Contents

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
    2.3.2 Properties .................................................... 9
  2.4 Class CommandAction .............................................. 10
    2.4.1 Methods ...................................................... 10
    2.4.2 Properties .................................................... 12
  2.5 Class CommandGeneratorAction ................................... 12
    2.5.1 Methods ...................................................... 12
    2.5.2 Properties .................................................... 14
  2.6 Class LazyAction .................................................. 14
    2.6.1 Methods ...................................................... 14
    2.6.2 Properties .................................................... 16
  2.7 Class FunctionAction .............................................. 17
    2.7.1 Methods ...................................................... 17
    2.7.2 Properties .................................................... 19
  2.8 Class ListAction .................................................. 19
    2.8.1 Methods ...................................................... 19
    2.8.2 Properties .................................................... 21
  2.9 Class ActionCaller ................................................ 21
    2.9.1 Methods ...................................................... 21
    2.9.2 Properties .................................................... 22
  2.10 Class ActionFactory ............................................... 22
    2.10.1 Methods ...................................................... 23
    2.10.2 Properties .................................................... 23

3 Module SCons.Builder  25
  3.1 Functions .......................................................... 26
  3.2 Variables .......................................................... 26
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Class DictCmdGenerator</td>
<td>27</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Methods</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>Class CallableSelector</td>
<td>28</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Methods</td>
<td>29</td>
</tr>
<tr>
<td>3.5</td>
<td>Class DictEmitter</td>
<td>30</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Methods</td>
<td>30</td>
</tr>
<tr>
<td>3.6</td>
<td>Class ListEmitter</td>
<td>31</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Methods</td>
<td>32</td>
</tr>
<tr>
<td>3.7</td>
<td>Class OverrideWarner</td>
<td>33</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Methods</td>
<td>33</td>
</tr>
<tr>
<td>3.8</td>
<td>Class EmitterProxy</td>
<td>34</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Methods</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Properties</td>
<td>36</td>
</tr>
<tr>
<td>3.9</td>
<td>Class BuilderBase</td>
<td>36</td>
</tr>
<tr>
<td>3.9.1</td>
<td>Methods</td>
<td>36</td>
</tr>
<tr>
<td>3.9.2</td>
<td>Properties</td>
<td>38</td>
</tr>
<tr>
<td>3.9.3</td>
<td>Class Variables</td>
<td>39</td>
</tr>
<tr>
<td>3.10</td>
<td>Class CompositeBuilder</td>
<td>39</td>
</tr>
<tr>
<td>3.10.1</td>
<td>Methods</td>
<td>39</td>
</tr>
<tr>
<td>3.10.2</td>
<td>Properties</td>
<td>40</td>
</tr>
<tr>
<td>4</td>
<td>Module SCons.CacheDir</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>Functions</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>Variables</td>
<td>41</td>
</tr>
<tr>
<td>4.3</td>
<td>Class CacheDir</td>
<td>41</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Methods</td>
<td>41</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Properties</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>Module SCons.Conftest</td>
<td>44</td>
</tr>
<tr>
<td>5.1</td>
<td>Functions</td>
<td>44</td>
</tr>
<tr>
<td>5.2</td>
<td>Variables</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>Module SCons.Debug</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>Functions</td>
<td>48</td>
</tr>
<tr>
<td>6.2</td>
<td>Variables</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>Module SCons.Defaults</td>
<td>50</td>
</tr>
<tr>
<td>7.1</td>
<td>Functions</td>
<td>50</td>
</tr>
<tr>
<td>7.2</td>
<td>Variables</td>
<td>51</td>
</tr>
<tr>
<td>7.3</td>
<td>Class NullCmdGenerator</td>
<td>52</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Methods</td>
<td>52</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Properties</td>
<td>53</td>
</tr>
<tr>
<td>7.4</td>
<td>Class Variable_Method_Caller</td>
<td>53</td>
</tr>
<tr>
<td>7.4.1</td>
<td>Methods</td>
<td>53</td>
</tr>
<tr>
<td>7.4.2</td>
<td>Properties</td>
<td>54</td>
</tr>
<tr>
<td>8</td>
<td>Module SCons.Environment</td>
<td>55</td>
</tr>
<tr>
<td>8.1</td>
<td>Functions</td>
<td>55</td>
</tr>
<tr>
<td>8.2</td>
<td>Variables</td>
<td>55</td>
</tr>
<tr>
<td>8.3</td>
<td>Class MethodWrapper</td>
<td>56</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Methods</td>
<td>56</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Properties</td>
<td>57</td>
</tr>
</tbody>
</table>
### 8.4 Class BuilderWrapper
- **Methods**: 58
- **Properties**: 59

### 8.5 Class BuilderDict
- **Methods**: 59
- **Properties**: 59

### 8.6 Class SubstitutionEnvironment
- **Methods**: 61
- **Properties**: 64
- **Class Variables**: 64

### 8.7 Class Base
- **Methods**: 64
- **Properties**: 74
- **Class Variables**: 74

### 8.8 Class OverrideEnvironment
- **Methods**: 75
- **Properties**: 84
- **Class Variables**: 84

### 8.9 Class Base
- **Methods**: 85
- **Properties**: 94
- **Class Variables**: 94

### 9 Module SCons.Errors
- **Functions**: 96
- **Variables**: 96
- **Class BuildError**: 96
  - **Methods**: 97
  - **Properties**: 99
- **Class InternalError**: 99
  - **Methods**: 99
  - **Properties**: 100
- **Class UserError**: 101
  - **Methods**: 101
  - **Properties**: 102
- **Class StopError**: 102
  - **Methods**: 103
  - **Properties**: 104
- **Class EnvironmentError**: 104
  - **Methods**: 104
  - **Properties**: 105
- **Class MSVCError**: 106
  - **Methods**: 106
  - **Properties**: 107
- **Class ExplicitExit**: 108
  - **Methods**: 108
  - **Properties**: 109

### 10 Module SCons.Executor
- **Functions**: 110
- **Variables**: 110
- **Class Batch**: 110
  - **Methods**: 110
  - **Properties**: 110
### 11 Module SCons.Job

11.1 Variables ................................................................. 121
11.2 Class InterruptState
   11.2.1 Methods .......................................................... 121
   11.2.2 Properties ...................................................... 122
11.3 Class Jobs ................................................................. 122
   11.3.1 Methods .......................................................... 123
   11.3.2 Properties ...................................................... 124
11.4 Class Serial ............................................................... 124
   11.4.1 Methods .......................................................... 124
   11.4.2 Properties ...................................................... 125
11.5 Class Worker ............................................................... 126
   11.5.1 Methods .......................................................... 126
   11.5.2 Properties ...................................................... 127
11.6 Class ThreadPool .......................................................... 127
   11.6.1 Methods .......................................................... 127
   11.6.2 Properties ...................................................... 129
11.7 Class Parallel ............................................................. 129
   11.7.1 Methods .......................................................... 129
   11.7.2 Properties ...................................................... 130

### 12 Module SCons.Memoize

12.1 Functions .................................................................. 131
12.2 Variables .................................................................. 132
12.3 Class Counter
   12.3.1 Methods .......................................................... 133
   12.3.2 Properties ...................................................... 134
12.4 Class CountValue
   12.4.1 Methods .......................................................... 134
   12.4.2 Properties ...................................................... 136
12.5 Class CountDict
   12.5.1 Methods .......................................................... 136
   12.5.2 Properties ...................................................... 137
12.6 Class Memoizer
   12.6.1 Methods .......................................................... 137
   12.6.2 Properties ...................................................... 138
12.7 Class Memoized_Metaclas
## 13 Package SCons.Node

- **13.1 Modules**
- **13.2 Functions**
- **13.3 Variables**

#### 13.4 Class NodeInfoBase
- **13.4.1 Methods**
- **13.4.2 Properties**
- **13.4.3 Class Variables**

#### 13.5 Class BuildInfoBase
- **13.5.1 Methods**
- **13.5.2 Properties**
- **13.5.3 Class Variables**

#### 13.6 Class Node
- **13.6.1 Methods**
- **13.6.2 Properties**
- **13.6.3 Class Variables**

#### 13.7 Class NodeList
- **13.7.1 Methods**

## 14 Module SCons.Node.Alias

- **14.1 Variables**

#### 14.2 Class AliasNameSpace
- **14.2.1 Methods**

#### 14.3 Class AliasNodeInfo
- **14.3.1 Methods**
- **14.3.2 Properties**
- **14.3.3 Class Variables**

#### 14.4 Class AliasBuildInfo
- **14.4.1 Methods**
- **14.4.2 Properties**
- **14.4.3 Class Variables**

#### 14.5 Class Alias
- **14.5.1 Methods**
- **14.5.2 Properties**
- **14.5.3 Class Variables**

## 15 Module SCons.Node.FS

- **15.1 Functions**
- **15.2 Variables**

#### 15.3 Class EntryProxyAttributeError
- **15.3.1 Methods**

#### 15.4 Class DiskChecker
- **15.4.1 Methods**

#### 15.5 Class EntryProxy
| 15.5.1 | Methods | ........................................ | 182 |
| 15.5.2 | Properties | ...................................... | 184 |
| 15.5.3 | Class Variables | .................................. | 184 |
| 15.6 | Class Base | ...................................... | 184 |
| 15.6.1 | Methods | ...................................... | 184 |
| 15.6.2 | Properties | .................................... | 197 |
| 15.6.3 | Class Variables | .................................. | 197 |
| 15.7 | Class Entry | ..................................... | 197 |
| 15.7.1 | Methods | ..................................... | 197 |
| 15.7.2 | Properties | ................................... | 210 |
| 15.7.3 | Class Variables | .................................. | 210 |
| 15.8 | Class LocalFS | ................................... | 211 |
| 15.8.1 | Methods | .................................. | 211 |
| 15.8.2 | Properties | .................................. | 212 |
| 15.8.3 | Class Variables | .................................. | 213 |
| 15.9 | Class FS | ...................................... | 213 |
| 15.9.1 | Methods | .................................. | 213 |
| 15.9.2 | Properties | .................................. | 216 |
| 15.9.3 | Class Variables | .................................. | 216 |
| 15.10 | Class DirNodeInfo | .................................. | 217 |
| 15.10.1 | Methods | .................................. | 217 |
| 15.10.2 | Properties | .................................. | 218 |
| 15.10.3 | Class Variables | .................................. | 218 |
| 15.11 | Class DirBuildInfo | .................................. | 218 |
| 15.11.1 | Methods | .................................. | 219 |
| 15.11.2 | Properties | .................................. | 219 |
| 15.11.3 | Class Variables | .................................. | 220 |
| 15.12 | Class Dir | ...................................... | 220 |
| 15.12.1 | Methods | .................................. | 220 |
| 15.12.2 | Properties | .................................. | 235 |
| 15.12.3 | Class Variables | .................................. | 235 |
| 15.13 | Class RootDir | ..................................... | 236 |
| 15.13.1 | Methods | .................................. | 236 |
| 15.13.2 | Properties | .................................. | 252 |
| 15.13.3 | Class Variables | .................................. | 252 |
| 15.14 | Class FileNodeInfo | .................................. | 252 |
| 15.14.1 | Methods | .................................. | 252 |
| 15.14.2 | Properties | .................................. | 254 |
| 15.14.3 | Class Variables | .................................. | 254 |
| 15.15 | Class FileBuildInfo | .................................. | 254 |
| 15.15.1 | Methods | .................................. | 254 |
| 15.15.2 | Properties | .................................. | 256 |
| 15.15.3 | Class Variables | .................................. | 256 |
| 15.16 | Class File | ...................................... | 256 |
| 15.16.1 | Methods | .................................. | 256 |
| 15.16.2 | Properties | .................................. | 270 |
| 15.16.3 | Class Variables | .................................. | 270 |
| 15.17 | Class FileFinder | ................................... | 271 |
| 15.17.1 | Methods | .................................. | 271 |
| 15.17.2 | Properties | .................................. | 272 |
| 15.17.3 | Class Variables | .................................. | 272 |
16 Module SCons.Node.Python
   16.1 Variables ........................................ 273
   16.2 Class ValueNodeInfo ................................ 273
      16.2.1 Methods ..................................... 273
      16.2.2 Properties .................................. 274
      16.2.3 Class Variables ............................... 274
   16.3 Class ValueBuildInfo ............................... 275
      16.3.1 Methods ..................................... 275
      16.3.2 Properties .................................. 276
      16.3.3 Class Variables ............................... 276
   16.4 Class Value ....................................... 276
      16.4.1 Methods ..................................... 276
      16.4.2 Properties .................................. 287
      16.4.3 Class Variables ............................... 287

17 Module SCons.PathList ................................ 288
   17.1 Functions ........................................ 288
   17.2 Variables ........................................ 288

