "built", in which case we call the appropriate Node method. In any event, we always call "`visited()`", which will handle any post-visit actions that must take place regardless of whether or not the target was an actual built target or a source Node.

Overrides: `SCons.Taskmaster.Task.executed` extit(inherited documentation)

`__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.

31.8 Class *TreePrinter*

31.8.1 Methods

`__init__(self, derived=False, prune=False, status=False)`

`get_all_children(self, node)`

`get_derived_children(self, node)`

`display(self, t)`

31.9 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.

31.9.1 Methods

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

31.9.2 Class Variables

Name	Description
values	Value: FakeOptionValues()

31.10 Class Stats

Known Subclasses: SCons.Script.Main.CountStats, SCons.Script.Main.MemStats

31.10.1 Methods

<code>__init__(self)</code>

<code>enable(self, outfp)</code>

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

31.11 Class CountStats

```

SCons.Script.Main.Stats └─ SCons.Script.Main.CountStats

```

31.11.1 Methods

<code>do_append(self, label)</code>

<code>do_print(self)</code>

<code>__init__(self)</code>

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

<code>enable(self, outfp)</code>

31.12 Class MemStats

SCons.Script.Main.Stats —
SCons.Script.Main.MemStats

31.12.1 Methods

<code>do_append(<i>self</i>, <i>label</i>)</code>

<code>do_print(<i>self</i>)</code>

<code>__init__(<i>self</i>)</code>

<code>do_nothing(<i>self</i>, *<i>args</i>, **<i>kw</i>)</code>

<code>enable(<i>self</i>, <i>outfp</i>)</code>
--

32 Module SCons.Script.SConscript'

SCons.Script.SConscript

This module defines the Python API provided to SConscript and SConstruct files.

32.1 Functions

get_calling_namespaces()

Return the locals and globals for the function that called into this module in the current call stack.

compute_exports(*exports*)

Compute a dictionary of exports given one of the parameters to the Export() function or the exports argument to SConscript().

Return(vars*, ***kw*)**

SConscript_exception(*file*=sys.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.

32.2 Variables

Name	Description
<code>--revision--</code>	Value: 'src/engine/SCons/Script/SConscript.py 3603 2008/10/10 05...

continued on next page

Name	Description
launch_dir	Value: <code>'/home/scons/release'</code>
GlobalDict	Value: <code>False</code>
global_exports	Value: <code>{}</code>
sconscript_chdir	Value: <code>False</code>
call_stack	Value: <code>[]</code>
stack_bottom	Value: <code>'% Stack boTTom %'</code>

32.3 Class *SConscriptReturn*



32.3.1 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__`

32.3.2 Properties

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

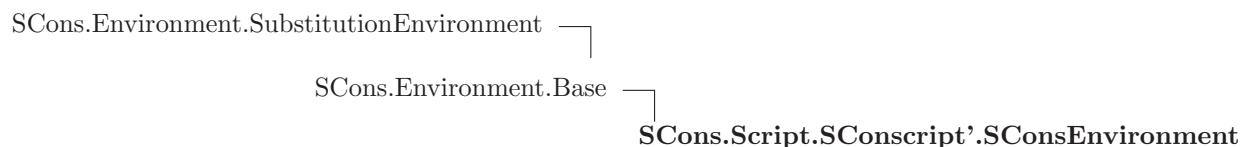
32.4 Class Frame

A frame on the SConstruct/SConscript call stack

32.4.1 Methods

```
__init__(self, fs, exports, sconscript)
```

32.5 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.

32.5.1 Methods

```
Configure(self, *args, **kw)
Overrides: SCons.Environment.Base.Configure
```

```
Default(self, *targets)
```

```
EnsureSConsVersion(self, major, minor, revision=0)
Exit abnormally if the SCons version is not late enough.
```

```
EnsurePythonVersion(self, major, minor)
Exit abnormally if the Python version is not late enough.
```

```
Exit(self, value=0)
```

```
Export(self, *vars)
```

```
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*=':', *delete_existing*=False)

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

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

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

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

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

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

CacheDir(*self*, *path*)

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

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

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

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*(':', *delete_existing*=False)

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

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

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

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

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 0x87a886c>, *lookup_list*=<class SCons.Environment.Null at 0x87a886c>, ***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.

32.5.2 Class Variables

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

32.6 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).

32.6.1 Methods

```
__init__(self, method_name, subst=0)
```

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


33 Module SCons.Sig

Place-holder for the old SCons.Sig module hierarchy

This is no longer used, but code out there (such as the NSIS module on the SCons wiki) may try to import SCons.Sig. If so, we generate a warning that points them to the line that caused the import, and don't die.

If someone actually tried to use the sub-modules or functions within the package (for example, SCons.Sig.MD5.signature()), then they'll still get an AttributeError, but at least they'll know where to start looking.

33.1 Variables

Name	Description
<code>--revision--</code>	Value: 'src/engine/SCons/Sig.py 3603 2008/10/10 05:46:45 scons'
<code>--doc--</code>	Value: ""Place-holder for the old SCons.Sig module hierar...
<code>msg</code>	Value: 'The SCons.Sig module no longer exists.\n Remove the f...
<code>default_calc</code>	Value: False
<code>default_module</code>	Value: False
<code>MD5</code>	Value: MD5Null()
<code>TimeStamp</code>	Value: TimeStampNull()

33.2 Class MD5Null

```

SCons.Util.Null └─ SCons.Sig.MD5Null

```

33.2.1 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)
```

```
__setattr__(self, name, value)
```

33.3 Class `TimeStampNull`



33.3.1 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)
```

```
__setattr__(self, name, value)
```

34 Module SCons.Subst

SCons.Subst

SCons string substitution.

34.1 Functions

SetAllowableExceptions(**excepts*)

raise_exception(*exception*, *target*, *s*)

quote_spaces(*arg*)

Generic function for putting double quotes around any string that has white space in it.

escape_list(*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...

34.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/Subst.py 3603 2008/10/10 05:46:45 scons'
<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

34.3 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.

34.3.1 Methods

```
__init__(self, lstr)
```

```
__str__(self)
```

```
escape(self, escape_func)
```

```
for_signature(self)
```

```
is_literal(self)
```

34.4 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.

34.4.1 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)
```

34.5 Class CmdStringHolder

```
UserString.UserString └─ SCons.Subst.CmdStringHolder
```

This is a special class used to hold strings generated by scons_subst() and scons_subst_list(). It defines a special method escape(). When passed a function with an escape algorithm for a particular platform, it will return the contained string with the proper escape sequences inserted.

34.5.1 Methods

```
__init__(self, cmd, literal=False)
```

Overrides: UserString.UserString.__init__

is_literal(*self*)**escape**(*self*, *escape_func*, *quote_func*=<function quote_spaces at 0x8469764>)

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=2147483647)``decode(self, encoding=False, errors=False)``encode(self, encoding=False, errors=False)``endswith(self, suffix, start=0, end=2147483647)``expandtabs(self, tabsize=8)``find(self, sub, start=0, end=2147483647)``index(self, sub, start=0, end=2147483647)``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=2147483647)``rindex(self, sub, start=0, end=2147483647)``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=2147483647)``strip(self, chars=False)``swapcase(self)``title(self)``translate(self, *args)``upper(self)``zfill(self, width)`

34.6 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...

34.6.1 Methods

`__init__(self, list, func)`

34.7 Class `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.

34.7.1 Methods

<code>--init--(self, nl)</code> Overrides: <code>UserList.UserList.__init__</code>

<code>--getattr--(self, attr)</code>

<code>--getitem--(self, i)</code> Overrides: <code>UserList.UserList.__getitem__</code>
--

<code>--getslice--(self, i, j)</code> Overrides: <code>UserList.UserList.__getslice__</code>

<code>--str--(self)</code>

<code>--repr--(self)</code> Overrides: <code>UserList.UserList.__repr__</code>

<code>--add--(self, other)</code>

<code>--cmp--(self, other)</code>

<code>--contains--(self, item)</code>

<code>--delitem--(self, i)</code>

<code>--delslice--(self, i, j)</code>

<code>--eq--(self, other)</code>

<code>--ge--(self, other)</code>

<code>--gt--(self, other)</code>

`__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)`

34.8 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.

34.8.1 Methods

<code>--init--(<i>self</i>, <i>nl</i>)</code>

<code>--getattr--(<i>self</i>, <i>attr</i>)</code>
--

<code>--str--(<i>self</i>)</code>

<code>--repr--(<i>self</i>)</code>

35 Module SCons.Taskmaster

Generic Taskmaster module for the SCons build engine.

This module contains the primary interface(s) between a wrapping user interface and the SCons build engine. There are two key classes here:

Taskmaster

This is the main engine for walking the dependency graph and calling things to decide what does or doesn't need to be built.

Task

This is the base class for allowing a wrapping interface to decide what does or doesn't actually need to be done. The intention is for a wrapping interface to subclass this as appropriate for different types of behavior it may need.

The canonical example is the SCons native Python interface, which has Task subclasses that handle its specific behavior, like printing "'foo' is up to date" when a top-level target doesn't need to be built, and handling the -c option by removing targets as its "build" action. There is also a separate subclass for suppressing this output when the -q option is used.

The Taskmaster instantiates a Task object for each (set of) target(s) that it decides need to be evaluated and/or built.

35.1 Functions

<code>dump_stats()</code>

<code>find_cycle(stack, visited)</code>

35.2 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>__revision__</code>	Value: 'src/engine/SCons/Taskmaster.py 3603 2008/10/10 05:46:45 ...
<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

continued on next page

Name	Description
CollectStats	Value: False
StatsNodes	Value: []
fmt	Value: '%(considered)3d %(already_handled)3d %(problem)3d %(chil...

35.3 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.)

35.3.1 Methods

<code>__init__(self)</code>
Instantiates a Taskmaster.Stats object, initializing all appropriate counters to zero.

35.4 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.

35.4.1 Methods

<code>__init__(self, tm, targets, top, node)</code>

display(*self*, *message*)

Hook to allow the calling interface to display a message.

This hook gets called as part of preparing a task for execution (that is, a Node to be built). As part of figuring out what Node should be built next, the actually target list may be altered, along with a message describing the alteration. The calling interface can subclass Task and provide a concrete implementation of this method to see those messages.

prepare(*self*)

Called just before the task is executed.

This is mainly intended to give the target Nodes a chance to unlink underlying files and make all necessary directories before the Action is actually called to build the targets.

get_target(*self*)

Fetch the target being built or updated by this task.

needs_execute(*self*)

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

35.5 Class Taskmaster

The Taskmaster for walking the dependency DAG.

35.5.1 Methods

```
__init__(self, targets=[], tasker=<class SCons.Taskmaster.Task at 0x8b63b9c>, 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 0x8b6b25c>)

Perform clean-up about nodes that will never be built.

stop(*self*)

Stops the current build completely.

cleanup(*self*)

Check for dependency cycles.

36 Module SCons.Util

SCons.Util

Various utility functions go here.

36.1 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*.

splittext(*path*)

Same as `os.path.splittext()` but faster.

updrive(*path*)

Make the drive letter (if any) upper case.
This is useful because Windows is inconsitent on the case of the drive letter, which can cause inconsistencies when calculating command signatures.

get_environment_var(*varstr*)

Given a string, first determine if it looks like a reference to a single environment variable, like "\$FOO" or "\${FOO}". If so, return that variable with no decorations ("FOO"). If not, return None.

```
render_tree(root, child_func, prune=0, margin=[0], visited={})
```

Render a tree of nodes into an ASCII tree view.

root - the root node of the tree

child_func - the function called to get the children of a node

prune - don't visit the same node twice

margin - the format of the left margin to use for 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, isinstance=<built-in function isinstance>, StringTypes=(<type 'str'>, <type 'unicode'>, <class UserString.UserString at 0xb7befb6c>), SequenceTypes=(<type 'list'>, <type 'tuple'>, <class UserList.UserList at 0x83b13e4>), do_flatten=<function do_flatten at 0x83b13e4>)
```

Flatten a sequence to a non-nested list.

Flatten() converts either a single scalar or a nested sequence to a non-nested list. Note that flatten() considers strings to be scalars instead of sequences like Python would.

```
flatten_sequence(sequence, isinstance=<built-in function isinstance>, StringTypes=(<type 'str'>, <type 'unicode'>, <class UserString.UserString at 0xb7befb6c>), SequenceTypes=(<type 'list'>, <type 'tuple'>, <class UserList.UserList at 0x83b13e4>), do_flatten=<function do_flatten at 0x83b13e4>)
```

Flatten a sequence to a non-nested list.

Same as flatten(), but it does not handle the single scalar case. This is slightly more efficient when one knows that the sequence to flatten can not be a scalar.

```
to_String(s, isinstance=<built-in function isinstance>, str=<type 'str'>, UserString=<class UserString.UserString at 0xb7befb6c>, BaseStringTypes=(<type 'str'>, <type 'unicode'>))
```

```
to_String_for_subst(s, isinstance=<built-in function isinstance>, join=<function join at 0xb7d7be2c>, str=<type 'str'>, to_String=<function to_String at 0x83b148c>, BaseStringTypes=(<type 'str'>, <type 'unicode'>), SequenceTypes=(<type 'list'>, <type 'tuple'>, <class UserList.UserList at 0x83b13e4>), UserString=<class UserString.UserString at 0xb7befb6c>)
```

```
to_String_for_signature(obj, to_String_for_subst=<function to_String_for_subst at 0x83b14c4>, 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, *pathext*=False, *reject*=[])**PrependPath**(*oldpath*, *newpath*, *sep*=':', *delete_existing*=False)

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 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:   "/biz/boom:/foo:/foo/bar"
```

If `delete_existing` is 0, then adding a path that exists will not move it to the beginning; it will stay where it is in the list.

AppendPath(*oldpath*, *newpath*, *sep*=':', *delete_existing*=False)

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"
```

If `delete_existing` is 0, then adding a path that exists will not move it to the end; it will stay where it is in the list.

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 $O(N \log_2 N)$ 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*)

MD5filesignature(*fname*, *chunksize*=65536)

MD5collect(*signatures*)

Collects a list of signatures into an aggregate signature.

signatures - a list of signatures
returns - the aggregate signature

36.2 Variables

Name	Description
UnicodeType	Value: types.UnicodeType
DictTypes	Value: dict, UserDict
ListTypes	Value: list, UserList
SequenceTypes	Value: (<type 'list'>, <type 'tuple'>, <class UserList.UserList ...

continued on next page

Name	Description
StringTypes	Value: (<type 'str'>, <type 'unicode'>, <class UserString.UserSt...
BaseStringTypes	Value: (<type 'str'>, <type 'unicode'>)
d	Value: {<type 'instance'>: <function _semi_deepcopy_inst at 0x83...
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
HKEY_CURRENT_USER	Value: hkey_mod.HKEY_CURRENT_USER
HKEY_USERS	Value: hkey_mod.HKEY_USERS
display	Value: SCons.Util.display
md5	Value: True

36.3 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.

36.3.1 Methods

<code>--call--(self, *args, **kwargs)</code>
<code>--add--(self, other)</code>
<code>--cmp--(self, other)</code>
<code>--contains--(self, item)</code>
<code>--delitem--(self, i)</code>
<code>--delslice--(self, i, j)</code>
<code>--eq--(self, other)</code>
<code>--ge--(self, other)</code>
<code>--getitem--(self, i)</code>

```
__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)
```

36.4 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' ]
```

36.4.1 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)
```

36.5 Class DisplayEngine

36.5.1 Methods

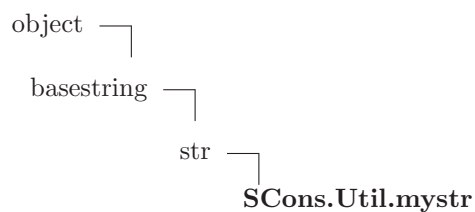
```
__init__(self)
```

```
print_it(self, text, append_newline=False)
```

```
dont_print(self, text, append_newline=False)
```

```
set_mode(self, mode)
```

36.6 Class mystr



36.6.1 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

lstrip(*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

36.6.2 Properties

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

36.7 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

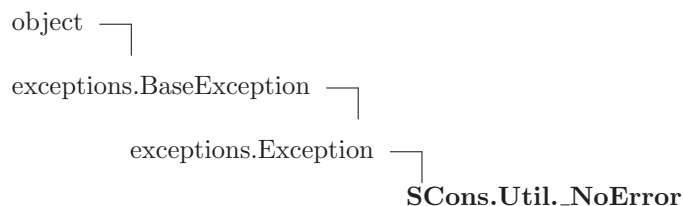
```
x = objA.var1
```

Inherit from this class to create a Proxy.

36.7.1 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>

36.8 Class `_NoError`



36.8.1 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__ 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__
```

36.8.2 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>

36.9 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.