18 Module SCons.SConf ................................... 289
   18.1 Functions ........................................ 289
   18.2 Variables ........................................ 290
   18.3 Class SConfWarning ............................... 291
      18.3.1 Methods ..................................... 291
      18.3.2 Properties .................................. 292
   18.4 Class SConfError ................................ 292
      18.4.1 Methods ..................................... 293
      18.4.2 Properties .................................. 294
   18.5 Class ConfigureDryRunError ...................... 294
      18.5.1 Methods ..................................... 294
      18.5.2 Properties .................................. 296
   18.6 Class ConfigureCacheError ....................... 296
      18.6.1 Methods ..................................... 296
      18.6.2 Properties .................................. 297
   18.7 Class SConfBuildInfo ............................. 298
      18.7.1 Methods ..................................... 298
      18.7.2 Properties .................................. 299
      18.7.3 Class Variables ............................... 299
   18.8 Class Streamer .................................... 300
      18.8.1 Methods ..................................... 300
      18.8.2 Properties .................................. 301
   18.9 Class SConfBuildTask ............................. 301
      18.9.1 Methods ..................................... 301
      18.9.2 Properties .................................. 306
   18.10 Class SConfBase .................................. 307
      18.10.1 Methods .................................... 307
      18.10.2 Properties ................................ 309
   18.11 Class CheckContext ............................... 310
      18.11.1 Methods .................................... 310
      18.11.2 Properties ................................ 312

19 Module SCons.SConsign ................................ 313
## 19.1 Functions ........................................ 313
## 19.2 Variables ......................................... 313
## 19.3 Class SConsignEntry ................................ 313
  19.3.1 Methods ....................................... 314
  19.3.2 Properties .................................... 314
  19.3.3 Class Variables ................................ 315
## 19.4 Class Base ......................................... 315
  19.4.1 Methods ....................................... 315
  19.4.2 Properties .................................... 316
## 19.5 Class DB ........................................... 316
  19.5.1 Methods ....................................... 317
  19.5.2 Properties .................................... 318
## 19.6 Class Dir ........................................... 318
  19.6.1 Methods ....................................... 318
  19.6.2 Properties .................................... 319
## 19.7 Class DirFile ....................................... 320
  19.7.1 Methods ....................................... 320
  19.7.2 Properties .................................... 321
## 19.8 Class DB ........................................... 322
  19.8.1 Methods ....................................... 322
  19.8.2 Properties .................................... 323

## 20 Package SCons.Scanner ................................... 324
  20.1 Modules ........................................... 324
  20.2 Functions .......................................... 324
  20.3 Variables .......................................... 324
  20.4 Class FindPathDirs .................................. 325
    20.4.1 Methods ...................................... 325
    20.4.2 Properties .................................... 326
  20.5 Class Base ......................................... 326
    20.5.1 Methods ...................................... 326
    20.5.2 Properties .................................... 328
  20.6 Class Selector ....................................... 329
    20.6.1 Methods ...................................... 331
    20.6.2 Properties .................................... 333
  20.7 Class Current ....................................... 333
    20.7.1 Methods ...................................... 335
    20.7.2 Properties .................................... 337
  20.8 Class Classic ........................................ 337
    20.8.1 Methods ...................................... 337
    20.8.2 Properties .................................... 339
  20.9 Class ClassicCPP .................................... 339
    20.9.1 Methods ...................................... 339
    20.9.2 Properties .................................... 341

## 21 Module SCons.Scanner.C ................................ 342
  21.1 Functions .......................................... 342
  21.2 Variables .......................................... 342
  21.3 Class SConsCPPScanner ............................... 342
    21.3.1 Methods ...................................... 342
    21.3.2 Properties .................................... 346
  21.4 Class SConsCPPScannerWrapper ......................... 347
21.4.1 Methods ........................................ 347
21.4.2 Properties ..................................... 348

22 Module SCons.Scanner.D 349
22.1 Functions ......................................... 349
22.2 Variables ........................................ 349
22.3 Class D ............................................ 349
   22.3.1 Methods ..................................... 349
   22.3.2 Properties .................................. 351

23 Module SCons.Scanner.Dir 352
23.1 Functions ......................................... 352
23.2 Variables ........................................ 352

24 Module SCons.Scanner.Fortran 353
24.1 Functions ......................................... 353
24.2 Variables ........................................ 353
24.3 Class F90Scanner .................................. 353
   24.3.1 Methods ..................................... 353
   24.3.2 Properties .................................. 355

25 Module SCons.Scanner.IDL 356
25.1 Functions ......................................... 356
25.2 Variables ........................................ 356

26 Module SCons.Scanner.LaTeX 357
26.1 Functions ......................................... 357
26.2 Variables ........................................ 357
26.3 Class FindENVPathDirs ............................. 357
   26.3.1 Methods ..................................... 357
   26.3.2 Properties .................................. 358
26.4 Class LaTeX ....................................... 358
   26.4.1 Methods ..................................... 361
   26.4.2 Properties .................................. 363
   26.4.3 Class Variables ............................... 363

27 Module SCons.Scanner.Prog 364
27.1 Functions ......................................... 364
27.2 Variables ........................................ 364

28 Module SCons.Scanner.RC 365
28.1 Functions ......................................... 365
28.2 Variables ........................................ 365

29 Package SCons.Script 366
29.1 Modules ........................................... 366
29.2 Functions ......................................... 366
29.3 Variables ........................................ 366
29.4 Class TargetList .................................. 370
   29.4.1 Methods ..................................... 370

30 Module SCons.Script.Interactive 373
30.1 Functions ......................................... 373
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>32.5.1</td>
<td>Methods</td>
<td>408</td>
</tr>
<tr>
<td>32.5.2</td>
<td>Properties</td>
<td>418</td>
</tr>
<tr>
<td>32.5.3</td>
<td>Class Variables</td>
<td>418</td>
</tr>
<tr>
<td>32.6</td>
<td>Class DefaultEnvironmentCall</td>
<td>418</td>
</tr>
<tr>
<td>32.6.1</td>
<td>Methods</td>
<td>418</td>
</tr>
<tr>
<td>32.6.2</td>
<td>Properties</td>
<td>419</td>
</tr>
<tr>
<td>33</td>
<td>Module SCons.Sig</td>
<td>420</td>
</tr>
<tr>
<td>33.1</td>
<td>Variables</td>
<td>420</td>
</tr>
<tr>
<td>33.2</td>
<td>Class MD5Null</td>
<td>420</td>
</tr>
<tr>
<td>33.2.1</td>
<td>Methods</td>
<td>420</td>
</tr>
<tr>
<td>33.2.2</td>
<td>Properties</td>
<td>421</td>
</tr>
<tr>
<td>33.3</td>
<td>Class TimeStampNull</td>
<td>422</td>
</tr>
<tr>
<td>33.3.1</td>
<td>Methods</td>
<td>422</td>
</tr>
<tr>
<td>33.3.2</td>
<td>Properties</td>
<td>423</td>
</tr>
<tr>
<td>34</td>
<td>Module SCons.Subst</td>
<td>424</td>
</tr>
<tr>
<td>34.1</td>
<td>Functions</td>
<td>424</td>
</tr>
<tr>
<td>34.2</td>
<td>Variables</td>
<td>425</td>
</tr>
<tr>
<td>34.3</td>
<td>Class Literal</td>
<td>425</td>
</tr>
<tr>
<td>34.3.1</td>
<td>Methods</td>
<td>426</td>
</tr>
<tr>
<td>34.3.2</td>
<td>Properties</td>
<td>427</td>
</tr>
<tr>
<td>34.4</td>
<td>Class SpecialAttrWrapper</td>
<td>427</td>
</tr>
<tr>
<td>34.4.1</td>
<td>Methods</td>
<td>427</td>
</tr>
<tr>
<td>34.4.2</td>
<td>Properties</td>
<td>428</td>
</tr>
<tr>
<td>34.5</td>
<td>Class CmdStringHolder</td>
<td>428</td>
</tr>
<tr>
<td>34.5.1</td>
<td>Methods</td>
<td>428</td>
</tr>
<tr>
<td>34.6</td>
<td>Class NLWrapper</td>
<td>431</td>
</tr>
<tr>
<td>34.6.1</td>
<td>Methods</td>
<td>432</td>
</tr>
<tr>
<td>34.6.2</td>
<td>Properties</td>
<td>432</td>
</tr>
<tr>
<td>34.7</td>
<td>Class Targets_or_Sources</td>
<td>433</td>
</tr>
<tr>
<td>34.7.1</td>
<td>Methods</td>
<td>433</td>
</tr>
<tr>
<td>34.8</td>
<td>Class Target_or_Source</td>
<td>434</td>
</tr>
<tr>
<td>34.8.1</td>
<td>Methods</td>
<td>434</td>
</tr>
<tr>
<td>34.8.2</td>
<td>Properties</td>
<td>435</td>
</tr>
<tr>
<td>34.9</td>
<td>Class NullNodeList</td>
<td>436</td>
</tr>
<tr>
<td>34.9.1</td>
<td>Methods</td>
<td>436</td>
</tr>
<tr>
<td>34.9.2</td>
<td>Properties</td>
<td>437</td>
</tr>
<tr>
<td>35</td>
<td>Module SCons.Taskmaster</td>
<td>438</td>
</tr>
<tr>
<td>35.1</td>
<td>Functions</td>
<td>438</td>
</tr>
<tr>
<td>35.2</td>
<td>Variables</td>
<td>438</td>
</tr>
<tr>
<td>35.3</td>
<td>Class Stats</td>
<td>439</td>
</tr>
<tr>
<td>35.3.1</td>
<td>Methods</td>
<td>439</td>
</tr>
<tr>
<td>35.3.2</td>
<td>Properties</td>
<td>440</td>
</tr>
<tr>
<td>35.4</td>
<td>Class Task</td>
<td>440</td>
</tr>
<tr>
<td>35.4.1</td>
<td>Methods</td>
<td>441</td>
</tr>
<tr>
<td>35.4.2</td>
<td>Properties</td>
<td>445</td>
</tr>
<tr>
<td>35.5</td>
<td>Class AlwaysTask</td>
<td>445</td>
</tr>
<tr>
<td>35.5.1</td>
<td>Methods</td>
<td>446</td>
</tr>
<tr>
<td>35.5.2</td>
<td>Properties</td>
<td>450</td>
</tr>
<tr>
<td>35.6</td>
<td>Class OutOfDateTask</td>
<td>450</td>
</tr>
<tr>
<td>Section</td>
<td>Details</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>35.6.1</td>
<td>Methods</td>
<td>450</td>
</tr>
<tr>
<td>35.6.2</td>
<td>Properties</td>
<td>455</td>
</tr>
<tr>
<td>35.7</td>
<td>Class Taskmaster</td>
<td>455</td>
</tr>
<tr>
<td>35.7.1</td>
<td>Methods</td>
<td>455</td>
</tr>
<tr>
<td>35.7.2</td>
<td>Properties</td>
<td>457</td>
</tr>
<tr>
<td>36.1</td>
<td>Functions</td>
<td>459</td>
</tr>
<tr>
<td>36.2</td>
<td>Variables</td>
<td>464</td>
</tr>
<tr>
<td>36.3</td>
<td>Class NodeList</td>
<td>465</td>
</tr>
<tr>
<td>36.3.1</td>
<td>Methods</td>
<td>465</td>
</tr>
<tr>
<td>36.4</td>
<td>Class DisplayEngine</td>
<td>467</td>
</tr>
<tr>
<td>36.4.1</td>
<td>Methods</td>
<td>467</td>
</tr>
<tr>
<td>36.4.2</td>
<td>Properties</td>
<td>468</td>
</tr>
<tr>
<td>36.4.3</td>
<td>Class Variables</td>
<td>468</td>
</tr>
<tr>
<td>36.5</td>
<td>Class Proxy</td>
<td>468</td>
</tr>
<tr>
<td>36.5.1</td>
<td>Methods</td>
<td>469</td>
</tr>
<tr>
<td>36.5.2</td>
<td>Properties</td>
<td>470</td>
</tr>
<tr>
<td>36.6</td>
<td>Class Delegate</td>
<td>470</td>
</tr>
<tr>
<td>36.6.1</td>
<td>Methods</td>
<td>470</td>
</tr>
<tr>
<td>36.6.2</td>
<td>Properties</td>
<td>471</td>
</tr>
<tr>
<td>36.7</td>
<td>Class _NoError</td>
<td>471</td>
</tr>
<tr>
<td>36.7.1</td>
<td>Methods</td>
<td>471</td>
</tr>
<tr>
<td>36.7.2</td>
<td>Properties</td>
<td>473</td>
</tr>
<tr>
<td>36.8</td>
<td>Class WindowsError</td>
<td>473</td>
</tr>
<tr>
<td>36.8.1</td>
<td>Methods</td>
<td>473</td>
</tr>
<tr>
<td>36.8.2</td>
<td>Properties</td>
<td>474</td>
</tr>
<tr>
<td>36.9</td>
<td>Class CLVar</td>
<td>475</td>
</tr>
<tr>
<td>36.9.1</td>
<td>Methods</td>
<td>475</td>
</tr>
<tr>
<td>36.10</td>
<td>Class OrderedDict</td>
<td>477</td>
</tr>
<tr>
<td>36.10.1</td>
<td>Methods</td>
<td>477</td>
</tr>
<tr>
<td>36.11</td>
<td>Class Selector</td>
<td>478</td>
</tr>
<tr>
<td>36.11.1</td>
<td>Methods</td>
<td>478</td>
</tr>
<tr>
<td>36.12</td>
<td>Class LogicalLines</td>
<td>480</td>
</tr>
<tr>
<td>36.12.1</td>
<td>Methods</td>
<td>480</td>
</tr>
<tr>
<td>36.12.2</td>
<td>Properties</td>
<td>481</td>
</tr>
<tr>
<td>36.13</td>
<td>Class UniqueList</td>
<td>481</td>
</tr>
<tr>
<td>36.13.1</td>
<td>Methods</td>
<td>481</td>
</tr>
<tr>
<td>36.14</td>
<td>Class Unbuffered</td>
<td>483</td>
</tr>
<tr>
<td>36.14.1</td>
<td>Methods</td>
<td>483</td>
</tr>
<tr>
<td>36.14.2</td>
<td>Properties</td>
<td>484</td>
</tr>
<tr>
<td>36.15</td>
<td>Class Null</td>
<td>484</td>
</tr>
<tr>
<td>36.15.1</td>
<td>Methods</td>
<td>484</td>
</tr>
<tr>
<td>36.15.2</td>
<td>Properties</td>
<td>485</td>
</tr>
<tr>
<td>36.16</td>
<td>Class NullSeq</td>
<td>486</td>
</tr>
<tr>
<td>36.16.1</td>
<td>Methods</td>
<td>486</td>
</tr>
<tr>
<td>36.16.2</td>
<td>Properties</td>
<td>487</td>
</tr>
</tbody>
</table>