36.9.1 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)`

36.10 Class `OrderedDict`

```

UserDict.UserDict └─
                    SCons.Util.OrderedDict

```

Known Subclasses: `SCons.Util.Selector`

36.10.1 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*)

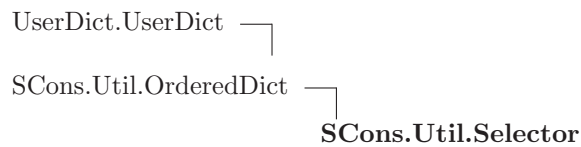
iteritems(*self*)

iterkeys(*self*)

itervalues(*self*)

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

36.11 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.

36.11.1 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 (<i>self</i>) Overrides: UserDict.UserDict.items
--

iteritems (<i>self</i>)

iterkeys (<i>self</i>)

itervalues (<i>self</i>)

keys (<i>self</i>) Overrides: UserDict.UserDict.keys
--

pop (<i>self</i> , <i>key</i> , * <i>args</i>)

popitem (<i>self</i>) Overrides: UserDict.UserDict.popitem
--

setdefault (<i>self</i> , <i>key</i> , <i>failobj</i> =False) Overrides: UserDict.UserDict.setdefault
--

update (<i>self</i> , <i>dict</i>) Overrides: UserDict.UserDict.update
--

values (<i>self</i>) Overrides: UserDict.UserDict.values
--

36.12 Class *LogicalLines*

36.12.1 Methods

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

readline (<i>self</i>)

readlines (<i>self</i>)

36.13 Class *UniqueList*

```

UserList.UserList └─ SCons.Util.UniqueList

```

36.13.1 Methods

__init__ (<i>self</i> , <i>seq</i> =[]) Overrides: UserList.UserList.__init__
--

`__lt__`(*self*, *other*)Overrides: `UserList.UserList.__lt__`**`__le__`**(*self*, *other*)Overrides: `UserList.UserList.__le__`**`__eq__`**(*self*, *other*)Overrides: `UserList.UserList.__eq__`**`__ne__`**(*self*, *other*)Overrides: `UserList.UserList.__ne__`**`__gt__`**(*self*, *other*)Overrides: `UserList.UserList.__gt__`**`__ge__`**(*self*, *other*)Overrides: `UserList.UserList.__ge__`**`__cmp__`**(*self*, *other*)Overrides: `UserList.UserList.__cmp__`**`__len__`**(*self*)Overrides: `UserList.UserList.__len__`**`__getitem__`**(*self*, *i*)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)
```

36.14 Class Unbuffered

A proxy class that wraps a file object, flushing after every write, and delegating everything else to the wrapped object.

36.14.1 Methods

```
__init__(self, file)
```

```
write(self, arg)
```

```
__getattr__(self, attr)
```

36.15 Class Null

Known Subclasses: SCons.Sig.MD5Null, SCons.Sig.TimeStampNull

Null objects always and reliably "do nothing."

36.15.1 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)
```


37 Package SCons.Variables

engine.SCons.Variables

This file defines the Variables class that is used to add user-friendly customizable variables to an SCons build.

37.1 Modules

- **BoolVariable'**: engine.SCons.Variables.BoolVariable
This file defines the option type for SCons implementing true/false values.
(Section 38, p. 375)
- **EnumVariable'**: engine.SCons.Variables.EnumVariable
This file defines the option type for SCons allowing only specified input-values.
(Section 39, p. 376)
- **ListVariable'**: engine.SCons.Variables.ListVariable
This file defines the option type for SCons implementing 'lists'.
(Section 40, p. 377)
- **PackageVariable'**: engine.SCons.Variables.PackageVariable
This file defines the option type for SCons implementing 'package activation'.
(Section 41, p. 378)
- **PathVariable'**: SCons.Variables.PathVariable
This file defines an option type for SCons implementing path settings.
(Section 42, p. 379)

37.2 Variables

Name	Description
__revision__	Value: 'src/engine/SCons/Variables/__init__.py 3603 2008/10/10 0...

37.3 Class Variables

37.3.1 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*=[])

37.3.2 Class Variables

Name	Description
instance	Holds all the options, updates the environment with the variables, and renders the help text. Value: False
format	Value: ' <i>\n</i> s: %s <i>\n</i> default: %s <i>\n</i> actual: %s <i>\n</i> '
format_	Value: ' <i>\n</i> s: %s <i>\n</i> default: %s <i>\n</i> actual: %s <i>\n</i> aliases: ...

38 Module *SCons.Variables.BoolVariable*

`engine.SCons.Variables.BoolVariable`

This file defines the option type for SCons implementing true/false values.

Usage example:

```
opts = Variables()
opts.Add(BoolVariable('embedded', 'build for an embedded system', 0))
...
if env['embedded'] == 1:
    ...
```

38.1 Functions

BoolVariable (<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> .

39 Module *SCons.Variables.EnumVariable*

`engine.SCons.Variables.EnumVariable`

This file defines the option type for SCons allowing only specified input-values.

Usage example:

```
opts = Variables()
opts.Add(EnumVariable('debug', 'debug output and symbols', 'no',
                      allowed_values=('yes', 'no', 'full'),
                      map={}, ignorecase=2))
...
if env['debug'] == 'full':
    ...
```

39.1 Functions

EnumVariable(*key, help, default, allowed_values, map={}, ignorecase=0*)

The input parameters describe a option with only certain values allowed. They are returned with an appropriate converter and validator appended. The result is usable for input to `Variables.Add()`.

'key' and 'default' are the values to be passed on to `Variables.Add()`.

'help' will be appended by the allowed values automatically

'allowed_values' is a list of strings, which are 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).

40 Module *SCons.Variables.ListVariable*

`engine.SCons.Variables.ListVariable`

This file defines the option type for SCons implementing 'lists'.

A 'list' option may either be 'all', 'none' or a list of names separated by comma. After the option has been processed, the option value holds either the named list elements, all list 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(...)
```

40.1 Functions

ListVariable(*key, help, default, names, map={}*)

The input parameters describe a 'package list' option, thus they are returned with the correct converter and validator appended. The result is usable for input to `opts.Add()` .

A 'package list' option may either be 'all', 'none' or a list of package names (separated by space).

41 Module *SCons.Variables.PackageVariable*

`engine.SCons.Variables.PackageVariable`

This file defines the option type for SCons implementing 'package activation'.

To be used whenever a 'package' may be enabled/disabled and the package path may be specified.

Usage example:

Examples:

```
x11=no    (disables X11 support)
x11=yes   (will search for the package installation dir)
x11=/usr/local/X11 (will check this path for 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 ...
```

41.1 Functions

`PackageVariable(key, help, default, searchfunc=False)`

The input parameters describe a 'package list' option, thus they are returned with the correct converter and validator appended. The result is usable for input to `opts.Add()` .

A 'package list' option may either be 'all', 'none' or a list of package names (seperated by space).

42 Module *SCons.Variables.PathVariable*

SCons.Variables.PathVariable

This file defines an option type for SCons implementing path settings.

To be used whenever a 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'))
```

42.1 Variables

Name	Description
PathVariable	Value: <SCons.Variables.PathVariable._PathVariableClass instance...

43 Module *SCons.Warnings*

SCons.Warnings

This file implements the warnings framework for *SCons*.

43.1 Functions

suppressWarningClass(*clazz*)

Suppresses all warnings that are of type *clazz* or derived from *clazz*.

enableWarningClass(*clazz*)

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.

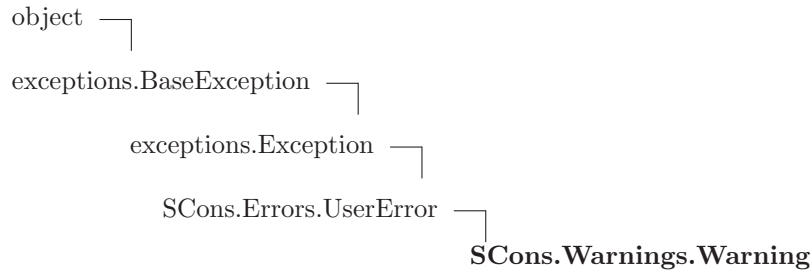
For example, `'deprecated'` will enable the `DeprecatedWarning` class. `'no-dependency'` will disable the `.DependencyWarning` class.

As a special case, `--warn=all` and `--warn=no-all` will enable or disable (respectively) the base `Warning` class of all warnings.

43.2 Variables

Name	Description
<code>--revision--</code>	Value: 'src/engine/SCons/Warnings.py 3603 2008/10/10 05:46:45 sc...

43.3 Class Warning



Known Subclasses: SCons.SConf.SConfWarning, SCons.Warnings.CacheWriteErrorWarning, SCons.Warnings.CorruptSCo...
 SCons.Warnings.DependencyWarning, SCons.Warnings.DeprecatedWarning, SCons.Warnings.DuplicateEnvironmentWarning,
 SCons.Warnings.LinkWarning, SCons.Warnings.MisleadingKeywordsWarning, SCons.Warnings.MissingSConsriptWarning,
 SCons.Warnings.NoMD5ModuleWarning, SCons.Warnings.NoMetaclassSupportWarning, SCons.Warnings.NoObjectCountW...
 SCons.Warnings.NoParallelSupportWarning, SCons.Warnings.ReservedVariableWarning, SCons.Warnings.StackSizeWarning

43.3.1 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__(...)

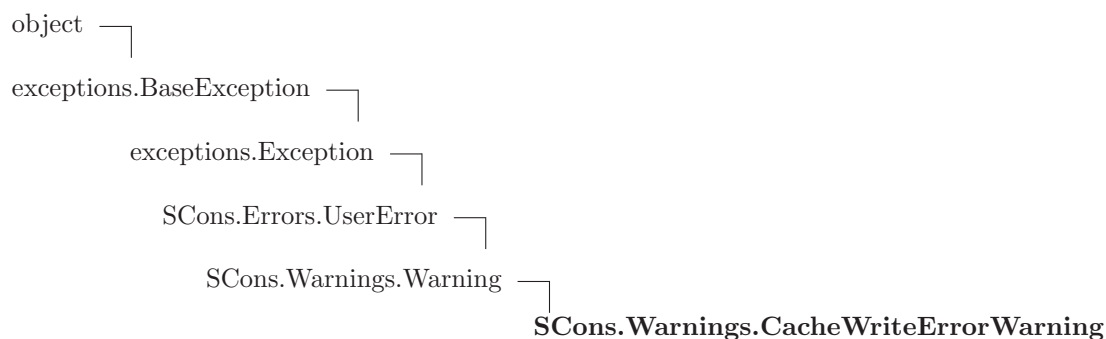
__str__(x)

str(x)

Overrides: object.__str__
43.3.2 Properties

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

43.4 Class `CacheWriteErrorWarning`



43.4.1 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__
```

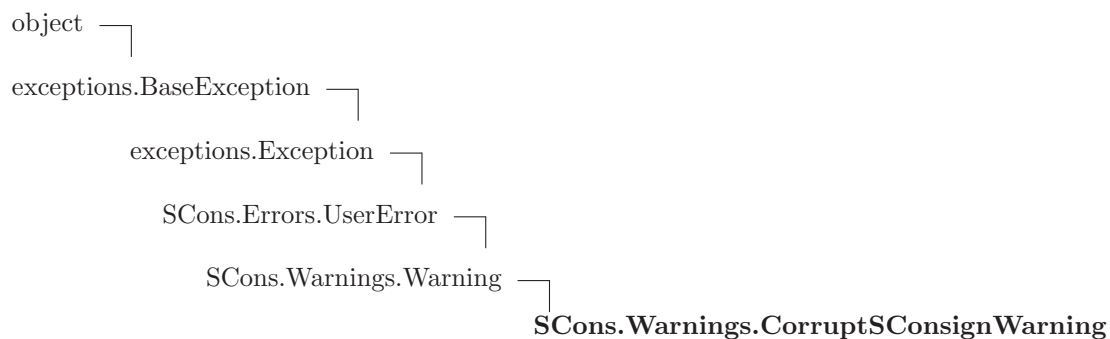
```
__setstate__(...)
```

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

43.4.2 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>

43.5 Class *CorruptSConsignWarning*



43.5.1 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__
```

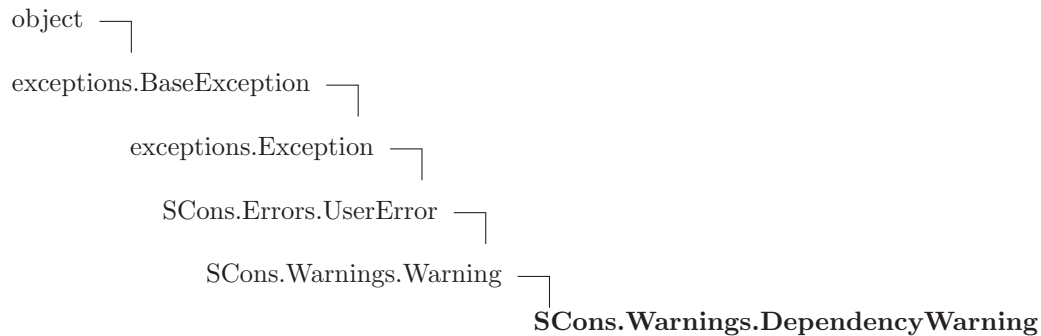
```
__setstate__(...)
```

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

43.5.2 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>

43.6 Class DependencyWarning



43.6.1 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__` `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)`

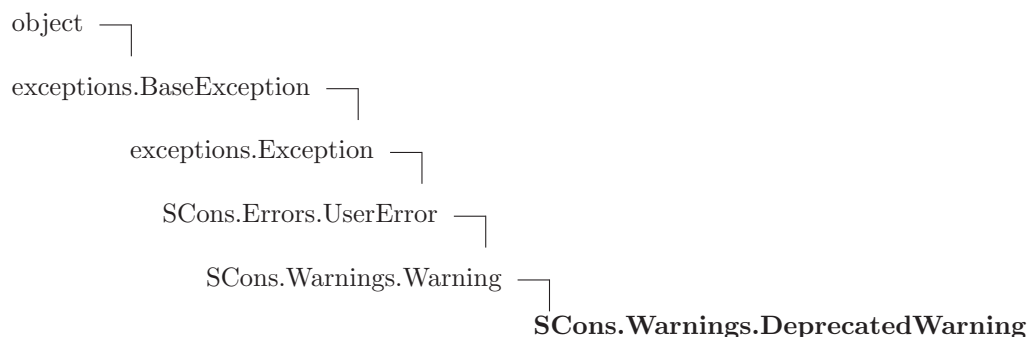
`str(x)`

Overrides: `object.__str__`

43.6.2 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>

43.7 Class *DeprecatedWarning*



Known Subclasses: *SCons.Warnings.DeprecatedCopyWarning*, *SCons.Warnings.DeprecatedSourceSignaturesWarning*, *SCons.Warnings.DeprecatedTargetSignaturesWarning*, *SCons.Warnings.PythonVersionWarning*