37 Package SCons.Variables

<table>
<thead>
<tr>
<th>Section</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>37.1</td>
<td>Modules</td>
</tr>
<tr>
<td>37.2</td>
<td>Variables</td>
</tr>
<tr>
<td>37.3</td>
<td>Class Variables</td>
</tr>
<tr>
<td>43.13</td>
<td>Class NoMD5ModuleWarning</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>43.13.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.13.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.14</td>
<td>Class NoMetaclassSupportWarning</td>
</tr>
<tr>
<td>43.14.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.14.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.15</td>
<td>Class NoObjectCountWarning</td>
</tr>
<tr>
<td>43.15.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.15.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.16</td>
<td>Class NoParallelSupportWarning</td>
</tr>
<tr>
<td>43.16.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.16.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.17</td>
<td>Class ReservedVariableWarning</td>
</tr>
<tr>
<td>43.17.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.17.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.18</td>
<td>Class StackSizeWarning</td>
</tr>
<tr>
<td>43.18.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.18.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.19</td>
<td>Class VisualCMissingWarning</td>
</tr>
<tr>
<td>43.19.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.19.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.20</td>
<td>Class VisualVersionMismatch</td>
</tr>
<tr>
<td>43.20.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.20.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.21</td>
<td>Class VisualStudioMissingWarning</td>
</tr>
<tr>
<td>43.21.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.21.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.22</td>
<td>Class FortranCxxMixWarning</td>
</tr>
<tr>
<td>43.22.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.22.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.23</td>
<td>Class FutureDeprecatedWarning</td>
</tr>
<tr>
<td>43.23.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.23.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.24</td>
<td>Class DeprecatedWarning</td>
</tr>
<tr>
<td>43.24.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.24.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.25</td>
<td>Class MandatoryDeprecatedWarning</td>
</tr>
<tr>
<td>43.25.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.25.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.26</td>
<td>Class PythonVersionWarning</td>
</tr>
<tr>
<td>43.26.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.26.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.27</td>
<td>Class DeprecatedSourceCodeWarning</td>
</tr>
<tr>
<td>43.27.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.27.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.28</td>
<td>Class DeprecatedBuildDirWarning</td>
</tr>
<tr>
<td>43.28.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.28.2</td>
<td>Properties</td>
</tr>
<tr>
<td>43.29</td>
<td>Class TaskmasterNeedsExecuteWarning</td>
</tr>
<tr>
<td>43.29.1</td>
<td>Methods</td>
</tr>
<tr>
<td>43.29.2</td>
<td>Properties</td>
</tr>
</tbody>
</table>
### 43.30 Class DeprecatedCopyWarning
- Methods
- Properties

### 43.31 Class DeprecatedOptionsWarning
- Methods
- Properties

### 43.32 Class DeprecatedSourceSignaturesWarning
- Methods
- Properties

### 43.33 Class DeprecatedTargetSignaturesWarning
- Methods
- Properties

### 43.34 Class DeprecatedDebugOptionsWarning
- Methods
- Properties

### 43.35 Class DeprecatedSigModuleWarning
- Methods
- Properties

### 43.36 Class DeprecatedBuilderKeywordsWarning
- Methods
- Properties

### 44 Package SCons.compat
- Modules
- Functions
- Variables

### 45 Module SCons.compat._scons_builtins
- Functions
- Variables
- Class memoryview
- Methods
- Properties

### 46 Module SCons.compat._scons_collections
- Variables

### 47 Module SCons.compat._scons_dbm
- Functions
- Variables
- Class error
- Methods
- Properties

### 48 Module SCons.compat._scons_hashlib
- Functions
- Variables
- Class md5obj
- Methods
- Properties
- Class Variables

---

15
48.4.1 Methods .................................................. 578
48.4.2 Properties ............................................ 579
48.4.3 Class Variables ....................................... 579

49 Module SCons.compat._scons_io 580
49.1 Variables ............................................... 580

50 Module SCons.compat._scons_sets 581
50.1 Class BaseSet ........................................... 581
50.1.1 Methods ............................................. 582
50.1.2 Properties .......................................... 585
50.2 Class ImmutableSet ..................................... 585
50.2.1 Methods ............................................. 585
50.2.2 Properties .......................................... 588
50.3 Class Set ................................................ 588
50.3.1 Methods ............................................. 589
50.3.2 Properties .......................................... 593

51 Module SCons.compat._scons_subprocess 594
51.1 Functions ............................................... 600
51.2 Variables ............................................... 601
51.3 Class CalledProcessError ......................... 601
51.3.1 Methods ............................................. 601
51.3.2 Properties .......................................... 602
51.4 Class Popen .............................................. 603
51.4.1 Methods ............................................. 603
51.4.2 Properties .......................................... 604

52 Module SCons.cpp 605
52.1 Functions ............................................... 605
52.2 Variables ............................................... 605
52.3 Class FunctionEvaluator ......................... 605
52.3.1 Methods ............................................. 606
52.3.2 Properties .......................................... 607
52.4 Class PreProcessor ................................. 607
52.4.1 Methods ............................................. 607
52.4.2 Properties .......................................... 611
52.5 Class DumbPreProcessor ......................... 611
52.5.1 Methods ............................................. 611
52.5.2 Properties .......................................... 615

53 Module SCons.dblite 616
53.1 Functions ............................................... 616
53.2 Variables ............................................... 616
53.3 Class dblite ............................................ 616
53.3.1 Methods ............................................. 616
53.3.2 Properties .......................................... 617

54 Module SCons.exitfuncs 618
54.1 Functions ............................................... 618
54.2 Variables ............................................... 618

55 Module md5 619
55.1 Variables ................................................................. 619
1 Package SCons

SCons

The main package for the SCons software construction utility.

Version: 2.0.0.final.0
Date: 2010/06/14 22:05:46

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. 25)*
- **CacheDir**: CacheDir support
  *(Section 4, p. 41)*
- **Conftest**: SCons.Conftest
  Autoconf-like configuration support; low level implementation of tests. *(Section 5, p. 44)*
- **Debug**: SCons.Debug
  Code for debugging SCons internal things. *(Section 6, p. 48)*
- **Defaults**: SCons.Defaults
  Builders and other things for the local site. *(Section 7, p. 50)*
- **Environment**: SCons.Environment
  Base class for construction Environments. *(Section 8, p. 55)*
- **Errors**: SCons.Errors
  This file contains the exception classes used to handle internal and user errors in SCons. *(Section 9, p. 96)*
- **Executor**: SCons.Executor
  A module for executing actions with specific lists of target and source Nodes. *(Section 10, p. 110)*
- **Job**: SCons.Job
  This module defines the Serial and Parallel classes that execute tasks to complete a build. *(Section 11, p. 121)*
- **Memoize**: Memoizer
  A metaclass implementation to count hits and misses of the computed values that various methods cache in memory. *(Section 12, p. 131)*
- **Node**: SCons.Node
  The Node package for the SCons software construction utility. *(Section 13, p. 141)*
  - **Alias**: scons.Node.Alias
    Alias nodes. *(Section 14, p. 160)*
- **FS**: scons.Node.FS
  File system nodes.
  *(Section 15, p. 177)*

- **Python**: scons.Node.Python
  Python nodes.
  *(Section 16, p. 273)*

- **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. 288)*

- **SConf**: SCons.SConf
  Autoconf-like configuration support.
  *(Section 18, p. 289)*

- **SConsign**: SCons.SConsign
  Writing and reading information to the .sconsign file or files.
  *(Section 19, p. 313)*

- **Scanner**: SCons.Scanner
  The Scanner package for the SCons software construction utility.
  *(Section 20, p. 324)*
  - **C**: SCons.Scanner.C
    This module implements the dependency scanner for C/C++ code.
    *(Section 21, p. 342)*
  - **D**: SCons.Scanner.D
    Scanner for the Digital Mars "D" programming language.
    *(Section 22, p. 349)*
  - **Dir** *(Section 23, p. 352)*
  - **Fortran**: SCons.Scanner.Fortran
    This module implements the dependency scanner for Fortran code.
    *(Section 24, p. 353)*
  - **IDL**: SCons.Scanner.IDL
    This module implements the dependency scanner for IDL (Interface Definition Language) files.
    *(Section 25, p. 356)*
  - **LaTeX**: SCons.Scanner.LaTeX
    This module implements the dependency scanner for LaTeX code.
    *(Section 26, p. 357)*
  - **Prog** *(Section 27, p. 364)*
  - **RC**: SCons.Scanner.RC
    This module implements the dependency scanner for RC (Interface Definition Language) files.
    *(Section 28, p. 365)*

- **Script**: SCons.Script
  This file implements the main() function used by the scons script.
  *(Section 29, p. 366)*
  - **Interactive**: SCons interactive mode
    *(Section 30, p. 373)*
  - **Main**: SCons.Script
    This file implements the main() function used by the scons script.
    *(Section 31, p. 377)*
  - **SConscript**: SCons.Script.SConscript
    This module defines the Python API provided to SConscript and SConstruct files.
    *(Section 32, p. 404)*
• **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. 420)*

• **Subst**: SCons.Subst
  SCons string substitution.
  *(Section 34, p. 424)*

• **Taskmaster**: Generic Taskmaster module for the SCons build engine.
  *(Section 35, p. 438)*

• **Util**: SCons.Util
  Various utility functions go here.
  *(Section 36, p. 459)*

• **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. 488)*
  - **BoolVariable**: engine.SCons.Variables.BoolVariable
    This file defines the option type for SCons implementing true/false values.
    *(Section 38, p. 492)*
  - **EnumVariable**: engine.SCons.Variables.EnumVariable
    This file defines the option type for SCons allowing only specified input-values.
    *(Section 39, p. 493)*
  - **ListVariable**: engine.SCons.Variables.ListVariable
    This file defines the option type for SCons implementing ‘lists’.
    *(Section 40, p. 494)*
  - **PackageVariable**: engine.SCons.Variables.PackageVariable
    This file defines the option type for SCons implementing ‘package activation’.
    *(Section 41, p. 495)*
  - **PathVariable**: SCons.Variables.PathVariable
    This file defines an option type for SCons implementing path settings.
    *(Section 42, p. 496)*

• **Warnings**: SCons.Warnings
  This file implements the warnings framework for SCons.
  *(Section 43, p. 498)*