43.7.1 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]
```

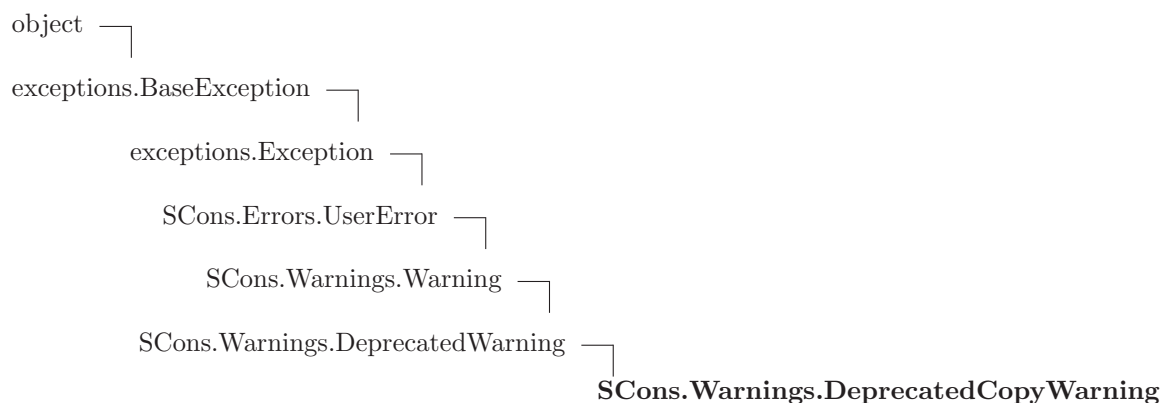
Use of negative indices is not supported.

<code>__hash__(x)</code>
<code>hash(x)</code>
<code>__init__(...)</code>
<code>x.__init__(...)</code> initializes <code>x</code> ; see <code>x.__class__.__doc__</code> for signature
Overrides: <code>exceptions.BaseException.__init__</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>
<code>__reduce__(...)</code>
helper for pickle
Overrides: <code>object.__reduce__</code> <code>exitit</code> (inherited documentation)
<code>__reduce_ex__(...)</code>
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>
<code>__str__(x)</code>
<code>str(x)</code>
Overrides: <code>object.__str__</code>

43.7.2 Properties

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

43.8 Class *DeprecatedCopyWarning*



43.8.1 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)
```

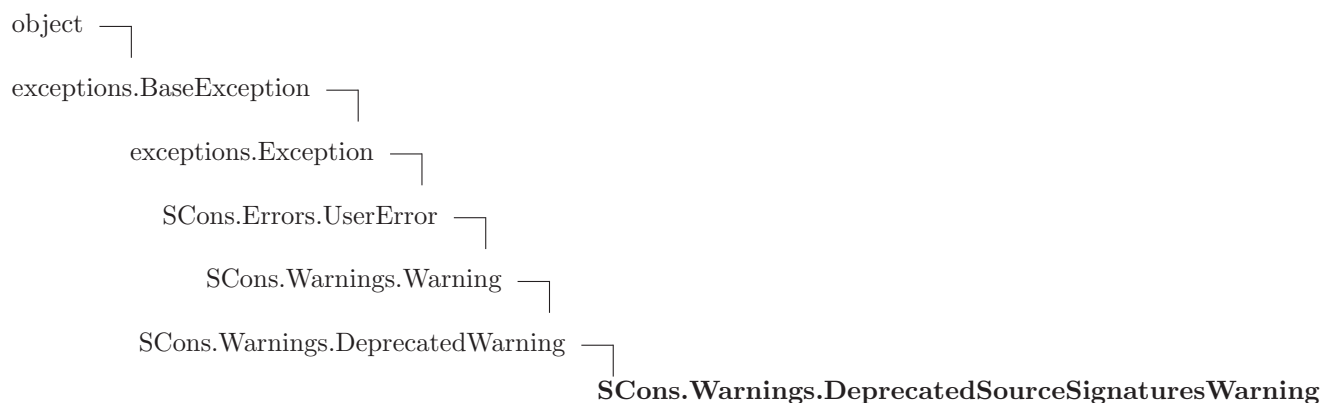
`str(x)`

Overrides: `object.__str__`

43.8.2 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>

43.9 Class *DeprecatedSourceSignaturesWarning*



43.9.1 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)
```

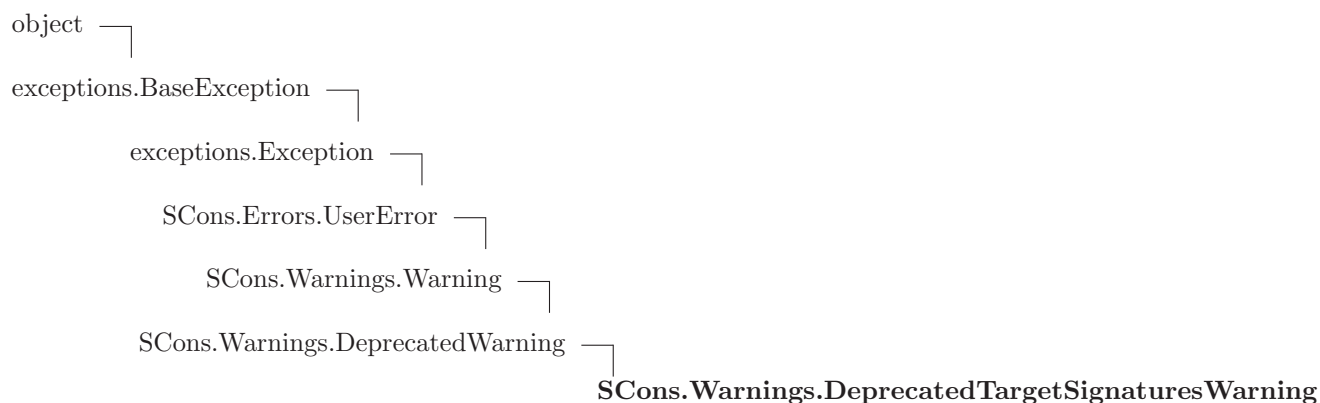
```
str(x)
```

Overrides: `object.__str__`

43.9.2 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>

43.10 Class `DeprecatedTargetSignaturesWarning`



43.10.1 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)
```

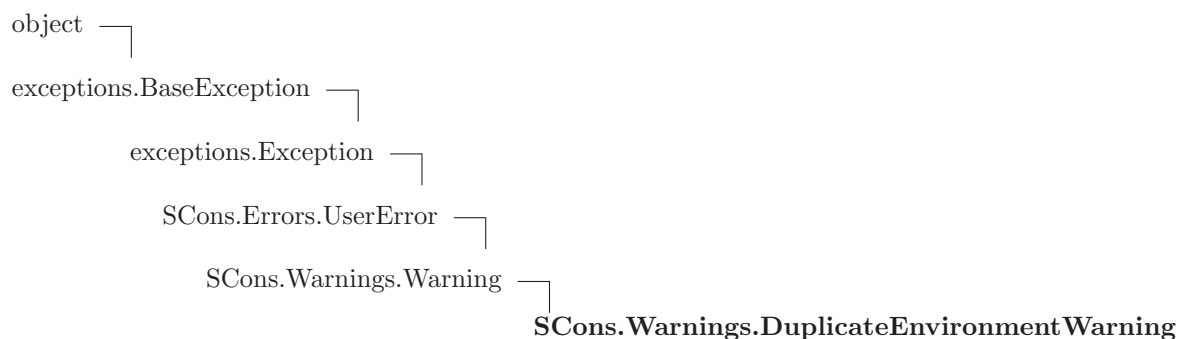
```
str(x)
```

Overrides: `object.__str__`

43.10.2 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>

43.11 Class `DuplicateEnvironmentWarning`



43.11.1 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__
```

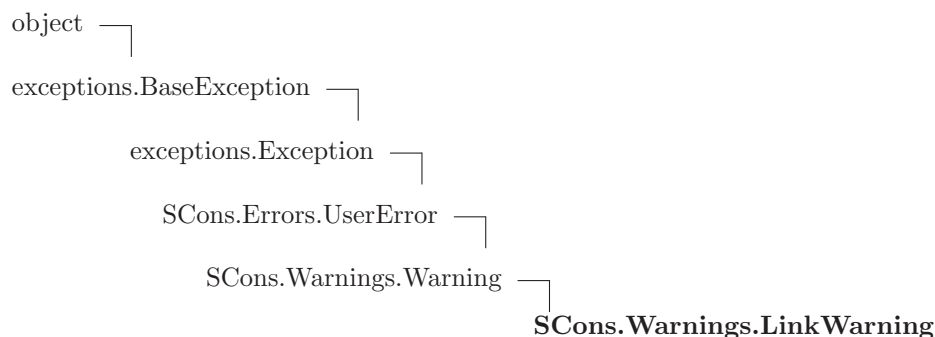
```
__setstate__(...)
```

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

43.11.2 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>

43.12 Class *LinkWarning*



Known Subclasses: `SCons.Warnings.FortranCxxMixWarning`

43.12.1 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 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__
```

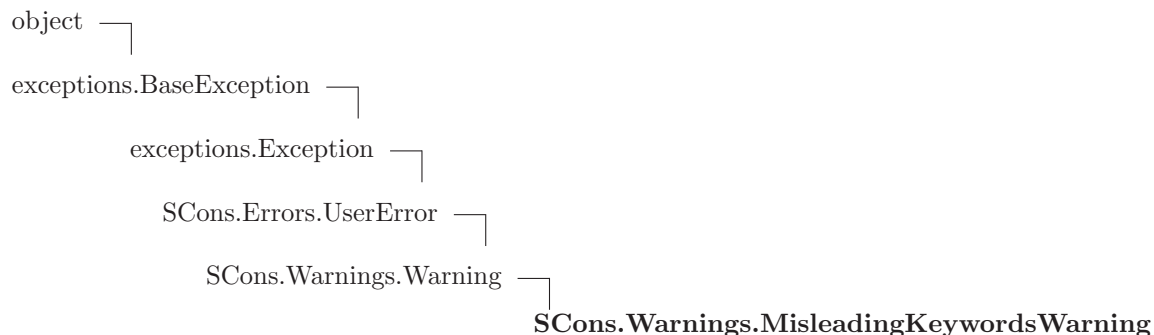
```
__setstate__(...)
```

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

43.12.2 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>

43.13 Class `MisleadingKeywordsWarning`



43.13.1 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__` `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)`

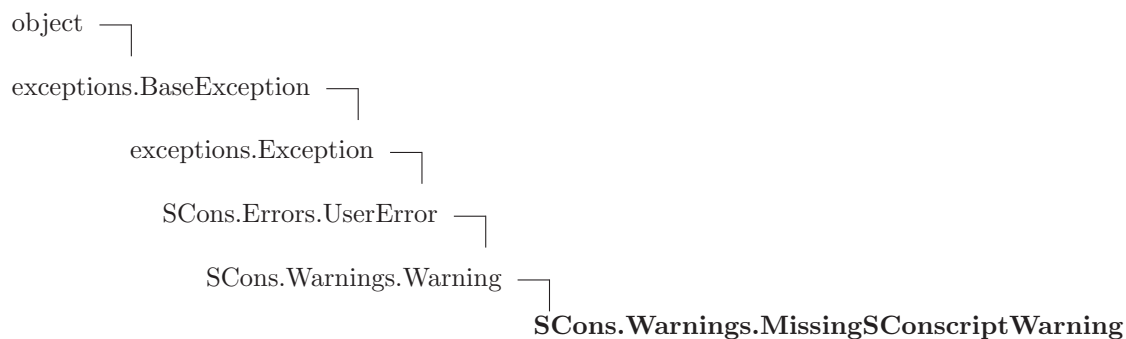
`str(x)`

Overrides: `object.__str__`

43.13.2 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>

43.14 Class `MissingSConscriptWarning`



43.14.1 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__(...)

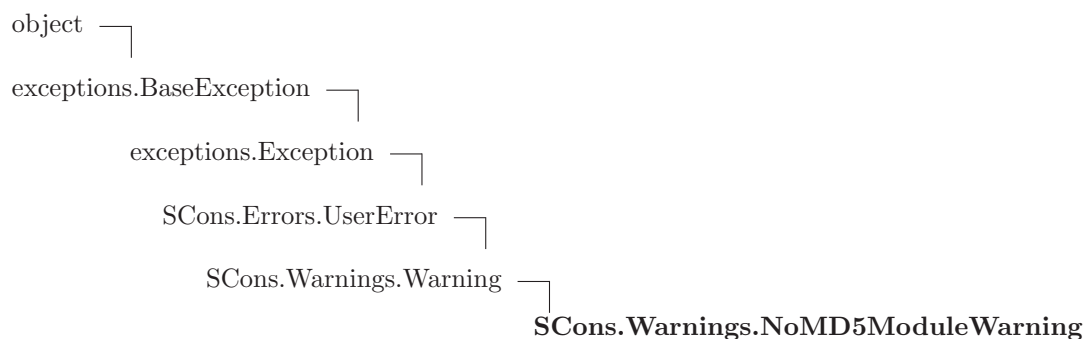
__str__(x)

str(x)

Overrides: object.__str__
43.14.2 Properties

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

43.15 Class NoMD5ModuleWarning



43.15.1 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__
```

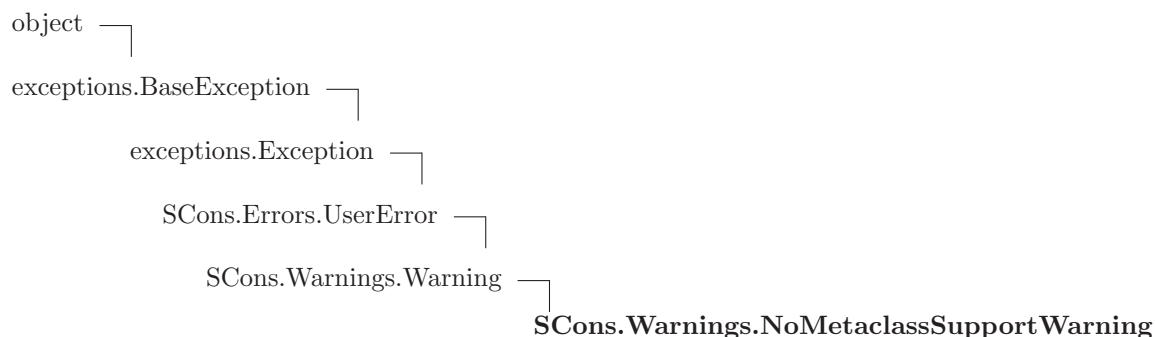
```
__setstate__(...)
```

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

43.15.2 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>

43.16 Class NoMetaclassSupportWarning



43.16.1 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__
```

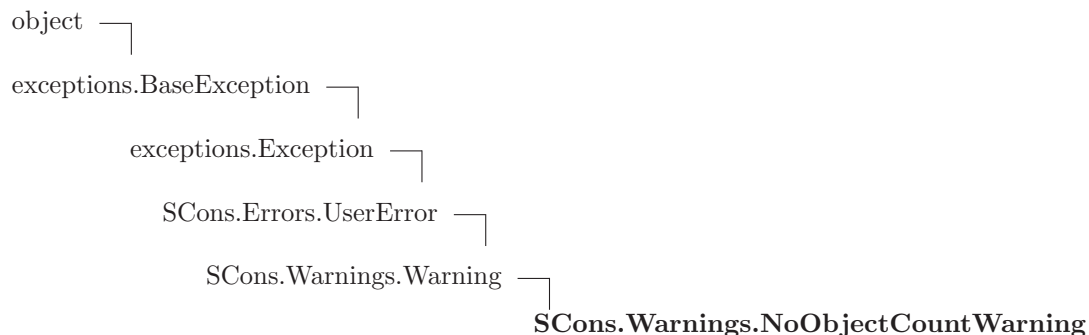
```
__setstate__(...)
```

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

43.16.2 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>

43.17 Class NoObjectCountWarning



43.17.1 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 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__(...)`