• **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. 567)*
  - **scons_builtins**: Compatibility idioms for builtins names
    This module adds names to the builtins module for things that we want to use in SCons but which don’t show up until later Python versions than the earliest ones we support.
    *(Section 45, p. 569)*
  - **scons_collections**: collections compatibility module for older (pre-2.4) Python versions
    This does not not NOT (repeat, *NOT*) provide complete collections functionality.
    *(Section 46, p. 572)*
  - **scons_dbm**: dbm compatibility module for Python versions that don’t have dbm.
    *(Section 47, p. 573)*
  - **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 48, p. 576)*
- **scons_io**: Io compatibility module for older (pre-2.6) Python versions
  This does not NOT (repeat, *NOT*) provide complete io functionality.
  *(Section 49, p. 580)*
- **scons_sets**: Classes to represent arbitrary sets (including sets of sets).
  *(Section 50, p. 581)*
- **scons_subprocess**: Subprocess - Subprocesses with accessible I/O streams
  This module allows you to spawn processes, connect to their input/output/error pipes, and obtain their return codes.
  *(Section 51, p. 594)*
- **cpp**: SCons C Pre-Processor module
  *(Section 52, p. 605)*
- **dblite** *(Section 53, p. 616)*
- **exitfuncs**: SCons.exitfuncs
  Register functions which are executed when SCons exits for any reason.
  *(Section 54, p. 618)*

### 1.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__build__</code></td>
<td>Value: 'r5023’</td>
</tr>
<tr>
<td><code>__buildds</code></td>
<td>Value: 'scons-dev’</td>
</tr>
<tr>
<td><code>__developer__</code></td>
<td>Value: 'scons’</td>
</tr>
<tr>
<td><code>__revision__</code></td>
<td>Value: 'src/engine/SCons/<strong>init</strong>.py 5023 2010/06/14 22:05:46 sc...</td>
</tr>
</tbody>
</table>
Module SCons.Action

SCons.Action

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

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

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

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

- CommandAction
- CommandGeneratorAction
- FunctionAction
- ListAction

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

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

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

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

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

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

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

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

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

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

2.1 Functions

rfile(n)
default_exitstatfunc(s)
remove_set_lineno_codes(x)

Action(act, *args, **kw)
A factory for action objects.

get_default_ENV(env)

2.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>revision</em></td>
<td>Value: 'src/engine/SCons/Action.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
<tr>
<td>print_actions</td>
<td>Value: 1</td>
</tr>
<tr>
<td>execute_actions</td>
<td>Value: 1</td>
</tr>
</tbody>
</table>

continued on next page
### Class ActionBase

**object**

```
SCons.Action.ActionBase
```


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

#### 2.3.1 Methods

- `_cmp_(self, other)`
- `no_batch_key(self, env, target, source)`
- `batch_key(self, env, target, source)`
- `genstring(self, target, source, env)`
- `get_contents(self, target, source, env)`
- `_add_(self, other)`
- `_radd_(self, other)`
- `presub_lines(self, env)`
- `get_varlist(self, target, source, env, executor=False)`
- `get_targets(self, env, executor)`

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

- `_delattr_()`
  ```
x._delattr_('name') == del x.name
  ```

24
```python
__getattribute__(...)  
x.__getattribute__("name") == x.name

__hash__(x)  
hash(x)

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

__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)

2.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
2.4 Class CommandAction

object

SCons.Action.ActionBase

SCons.Action._ActionAction

SCons.Action.CommandAction

Known Subclasses: SCons.Action.LazyAction

Class for command-execution actions.

2.4.1 Methods

__init__(self, cmd, **kw)
Overrides: SCons.Action._ActionAction.__init__

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

process(self, target, source, env, executor=False)

strfunction(self, target, source, env, executor=False)

execute(self, target, source, env, executor=False)

Execute a command action.

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

get_presig(self, target, source, env, executor=False)

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

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

get_implicit_deps(self, target, source, env, executor=False)

__add__(self, other)
**Class CommandAction**

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmp_</code></td>
<td><code>(self, other)</code></td>
<td>Compare the command action with another object</td>
</tr>
<tr>
<td><code>_delattr_</code></td>
<td><code>(...)</code></td>
<td>Delete an attribute from the command action object</td>
</tr>
<tr>
<td><code>_getattribute_</code></td>
<td><code>(...)</code></td>
<td>Get an attribute from the command action object</td>
</tr>
<tr>
<td><code>_hash_</code></td>
<td><code>(x)</code></td>
<td>Return the hash of the command action object</td>
</tr>
<tr>
<td><code>_new_</code></td>
<td><code>(T, S, ...)</code></td>
<td>Return a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>_radd_</code></td>
<td><code>(self, other)</code></td>
<td>Add another object to the command action object</td>
</tr>
<tr>
<td><code>_reduce_</code></td>
<td><code>(...)</code></td>
<td>Helper for pickle serialization</td>
</tr>
<tr>
<td><code>_reduce_ex_</code></td>
<td><code>(...)</code></td>
<td>Helper for pickle serialization</td>
</tr>
<tr>
<td><code>_repr_</code></td>
<td><code>(x)</code></td>
<td>Return the string representation of the command action object</td>
</tr>
<tr>
<td><code>_setattr_</code></td>
<td><code>(...)</code></td>
<td>Set an attribute value in the command action object</td>
</tr>
<tr>
<td><code>batch_key</code></td>
<td><code>(self, env, target, source)</code></td>
<td>Batch key method</td>
</tr>
<tr>
<td><code>genstring</code></td>
<td><code>(self, target, source, env)</code></td>
<td>Generate a string representation of the command action object</td>
</tr>
<tr>
<td><code>get_contents</code></td>
<td><code>(self, target, source, env)</code></td>
<td>Get contents method</td>
</tr>
</tbody>
</table>
get_targets(self, env, executor)

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

get_varlist(self, target, source, env, executor=False)

no_batch_key(self, env, target, source)

presub_lines(self, env)

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

2.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

2.5 Class CommandGeneratorAction

object

SCons.Action.ActionBase

SCons.Action.CommandGeneratorAction

Known Subclasses: SCons.Action.LazyAction

Class for command-generator actions.

2.5.1 Methods

__init__(self, generator, kw)

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

__str__(self)

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

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

genstring(self, target, source, env, executor=False)
Class `CommandGeneratorAction Module SCons.Action`

```python
```

#### get_presig(self, target, source, env, executor=False)

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

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

#### get_implicit_deps(self, target, source, env, executor=False)

```python
get_varlist(self, target, source, env, executor=False)
```


#### get_targets(self, env, executor=True)

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

Overrides: SCons.Action.ActionBase.get_targets

#### add__(self, other)

#### cmp__(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
```

#### radd__(self, other)

#### reduce__(...)

helper for pickle
Class LazyAction

Module SCons.Action

```python
__reduce_ex__(...)  
helper for pickle
```

```python
__repr__(x)  
repr(x)
```

```python
__setattr__(...)  
x.__setattr__(‘name’, value) ==> x.name = value
```

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

```python
no_batch_key(self, env, target, source)
```

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

### 2.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

### 2.6 Class LazyAction

```
object
  | SCons.Action.ActionBase
  | SCons.Action.CommandGeneratorAction

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

SCons.Action.LazyAction
```

### 2.6.1 Methods

```python
__init__(self, var, kw)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
```

Overrides: SCons.Action.CommandGeneratorAction.__init__

30
get_parent_class(self, env)

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

get_presig(self, target, source, env)
Return the signature contents of this action’s command line.
This strips $(-$) and everything in between the string,
since those parts don’t affect signatures.
Overrides: SCons.Action.CommandGeneratorAction.get_presig

get_varlist(self, target, source, env, executor=False)

__add__(self, other)

__cmp__(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

__radd__(self, other)

__reduce__(...)
helper for pickle

__reduce_ex__(...)
helper for pickle

__repr__(x)
repr(x)
Class LazyAction

```python
setattr(x, '_setattr_(...)

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

```python
_str__(self)
str(x)
```

Overrides: object._str_ (inherited documentation)

```python
batch_key(self, env, target, source)
```

Overrides: SCons.Action.ActionBase.batch_key

```python
execute(self, target, source, env, executor=False)
```

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.

```python
genstring(self, target, source, env, executor=False)
```


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

```python
get_implicit_deps(self, target, source, env, executor=False)
```

```python
get_targets(self, env, executor)
```

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

Overrides: SCons.Action.ActionBase.get_targets (inherited documentation)

```python
no_batch_key(self, env, target, source)
```

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

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

```python
process(self, target, source, env, executor=False)
```

```python
strfunction(self, target, source, env, executor=False)
```

2.6.2 Properties
2.7 Class FunctionAction

Object  ▼
SCons.Action.ActionBase  ▼
SCons.Action._ActionAction  ▼
SCons.Action.FunctionAction

Class for Python function actions.

2.7.1 Methods

```python
__init__(self, execfunction, kw)
Overrides: SCons.Action._ActionAction.__init__

function_name(self)

strfunction(self, target, source, env, executor=False)

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

execute(self, target, source, env, executor=False)

get_presig(self, target, source, env)
Return the signature contents of this callable action.

get_implicit_deps(self, target, source, env)

__add__(self, other)


cmp__(self, other)

__delattr__(self, other)

x.__delattr__('name') == del x.name
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__getattribute__(...)</code></td>
<td><code>x.__getattribute__('name') ==&gt; x.name</code></td>
</tr>
<tr>
<td><code>__hash__()</code></td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><code>__new__(T, S, ...)</code></td>
<td><strong>Return Value</strong>&lt;br&gt;a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>__radd__(self, other)</code></td>
<td></td>
</tr>
<tr>
<td><code>__reduce__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__repr__(x)</code></td>
<td><code>repr(x)</code></td>
</tr>
<tr>
<td><code>__setattr__(...)</code></td>
<td><code>x.__setattr__('name', value) ==&gt; x.name = value</code></td>
</tr>
<tr>
<td><code>batch_key(self, env, target, source)</code></td>
<td></td>
</tr>
<tr>
<td><code>genstring(self, target, source, env)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_contents(self, target, source, env)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_targets(self, env, executor)</code></td>
<td>Returns the type of targets ($TARGETS, $CHANGED_TARGETS) used by this action.</td>
</tr>
<tr>
<td><code>get_varlist(self, target, source, env, executor=False)</code></td>
<td></td>
</tr>
<tr>
<td><code>no_batch_key(self, env, target, source)</code></td>
<td></td>
</tr>
<tr>
<td><code>presub_lines(self, env)</code></td>
<td></td>
</tr>
</tbody>
</table>
print_cmd_line(self, s, target, source, env)

2.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute 'class_' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

2.8 Class ListAction

object

SCons.Action.ActionBase

SCons.Action.ListAction

Class for lists of other actions.

2.8.1 Methods

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

genstring(self, target, source, env)

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

presub_lines(self, env)
Overrides: SCons.Action.ActionBase.psub_lines

get_presig(self, target, source, env)
Return the signature contents of this action list.
Simple concatenation of the signatures of the elements.


get_implicit_deps(self, target, source, env)

get_varlist(self, target, source, env, executor=False)
### Class ListAction

**Module SCons.Action**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__add__</code></td>
<td>Add two actions</td>
</tr>
<tr>
<td><code>__cmp__</code></td>
<td>Compare two actions</td>
</tr>
<tr>
<td><code>__delattr__</code></td>
<td>Remove an attribute from an object</td>
</tr>
<tr>
<td><code>x.__delattr__('name')</code></td>
<td>Delete attribute <code>name</code> from <code>x</code>.</td>
</tr>
<tr>
<td><code>__getattr__</code></td>
<td>Get an attribute from an object</td>
</tr>
<tr>
<td><code>x.__getattr__('name')</code></td>
<td>Get attribute <code>name</code> from <code>x</code>.</td>
</tr>
<tr>
<td><code>__hash__</code></td>
<td>Get the hash of the object</td>
</tr>
<tr>
<td><code>hash(x)</code></td>
<td>Compute the hash of <code>x</code>.</td>
</tr>
<tr>
<td><code>__new__</code></td>
<td>Create a new object</td>
</tr>
<tr>
<td><code>T.__new__(S)</code></td>
<td>Return a new object with type <code>S</code>, a subtype of <code>T</code></td>
</tr>
<tr>
<td><code>__radd__</code></td>
<td>Add a number to an object</td>
</tr>
<tr>
<td><code>__reduce__</code></td>
<td>Reduce the object for pickle serialization</td>
</tr>
<tr>
<td><code>__reduce_ex__</code></td>
<td>Reduce the object for pickle serialization</td>
</tr>
<tr>
<td><code>__repr__</code></td>
<td>Represent the object as a string</td>
</tr>
<tr>
<td><code>repr(x)</code></td>
<td>Represent <code>x</code> as a string</td>
</tr>
<tr>
<td><code>__setattr__</code></td>
<td>Set an attribute of an object</td>
</tr>
<tr>
<td><code>x.__setattr__('name', value)</code></td>
<td>Set attribute <code>name</code> to <code>value</code> in <code>x</code>.</td>
</tr>
<tr>
<td><code>batch_key</code></td>
<td>Batch key function</td>
</tr>
<tr>
<td><code>get_contents</code></td>
<td>Get contents from a target</td>
</tr>
<tr>
<td><code>get_targets</code></td>
<td>Get targets from an environment and executor</td>
</tr>
<tr>
<td><code>get_targets</code></td>
<td>Get targets from an environment and executor</td>
</tr>
</tbody>
</table>

Returns the type of targets ($TARGETS, $CHANGED_TARGETS) used by this action.
no_batch_key(self, env, target, source)

2.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

2.9 Class ActionCaller

object —

\[\text{SCons.Action.ActionCaller}\]

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

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

2.9.1 Methods

__init__(self, parent, args, kw)

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

Overrides: object.__init__ (inherited documentation)

get_contents(self, target, source, env)

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

subst_args(self, target, source, env)

subst_kw(self, target, source, env)

__call__(self, target, source, env, executor=False)

strfunction(self, target, source, env)

__str__(self)

str(x)

Overrides: object.__str__ (inherited documentation)

__delattr__(...)

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

```python
__init__(self, actfunc, strfunc, convert=<function <lambda> at 0x843cd14>)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)
```

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

```python
__delattr__(...)
x.__delattr__(‘name’) ==> del x.name
```

```python
__getattribute__(...)
x.__getattribute__(‘name’) ==> x.name
```

```python
__hash__(x)
hash(x)
```

```python
__new__(T, S, ...)
Return Value
    a new object with type S, a subtype of T
```

```python
__reduce__(...)
helper for pickle
```

```python
__reduce_ex__(...)
helper for pickle
```

```python
__repr__(x)
repr(x)
```

```python
__setattr__(...)
x.__setattr__(‘name’, value) ==> x.name = value
```

```python
__str__(x)
str(x)
```

2.10.2 Properties

*continued on next page*
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.class_</td>
<td>Value: &lt;attribute '.<em>class</em>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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:

```python
_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.

```python
get_name()
```

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

```python
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

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

`Builder(**kw)`

A factory for builder objects.

```python
is_a_Builder(obj)
```

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

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

### 3.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>revision</td>
<td>Value: 'src/engine/SCons/Builder.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
<tr>
<td>misleading_keywords</td>
<td>Value: {'sources': 'source', 'targets': 'target'}</td>
</tr>
</tbody>
</table>
3.3 Class 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=1)
  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
### 3.4 Class CallableSelector

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

```python
copy(self)
Overrides: UserDict.UserDict.copy

fromkeys(cls, iterable, value=False)

get(self, key, failobj=False)

has_key(self, key)

items(self)
Overrides: UserDict.UserDict.items

iteritems(self)

iterkeys(self)

itervalues(self)

keys(self)
Overrides: UserDict.UserDict.keys

pop(self, key, *args)

popitem(self)
Overrides: UserDict.UserDict.popitem

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

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

values(self)
Overrides: UserDict.UserDict.values
```
### 3.4.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Override</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__</code></td>
<td><code>(self, env, source)</code></td>
<td>SCons.Util.Selector.<strong>call</strong></td>
</tr>
<tr>
<td><code>__cmp__</code></td>
<td><code>(self, dict)</code></td>
<td></td>
</tr>
<tr>
<td><code>__contains__</code></td>
<td><code>(self, key)</code></td>
<td></td>
</tr>
<tr>
<td><code>__delitem__</code></td>
<td><code>(self, key)</code></td>
<td>UserDict.UserDict.<strong>delitem</strong></td>
</tr>
<tr>
<td><code>__getitem__</code></td>
<td><code>(self, key)</code></td>
<td></td>
</tr>
<tr>
<td><code>__init__</code></td>
<td><code>(self, dict=False)</code></td>
<td>UserDict.UserDict.<strong>init</strong></td>
</tr>
<tr>
<td><code>__len__</code></td>
<td><code>(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>__repr__</code></td>
<td><code>(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>__setitem__</code></td>
<td><code>(self, key, item)</code></td>
<td>UserDict.UserDict.<strong>setitem</strong></td>
</tr>
<tr>
<td><code>clear</code></td>
<td><code>(self)</code></td>
<td>UserDict.UserDict.clear</td>
</tr>
<tr>
<td><code>copy</code></td>
<td><code>(self)</code></td>
<td>UserDict.UserDict.copy</td>
</tr>
<tr>
<td><code>fromkeys</code></td>
<td><code>(cls, iterable, value=False)</code></td>
<td></td>
</tr>
<tr>
<td><code>get</code></td>
<td><code>(self, key, failobj=False)</code></td>
<td></td>
</tr>
<tr>
<td><code>has_key</code></td>
<td><code>(self, key)</code></td>
<td></td>
</tr>
<tr>
<td><code>items</code></td>
<td><code>(self)</code></td>
<td>UserDict.UserDict.items</td>
</tr>
<tr>
<td><code>iteritems</code></td>
<td><code>(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>iterkeys</code></td>
<td><code>(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>itervalues</code></td>
<td><code>(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>keys</code></td>
<td><code>(self)</code></td>
<td>UserDict.UserDict.keys</td>
</tr>
</tbody>
</table>
Class DictEmitter

UserDict.UserDict

SCons.Util.OrderedDict

SCons.Util.Selector

SCons.Builder.DictEmitter

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

3.5.1 Methods

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

__cmp__(self, dict)

__contains__(self, key)

__delitem__(self, key)
Overrrides: UserDict.UserDict.__delitem__

__getitem__(self, key)

__init__(self, dict=False)
Overrrides: UserDict.UserDict.__init__

__len__(self)
3.6 Class ListEmitter

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

3.6.1 Methods

```python
__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)
```
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

________________________
<p>| <em>init</em>  | (self, dict) |
| Overrides: UserDict.UserDict.<em>init</em> |
| warn     | (self)       |
| <strong>cmp</strong>  | (self, dict) |
| <strong>contains</strong> | (self, key) |</p>
<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>delitem</strong>(self, key)</td>
</tr>
<tr>
<td><strong>getitem</strong>(self, key)</td>
</tr>
<tr>
<td><strong>len</strong>(self)</td>
</tr>
<tr>
<td><strong>repr</strong>(self)</td>
</tr>
<tr>
<td><strong>setitem</strong>(self, key, item)</td>
</tr>
<tr>
<td>clear(self)</td>
</tr>
<tr>
<td>copy(self)</td>
</tr>
<tr>
<td>fromkeys(cls, iterable, value=False)</td>
</tr>
<tr>
<td>get(self, key, failobj=False)</td>
</tr>
<tr>
<td>has_key(self, key)</td>
</tr>
<tr>
<td>items(self)</td>
</tr>
<tr>
<td>iteritems(self)</td>
</tr>
<tr>
<td>iterkeys(self)</td>
</tr>
<tr>
<td>itervalues(self)</td>
</tr>
<tr>
<td>keys(self)</td>
</tr>
<tr>
<td>pop(self, key, *args)</td>
</tr>
<tr>
<td>popitem(self)</td>
</tr>
<tr>
<td>setdefault(self, key, failobj=False)</td>
</tr>
<tr>
<td>update(self, dict=False, **kwargs)</td>
</tr>
<tr>
<td>values(self)</td>
</tr>
</tbody>
</table>

### 3.8 Class EmitterProxy

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

3.8.1 Methods

```python
__init__(self, var)
x.__init__(...) initializes x; see x.__class__.doc for signature
Overrides: object.__init__(inherited documentation)
```

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

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

```python
__delattr__(...)
x.__delattr__('name') == del x.name
```

```python
__getattr__(...)
x.__getattr__('name') == x.name
```

```python
__hash__(x)
hash(x)
```

```python
__new__(T, S, ...)
Return Value
   a new object with type S, a subtype of T
```

```python
__reduce__(...)
helper for pickle
```

```python
__reduce_ex__(...)
helper for pickle
```

```python
__repr__(x)
repr(x)
```

```python
__setattr__(...)
x.__setattr__('name', value) == x.name = value
```
3.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

3.9 Class BuilderBase

object

SCons.Builder.BuilderBase

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

3.9.1 Methods

```python
__init__(self, action=..., prefix='', suffix='', src_suffix='', target_factory=False,
source_factory=False, target_scaner=False, source_scanner=False, emitter=False, multi=0,
env=..., single_source=0, name=..., chdir=<class 'SCons.Builder.Null'>, is_explicit=1,
src_builder=False, ensure_suffix=False, **overrides)
```

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

Overrides: `object.__init__(inherited documentation)`

```python
__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).

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

splitext(self, path, env=False)

```python
__call__(self, env, target=..., source=..., chdir=<class 'SCons.Builder.Null'>, **kw)
```

adjust_suffix(self, suff)

get_prefix(self, env, sources=[])
```python
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_srcBuilders(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.)

```python
_delattr__(...)  
ex._delattr__('name') ===> del x.name
```

```python
_getattribute__(...)  
ex._getattribute__('name') ===> x.name
```

```python
_hash__(x)  
hash(x)
```

```python
_new__(T, S, ...)  
Return Value  
a new object with type S, a subtype of T
```

```python
_reduce__(...)  
helper for pickle
```

```python
_reduce_ex__(...)  
helper for pickle
```

```python
_repr__(x)  
repr(x)
```

```python
_setattr__(...)  
ex._setattr__('name', value) ===> x.name = value
```

```python
_str__(x)  
str(x)
```

### 3.9.2 Properties
3.9.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

3.10 Class CompositeBuilder

object

```
```

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

```python
__init__(self, builder, cmdgen)
```
Wrap an object as a Proxy object
Overrides: SCons.Util.Proxy._init_, extit(inherited documentation)

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

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

```python
add_action(self, suffix, action)
```

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

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

```python
x.__delattr__('name') == del x.name
```

```python
__getattr__(self, name)
```
Retrieve an attribute from the wrapped object. If the named attribute doesn’t exist, AttributeError is raised
Class CompositeBuilder

Module SCons.Builder

```python
__getattribute__(...)  
x.__getattribute__('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)

get(self)  
Retrieve the entire wrapped object
```

3.10.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
4 Module SCons.CacheDir

CacheDir support

4.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheRetrieveFunc(target, source, env)</td>
<td></td>
</tr>
<tr>
<td>CacheRetrieveString(target, source, env)</td>
<td></td>
</tr>
<tr>
<td>CachePushFunc(target, source, env)</td>
<td></td>
</tr>
</tbody>
</table>

4.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: <code>src/engine/SCons/CacheDir.py 5023 2010/06/14 22:05:46 sc...</code></td>
</tr>
<tr>
<td><strong>doc</strong></td>
<td>Value: ...</td>
</tr>
<tr>
<td>cache_enabled</td>
<td>Value: True</td>
</tr>
<tr>
<td>cache_debug</td>
<td>Value: False</td>
</tr>
<tr>
<td>cache_force</td>
<td>Value: False</td>
</tr>
<tr>
<td>cache_show</td>
<td>Value: False</td>
</tr>
<tr>
<td>CacheRetrieveSilent</td>
<td>Value: SCons.Action.Action(CacheRetrieveFunc, None)</td>
</tr>
</tbody>
</table>

4.3 Class CacheDir

```
object | SCons.CacheDir.CacheDir
```

4.3.1 Methods

```
__init__ (self, path)  # overrides object.__init__ (inherited documentation)
```

CacheDebug(self, fmt, target, cachefile)

is_enabled(self)
cachePath (self, node)

retrieve (self, node)

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

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

push (self, node)

push_if_forced (self, node)

__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
Class CacheDir

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__reduce__</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__repr__</code></td>
<td><code>repr(x)</code></td>
</tr>
<tr>
<td><code>__setattr__</code></td>
<td><code>x.__setattr__('name', value) &lt;=&gt; x.name = value</code></td>
</tr>
<tr>
<td><code>__str__</code></td>
<td><code>str(x)</code></td>
</tr>
</tbody>
</table>

4.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_class_</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>
5 Module SCons.Conftest

SCons.Conftest

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

5.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckBuilder</td>
<td>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. &quot;language&quot; should be &quot;C&quot; or &quot;C++&quot; and is used to select the compiler. Default is &quot;C&quot;. &quot;text&quot; may be used to specify the code to be build. Returns an empty string for success, an error message for failure.</td>
</tr>
<tr>
<td>CheckCC</td>
<td>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.</td>
</tr>
<tr>
<td>CheckSHCC</td>
<td>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.</td>
</tr>
<tr>
<td>CheckCXX</td>
<td>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.</td>
</tr>
</tbody>
</table>
### 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.
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogInputFiles</td>
<td>Value: 1</td>
</tr>
<tr>
<td>LogErrorMessages</td>
<td>Value: 1</td>
</tr>
</tbody>
</table>
6 Module SCons.Debug

SCons.Debug

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

6.1 Functions

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Debug.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
<tr>
<td>tracked_classes</td>
<td>Value: {'Action.CommandAction': [weakref at 0x875b7d4; to 'Lazy...}</td>
</tr>
<tr>
<td>caller_bases</td>
<td>Value: {}</td>
</tr>
<tr>
<td>caller_dicts</td>
<td>Value: {}</td>
</tr>
</tbody>
</table>

continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shorten_list</td>
<td>Value: [('/scons/SCons/', 1), ('/src/engine/SCons/', 1), ('/usr/...</td>
</tr>
<tr>
<td>TraceFP</td>
<td>Value: {}</td>
</tr>
<tr>
<td>TraceDefault</td>
<td>Value: '/dev/tty'</td>
</tr>
<tr>
<td>TimeStampDefault</td>
<td>Value: False</td>
</tr>
<tr>
<td>StartTime</td>
<td>Value: 1276578476.45</td>
</tr>
<tr>
<td>PreviousTime</td>
<td>Value: 1276578476.45</td>
</tr>
</tbody>
</table>
7 Module SCons.Defaults

SCons.Defaults

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

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

7.1 Functions

DefaultEnvironment(*args, **kw)

Initial public entry point for creating the default construction Environment.