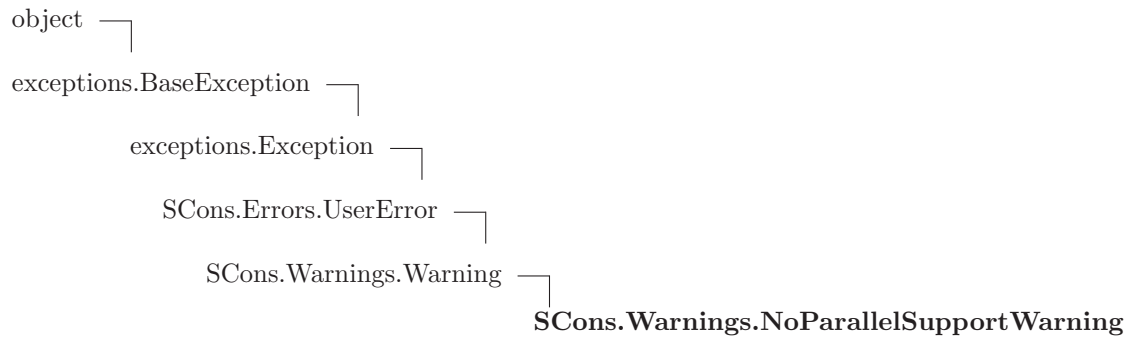
`__str__(x)`

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

43.17.2 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>

43.18 Class NoParallelSupportWarning



43.18.1 Methods

<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__</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>

__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__(...)

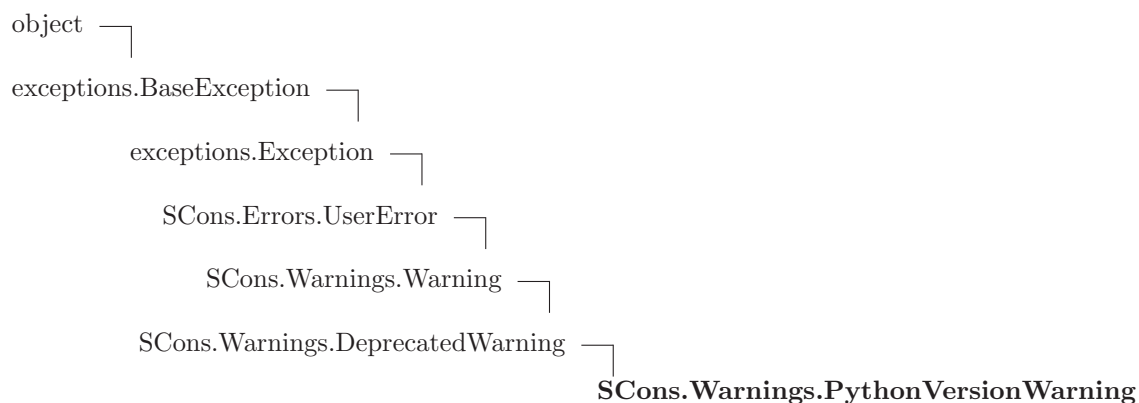
__str__(x)

str(x)

Overrides: object.__str__
43.18.2 Properties

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

43.19 Class `PythonVersionWarning`



43.19.1 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 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)
```

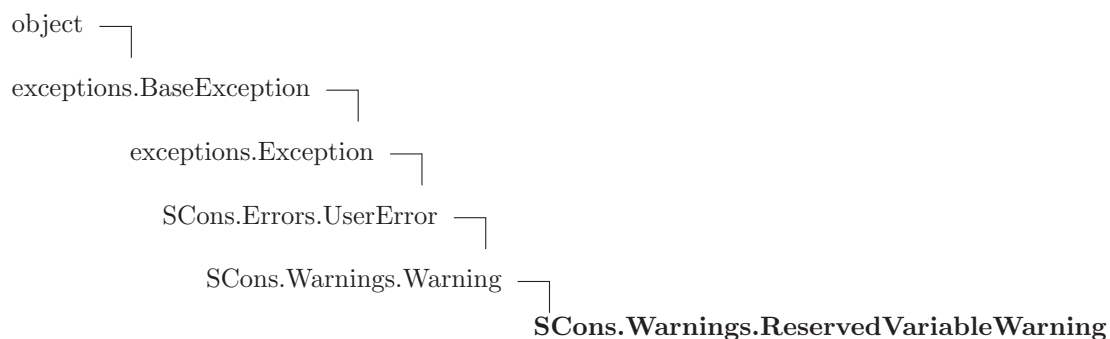
```
str(x)
```

Overrides: `object.__str__`

43.19.2 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>

43.20 Class ReservedVariableWarning



43.20.1 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__
```

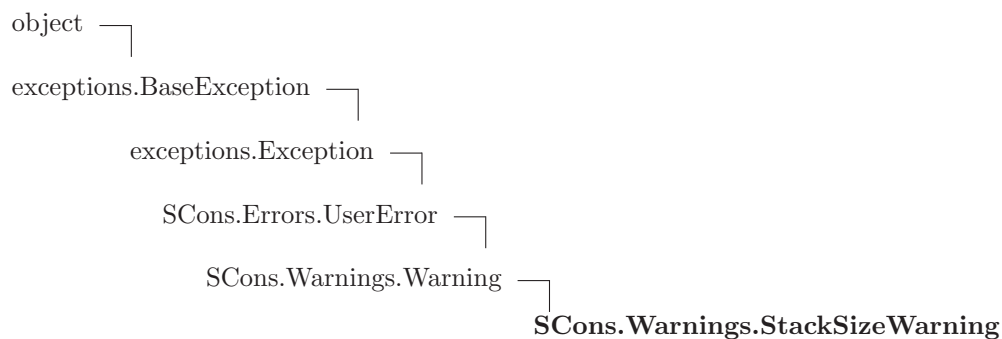
```
__setstate__(...)
```

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

43.20.2 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>

43.21 Class StackSizeWarning



43.21.1 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 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__
```

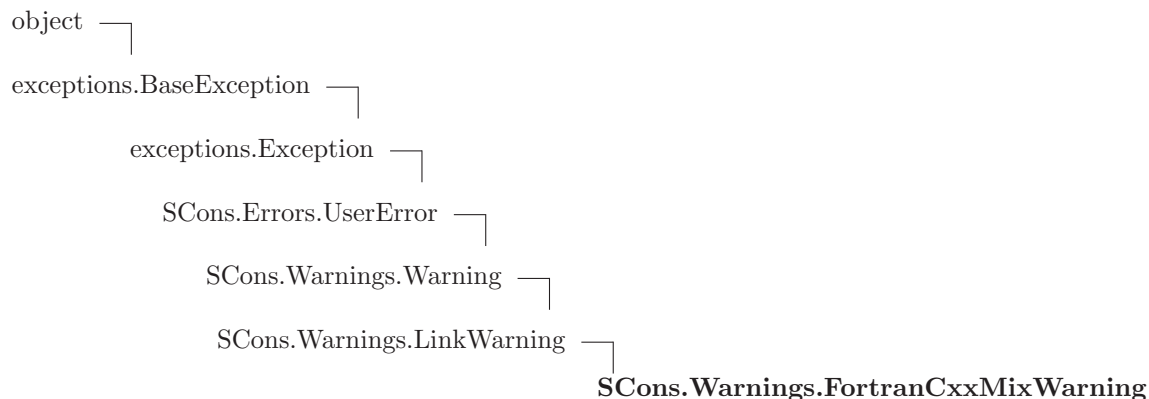
```
__setstate__(...)
```

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

43.21.2 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>

43.22 Class FortranCxxMixWarning



43.22.1 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 Valuea 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__`

43.22.2 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>

44 Package SCons.compat

SCons compatibility package for old Python versions

This subpackage holds modules that provide backwards-compatible implementations of various things that we'd like to use in SCons but which only show up in later versions of Python than the early, old version(s) we still support.

Other code will not generally reference things in this package through the SCons.compat namespace. The modules included here add things to the `__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.

44.1 Modules

- **`_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.
(Section 45, p. 421)
- **`_scons_hashlib`**: hashlib backwards-compatibility module for older (pre-2.5) Python versions
This does not NOT (repeat, **NOT**) provide complete hashlib functionality.
(Section 46, p. 423)
- **`_scons_itertools`**: Implementations of itertools functions for Python versions that don't have iterators.
(Section 47, p. 425)
- **`_scons_optparse`**: optparse - a powerful, extensible, and easy-to-use option parser.

(Section 48, p. 427)

- **`_scons_sets`**: Classes to represent arbitrary sets (including sets of sets).
(Section 49, p. 449)
- **`_scons_sets15`** (Section 50, p. 462)
- **`_scons_shlex`**: A lexical analyzer class for simple shell-like syntaxes.
(Section 51, p. 464)
- **`_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 52, p. 465)
- **`_scons_textwrap`**: Text wrapping and filling.
(Section 53, p. 476)
- **`_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.
(Section 54, p. 479)

44.2 Functions

```
import_as(module, name)
```

Imports the specified module (from our local directory) as the specified name.

44.3 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>__revision__</code>	Value: 'src/engine/SCons/compat/__init__.py 3603 2008/10/10 05:4...
<code>version_string</code>	Value: <code>string.split(sys.version) [0]</code>
<code>version_ints</code>	Value: <code>map(int, string.split(version_string, '.'))</code>

45 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.

45.1 Functions

<code>is_String(<i>obj</i>)</code>

45.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/compat/_scons_UserString.py 3603 2008/1...
<code>__doc__</code>	Value: ...

45.3 Class *UserString*

45.3.1 Methods

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

<code>__str__(<i>self</i>)</code>

<code>__repr__(<i>self</i>)</code>

<code>__int__(<i>self</i>)</code>

<code>__long__(<i>self</i>)</code>

<code>__float__(<i>self</i>)</code>

<code>__complex__(<i>self</i>)</code>

<code>__hash__(<i>self</i>)</code>

<code>__cmp__(<i>self</i>, <i>string</i>)</code>
--

<code>__contains__(<i>self</i>, <i>char</i>)</code>

```
__len__(self)
```

```
__getitem__(self, index)
```

```
__getslice__(self, start, end)
```

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

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

```
__mul__(self, n)
```

```
__rmul__(self, n)
```

46 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.

46.1 Functions

<code>md5(string='')</code>

46.2 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>__revision__</code>	Value: 'src/engine/SCons/compat/_scons_hashlib.py 3603 2008/10/1...

46.3 Class `md5obj`

46.3.1 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>

46.3.2 Class Variables

Name	Description
<code>md5_module</code>	Value: <module 'md5' from '/usr/lib/python2.5/md5.pyc'>

46.4 Class `md5obj`

46.4.1 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>

46.4.2 Class Variables

Name	Description
<code>md5_module</code>	Value: <module 'md5' from '/usr/lib/python2.5/md5.pyc'>

47 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.

47.1 Functions

<code>chain(*iterables)</code>

<code>count(n=0)</code>

<code>cycle(iterable)</code>

<code>dropwhile(predicate, iterable)</code>

<code>groupby(iterable, *args)</code>

<code>ifilter(predicate, iterable)</code>

<code>ifilterfalse(predicate, iterable)</code>
--

<code>imap(function, *iterables)</code>

<code>islice(*args, **kw)</code>

<code>izip(*iterables)</code>

<code>repeat(*args, **kw)</code>

starmap (<i>*args</i> , <i>**kw</i>)

takewhile (<i>predicate</i> , <i>iterable</i>)

tee (<i>*args</i> , <i>**kw</i>)

47.2 Variables

Name	Description
<code>--revision--</code>	Value: 'src/engine/SCons/compat/_scons_itertools.py 3603 2008/10...
<code>--doc--</code>	Value: ...

48 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.

48.1 Variables

Name	Description
<code>SUPPRESS_HELP</code>	Value: 'SUPPRESSHELP'
<code>SUPPRESS_USAGE</code>	Value: 'SUPPRESSUSAGE'

48.2 Class *OptParseError*



Known Subclasses: *SCons.compat._scons_optparse.BadOptionError*, *SCons.compat._scons_optparse.OptionError*, *SCons.compat._scons_optparse.OptionValueError*

48.2.1 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__` `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__(...)
```

48.2.2 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>

48.3 Class *OptionError*

```
object └─
```

```
exceptions.BaseException └─
```

```
exceptions.Exception └─
```

```
SCons.compat._scons_optparse.ParseError └─
```

```
SCons.compat._scons_optparse.OptionError
```

Known Subclasses: `SCons.compat._scons_optparse.OptionConflictError`

Raised if an `Option` instance is created with invalid or inconsistent arguments.

48.3.1 Methods

`__init__(self, msg, option)`
`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature
 Overrides: `SCons.compat._scons_optparse.OptionParserError.__init__`

`__str__(self)`
`str(x)`
 Overrides: `SCons.compat._scons_optparse.OptionParserError.__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`

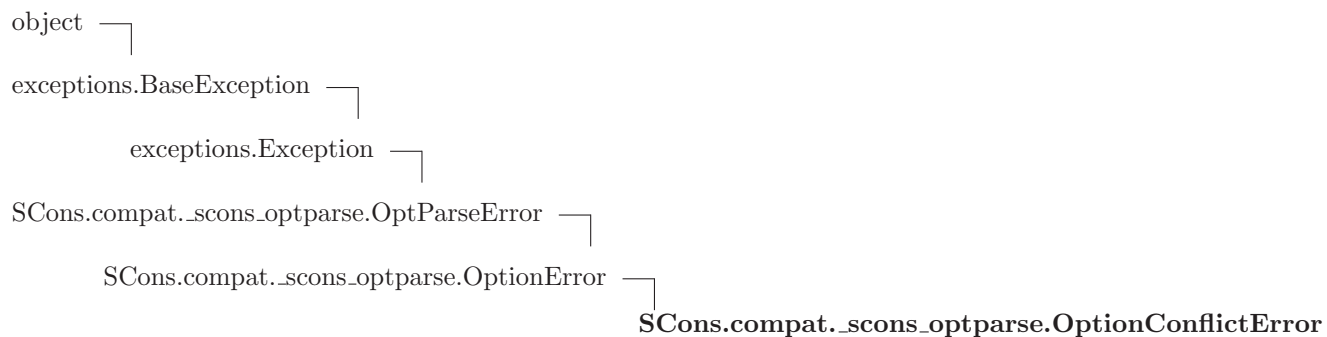
 Overrides: `object.__setattr__`

`__setstate__(...)`

48.3.2 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>

48.4 Class `OptionConflictError`



Raised if conflicting options are added to an `OptionParser`.

48.4.1 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__(self, msg, option)`

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signatureOverrides: `SCons.compat._scons_optparse.OptParseError.__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__` `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__`

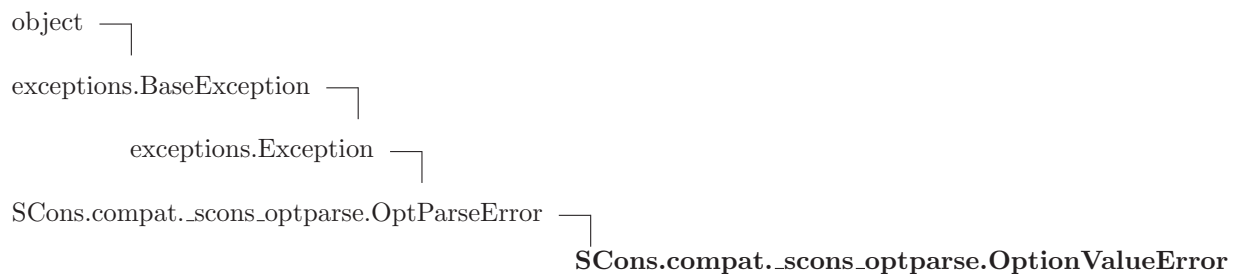
`__setstate__(...)`

```
__str__(self)
str(x)
Overrides: SCons.compat._scons_optparse.OptParseError.__str__
```

48.4.2 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>

48.5 Class `OptionValueError`



Raised if an invalid option value is encountered on the command line.

48.5.1 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__(self, msg)`

`x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signatureOverrides: `exceptions.Exception.__init__` `exitit`(inherited documentation)

`__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__` `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__`

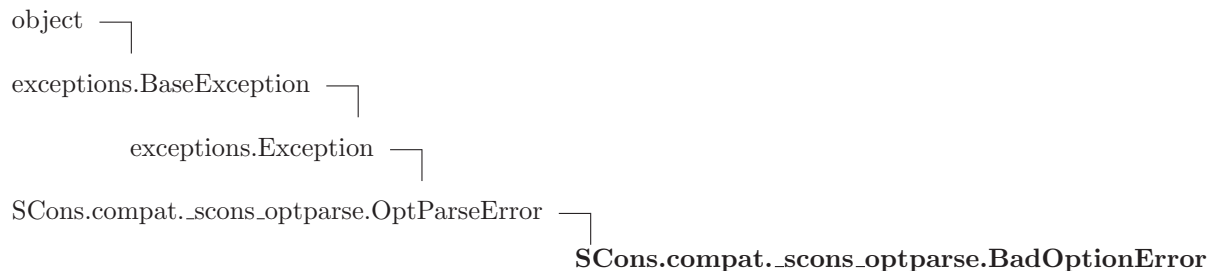
`__setstate__(...)`

`__str__(self)`

`str(x)`Overrides: `exceptions.BaseException.__str__` `exitit`(inherited documentation)**48.5.2 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>

48.6 Class *BadOptionError*



Known Subclasses: *SCons.compat._scons_optparse.AmbiguousOptionError*

Raised if an invalid option is seen on the command line.

48.6.1 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__(...)`

48.6.2 Properties

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

48.7 Class *HelpFormatter***Known Subclasses:** `SCons.compat._scons_optparse.IndentedHelpFormatter`, `SCons.compat._scons_optparse.TitledHelpForm`

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").

```

48.7.1 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>
--

indent(*self*)**dedent**(*self*)**format_usage**(*self*, *usage*)**format_heading**(*self*, *heading*)**format_description**(*self*, *description*)**format_epilog**(*self*, *epilog*)**expand_default**(*self*, *option*)**format_option**(*self*, *option*)**store_option_strings**(*self*, *parser*)**format_option_strings**(*self*, *option*)

Return a comma-separated list of option strings & metavariables.

48.7.2 Class Variables

Name	Description
NO_DEFAULT_VALUE	Value: 'none'

48.8 Class IndentedHelpFormatter

SCons.compat._scons_optparse.HelpFormatter

SCons.compat._scons_optparse.IndentedHelpFormatter

Format help with indented section bodies.

48.8.1 Methods

__init__(*self*, *indent_increment*=2, *max_help_position*=24, *width*=False, *short_first*=False)

Overrides: SCons.compat._scons_optparse.HelpFormatter.__init__

format_usage(*self*, *usage*)

Overrides: SCons.compat._scons_optparse.HelpFormatter.format_usage

format_heading(*self*, *heading*)

Overrides: SCons.compat._scons_optparse.HelpFormatter.format_heading

`dedent(self)``expand_default(self, option)``format_description(self, description)``format_epilog(self, epilog)``format_option(self, option)``format_option_strings(self, option)`

Return a comma-separated list of option strings & metavariables.

`indent(self)``set_long_opt_delimiter(self, delim)``set_parser(self, parser)``set_short_opt_delimiter(self, delim)``store_option_strings(self, parser)`

48.8.2 Class Variables

Name	Description
NO_DEFAULT_VALUE	Value: 'none'

48.9 Class *TitledHelpFormatter*

*SCons.compat._scons_optparse.HelpFormatter**SCons.compat._scons_optparse.TitledHelpFormatter*

Format help with underlined section headers.

48.9.1 Methods

`__init__(self, indent.increment=0, max_help_position=24, width=False, short_first=0)`
 Overrides: *SCons.compat._scons_optparse.HelpFormatter.__init__*

`format_usage(self, usage)`
 Overrides: *SCons.compat._scons_optparse.HelpFormatter.format_usage*

format_heading(*self*, *heading*)

Overrides: SCons.compat._scons_optparse.HelpFormatter.format_heading

dedent(*self*)

expand_default(*self*, *option*)

format_description(*self*, *description*)

format_epilog(*self*, *epilog*)

format_option(*self*, *option*)

format_option_strings(*self*, *option*)

Return a comma-separated list of option strings & metavariables.

indent(*self*)

set_long_opt_delimiter(*self*, *delim*)

set_parser(*self*, *parser*)

set_short_opt_delimiter(*self*, *delim*)

store_option_strings(*self*, *parser*)

48.9.2 Class Variables

Name	Description
NO_DEFAULT_VALUE	Value: 'none'

48.10 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

```

48.10.1 Methods

```
__init__(self, *opts, **attrs)
```

```
__str__(self)
```

```
__repr__(self)
```

```
takes_value(self)
```

```
get_opt_string(self)
```

```
check_value(self, opt, value)
```

```
convert_value(self, opt, value)
```

```
process(self, opt, value, values, parser)
```

```
take_action(self, action, dest, opt, value, values, parser)
```

48.10.2 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 0x84f5294>, 'complex...'
CHECK_METHODS	Value: [_check_action, _check_type, _check_choice, _check_dest, ...]

48.11 Class Values

48.11.1 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)
```

48.12 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:

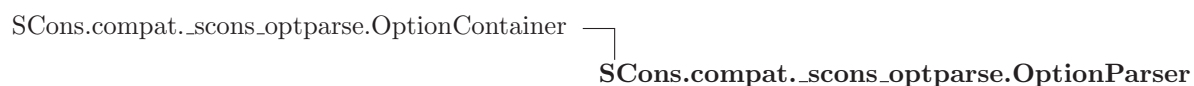
```
option_list : [Option]
    the list of Option objects contained by this OptionContainer
_short_opt : { string : Option }
    dictionary mapping short option strings, eg. "-f" or "-X",
    to the Option instances that implement them. If an Option
    has multiple short option strings, it will appear 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.

48.12.1 Methods`__init__(self, option_class, conflict_handler, description)``set_conflict_handler(self, handler)``set_description(self, description)``get_description(self)``destroy(self)`see `OptionParser.destroy()`.`add_option(Option)``add_option(opt_str, ..., kwarg=val, ...)``add_options(self, option_list)``get_option(self, opt_str)``has_option(self, opt_str)``remove_option(self, opt_str)``format_option_help(self, formatter)``format_description(self, formatter)``format_help(self, formatter)`**48.13 Class *OptionGroup***`SCons.compat._scons_optparse.OptionContainer``SCons.compat._scons_optparse.OptionGroup`**48.13.1 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*)

48.14 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', 'largs', 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.

48.14.1 Methods

```
__init__(self, usage=False, option_list=False, option_class=<class
SCons.compat._scons_optparse.Option at 0x84ed9ec>, 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*)

48.14.2 Class Variables

Name	Description
<code>standard_option_list</code>	Value: []

49 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'.

49.1 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.

49.1.1 Methods**`__init__(self)`**

This is an abstract class.

Overrides: `object.__init__`**`__len__(self)`**

Return the number of elements of a set.

`__repr__(self)`

Return string representation of a set.

This looks like `'Set([<list of elements>])'`.Overrides: `object.__repr__`**`__str__(self)`**

Return string representation of a set.

This looks like `'Set([<list of elements>])'`.Overrides: `object.__str__`**`__iter__(self)`**

Return an iterator over the elements or a set.

This is the keys iterator for the underlying dict.

`__cmp__(self, other)`**`__eq__(self, other)`****`__ne__(self, other)`****`copy(self)`**

Return a shallow copy of a set.

`__copy__(self)`

Return a shallow copy of a set.

`__deepcopy__(self, memo)`Return a deep copy of a set; used by `copy` module.

`--or--(self, other)`

Return the union of two sets as a new set.

(I.e. all elements that are in either set.)

`union(self, other)`

Return the union of two sets as a new set.

(I.e. all elements that are in either set.)

`--and--(self, other)`

Return the intersection of two sets as a new set.

(I.e. all elements that are in both sets.)

`intersection(self, other)`

Return the intersection of two sets as a new set.

(I.e. all elements that are in both sets.)

`--xor--(self, other)`

Return the symmetric difference of two sets as a new set.

(I.e. all elements that are in exactly one of the sets.)

`symmetric_difference(self, other)`

Return the symmetric difference of two sets as a new set.

(I.e. all elements that are in exactly one of the sets.)

`--sub--(self, other)`

Return the difference of two sets as a new Set.

(I.e. all elements that are in this set and not in the other.)

`difference(self, other)`

Return the difference of two sets as a new Set.

(I.e. all elements that are in this set and not in the other.)

__contains__(*self*, *element*)

Report whether an element is a member of a set.

(Called in response to the expression 'element in self'.)

issubset(*self*, *other*)

Report whether another set contains this set.

issuperset(*self*, *other*)

Report whether this set contains another set.

__le__(*self*, *other*)

Report whether another set contains this set.

__ge__(*self*, *other*)

Report whether this set contains another set.

__lt__(*self*, *other*)

__gt__(*self*, *other*)

__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
```

49.1.2 Properties

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

49.2 Class ImmutableSet



Immutable set class.

49.2.1 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__ extit(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.)

49.2.2 Properties

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

49.3 Class Set

object

SCons.compat._scons_sets.BaseSet

SCons.compat._scons_sets.Set

Mutable set class.

49.3.1 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.)

<code>issubset(<i>self</i>, <i>other</i>)</code>
--

Report whether another set contains this set.

<code>issuperset(<i>self</i>, <i>other</i>)</code>
--

Report whether this set contains another set.

<code>symmetric_difference(<i>self</i>, <i>other</i>)</code>
--

Return the symmetric difference of two sets as a new set.

(I.e. all elements that are in exactly one of the sets.)
--

<code>union(<i>self</i>, <i>other</i>)</code>

Return the union of two sets as a new set.
--

(I.e. all elements that are in either set.)

49.3.2 Properties

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

50 Module *SCons.compat._scons_sets15*

50.1 Class Set

The set class. It can contain mutable objects.

50.1.1 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.

51 Module *SCons.compat._scons_shlex*

A lexical analyzer class for simple shell-like syntaxes.

51.1 Functions

<code>split(s, comments=False)</code>

51.2 Class *shlex*

A lexical analyzer class for simple shell-like syntaxes.

51.2.1 Methods

<code>__init__(self, instream=False, infile=False, posix=False)</code>
--

<code>push_token(self, tok)</code>

Push a token onto the stack popped by the <code>get_token</code> method

<code>push_source(self, newstream, newfile=False)</code>
--

Push an input source onto the lexer's input source stack.

<code>pop_source(self)</code>

Pop the input source stack.

<code>get_token(self)</code>

Get a token from the input stream (or from stack if it's nonempty)
--

<code>read_token(self)</code>

<code>sourcehook(self, newfile)</code>
--

Hook called on a filename to be sourced.
--

<code>error_leader(self, infile=False, lineno=False)</code>

Emit a C-compiler-like, Emacs-friendly error-message leader.
--

<code>__iter__(self)</code>

<code>next(self)</code>

52 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.

52.1 Functions

<p>call(*popenargs, **kwargs)</p> <hr/> <p>Run command with arguments. Wait for command to complete, then return the returncode attribute.</p> <p>The arguments are the same as for the Popen constructor. Example:</p> <pre>retcode = call(["ls", "-l"])</pre>
--

```
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"])
```

52.2 Variables

Name	Description
PIPE	Value: -1
STDOUT	Value: -2

52.3 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.

52.3.1 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__` `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__(...)`

52.3.2 Properties

continued on next page

Name	Description
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>

52.4 Class Popen

object  **SCons.compat.scons_subprocess.Popen**

52.4.1 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)`

52.4.2 Properties

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

53 Module *SCons.compat._scons_textwrap*

Text wrapping and filling.

53.1 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.

53.2 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'.

53.2.1 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.

53.2.2 Class Variables

Name	Description
<code>whitespace.trans</code>	Value: <code>'\x00\x01\x02\x03\x04\x05\x06\x07\x08\x0e\x0f\x10\x1...</code>
<code>unicode_whitespace.trans</code>	Value: {9: 32, 10: 32, 11: 32, 12: 32, 13: 32, 32: 32}
<code>uspace</code>	Value: 32
<code>wordsep_re</code>	Value: <code>re.compile(r'(\s [\^s\w]*\w{2,}-(?=\w{2,}) (?<=[\w!"'&\...</code>
<code>sentence_end_re</code>	Value: <code>re.compile(r'[abcdefghijklmnopqrstuvwxyz][\.\!?\]["'']?)</code>

continued on next page

Name	Description
x	Value: 32

54 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.

54.1 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.

54.2 Variables

Name	Description
<code>__doc__</code>	Value: ...
<code>__revision__</code>	Value: 'src/engine/SCons/compat/builtins.py 3603 2008/10/10 05:4...'
<code>False</code>	Value: False
<code>True</code>	Value: True

55 Module SCons.cpp

SCons C Pre-Processor module

55.1 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.

55.2 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>

55.3 Class FunctionEvaluator

Handles delayed evaluation of a #define function call.

55.3.1 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.

55.4 Class PreProcessor

Known Subclasses: SCons.cpp.DumbPreProcessor, SCons.Scanner.C.SConsCPPScanner

The main workhorse class for handling C pre-processing.

55.4.1 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.

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.

55.5 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.

55.5.1 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.

56 Module SCons.dblite

56.1 Functions

`corruption_warning(filename)`

`is_string(s)`

`unicode(s)`

`open(file, flag=False, mode=438)`

56.2 Variables

Name	Description
<code>keep_all_files</code>	Value: 0
<code>ignore_corrupt_dbfiles</code>	Value: 0
<code>dblite_suffix</code>	Value: <code>' .dblite'</code>
<code>tmp_suffix</code>	Value: <code>' .tmp'</code>

56.3 Class dblite

56.3.1 Methods

`__init__(self, file_base_name, flag, mode)`

`__del__(self)`

`sync(self)`

`__getitem__(self, key)`

`__setitem__(self, key, value)`

`keys(self)`

`has_key(self, key)`

`__contains__(self, key)`

`iterkeys(self)`

`__iter__(self)`

`__len__(self)`

57 Module *SCons.exitfuncs*

SCons.exitfuncs

Register functions which are executed when *SCons* exits for any reason.