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

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

StaticObjectEmitter(target, source, env)

SharedObjectEmitter(target, source, env)

SharedFlagChecker(source, target, env)

get_paths_str(dest)

chmod_func(dest, mode)

chmod_strfunc(dest, mode)

copy_func(dest, src)

delete_func(dest, must_exist=0)

delete_strfunc(dest, must_exist=0)
Variables

### mkdir_func(dest)

### move_func(dest, src)

### touch_func(dest)

### processDefines(defs)

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

#### 7.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Defaults.py 5023 2010/06/14 22:05:46 sc...</td>
</tr>
<tr>
<td>SharedCheck</td>
<td>Value: SCons.Action.Action(SharedFlagChecker, None)</td>
</tr>
<tr>
<td>CScan</td>
<td>Value: SCons.Tool.CScanner</td>
</tr>
<tr>
<td>DScan</td>
<td>Value: SCons.Tool.DScanner</td>
</tr>
<tr>
<td>LaTeXScan</td>
<td>Value: SCons.Tool.LaTeXScanner</td>
</tr>
<tr>
<td>ObjSourceScan</td>
<td>Value: SCons.Tool.SourceFileScanner</td>
</tr>
<tr>
<td>ProgScan</td>
<td>Value: SCons.Tool.ProgramScanner</td>
</tr>
<tr>
<td>DirScanner</td>
<td>Value: SCons.Scanner.Dir.DirScanner()</td>
</tr>
<tr>
<td>DirEntryScanner</td>
<td>Value: SCons.Scanner.Dir.DirEntryScanner()</td>
</tr>
<tr>
<td>CAction</td>
<td>Value: SCons.Action.Action(&quot;$CCCOM&quot;, &quot;$CCCOMSTR&quot;)</td>
</tr>
<tr>
<td>ShCXXAction</td>
<td>Value: SCons.Action.Action(&quot;$CXXCOM&quot;, &quot;$CXXCOMSTR&quot;)</td>
</tr>
<tr>
<td>ShCXXAction</td>
<td>Value: SCons.Action.Action(&quot;$SHCXXCOM&quot;, &quot;$SHCXXCOMSTR&quot;)</td>
</tr>
<tr>
<td>ASAction</td>
<td>Value: SCons.Action.Action(&quot;$ASCOM&quot;, &quot;$ASCOMSTR&quot;)</td>
</tr>
<tr>
<td>ShLinkAction</td>
<td>Value: SCons.Action.Action(&quot;$SLLINKCOM&quot;, &quot;$SLLINKCOMSTR&quot;)</td>
</tr>
<tr>
<td>Chmod</td>
<td>Value: ActionFactory(chmod_func, chmod_strfunc)</td>
</tr>
<tr>
<td>Copy</td>
<td>Value: ActionFactory(copy_func, lambda dest, src: 'Copy(&quot;%s&quot;, &quot;%s&quot;)')</td>
</tr>
<tr>
<td>Delete</td>
<td>Value: ActionFactory(delete_func, delete_strfunc)</td>
</tr>
<tr>
<td>Mkdir</td>
<td>Value: ActionFactory(mkdir_func, lambda dir: 'Mkdir(%s) % get_p...')</td>
</tr>
<tr>
<td>Move</td>
<td>Value: ActionFactory(move_func, lambda dest, src: 'Move(&quot;%s&quot;, &quot;%s&quot;)')</td>
</tr>
<tr>
<td>Touch</td>
<td>Value: ActionFactory(touch_func, lambda file: 'Touch(%s) % get_p...')</td>
</tr>
<tr>
<td>ConstructionEnvironment</td>
<td>Value: {'BUILDERS': {}, 'CONFIGUREDIR': '/#.sconf_temp', 'CONFIG...}</td>
</tr>
</tbody>
</table>
7.3 Class NullCmdGenerator

```plaintext
object -> SCons.Defaults.NullCmdGenerator

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

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

Example usage:
```python
env['DO NOTHING'] = NullCmdGenerator
env['LINKCOM'] = '${DO NOTHING(''LINK $SOURCES $TARGET'')}"
```

7.3.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td><code>(self, cmd)</code></td>
<td><code>x.__init__(...)</code> initializes x; see <code>x.__class__.__doc__</code> for signature</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Overrides: <code>object.__init__</code> (inherited documentation)</td>
</tr>
<tr>
<td><code>__call__</code></td>
<td><code>(self, target, source, env, for_signature=False)</code></td>
<td></td>
</tr>
<tr>
<td><code>__delattr__</code></td>
<td><code>(...)</code></td>
<td><code>x.__delattr__('name')</code> == del x.name</td>
</tr>
<tr>
<td><code>__getattribute__</code></td>
<td><code>(...)</code></td>
<td><code>x.__getattribute__('name')</code> == x.name</td>
</tr>
<tr>
<td><code>__hash__</code></td>
<td><code>(x)</code></td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><code>__new__</code></td>
<td><code>(T, S, ...)</code></td>
<td>Return Value: a new object with type <code>S</code>, a subtype of <code>T</code></td>
</tr>
<tr>
<td><code>__reduce__</code></td>
<td><code>(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__</code></td>
<td><code>(...)</code></td>
<td>helper for pickle</td>
</tr>
</tbody>
</table>

68
7.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

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

call__(self, *args, **kw)

delattr__(...)  
x.__delattr__(‘name’) <==> del x.name

getattribute__(...)  
x.__getattribute__(‘name’) <==> x.name
7.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
8  Module SCons.Environment

SCons.Environment

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

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

8.1  Functions

alias_builder(env, target, source)

apply_tools(env, tools, toolpath)

copy_non_reserved_keywords(dict)

is_valid_construction_var(varstr)
Return if the specified string is a legitimate construction variable.

default_decide_source(dependency, target, prev_ni)

default_decide_target(dependency, target, prev_ni)

default_copy_from_cache(src, dst)

NoSubstitutionProxy(subject)

8.2  Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: ’src/engine/SCons/Environment.py 5023 2010/06/14 22:05:46...</td>
</tr>
<tr>
<td>Clean Targets</td>
<td>Value: {}</td>
</tr>
<tr>
<td>CalculatorArgs</td>
<td>Value: {}</td>
</tr>
<tr>
<td>AliasBuilder</td>
<td>Value: SCons.Builder.Builder(action= alias_builder, target_facto...</td>
</tr>
<tr>
<td>reserved_construction_var_names</td>
<td>Value: ['CHANGED_SOURCES', 'CHANGED_TARGETS', 'SOURCE', 'SOURCES...</td>
</tr>
<tr>
<td>future_reserved_construction_var_names</td>
<td>Value: []</td>
</tr>
</tbody>
</table>
8.3 Class MethodWrapper

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

```python
__init__(self, object, method, name=False)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)
```

```python
_call_(self, *args, **kwargs)
```

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

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

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

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

```python
__hash__(x)
hash(x)
```
_new_\( (T, S, ...) \)

Return Value

\[ \text{a new object with type } S, \text{ a subtype of } T \]

_reduce_\( (...) \)

helper for pickle

_reduce_ex_\( (...) \)

helper for pickle

_repr_\( (x) \)

\[ \text{repr}(x) \]

_setattr_\( (...) \)

\[ x._\text{setattr}('name', \text{value}) \iff x._\text{name} = \text{value} \]

_str_\( (x) \)

\[ \text{str}(x) \]

### 8.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

### 8.4 Class BuilderWrapper

```
object
   SCons.Environment.MethodWrapper
         SCons.Environment.BuilderWrapper
```

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

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

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

8.4.1 Methods

```python
__call__(self, target=False, source=<class 'SCons.Environment.Null'>, *args, **kw)
```

```python
__repr__(self)
repr(x)
```
Overrides: object.__repr__ (inherited documentation)

```python
__str__(self)
str(x)
```
Overrides: object.__str__ (inherited documentation)

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

```python
__setattr__(self, name, value)
x.__setattr__('name', value) == x.name = value
```
Overrides: object.__setattr__ (inherited documentation)

```python
__delattr__(...
```
```python
x.__delattr__('name') == del x.name
```

```python
__getattribute__(...
```
```python
x.__getattribute__('name') == x.name
```

```python
__hash__(x)
hash(x)
```

```python
__init__(self, object, method, name=False)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
```
Overrides: object.__init__ (inherited documentation)

```python
__new__(T, S, ...)
```
Return Value
```
a new object with type S, a subtype of T
```
Class BuilderDict

Module SCons.Environment

```python
__reduce__(...)  
helper for pickle

__reduce_ex__(...)  
helper for pickle

clone(self, new_object)

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

8.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

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

```python
__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)  
```

75
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__contains__</code></td>
<td>checks if a key is in the environment</td>
</tr>
<tr>
<td><code>__getitem__</code></td>
<td>retrieves the value associated with the key</td>
</tr>
<tr>
<td><code>__len__</code></td>
<td>returns the number of keys</td>
</tr>
<tr>
<td><code>__repr__</code></td>
<td>returns a string representation of the environment</td>
</tr>
<tr>
<td><code>clear</code></td>
<td>removes all key-value pairs from the environment</td>
</tr>
<tr>
<td><code>copy</code></td>
<td>returns a copy of the environment</td>
</tr>
<tr>
<td><code>fromkeys</code></td>
<td>creates an environment from a dictionary</td>
</tr>
<tr>
<td><code>get</code></td>
<td>retrieves the value associated with the key</td>
</tr>
<tr>
<td><code>has_key</code></td>
<td>checks if a key is in the environment</td>
</tr>
<tr>
<td><code>items</code></td>
<td>returns an iterator of (key, value) pairs</td>
</tr>
<tr>
<td><code>iteritems</code></td>
<td>returns an iterator of key-value pairs</td>
</tr>
<tr>
<td><code>iterkeys</code></td>
<td>returns an iterator of keys</td>
</tr>
<tr>
<td><code>itervalues</code></td>
<td>returns an iterator of values</td>
</tr>
<tr>
<td><code>keys</code></td>
<td>returns an iterator of keys</td>
</tr>
<tr>
<td><code>pop</code></td>
<td>removes and returns the value associated with the key</td>
</tr>
<tr>
<td><code>popitem</code></td>
<td>removes and returns the key-value pair</td>
</tr>
<tr>
<td><code>setdefault</code></td>
<td>sets the value associated with the key</td>
</tr>
<tr>
<td><code>values</code></td>
<td>returns an iterator of values</td>
</tr>
</tbody>
</table>

### 8.6 Class SubstitutionEnvironment

```
object
    SCons.Environment.SubstitutionEnvironment
```

**Known Subclasses:** `SCons.Environment.Base`

Base class for different flavors of construction environments.

This class contains a minimal set of methods that handle construction variable expansion and conversion of strings to Nodes, which may or
Class SubstitutionEnvironment Module SCons.Environment

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

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

8.6.1 Methods

_init_(self, **kw)
Initialization of an underlying SubstitutionEnvironment class.
Overrides: object.__init__

_cmp_(self, other)

_delitem_(self, key)

_getitem_(self, key)

_setitem_(self, key, value)

get(self, key, default=False)
Emulates the get() method of dictionaries.

has_key(self, key)

_contains_(self, key)

items(self)

arg2nodes(self, args, node_factory=<class 'SCons.Environment.Null'>, lookup_list=<class 'SCons.Environment.Null'>, **kw)
### gvars(self)

### lvars(self)

### subst(self, string, raw=False, target=False, source=False, conv=False, executor=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=False, target=False, source=False)

### subst_list(self, string, raw=False, target=False, source=False, conv=False, executor=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=False, target=False, source=False, conv=False, executor=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=1, 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.

_delattr_(...)  
x._delattr_('name') <= del x.name

_getattribute_(...)  
x._getattribute_('name') <= x.name

_hash_(x)  
hash(x)
Class Base

Module SCons.Environment

---

_.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)

---

8.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>.class</em></td>
<td>Value: &lt;attribute ‘<em>.class</em>’ of ‘object’ objects&gt;</td>
</tr>
</tbody>
</table>

8.6.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>.metaclass</em></td>
<td>Value: SCons.Memoize.Memoized_Metaclass</td>
</tr>
</tbody>
</table>

8.7 Class Base

object

SCons.Environment.SubstitutionEnvironment

SCons.Environment.Base


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=1)

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

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

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

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

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

**Builder**(self, **kw)**
### Class Base Module `SCons.Environment`

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| **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 a dependency.