57.1 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*

57.2 Variables

Name	Description
<code>__revision__</code>	Value: 'src/engine/SCons/exitfuncs.py 3603 2008/10/10 05:46:45 s...'
<code>x</code>	Value: <code>sys.exitfunc</code>

58 Module md5

58.1 Variables

Name	Description
blocksize	Value: False
digest_size	Value: 16

Index

- cmd.Cmd.cmdloop (*function*), 291
- cmd.Cmd.columnize (*function*), 292
- cmd.Cmd.complete (*function*), 292
- cmd.Cmd.complete_help (*function*), 292
- cmd.Cmd.completedefault (*function*), 292
- cmd.Cmd.completenames (*function*), 292
- cmd.Cmd.emptyline (*function*), 292
- cmd.Cmd.get_names (*function*), 292
- cmd.Cmd.parseline (*function*), 292
- cmd.Cmd.postcmd (*function*), 292
- cmd.Cmd.postloop (*function*), 292
- cmd.Cmd.precmd (*function*), 293
- cmd.Cmd.preloop (*function*), 293
- cmd.Cmd.print_topics (*function*), 293
- exceptions.BaseException.__getitem__ (*function*), 74, 76, 78, 79, 81, 83, 85, 230, 232, 234, 235, 295, 312, 362, 382, 384, 386, 387, 389, 391, 393, 395, 397, 399, 400, 402, 404, 406, 407, 409, 411, 413, 415, 416, 428, 430, 432, 433, 435, 473
- exceptions.BaseException.__getslice__ (*function*), 74, 76, 78, 79, 81, 83, 85, 230, 232, 234, 235, 295, 312, 362, 382, 384, 386, 388, 389, 391, 393, 395, 397, 399, 401, 402, 404, 406, 408, 409, 411, 413, 415, 417, 428, 430, 432, 433, 435, 473
- exceptions.BaseException.__setstate__ (*function*), 75, 77, 79, 80, 82, 84, 85, 231, 233, 234, 236, 296, 313, 363, 383, 385, 387, 388, 390, 392, 394, 396, 398, 400, 401, 403, 405, 407, 408, 410, 412, 414, 416, 417, 429, 431, 432, 434, 436, 473
- md5 (*module*), 492
- object.__delattr__ (*function*), 93, 352, 452, 454, 458, 474
- object.__getattr__ (*function*), 93, 452, 454, 459, 474
- object.__hash__ (*function*), 75, 76, 78, 80, 81, 83, 85, 93, 230, 232, 234, 235, 296, 312, 362, 382, 384, 386, 388, 389, 391, 393, 395, 397, 399, 401, 402, 404, 406, 408, 409, 411, 413, 415, 417, 428, 430, 432, 434, 436, 452, 473, 475
- object.__init__ (*function*), 353
- object.__new__ (*function*), 93, 452, 454, 459, 475
- object.__reduce__ (*function*), 93, 101, 354, 452, 455, 459, 475
- object.__reduce_ex__ (*function*), 75, 77, 78, 80, 82, 83, 85, 93, 101, 231, 232, 234, 236, 296, 313, 354, 363, 383, 385, 386, 388, 390, 392, 394, 396, 398, 399, 401, 403, 405, 406, 408, 410, 412, 414, 415, 417, 429, 430, 432, 434, 436, 452, 455, 459, 473, 475
- object.__repr__ (*function*), 475
- object.__setattr__ (*function*), 93, 354, 452, 455, 460, 475
- object.__str__ (*function*), 94, 101, 475
- SCons (*package*), 2–5
 - SCons.Action (*module*), 6–14
 - SCons.Builder (*module*), 15–27
 - SCons.Builder.Builder (*function*), 16
 - SCons.Builder.BuilderBase (*class*), 25–27
 - SCons.Builder.CallableSelector (*class*), 18–20
 - SCons.Builder.CompositeBuilder (*class*), 27
 - SCons.Builder.DictCmdGenerator (*class*), 16–18
 - SCons.Builder.DictEmitter (*class*), 20–21
 - SCons.Builder.EmitterProxy (*class*), 24–25
 - SCons.Builder.ListEmitter (*class*), 21–23
 - SCons.Builder.OverrideWarner (*class*), 23–24
 - SCons.CacheDir (*module*), 28–29
 - SCons.CacheDir.CacheDir (*class*), 28–29
 - SCons.CacheDir.CachePushFunc (*function*), 28
 - SCons.CacheDir.CacheRetrieveFunc (*function*), 28
 - SCons.CacheDir.CacheRetrieveString (*function*), 28
 - SCons.compat (*package*), 419–420
 - SCons.compat._scons_hashlib (*module*), 423–424
 - SCons.compat._scons_itertools (*module*), 425–426
 - SCons.compat._scons_optparse (*module*), 427–448
 - SCons.compat._scons_sets (*module*), 449–461
 - SCons.compat._scons_sets15 (*module*), 462–463
 - SCons.compat._scons_shlex (*module*), 464
 - SCons.compat._scons_subprocess (*module*), 465–475
 - SCons.compat._scons_textwrap (*module*), 476–478
 - SCons.compat._scons_UserString (*module*), 421–422
 - SCons.compat.builtins (*module*), 479–480
 - SCons.compat.import_as (*function*), 420
 - SCons.Conftest (*module*), 30–33
 - SCons.cpp (*module*), 481–489
 - SCons.dblite (*module*), 490
 - SCons.dblite.corruption_warning (*function*), 490

- SCons.dblite.dblite (*class*), 490
- SCons.dblite.is_string (*function*), 490
- SCons.dblite.open (*function*), 490
- SCons.dblite.unicode (*function*), 490
- SCons.Debug (*module*), 34–35
 - SCons.Debug.caller_stack (*function*), 34
 - SCons.Debug.caller_trace (*function*), 34
 - SCons.Debug.countLoggedInstances (*function*), 34
 - SCons.Debug.dump_caller_counts (*function*), 34
 - SCons.Debug.dumpLoggedInstances (*function*), 34
 - SCons.Debug.fetchLoggedInstances (*function*), 34
 - SCons.Debug.func_shorten (*function*), 34
 - SCons.Debug.listLoggedInstances (*function*), 34
 - SCons.Debug.logInstanceCreation (*function*), 34
 - SCons.Debug.memory (*function*), 34
 - SCons.Debug.string_to_classes (*function*), 34
 - SCons.Debug.Trace (*function*), 34
- SCons.Defaults (*module*), 36–38
- SCons.Environment (*module*), 39–73
 - SCons.Environment.alias_builder (*function*), 39
 - SCons.Environment.apply_tools (*function*), 39
 - SCons.Environment.Base (*class*), 45–54, 64–73
 - SCons.Environment.build_source (*function*), 39
 - SCons.Environment.BuilderDict (*class*), 41–42
 - SCons.Environment.BuilderWrapper (*class*), 40–41
 - SCons.Environment.copy_non_reserved_keywords (*function*), 39
 - SCons.Environment.default_copy_from_cache (*function*), 39
 - SCons.Environment.default_decide_source (*function*), 39
 - SCons.Environment.default_decide_target (*function*), 39
 - SCons.Environment.is_valid_construction_var (*function*), 39
 - SCons.Environment.MethodWrapper (*class*), 40
 - SCons.Environment.NoSubstitutionProxy (*function*), 39
 - SCons.Environment.OverrideEnvironment (*class*), 54–64
 - SCons.Environment.SubstitutionEnvironment (*class*), 42–45
- SCons.Errors (*module*), 74–86
- SCons.Executor (*module*), 87–90
 - SCons.Executor.Executor (*class*), 87–89
 - SCons.Executor.get_NullEnvironment (*function*), 87
 - SCons.Executor.Null (*class*), 89–90
- SCons.exitfuncs (*module*), 491
 - SCons.exitfuncs.register (*function*), 491
- SCons.Job (*module*), 91–95
 - SCons.Job.InterruptState (*class*), 91
 - SCons.Job.Jobs (*class*), 91–92
 - SCons.Job.Parallel (*class*), 95
 - SCons.Job.Serial (*class*), 92
 - SCons.Job.ThreadPool (*class*), 94–95
 - SCons.Job.Worker (*class*), 92–94
- SCons.Memoize (*module*), 96–102
- SCons.Node (*package*), 103–117
 - SCons.Node.Alias (*module*), 118–131
 - SCons.Node.FS (*module*), 132–213
 - SCons.Node.Python (*module*), 214–226
- SCons.PathList (*module*), 227
 - SCons.PathList.node_conv (*function*), 227
 - SCons.PathList.PathList (*function*), 227
- SCons.Scanner (*module*)
 - SCons.Scanner.Base (*class*), 253–255
 - SCons.Scanner.Classic (*class*), 261–262
 - SCons.Scanner.ClassicCPP (*class*), 262–264
 - SCons.Scanner.Current (*class*), 258–261
 - SCons.Scanner.FindPathDirs (*class*), 253
 - SCons.Scanner.Scanner (*function*), 252
 - SCons.Scanner.Selector (*class*), 255–258
- SCons.Scanner (*package*), 252–264
 - SCons.Scanner.C (*module*), 265–269
 - SCons.Scanner.D (*module*), 270–271
 - SCons.Scanner.Dir (*module*), 272
 - SCons.Scanner.Fortran (*module*), 273–274
 - SCons.Scanner.IDL (*module*), 275
 - SCons.Scanner.LaTeX (*module*), 276–280
 - SCons.Scanner.Prog (*module*), 281
 - SCons.Scanner.RC (*module*), 282
- SCons.SConf (*module*), 228–245
 - SCons.SConf.CheckCC (*function*), 228
 - SCons.SConf.CheckCHheader (*function*), 228
 - SCons.SConf.CheckContext (*class*), 244–245
 - SCons.SConf.CheckCXX (*function*), 228
 - SCons.SConf.CheckCXXHeader (*function*), 229
 - SCons.SConf.CheckDeclaration (*function*), 228
 - SCons.SConf.CheckFunc (*function*), 228
 - SCons.SConf.CheckHeader (*function*), 228
 - SCons.SConf.CheckLib (*function*), 229
 - SCons.SConf.CheckLibWithHeader (*function*), 229
 - SCons.SConf.CheckSHCC (*function*), 228
 - SCons.SConf.CheckSHCXX (*function*), 228

- SCons.SConf.CheckType (function), 228
- SCons.SConf.CheckTypeSize (function), 228
- SCons.SConf.ConfigureCacheError (class), 235–237
- SCons.SConf.ConfigureDryRunError (class), 233–235
- SCons.SConf.CreateConfigHBuilder (function), 228
- SCons.SConf.createIncludesFromHeaders (function), 228
- SCons.SConf.SConf (function), 228
- SCons.SConf.SConfBase (class), 242–244
- SCons.SConf.SConfBuildInfo (class), 237–238
- SCons.SConf.SConfBuildTask (class), 238–242
- SCons.SConf.SConfError (class), 231–233
- SCons.SConf.SConfWarning (class), 230–231
- SCons.SConf.SetBuildType (function), 228
- SCons.SConf.SetCacheMode (function), 228
- SCons.SConf.SetProgressDisplay (function), 228
- SCons.SConf.Streamer (class), 238
- SCons.SConsign (module), 246–251
 - SCons.SConsign.Base (class), 247
 - SCons.SConsign.corrupt_dblite_warning (function), 246
 - SCons.SConsign.DB (class), 247–248, 250–251
 - SCons.SConsign.Dir (class), 248–249
 - SCons.SConsign.DirFile (class), 249–250
 - SCons.SConsign.File (function), 246
 - SCons.SConsign.Get_Database (function), 246
 - SCons.SConsign.Reset (function), 246
 - SCons.SConsign.SConsignEntry (class), 246–247
 - SCons.SConsign.write (function), 246
- SCons.Script (module)
 - SCons.Script.HelpFunction (function), 283
 - SCons.Script.Options (function), 283
 - SCons.Script.TargetList (class), 287–289
 - SCons.Script.Variables (function), 283
- SCons.Script (package), 283–289
 - SCons.Script.Interactive (module), 290–293
 - SCons.Script.Main (module), 294–310
 - SCons.Script.SConscript' (module), 311–323
- SCons.Sig (module), 324–325
 - SCons.Sig.MD5Null (class), 324–325
 - SCons.Sig.TimeStampNull (class), 325
- SCons.Subst (module), 326–334
 - SCons.Subst.CmdStringHolder (class), 328–331
 - SCons.Subst.escape_list (function), 326
 - SCons.Subst.Literal (class), 327–328
 - SCons.Subst.NLWrapper (class), 331
 - SCons.Subst.quote_spaces (function), 326
 - SCons.Subst.raise_exception (function), 326
 - SCons.Subst.scons_subst (function), 326
 - SCons.Subst.scons_subst_list (function), 326
 - SCons.Subst.scons_subst_once (function), 327
 - SCons.Subst.SetAllowableExceptions (function), 326
 - SCons.Subst.SpecialAttrWrapper (class), 328
 - SCons.Subst.subst_dict (function), 326
 - SCons.Subst.Target_or_Source (class), 333–334
 - SCons.Subst.Targets_or_Sources (class), 331–333
- SCons.Taskmaster (module), 335–341
 - SCons.Taskmaster.dump_stats (function), 335
 - SCons.Taskmaster.find_cycle (function), 335
 - SCons.Taskmaster.Stats (class), 336
 - SCons.Taskmaster.Task (class), 336–340
 - SCons.Taskmaster.Taskmaster (class), 340–341
- SCons.Util (module), 342–371
- SCons.Variables (package), 372–374
 - SCons.Variables.BoolVariable' (module), 375
 - SCons.Variables.EnumVariable' (module), 376
 - SCons.Variables.ListVariable' (module), 377
 - SCons.Variables.PackageVariable' (module), 378
 - SCons.Variables.PathVariable' (module), 379–380
 - SCons.Variables.Variables (class), 372–374
- SCons.Warnings (module), 381–418
- SCons.Action._ActionAction (class)
 - SCons.Action._ActionAction.__call__ (method), 9, 12
 - SCons.Action._ActionAction.print_cmd_line (method), 9, 11, 13
- SCons.Action.Action (function), 7
- SCons.Action.ActionBase (class), 8
 - SCons.Action.ActionBase.__add__ (method), 8–13
 - SCons.Action.ActionBase.__cmp__ (method), 8–13
 - SCons.Action.ActionBase.__radd__ (method), 8–13
 - SCons.Action.ActionBase.genstring (method), 8, 9, 12
 - SCons.Action.ActionBase.get_executor (method), 8–13
 - SCons.Action.ActionBase.presub_lines (method), 8–12
- SCons.Action.ActionCaller (class), 13–14
 - SCons.Action.ActionCaller.__call__ (method), 14
 - SCons.Action.ActionCaller.__init__ (method), 14
 - SCons.Action.ActionCaller.__str__ (method), 14
 - SCons.Action.ActionCaller.get_contents (method), 14

- SCons.Action.ActionCaller.strfunction (method), 14
- SCons.Action.ActionCaller.subst (method), 14
- SCons.Action.ActionCaller.subst_args (method), 14
- SCons.Action.ActionCaller.subst_kw (method), 14
- SCons.Action.ActionFactory (class), 14
 - SCons.Action.ActionFactory.__call__ (method), 14
 - SCons.Action.ActionFactory.__init__ (method), 14
- SCons.Action.CommandAction (class), 8–9
 - SCons.Action.CommandAction.__str__ (method), 8
 - SCons.Action.CommandAction.execute (method), 8, 11
 - SCons.Action.CommandAction.get_contents (method), 9
 - SCons.Action.CommandAction.get_implicit_deps (method), 9
 - SCons.Action.CommandAction.process (method), 8, 11
 - SCons.Action.CommandAction.strfunction (method), 8, 11
- SCons.Action.CommandGeneratorAction (class), 9–10
 - SCons.Action.CommandGeneratorAction.__call__ (method), 10
 - SCons.Action.CommandGeneratorAction.__init__ (method), 10
 - SCons.Action.CommandGeneratorAction.__str__ (method), 10, 11
 - SCons.Action.CommandGeneratorAction.get_contents (method), 10
 - SCons.Action.CommandGeneratorAction.get_implicit_deps (method), 10, 11
- SCons.Action.default_exitstatfunc (function), 7
- SCons.Action.FunctionAction (class), 12–13
 - SCons.Action.FunctionAction.__str__ (method), 12
 - SCons.Action.FunctionAction.execute (method), 12
 - SCons.Action.FunctionAction.function_name (method), 12
 - SCons.Action.FunctionAction.get_contents (method), 12
 - SCons.Action.FunctionAction.get_implicit_deps (method), 12
 - SCons.Action.FunctionAction.strfunction (method), 12
- SCons.Action.LazyAction (class), 10–12
 - SCons.Action.LazyAction.get_parent_class (method), 11
- SCons.Action.ListAction (class), 13
 - SCons.Action.ListAction.__call__ (method), 13
 - SCons.Action.ListAction.__init__ (method), 13
 - SCons.Action.ListAction.__str__ (method), 13
 - SCons.Action.ListAction.get_contents (method), 13
 - SCons.Action.ListAction.get_implicit_deps (method), 13
- SCons.Action.remove_set_lineno_codes (function), 7
- SCons.Action.rfile (function), 7
- SCons.Conftest.CheckBuilder (function), 30
- SCons.Conftest.CheckCC (function), 30
- SCons.Conftest.CheckCXX (function), 30
- SCons.Conftest.CheckDeclaration (function), 32
- SCons.Conftest.CheckFunc (function), 31
- SCons.Conftest.CheckHeader (function), 31
- SCons.Conftest.CheckLib (function), 32
- SCons.Conftest.CheckSHCC (function), 30
- SCons.Conftest.CheckSHCXX (function), 30
- SCons.Conftest.CheckType (function), 31
- SCons.Conftest.CheckTypeSize (function), 31
- SCons.cpp.CPP_to_Python (function), 481
- SCons.cpp.CPP_to_Python_Ops.Sub (function), 481
- SCons.cpp.DumbPreProcessor (class), 485–489
- SCons.cpp.FunctionEvaluator (class), 481–482
 - SCons.cpp.FunctionEvaluator.__call__ (method), 482
 - SCons.cpp.FunctionEvaluator.__init__ (method), 482
- SCons.cpp.PreProcessor (class), 482–485
 - SCons.cpp.PreProcessor.__call__ (method), 266, 482, 485
 - SCons.cpp.PreProcessor.__init__ (method), 482
 - SCons.cpp.PreProcessor.all.include (method), 266, 482, 486
 - SCons.cpp.PreProcessor.do.define (method), 266, 482, 486
 - SCons.cpp.PreProcessor.do.elif (method), 266, 482, 486
 - SCons.cpp.PreProcessor.do.else (method), 266, 482, 486
 - SCons.cpp.PreProcessor.do.endif (method), 266, 483, 486
 - SCons.cpp.PreProcessor.do.if (method), 266, 483, 486
 - SCons.cpp.PreProcessor.do.ifdef (method), 266, 483, 486
 - SCons.cpp.PreProcessor.do.ifndef (method), 266, 483, 486
 - SCons.cpp.PreProcessor.do.import (method), 267, 483, 486
 - SCons.cpp.PreProcessor.do.include (method), 267, 483, 486, 487

- SCons.cpp.PreProcessor.do_nothing (*method*), 267, 483, 487
 SCons.cpp.PreProcessor.do_undef (*method*), 267, 483, 487
 SCons.cpp.PreProcessor.eval_expression (*method*), 267, 483, 487
 SCons.cpp.PreProcessor.finalize_result (*method*), 484, 487
 SCons.cpp.PreProcessor.find_include_file (*method*), 484, 487
 SCons.cpp.PreProcessor.initialize_result (*method*), 484, 487
 SCons.cpp.PreProcessor.process_contents (*method*), 267, 484, 487
 SCons.cpp.PreProcessor.read_file (*method*), 484, 487
 SCons.cpp.PreProcessor.resolve_include (*method*), 267, 484, 487
 SCons.cpp.PreProcessor.restore (*method*), 268, 484, 488
 SCons.cpp.PreProcessor.save (*method*), 268, 484, 488
 SCons.cpp.PreProcessor.scons_current_file (*method*), 268, 484, 488
 SCons.cpp.PreProcessor.start_handling_includes (*method*), 268, 484, 488
 SCons.cpp.PreProcessor.stop_handling_includes (*method*), 268, 485, 488
 SCons.cpp.PreProcessor.tupleize (*method*), 268, 485, 488
 SCons.Defaults.chmod_func (*function*), 36
 SCons.Defaults.chmod_strfunc (*function*), 36
 SCons.Defaults.copy_func (*function*), 36
 SCons.Defaults.DefaultEnvironment (*function*), 36
 SCons.Defaults.delete_func (*function*), 36
 SCons.Defaults.delete_strfunc (*function*), 36
 SCons.Defaults.get_paths_str (*function*), 36
 SCons.Defaults.mkdir_func (*function*), 36
 SCons.Defaults.move_func (*function*), 37
 SCons.Defaults.NullCmdGenerator (*class*), 37–38
 SCons.Defaults.NullCmdGenerator.__call__ (*method*), 38
 SCons.Defaults.NullCmdGenerator.__init__ (*method*), 38
 SCons.Defaults.SharedFlagChecker (*function*), 36
 SCons.Defaults.SharedObjectEmitter (*function*), 36
 SCons.Defaults.StaticObjectEmitter (*function*), 36
 SCons.Defaults.touch_func (*function*), 37
 SCons.Defaults.Variable_Method_Caller (*class*), 38
 SCons.Defaults.Variable_Method_Caller.__call__ (*method*), 38
 SCons.Defaults.Variable_Method_Caller.__init__ (*method*), 38
 SCons.Errors.BuildError (*class*), 74–76
 SCons.Errors.EnvironmentError (*class*), 81–83
 SCons.Errors.ExplicitExit (*class*), 83–84
 SCons.Errors.InternalError (*class*), 76–77
 SCons.Errors.StopError (*class*), 79–81
 SCons.Errors.TaskmasterException (*class*), 84–86
 SCons.Errors.UserError (*class*), 77–79
 SCons.Memoize.CountDict (*class*), 99–100
 SCons.Memoize.CountDict.__call__ (*method*), 99
 SCons.Memoize.Counter (*class*), 98
 SCons.Memoize.Counter.__cmp__ (*method*), 98, 99
 SCons.Memoize.Counter.__init__ (*method*), 98, 99
 SCons.Memoize.Counter.display (*method*), 98, 99
 SCons.Memoize.CountValue (*class*), 98–99
 SCons.Memoize.CountValue.__call__ (*method*), 99
 SCons.Memoize.Dump (*function*), 97
 SCons.Memoize.EnableMemoization (*function*), 97
 SCons.Memoize.Memoized_Metaclass (*class*), 100–102
 SCons.Memoize.Memoizer (*class*), 100
 SCons.Memoize.Memoizer.__init__ (*method*), 100
 SCons.Node.Annotate (*function*), 103
 SCons.Node.BuildInfoBase (*class*), 105
 SCons.Node.BuildInfoBase.__init__ (*method*), 105, 120, 165, 197, 215, 237
 SCons.Node.BuildInfoBase.merge (*method*), 105, 120, 165, 197, 215, 237
 SCons.Node.classname (*function*), 103
 SCons.Node.do_nothing (*function*), 103
 SCons.Node.get_children (*function*), 103
 SCons.Node.ignore_cycle (*function*), 103
 SCons.Node.Node (*class*), 105–115
 SCons.Node.Node.__init__ (*method*), 105
 SCons.Node.Node.add_dependency (*method*), 105, 122, 139, 149, 172, 182, 203, 217
 SCons.Node.Node.add_ignore (*method*), 105, 122, 139, 150, 172, 182, 203, 217
 SCons.Node.Node.add_prerequisite (*method*), 105, 122, 139, 150, 172, 183, 204, 217
 SCons.Node.Node.add_source (*method*), 106, 122, 139, 150, 172, 183, 204, 217
 SCons.Node.Node.add_to_implicit (*method*), 106, 123, 139, 150, 172, 183, 204, 217
 SCons.Node.Node.add_to_waiting_parents (*method*), 106, 123, 139, 150, 172, 183, 204, 217
 SCons.Node.Node.add_to_waiting_s.e (*method*), 106, 123, 139, 150, 172, 183, 204, 218
 SCons.Node.Node.add_wkid (*method*), 106, 123, 139, 150, 172, 183, 204, 218

- SCons.Node.Node.all_children (*method*), 106, 123, 139, 150, 172, 183, 204, 218
- SCons.Node.Node.alter_targets (*method*), 106, 123, 139, 150, 218
- SCons.Node.Node.build (*method*), 106, 139, 150, 204
- SCons.Node.Node.builder_set (*method*), 106, 123, 140, 151, 172, 183, 218
- SCons.Node.Node.built (*method*), 106, 123, 140, 151, 172, 183, 218
- SCons.Node.Node.changed (*method*), 106, 123, 140, 151, 172, 183, 204, 218
- SCons.Node.Node.changed_since_last_build (*method*), 107, 140
- SCons.Node.Node.children (*method*), 107, 123, 140, 151, 173, 184, 205, 218
- SCons.Node.Node.children_are_up_to_date (*method*), 107, 124, 141, 151, 173, 184, 205, 218
- SCons.Node.Node.clear (*method*), 107, 124, 141, 151, 173, 184, 205, 219
- SCons.Node.Node.clear_memoized_values (*method*), 108, 124, 141, 151, 173, 185, 205, 219
- SCons.Node.Node.Decider (*method*), 105, 122, 139, 149, 171, 182, 203, 217
- SCons.Node.Node.del_binfo (*method*), 108, 124, 141, 152, 173, 185, 205, 219
- SCons.Node.Node.disambiguate (*method*), 108, 124, 141, 173, 185, 205, 219
- SCons.Node.Node.do_not_store_info (*method*), 108, 124, 141, 152, 173, 185, 205, 219
- SCons.Node.Node.env_set (*method*), 108, 124, 141, 152, 173, 185, 205, 219
- SCons.Node.Node.executor_cleanup (*method*), 108, 124, 141, 152, 173, 185, 205, 219
- SCons.Node.Node.exists (*method*), 108, 124, 219
- SCons.Node.Node.explain (*method*), 108, 124, 141, 152, 174, 185, 205, 219
- SCons.Node.Node.for_signature (*method*), 108, 124, 219
- SCons.Node.Node.get_abspath (*method*), 108, 125, 219
- SCons.Node.Node.get_binfo (*method*), 108, 125, 141, 152, 174, 186, 206, 220
- SCons.Node.Node.get_build_env (*method*), 109, 125, 141, 152, 174, 186, 206, 220
- SCons.Node.Node.get_build_scanner_path (*method*), 109, 125, 142, 153, 174, 186, 206, 220
- SCons.Node.Node.get_builder (*method*), 109, 125, 142, 153, 174, 186, 206, 220
- SCons.Node.Node.get_cachedir_csig (*method*), 109, 125, 142, 153, 175, 187, 220
- SCons.Node.Node.get_csig (*method*), 109, 142, 153
- SCons.Node.Node.get_env (*method*), 109, 125, 142, 153, 175, 187, 206, 220
- SCons.Node.Node.get_env_scanner (*method*), 109, 125, 142, 153, 207, 220
- SCons.Node.Node.get_executor (*method*), 109, 126, 142, 153, 175, 187, 207, 220
- SCons.Node.Node.get_found_includes (*method*), 109, 126, 142, 153, 220
- SCons.Node.Node.get_implicit_deps (*method*), 109, 126, 142, 153, 175, 187, 207, 221
- SCons.Node.Node.get_ninfo (*method*), 110, 126, 142, 153, 175, 187, 207, 221
- SCons.Node.Node.get_source_scanner (*method*), 110, 126, 142, 154, 175, 188, 207, 221
- SCons.Node.Node.get_state (*method*), 110, 126, 143, 154, 175, 188, 207, 221
- SCons.Node.Node.get_stored_implicit (*method*), 110, 126, 143, 154, 175, 188, 221
- SCons.Node.Node.get_stored_info (*method*), 110, 126, 143, 154, 175, 188, 221
- SCons.Node.Node.get_string (*method*), 110, 126, 143, 154, 175, 188, 207, 221
- SCons.Node.Node.get_subst_proxy (*method*), 110, 127, 222
- SCons.Node.Node.get_suffix (*method*), 111, 127, 222
- SCons.Node.Node.get_target_scanner (*method*), 111, 127, 143, 155, 208, 222
- SCons.Node.Node.has_builder (*method*), 111, 112, 127, 128, 143, 144, 155, 156, 176, 190, 208, 209, 222, 223
- SCons.Node.Node.has_explicit_builder (*method*), 111, 127, 143, 155, 176, 190, 208, 222
- SCons.Node.Node.is_derived (*method*), 111, 128, 144, 155, 177, 191, 209, 223
- SCons.Node.Node.is_literal (*method*), 111, 128, 144, 156, 177, 191, 209, 223
- SCons.Node.Node.is_up_to_date (*method*), 112, 144, 156
- SCons.Node.Node.make_ready (*method*), 112, 144, 156, 177, 191
- SCons.Node.Node.missing (*method*), 112, 128, 144, 156, 177, 191, 209, 223
- SCons.Node.Node.new_binfo (*method*), 112, 128, 145, 157, 177, 192, 210, 223
- SCons.Node.Node.new_ninfo (*method*), 112, 128, 145, 178, 192, 210, 223
- SCons.Node.Node.postprocess (*method*), 112, 128, 145, 157, 178, 192, 210, 223

- SCons.Node.Node.prepare (*method*), 112, 128, 145, 157, 223
- SCons.Node.Node.remove (*method*), 113, 129, 145, 157, 178, 193, 224
- SCons.Node.Node.render_include_tree (*method*), 113, 129, 145, 157, 178, 193, 210, 224
- SCons.Node.Node.reset_executor (*method*), 113, 129, 146, 158, 178, 193, 210, 224
- SCons.Node.Node.retrieve_from_cache (*method*), 113, 129, 146, 158, 178, 193, 224
- SCons.Node.Node.rexists (*method*), 113, 129, 224
- SCons.Node.Node.scan (*method*), 113, 129, 146, 158, 178, 193, 210, 224
- SCons.Node.Node.scanner_key (*method*), 114, 130, 146, 225
- SCons.Node.Node.select_scanner (*method*), 114, 130, 146, 158, 178, 194, 210, 225
- SCons.Node.Node.set_always_build (*method*), 114, 130, 146, 158, 179, 194, 210, 225
- SCons.Node.Node.set_executor (*method*), 114, 130, 146, 158, 179, 194, 210, 225
- SCons.Node.Node.set_explicit (*method*), 114, 130, 146, 158, 179, 194, 211, 225
- SCons.Node.Node.set_nocache (*method*), 114, 130, 146, 158, 179, 194, 211, 225
- SCons.Node.Node.set_noclean (*method*), 114, 130, 146, 159, 179, 194, 211, 225
- SCons.Node.Node.set_precious (*method*), 114, 130, 147, 159, 179, 194, 211, 225
- SCons.Node.Node.set_specific_source (*method*), 114, 130, 147, 159, 179, 194, 211, 225
- SCons.Node.Node.set_state (*method*), 114, 130, 147, 159, 179, 194, 211, 225
- SCons.Node.Node.state_has_changed (*method*), 114, 130, 147, 159, 179, 195, 211, 225
- SCons.Node.Node.store_info (*method*), 114, 130, 147, 159, 180, 195, 225
- SCons.Node.Node.visited (*method*), 114, 130, 147, 160, 180, 195, 225
- SCons.Node.NodeInfoBase (*class*), 104–105
 - SCons.Node.NodeInfoBase.__init__ (*method*), 104, 119, 164, 196, 214
 - SCons.Node.NodeInfoBase.convert (*method*), 104, 119, 164, 196, 214
 - SCons.Node.NodeInfoBase.format (*method*), 104, 119, 164, 196, 214
 - SCons.Node.NodeInfoBase.merge (*method*), 104, 120, 164, 196, 214
 - SCons.Node.NodeInfoBase.update (*method*), 104, 120, 164, 196, 214
- SCons.Node.NodeList (*class*), 115–116
 - SCons.Node.NodeList.__str__ (*method*), 115
- SCons.Node.Walker (*class*), 116–117
 - SCons.Node.Walker.__init__ (*method*), 117
 - SCons.Node.Walker.is_done (*method*), 117
 - SCons.Node.Walker.next (*method*), 117
- SCons.Scanner.Dir.DirEntryScanner (*function*), 272
- SCons.Scanner.Dir.DirScanner (*function*), 272
- SCons.Scanner.Dir.do_not_scan (*function*), 272
- SCons.Scanner.Dir.only_dirs (*function*), 272
- SCons.Scanner.Dir.scan_in_memory (*function*), 272
- SCons.Scanner.Dir.scan_on_disk (*function*), 272
- SCons.Script.Interactive.interact (*function*), 290
- SCons.Script.Interactive.SConsInteractiveCmd (*class*), 290–293
 - SCons.Script.Interactive.SConsInteractiveCmd.do_build (*method*), 291
 - SCons.Script.Interactive.SConsInteractiveCmd.do_clean (*method*), 291
 - SCons.Script.Interactive.SConsInteractiveCmd.do_EOF (*method*), 291
 - SCons.Script.Interactive.SConsInteractiveCmd.do_exit (*method*), 291
 - SCons.Script.Interactive.SConsInteractiveCmd.do_shell (*method*), 291
 - SCons.Script.Interactive.SConsInteractiveCmd.do_version (*method*), 291
- SCons.Util.NoError (*class*), 361–363
- SCons.Util.AddMethod (*function*), 347
- SCons.Util.adjustixes (*function*), 346
- SCons.Util.AppendPath (*function*), 345
- SCons.Util.CallableComposite (*class*), 348–350
 - SCons.Util.CallableComposite.__call__ (*method*), 348
- SCons.Util.case_sensitive_suffixes (*function*), 346
- SCons.Util.CLVar (*class*), 363–365
 - SCons.Util.CLVar.__coerce__ (*method*), 364
 - SCons.Util.CLVar.__str__ (*method*), 364
- SCons.Util.containsAll (*function*), 342
- SCons.Util.containsAny (*function*), 342
- SCons.Util.containsOnly (*function*), 342
- SCons.Util.dictify (*function*), 342
- SCons.Util.DisplayEngine (*class*), 351–352
 - SCons.Util.DisplayEngine.__init__ (*method*), 352
 - SCons.Util.DisplayEngine.dont_print (*method*), 352
 - SCons.Util.DisplayEngine.print_it (*method*), 352
 - SCons.Util.DisplayEngine.set_mode (*method*), 352
- SCons.Util.do_flatten (*function*), 343
- SCons.Util.flatten (*function*), 343
- SCons.Util.flatten_sequence (*function*), 344
- SCons.Util.get_environment_var (*function*), 342
- SCons.Util.get_native_path (*function*), 346