**Literal** *(self, string)*

**Local** *(self, *targets)*

**MergeFlags** *(self, args, unique=1, 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=1)*

Use the specified function to parse the output of the command in order to modify the current environment. The 'command' can be a string or a list of strings representing a command and its arguments. 'Function' is an optional argument that takes the environment, the output of the command, and the unique flag. If no function is specified, MergeFlags, which treats the output as the result of a typical 'X-config' command (i.e. gtk-config), will merge the output into the appropriate variables.
ParseDepends(self, filename, must_exist=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=1)

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

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

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

Prepend values to existing construction variables in an Environment, if they’re not already there. If delete_existing is 1, removes existing values first, so values move to front.
<table>
<thead>
<tr>
<th><strong>Method</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RemoveMethod</strong>(self, function)</td>
<td>Removes the specified function’s MethodWrapper from the added_methods list, so we don’t re-bind it when making a clone.</td>
</tr>
<tr>
<td><strong>Replace</strong>(self, **kw)</td>
<td>Replace existing construction variables in an Environment with new construction variables and/or values.</td>
</tr>
<tr>
<td><strong>ReplaceIxes</strong>(self, path, old_prefix, old_suffix, new_prefix, new_suffix)</td>
<td>Replace old_prefix with new_prefix and old_suffix with new_suffix.</td>
</tr>
<tr>
<td><strong>Repository</strong>(self, *dirs, **kw)</td>
<td>Specifies that a <code>Repository</code> is used.</td>
</tr>
<tr>
<td><strong>Requires</strong>(self, target, prerequisite)</td>
<td>Specify that 'prerequisite' must be built before 'target', (but 'target' does not actually depend on 'prerequisite' and need not be rebuilt if it changes).</td>
</tr>
<tr>
<td><strong>SConsignFile</strong>(self, name='.sconsign', dbm_module=False)</td>
<td>Add a SConsign file.</td>
</tr>
<tr>
<td><strong>Scanner</strong>(self, *args, **kw)</td>
<td>Produce a scanner object for the given type.</td>
</tr>
<tr>
<td><strong>SetDefault</strong>(self, **kw)</td>
<td>Set default values for construction variables.</td>
</tr>
<tr>
<td><strong>SideEffect</strong>(self, side_effect, target)</td>
<td>Tell scons that side_effects are built as side effects of building targets.</td>
</tr>
<tr>
<td><strong>SourceCode</strong>(self, entry, builder)</td>
<td>Arrange for a source code builder for (part of) a tree.</td>
</tr>
<tr>
<td><strong>SourceSignatures</strong>(self, type)</td>
<td>Add a source code signature for the given type.</td>
</tr>
</tbody>
</table>
### Split

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

### Tool

### Value

### VariantDir

### WhereIs

Find prog in the path.

### __cmp__

### __contains__

### __delattr__

\[x.__delattr__('name') \leftrightarrow \text{del} \ x.name\]

### __delitem__

### __getattribute__

\[x.__getattribute__('name') \leftrightarrow x.name\]

### __getitem__

### __hash__

\[\text{hash}(x)\]
Class Base

Module SCons.Environment

```python
__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__

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

Return Value

a new object with type S, a subtype of T

```python
__reduce__(...)
```

helper for pickle

```python
__reduce_ex__(...)
```

helper for pickle

```python
__repr__(x)
```

repr(x)

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

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

```python
__str__(x)
```

str(x)

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

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

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

Emulates the get() method of dictionaries.

```python
get_CacheDir(self)
```

88
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>get_builder</strong> <em>(self, name)</em></td>
<td>Fetch the builder with the specified name from the environment.</td>
</tr>
<tr>
<td><strong>get_factory</strong> <em>(self, factory, default='File')</em></td>
<td>Return a factory function for creating Nodes for this construction environment.</td>
</tr>
<tr>
<td><strong>get_scanner</strong> <em>(self, key)</em></td>
<td>Find the appropriate scanner given a key (usually a file suffix).</td>
</tr>
<tr>
<td><strong>get_src_sig_type</strong> <em>(self)</em></td>
<td></td>
</tr>
<tr>
<td><strong>get_tgt_sig_type</strong> <em>(self)</em></td>
<td></td>
</tr>
<tr>
<td><strong>gvars</strong> <em>(self)</em></td>
<td></td>
</tr>
<tr>
<td><strong>has_key</strong> <em>(self, key)</em></td>
<td></td>
</tr>
<tr>
<td><strong>items</strong> <em>(self)</em></td>
<td></td>
</tr>
<tr>
<td><strong>lvars</strong> <em>(self)</em></td>
<td></td>
</tr>
<tr>
<td><strong>scanner_map_delete</strong> <em>(self, kw=False)</em></td>
<td>Delete the cached scanner map (if we need to).</td>
</tr>
<tr>
<td><strong>subst</strong> <em>(self, string, raw=0, target=False, source=False, conv=False, executor=False)</em></td>
<td>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.</td>
</tr>
<tr>
<td><strong>subst_kw</strong> <em>(self, kw, raw=0, target=False, source=False)</em></td>
<td></td>
</tr>
<tr>
<td><strong>subst_list</strong> <em>(self, string, raw=0, target=False, source=False, conv=False, executor=False)</em></td>
<td>Calls through to SCons.Subst.scons_subst_list(). See the documentation for that function.</td>
</tr>
</tbody>
</table>
**Class OverrideEnvironment**

**Module SCons.Environment**

```python
def subst_path(self, path, target=False, source=False):
    """Substitute a path list, turning EntryProxies into Nodes and leaving Nodes (and other objects) as-is."""
```

```python
def subst_target_source(self, string, raw=0, target=False, source=False, conv=False, executor=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 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

### 8.7.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: SCons.Memoize.MemoizedMetaclass</td>
</tr>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
</tbody>
</table>

### 8.8 Class 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

```python
_init__(self, subject, overrides={})
```

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

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

Overrides: SCons.Environment.Base._init__ extit(inherited documentation)

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

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

Overrides: object.__setattr__ extit(inherited documentation)

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

Overrides: SCons.Environment.SubstitutionEnvironment.__getitem__

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

Overrides: SCons.Environment.SubstitutionEnvironment.__setitem__

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

Overrides: SCons.Environment.SubstitutionEnvironment.__delitem__

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

Emulates the get() method of dictionaries.


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

Overrides: SCons.Environment.SubstitutionEnvironment.has_key

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

Overrides: SCons.Environment.SubstitutionEnvironment.__contains__
### Class OverrideEnvironment

Module SCons.Environment

**Dictionary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Dictionary(self)</code></td>
<td>Emulates the items() method of dictionaries.</td>
</tr>
</tbody>
</table>

**items**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>items(self)</code></td>
<td>Emulates the items() method of dictionaries.</td>
</tr>
<tr>
<td>Overrides: SCons.Environment.SubstitutionEnvironment.items</td>
<td></td>
</tr>
</tbody>
</table>

**gvars**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>gvars(self)</code></td>
<td></td>
</tr>
</tbody>
</table>

**lvars**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>lvars(self)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Replace**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Replace(self, **kw)</code></td>
<td>Replace existing construction variables in an Environment with new construction variables and/or values.</td>
</tr>
</tbody>
</table>

**Action**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Action(self, *args, **kw)</code></td>
<td></td>
</tr>
</tbody>
</table>

**AddMethod**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AddMethod(self, function, name=False)</code></td>
<td>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.</td>
</tr>
</tbody>
</table>

**AddPostAction**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AddPostAction(self, files, action)</code></td>
<td></td>
</tr>
</tbody>
</table>

**AddPreAction**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AddPreAction(self, files, action)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Alias**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Alias(self, target, source=[], action=False, **kw)</code></td>
<td></td>
</tr>
</tbody>
</table>

**AlwaysBuild**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>AlwaysBuild(self, *targets)</code></td>
<td></td>
</tr>
</tbody>
</table>

**Append**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Append(self, **kw)</code></td>
<td>Append values to existing construction variables in an Environment.</td>
</tr>
</tbody>
</table>
AppendENVPath(self, name, newpath, envname='ENV', sep=':', delete_existing=1)

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

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

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

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

BuildContext(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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FindIxes</strong>&lt;br&gt;<em>(self, paths, prefix, suffix)</em></td>
<td>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.</td>
</tr>
<tr>
<td><strong>FindSourceFiles</strong>&lt;br&gt;<em>(self, node='.')</em></td>
<td>Returns a list of all source files.</td>
</tr>
<tr>
<td><strong>Flatten</strong>&lt;br&gt;<em>(self, sequence)</em></td>
<td></td>
</tr>
<tr>
<td><strong>GetBuildPath</strong>&lt;br&gt;<em>(self, files)</em></td>
<td></td>
</tr>
<tr>
<td><strong>Glob</strong>&lt;br&gt;<em>(self, pattern, ondisk=True, source=False, strings=False)</em></td>
<td></td>
</tr>
<tr>
<td><strong>Ignore</strong>&lt;br&gt;<em>(self, target, dependency)</em></td>
<td>Ignore a dependency.</td>
</tr>
<tr>
<td><strong>Literal</strong>&lt;br&gt;<em>(self, string)</em></td>
<td></td>
</tr>
<tr>
<td><strong>Local</strong>&lt;br&gt;*(self, <em>targets)</em></td>
<td></td>
</tr>
<tr>
<td><strong>MergeFlags</strong>&lt;br&gt;<em>(self, args, unique=1, dict=False)</em></td>
<td>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.</td>
</tr>
<tr>
<td><strong>NoCache</strong>&lt;br&gt;*(self, <em>targets)</em></td>
<td>Tags a target so that it will not be cached</td>
</tr>
<tr>
<td><strong>NoClean</strong>&lt;br&gt;*(self, <em>targets)</em></td>
<td>Tags a target so that it will not be cleaned by -c</td>
</tr>
</tbody>
</table>
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=1)

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

ParseDepends(self, filename, must_exist=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**

```python
self, **kw)
```

Prepend values to existing construction variables in an Environment.

**PrependENVPath**

```python
self, name, newpath, envname='ENV', sep=':', delete_existing=1)
```

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

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

**PrependUnique**

```python
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**

```python
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**

```python
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**

```python
self, *dirs, **kw)
```

**Requires**

```python
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**

```python
self, name='\.sconsign', dbm_module=False)
```

97
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=1)

WhereIs(self, prog, path=False, pathext=False, reject=[]) Find prog in the path.

__cmp__(self, other)

__delattr__(...)  
x.__delattr__(‘name’) <=> del x.name
Class OverrideEnvironment

Module SCons.Environment

__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)

__str__(x)  

str(x)

arg2nodes(self, args, node_factory=<class 'SCons.Environment.Null'>, lookup_list=<class 'SCons.Environment.Null'>, **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, executor=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, executor=False)
Calls through to SCons.Subst.sconssubst_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, executor=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 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

8.8.3 Class Variables
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__metaclass__</code></td>
<td>Value: <code>SCons.Memoize.MemoizedMetaclasse</code></td>
</tr>
<tr>
<td>memoizer_counters</td>
<td>Value: <code>[]</code></td>
</tr>
</tbody>
</table>

8.9 Class Base

object


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)

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.

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

**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

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

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

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

### AppendUnique

`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

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

### Builder

`Builder(self, **kw)`

### CacheDir

`CacheDir(self, path)`

### Clean

`Clean(self, targets, files)`

### Clone

`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

`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

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

### Copy

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

### Decider

`Decider(self, function)`
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depends(self, target, dependency)</td>
<td>Explicitly specify that 'target's depend on 'dependency'.</td>
</tr>
<tr>
<td>Detect(self, progs)</td>
<td>Return the first available program in progs.</td>
</tr>
<tr>
<td>Dictionary(self, *args)</td>
<td></td>
</tr>
<tr>
<td>Dir(self, name, *args, **kw)</td>
<td></td>
</tr>
<tr>
<td>Dump(self, key=False)</td>
<td>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.</td>
</tr>
<tr>
<td>Entry(self, name, *args, **kw)</td>
<td></td>
</tr>
<tr>
<td>Environment(self, **kw)</td>
<td></td>
</tr>
<tr>
<td>Execute(self, action, *args, **kw)</td>
<td>Directly execute an action through an Environment</td>
</tr>
<tr>
<td>File(self, name, *args, **kw)</td>
<td></td>
</tr>
<tr>
<td>FindFile(self, file, dirs)</td>
<td></td>
</tr>
<tr>
<td>FindInstalledFiles(self)</td>
<td>returns the list of all targets of the Install and InstallAs Builder.</td>
</tr>
</tbody>
</table>
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=1, 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=1)

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

ParseDepends(self, filename, must_exist=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

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

Prepend values to existing construction variables in an Environment.