- SCons.Util.IDX (*function*), 343
- SCons.Util.is_Dict (*function*), 343
- SCons.Util.is_List (*function*), 343
- SCons.Util.is_Scalar (*function*), 343
- SCons.Util.is_Sequence (*function*), 343
- SCons.Util.is_String (*function*), 343
- SCons.Util.is_Tuple (*function*), 343
- SCons.Util.LogicalLines (*class*), 368
 - SCons.Util.LogicalLines.__init__ (*method*), 368
 - SCons.Util.LogicalLines.readline (*method*), 368
 - SCons.Util.LogicalLines.readlines (*method*), 368
- SCons.Util.make_path_relative (*function*), 347
- SCons.Util.MD5collect (*function*), 347
- SCons.Util.MD5filesignature (*function*), 347
- SCons.Util.MD5signature (*function*), 347
- SCons.Util.mystr (*class*), 352–361
- SCons.Util.NodeList (*class*), 350–351
 - SCons.Util.NodeList.__getattr__ (*method*), 350
 - SCons.Util.NodeList.__nonzero__ (*method*), 350
 - SCons.Util.NodeList.__str__ (*method*), 350
- SCons.Util.Null (*class*), 371
 - SCons.Util.Null.__call__ (*method*), 324, 325, 371
 - SCons.Util.Null.__delattr__ (*method*), 324, 325, 371
 - SCons.Util.Null.__getattr__ (*method*), 324, 325, 371
 - SCons.Util.Null.__init__ (*method*), 324, 325, 371
 - SCons.Util.Null.__new__ (*method*), 324, 325, 371
 - SCons.Util.Null.__nonzero__ (*method*), 324, 325, 371
 - SCons.Util.Null.__repr__ (*method*), 371
 - SCons.Util.Null.__setattr__ (*method*), 325, 371
- SCons.Util.OrderedDict (*class*), 365–366
- SCons.Util.PrependPath (*function*), 345
- SCons.Util.print_tree (*function*), 343
- SCons.Util.Proxy (*class*), 361
 - SCons.Util.Proxy.__cmp__ (*method*), 27, 135, 361
 - SCons.Util.Proxy.__getattr__ (*method*), 27, 361
 - SCons.Util.Proxy.__init__ (*method*), 135, 361
 - SCons.Util.Proxy.get (*method*), 27, 135, 361
- SCons.Util.RegGetValue (*function*), 344
- SCons.Util.RenameFunction (*function*), 347
- SCons.Util.render_tree (*function*), 342
- SCons.Util.Selector (*class*), 366–368
 - SCons.Util.Selector.__call__ (*method*), 367
- SCons.Util.semi_deepcopy (*function*), 344
- SCons.Util.Split (*function*), 346
- SCons.Util.splitext (*function*), 342
- SCons.Util.to_String (*function*), 344
- SCons.Util.to_String_for_signature (*function*), 344
- SCons.Util.to_String_for_subst (*function*), 344
- SCons.Util.Unbuffered (*class*), 370–371
 - SCons.Util.Unbuffered.__getattr__ (*method*), 370
 - SCons.Util.Unbuffered.__init__ (*method*), 370
 - SCons.Util.Unbuffered.write (*method*), 370
- SCons.Util.unique (*function*), 346
- SCons.Util.UniqueList (*class*), 368–370
- SCons.Util.uniquer (*function*), 346
- SCons.Util.uniquer_hashables (*function*), 346
- SCons.Util.updrive (*function*), 342
- SCons.Util.WhereIs (*function*), 345
- SCons.Warnings.CacheWriteErrorWarning (*class*), 384–385
- SCons.Warnings.CorruptSConsignWarning (*class*), 385–387
- SCons.Warnings.DependencyWarning (*class*), 387–389
- SCons.Warnings.DeprecatedCopyWarning (*class*), 391–393
- SCons.Warnings.DeprecatedSourceSignaturesWarning (*class*), 393–395
- SCons.Warnings.DeprecatedTargetSignaturesWarning (*class*), 395–397
- SCons.Warnings.DeprecatedWarning (*class*), 389–391
- SCons.Warnings.DuplicateEnvironmentWarning (*class*), 397–398
- SCons.Warnings.enableWarningClass (*function*), 381
- SCons.Warnings.FortranCxxMixWarning (*class*), 416–418
- SCons.Warnings.LinkWarning (*class*), 398–400
- SCons.Warnings.MisleadingKeywordsWarning (*class*), 400–402
- SCons.Warnings.MissingSConscriptWarning (*class*), 402–404
- SCons.Warnings.NoMD5ModuleWarning (*class*), 404–405
- SCons.Warnings.NoMetaclassSupportWarning (*class*), 405–407
- SCons.Warnings.NoObjectCountWarning (*class*), 407–409
- SCons.Warnings.NoParallelSupportWarning (*class*), 409–411
- SCons.Warnings.process_warn_strings (*function*), 381
- SCons.Warnings.PythonVersionWarning (*class*), 411–413
- SCons.Warnings.ReservedVariableWarning (*class*), 413–414
- SCons.Warnings.StackSizeWarning (*class*), 414–416
- SCons.Warnings.suppressWarningClass (*function*), 381
- SCons.Warnings.warn (*function*), 381
- SCons.Warnings.Warning (*class*), 382–384
- SCons.Warnings.warningAsException (*function*), 381
- str.__add__ (*function*), 352
- str.__contains__ (*function*), 352

str.__eq__ (function), 352
 str.__ge__ (function), 352
 str.__getitem__ (function), 352
 str.__getnewargs__ (function), 353
 str.__getslice__ (function), 353
 str.__gt__ (function), 353
 str.__le__ (function), 353
 str.__len__ (function), 353
 str.__lt__ (function), 353
 str.__mod__ (function), 353
 str.__mul__ (function), 353
 str.__ne__ (function), 353
 str.__rmod__ (function), 354
 str.__rmul__ (function), 354
 str.capitalize (function), 354
 str.center (function), 354
 str.count (function), 354
 str.decode (function), 355
 str.encode (function), 355
 str.endswith (function), 355
 str.expandtabs (function), 355
 str.find (function), 355
 str.index (function), 356
 str.isalnum (function), 356
 str.isalpha (function), 356
 str.isdigit (function), 356
 str.islower (function), 356
 str.isspace (function), 356
 str.istitle (function), 356
 str.isupper (function), 357
 str.join (function), 357
 str.ljust (function), 357
 str.lower (function), 357
 str.lstrip (function), 357
 str.partition (function), 357
 str.replace (function), 358
 str.rfind (function), 358
 str.rindex (function), 358
 str.rjust (function), 358
 str.rpartition (function), 358
 str.rsplit (function), 358
 str.rstrip (function), 359
 str.split (function), 359
 str.splitlines (function), 359
 str.startswith (function), 359
 str.strip (function), 359
 str.swapcase (function), 360
 str.title (function), 360
 str.translate (function), 360
 str.upper (function), 360
 str.zfill (function), 360

threading.Thread.getName (function), 94
 threading.Thread.isAlive (function), 94
 threading.Thread.isDaemon (function), 94
 threading.Thread.join (function), 94
 threading.Thread.setDaemon (function), 94
 threading.Thread.setName (function), 94
 threading.Thread.start (function), 94
 type.__call__ (function), 100
 type.__cmp__ (function), 100
 type.__subclasses__ (function), 101
 type.mro (function), 101

 UserDict.UserDict.__cmp__ (function), 17, 18, 20, 23, 42, 118, 366, 367
 UserDict.UserDict.__contains__ (function), 17, 18, 20, 23, 42, 118, 366, 367
 UserDict.UserDict.__delitem__ (function), 23, 118
 UserDict.UserDict.__getitem__ (function), 17, 19, 20, 23, 42, 118, 366, 367
 UserDict.UserDict.__init__ (function), 118
 UserDict.UserDict.__len__ (function), 17, 19, 20, 23, 42, 118, 366, 367
 UserDict.UserDict.__repr__ (function), 17, 19, 20, 23, 42, 118, 366, 367
 UserDict.UserDict.__setitem__ (function), 24, 118
 UserDict.UserDict.clear (function), 24, 42, 118
 UserDict.UserDict.copy (function), 24, 42, 118
 UserDict.UserDict.fromkeys (class method), 17, 19, 21, 24, 42, 119, 366, 367
 UserDict.UserDict.get (function), 17, 19, 21, 24, 42, 119, 366, 367
 UserDict.UserDict.has_key (function), 17, 19, 21, 24, 42, 119, 366, 367
 UserDict.UserDict.items (function), 24, 42, 119
 UserDict.UserDict.iteritems (function), 18, 19, 21, 24, 42, 119, 366, 368
 UserDict.UserDict.iterkeys (function), 18, 19, 21, 24, 42, 119, 366, 368
 UserDict.UserDict.itervalues (function), 18, 19, 21, 24, 42, 119, 366, 368
 UserDict.UserDict.keys (function), 24, 42, 119
 UserDict.UserDict.pop (function), 18, 19, 21, 24, 42, 119, 366, 368
 UserDict.UserDict.popitem (function), 24, 42, 119
 UserDict.UserDict.setdefault (function), 24, 42, 119
 UserDict.UserDict.update (function), 24, 119
 UserDict.UserDict.values (function), 24, 42, 119
 UserList.UserList.__add__ (function), 21, 115, 287, 332, 348, 350, 364
 UserList.UserList.__cmp__ (function), 22, 115, 287, 332, 348, 350, 364

- UserList.UserList.__contains__ (function), 22, 115, 287, 332, 348, 350, 364, 370
- UserList.UserList.__delitem__ (function), 22, 115, 287, 332, 348, 350, 364, 370
- UserList.UserList.__delslice__ (function), 22, 115, 287, 332, 348, 350, 364, 370
- UserList.UserList.__eq__ (function), 22, 115, 287, 332, 348, 350, 364
- UserList.UserList.__ge__ (function), 22, 115, 287, 332, 348, 350, 364
- UserList.UserList.__getitem__ (function), 22, 115, 287, 348, 350, 364
- UserList.UserList.__getslice__ (function), 22, 115, 287, 348, 350, 364
- UserList.UserList.__gt__ (function), 22, 115, 288, 332, 349, 350, 364
- UserList.UserList.__iadd__ (function), 22, 115, 288, 332, 349, 350, 364
- UserList.UserList.__imul__ (function), 22, 115, 288, 333, 349, 350, 364
- UserList.UserList.__init__ (function), 22, 115, 288, 349, 351
- UserList.UserList.__le__ (function), 22, 115, 288, 333, 349, 351, 364
- UserList.UserList.__len__ (function), 22, 116, 288, 333, 349, 351, 364
- UserList.UserList.__lt__ (function), 22, 116, 288, 333, 349, 351, 364
- UserList.UserList.__mul__ (function), 22, 116, 288, 333, 349, 351, 364
- UserList.UserList.__ne__ (function), 22, 116, 288, 333, 349, 351, 364
- UserList.UserList.__radd__ (function), 22, 116, 288, 333, 349, 351, 364
- UserList.UserList.__repr__ (function), 22, 116, 288, 349, 351, 364, 370
- UserList.UserList.__setitem__ (function), 22, 116, 288, 333, 349, 351, 365
- UserList.UserList.__setslice__ (function), 22, 116, 288, 333, 349, 351, 365
- UserList.UserList.append (function), 22, 116, 288, 333, 349, 351, 365
- UserList.UserList.count (function), 23, 116, 288, 333, 349, 351, 365
- UserList.UserList.extend (function), 23, 116, 288, 333, 349, 351, 365
- UserList.UserList.index (function), 23, 116, 288, 333, 349, 351, 365
- UserList.UserList.insert (function), 23, 116, 288, 333, 349, 351, 365
- UserList.UserList.pop (function), 23, 116, 288, 333, 349, 351, 365, 370
- UserList.UserList.remove (function), 23, 116, 288, 333, 349, 351, 365, 370
- UserList.UserList.reverse (function), 23, 116, 288, 333, 349, 351, 365
- UserList.UserList.sort (function), 23, 116, 288, 333, 349, 351, 365
- UserString.UserString.__add__ (function), 329
- UserString.UserString.__cmp__ (function), 329
- UserString.UserString.__complex__ (function), 329
- UserString.UserString.__contains__ (function), 329
- UserString.UserString.__float__ (function), 329
- UserString.UserString.__getitem__ (function), 329
- UserString.UserString.__getslice__ (function), 329
- UserString.UserString.__hash__ (function), 329
- UserString.UserString.__int__ (function), 329
- UserString.UserString.__len__ (function), 329
- UserString.UserString.__long__ (function), 329
- UserString.UserString.__mod__ (function), 329
- UserString.UserString.__mul__ (function), 329
- UserString.UserString.__radd__ (function), 329
- UserString.UserString.__repr__ (function), 329
- UserString.UserString.__str__ (function), 329
- UserString.UserString.capitalize (function), 329
- UserString.UserString.center (function), 329
- UserString.UserString.count (function), 330
- UserString.UserString.decode (function), 330
- UserString.UserString.encode (function), 330
- UserString.UserString.endswith (function), 330
- UserString.UserString.expandtabs (function), 330
- UserString.UserString.find (function), 330
- UserString.UserString.index (function), 330
- UserString.UserString.isalnum (function), 330
- UserString.UserString.isalpha (function), 330
- UserString.UserString.isdecimal (function), 330
- UserString.UserString.isdigit (function), 330
- UserString.UserString.islower (function), 330
- UserString.UserString.isnumeric (function), 330
- UserString.UserString.isspace (function), 330
- UserString.UserString.istitle (function), 330
- UserString.UserString.isupper (function), 330
- UserString.UserString.join (function), 330
- UserString.UserString.ljust (function), 330
- UserString.UserString.lower (function), 330
- UserString.UserString.lstrip (function), 330
- UserString.UserString.partition (function), 330
- UserString.UserString.replace (function), 330
- UserString.UserString.rfind (function), 330
- UserString.UserString.rindex (function), 331
- UserString.UserString.rjust (function), 331

UserString.UserString.rpartition (*function*), 331
UserString.UserString.rsplitleft (*function*), 331
UserString.UserString.rstrip (*function*), 331
UserString.UserString.split (*function*), 331
UserString.UserString.splitlines (*function*), 331
UserString.UserString.startswith (*function*), 331
UserString.UserString.strip (*function*), 331
UserString.UserString.swapcase (*function*), 331
UserString.UserString.title (*function*), 331
UserString.UserString.translate (*function*), 331
UserString.UserString.upper (*function*), 331
UserString.UserString.zfill (*function*), 331