### PrependENVPath

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

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

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

### PrependUnique

**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

**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

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

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

### ReplaceIxes

**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

**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=1)*
WhereIs(self, prog, path=False, pathext=False, reject=[])  
Find prog in the path.

__cmp__(self, other)

__contains__(self, key)

__delattr__(...)  
x.__delattr__('name') --> del x.name

__delitem__(self, key)

__getattribute__(...)  
x.__getattribute__('name') --> x.name

__getitem__(self, key)

__hash__(x)  
hash(x)

__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__

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

__reduce__(...)  
helper for pickle

__reduce_ex__(...)  
helper for pickle
```python
__repr__(x)
repr(x)

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

__setitem__(self, key, value)

__str__(x)
str(x)

arg2nodes(self, args, node_factory=<class 'SCons.Environment.Null'>, lookup_list=<class 'SCons.Environment.Null'>, **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, executor=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, executor=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, executor=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 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

8.9.3 Class Variables
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: SCons.Memoize.Memoized_Metaclasse</td>
</tr>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
</tbody>
</table>
9 Module SCons.Errors

SCons.Errors

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

9.1 Functions

convert_to_BuildError(status, exc_info=False)

Convert any return code a BuildError Exception.

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

9.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Errors.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
</tbody>
</table>

9.3 Class BuildError

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.BuildError

Errors occurring while building.

BuildError have the following attributes:

Information about the cause of the build error:

---

errstr : a description of the error message

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

exitstatus : SCons exit status due to this build error.
Class BuildError Module SCons.Errors

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

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

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

Information about the cause of the location of the error:

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

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

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

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

9.3.1 Methods

```python
__init__(self, node=False, errstr='Unknown error', status=2, exitstatus=2, filename=False, executor=False, action=False, command=False, exc_info=(None, None, None))
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.Exception.__init__(inheritad documentation)
```

```python
str_(self)
str(x)
Overrides: exceptions.BaseException.__str__(inherited documentation)
```
### Class BuildError

Module SCons.Errors

```python
_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)

_new_((T, S, ...)

Return Value

  a new object with type S, a subtype of T

Overides: exceptions.BaseException._new_

_reduce_((...)

helper for pickle

Overides: object._reduce_ extit(inherited documentation)

_reduce_ex_((...)

helper for pickle

_repr_((x)

repr(x)

Overides: object._repr_

_setattr_((...)

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

Overides: object._setattr_

_setstate_((...)
```
9.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: <code>&lt;attribute '_class_' of 'object' objects&gt;</code></td>
</tr>
<tr>
<td>args</td>
<td>Value: <code>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
<tr>
<td>message</td>
<td>Value: <code>&lt;member 'message' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
</tbody>
</table>

9.4 Class InternalError

```
object          \|--
\       \     \   \   \   \   \   
exceptions.BaseException  \     \   \   \   
                     \   \   \   \   \   
exceptions.Exception       \   \   \   \   \   
\                     \   \   \   \   \   
SCons.Errors.InternalError
```

9.4.1 Methods

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

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

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

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

```python
__hash__(x)  
hash(x)
```

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: exceptions.BaseException.__init__
```
Class InternalError

Module SCons.Errors

__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__

9.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
9.5 Class UserError

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

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

9.5.1 Methods

_delattr__(...)  
x._delattr_('name') == del x.name  
Overrides: object._delattr_

_getattribute__(...)  
x._getattribute_('name') == x.name  
Overrides: object._getattribute_

_getitem__(x, y)  
x[y]

_getslice__(x, i, j)  
x[i:j]

Use of negative indices is not supported.

_hash__(x)  
hash(x)

_init__(...)  
x._init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: exceptions.BaseException.__init_

_new__(T, S, ...)  

Return Value  
  a new object with type S, a subtype of T  
Overr).es: exceptions.BaseException.__new__
Class StopError

Overrides: object.__reduce_ex__ (inherited documentation)

Overrides: object.__repr__

Overrides: object.__setattr__

Overrides: object.__str__

9.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
<tr>
<td>args</td>
<td>Value: <code>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
<tr>
<td>message</td>
<td>Value: <code>&lt;member 'message' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
</tbody>
</table>

9.6 Class StopError

```
```
9.6.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_delattr_(...)</td>
<td>_delattr_('name') \implies \texttt{del x.name}</td>
</tr>
<tr>
<td></td>
<td>Overrides: \texttt{object._delattr_}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_getattr_(...)</td>
<td>_getattr_('name') \implies x.name</td>
</tr>
<tr>
<td></td>
<td>Overrides: \texttt{object._getattr_}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_getitem_(x, y)</td>
<td>\texttt{x[y]}</td>
</tr>
<tr>
<td>_getslice_(x, i, j)</td>
<td>\texttt{x[i:j]}</td>
</tr>
<tr>
<td></td>
<td>Use of negative indices is not supported.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_hash_(x)</td>
<td>hash(x)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_init_(...)</td>
<td>_init_(...) initializes x; see _class__._doc_ for signature</td>
</tr>
<tr>
<td></td>
<td>Overrides: exceptions.BaseException._init_</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_new_(T, S, ...)</td>
<td>Return Value \new object with type S, a subtype of T</td>
</tr>
<tr>
<td></td>
<td>Overrides: exceptions.BaseException._new_</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_reduce_(...)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td></td>
<td>Overrides: object._reduce_.\textit{extit(inherited documentation)}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_reduce_ex_(...)</td>
<td>helper for pickle</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_repr_(x)</td>
<td>repr(x)</td>
</tr>
<tr>
<td></td>
<td>Overrides: \texttt{object._repr_}</td>
</tr>
</tbody>
</table>
9.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
<tr>
<td>args</td>
<td>Value: <code>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
<tr>
<td>message</td>
<td>Value: <code>&lt;member 'message' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
</tbody>
</table>

9.7 Class EnvironmentError


9.7.1 Methods

<table>
<thead>
<tr>
<th><strong>delattr</strong>(...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.<strong>delattr</strong>('name')</td>
</tr>
<tr>
<td>Overrides: object.<strong>delattr</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>getattribute</strong>(...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x.<strong>getattribute</strong>('name')</td>
</tr>
<tr>
<td>Overrides: object.<strong>getattribute</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>getitem</strong>(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x[y]</td>
</tr>
</tbody>
</table>

120
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, ...)
Returns Value
    a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__

__reduce__(...)
helper for pickle
Overrides: object.__reduce__ (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
9.8 Class MSVCError

object
declares BaseException
declares Exception

declares StandardError

declares EnvironmentError

declares IOError

declares SCons.Errors.MSVCError

9.8.1 Methods

_delattr__(...)  

\[
\text{x.delattr('name')} \iff \text{del x.name}
\]

Overrides: object._delattr_

_getattribute__(...)  

\[
\text{x.getattribute('name')} \iff x.name
\]

Overrides: object._getattribute_

_getitem_(x, y)  

\[
x[y]
\]

_getslice_(x, i, j)  

\[
x[i:j]
\]

Use of negative indices is not supported.

_hash_(x)  

\[
\text{hash(x)}
\]
### Class MSVCError

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.EnvironmentError.__init__
```

```python
__new__(T, S, ...)  
Return Value  
a new object with type S, a subtype of T
Overrides: exceptions.EnvironmentError.__new__
```

```python
__reduce__(...)  
helper for pickle
Overrides: exceptions.BaseException.__reduce__
```

```python
__reduce_ex__(...)  
helper for pickle
```

```python
__repr__(x)  
repr(x)
Overrides: object.__repr__
```

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

```python
__setstate__(...)  

```

```python
__str__(x)  
str(x)
Overrides: exceptions.BaseException.__str__
```

### 9.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '.__class__' of 'object' objects&gt;</code></td>
</tr>
<tr>
<td>args</td>
<td>Value: <code>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
<tr>
<td>errno</td>
<td>Value: <code>&lt;member 'errno' of 'exceptions.EnvironmentError' objects&gt;</code></td>
</tr>
<tr>
<td>filename</td>
<td>Value: <code>&lt;member 'filename' of 'exceptions.EnvironmentError' objects&gt;</code></td>
</tr>
<tr>
<td>message</td>
<td>Value: <code>&lt;member 'message' of 'exceptions.EnvironmentError' objects&gt;</code></td>
</tr>
</tbody>
</table>

*continued on next page*
### 9.9 Class ExplicitExit

```
Class ExplicitExit

```

#### 9.9.1 Methods

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

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

Overrides: exceptions.Exception.__init__ (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_(inherited documentation)

_reducex_(...)  
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__

9.9.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
10 Module SCons.Executor

SCons.Executor

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

10.1 Functions

**rfile(node)**

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

**GetBatchExecutor(key)**

**AddBatchExecutor(key, executor)**

**get_NullEnvironment()**

Use singleton pattern for Null Environments.

10.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>revision</td>
<td>Value: ʼsrc/engine/SCons/Executor.py 5023 2010/06/14 22:05:46 sc...</td>
</tr>
<tr>
<td>nullenv</td>
<td>Value: False</td>
</tr>
</tbody>
</table>

10.3 Class Batch

```
object
```

SCons.Executor.Batch

Remembers exact association between targets and sources of executor.

10.3.1 Methods

```
__init__(self, targets=[], sources=[])  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: object.__init__ (inherited documentation)
```
Class TSList Module SCons.Executor

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

__getattribute__(...)  
x.__getattribute__('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)

10.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

10.4 Class TSList

UserList.UserList — SCons.Executor.TSList

A class that implements $TARGETS or $SOURCES expansions by wrapping
an executor Method. This class is used in the Executor.lvars() to delay creation of NodeList objects until they're needed.

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

10.4.1 Methods

```python
_init__(self, func)
Overrides: UserList.UserList._init_

_getattr__(self, attr)

_getitem__(self, i)
Overrides: UserList.UserList._getitem_

_getslice__(self, i, j)
Overrides: UserList.UserList._getslice_

_str__(self)

_repr__(self)
Overrides: UserList.UserList._repr_

_add__(self, other)

_cmp__(self, other)

_contains__(self, item)

_delitem__(self, i)

_delslice__(self, i, j)

_eq__(self, other)

_ge__(self, other)

_gt__(self, other)

_iadd__(self, other)

_imul__(self, n)

_le__(self, other)
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__len__()</code></td>
<td>Returns the length of the object.</td>
</tr>
<tr>
<td><code>__lt__()</code></td>
<td>Less than comparison.</td>
</tr>
<tr>
<td><code>__mul__()</code></td>
<td>Multiplication.</td>
</tr>
<tr>
<td><code>__ne__()</code></td>
<td>Not equal comparison.</td>
</tr>
<tr>
<td><code>__radd__()</code></td>
<td>Right addition.</td>
</tr>
<tr>
<td><code>__rmul__()</code></td>
<td>Right multiplication.</td>
</tr>
<tr>
<td><code>__setitem__()</code></td>
<td>Set item at index.</td>
</tr>
<tr>
<td><code>__setslice__()</code></td>
<td>Set slice from index to another object.</td>
</tr>
<tr>
<td><code>append()</code></td>
<td>Append an item to the object.</td>
</tr>
<tr>
<td><code>count()</code></td>
<td>Count occurrences of an item.</td>
</tr>
<tr>
<td><code>extend()</code></td>
<td>Extend the object with another object.</td>
</tr>
<tr>
<td><code>index()</code></td>
<td>Index of the item.</td>
</tr>
<tr>
<td><code>insert()</code></td>
<td>Insert an item at index.</td>
</tr>
<tr>
<td><code>pop()</code></td>
<td>Pop an item at index or last one if index is -1</td>
</tr>
<tr>
<td><code>remove()</code></td>
<td>Remove an item.</td>
</tr>
<tr>
<td><code>reverse()</code></td>
<td>Reverse the object.</td>
</tr>
<tr>
<td><code>sort()</code></td>
<td>Sort the object.</td>
</tr>
</tbody>
</table>

### 10.5 Class TSObject

```
object __init__(self, *args, **kwds)
```

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td>initializes x; see <code>.</code> for signature</td>
</tr>
<tr>
<td><code>__getattribute__</code></td>
<td></td>
</tr>
<tr>
<td><code>__str__</code></td>
<td></td>
</tr>
<tr>
<td><code>__repr__</code></td>
<td></td>
</tr>
<tr>
<td><code>__delattr__</code></td>
<td>x.<strong>delattr</strong>('name') == del x.name</td>
</tr>
<tr>
<td><code>__getattribute__</code></td>
<td></td>
</tr>
<tr>
<td><code>__hash__</code></td>
<td></td>
</tr>
<tr>
<td><code>__new__</code></td>
<td>a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>__reduce__</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__setattr__</code></td>
<td>x.<strong>setattr</strong>('name', value) == x.name = value</td>
</tr>
</tbody>
</table>

10.5.2 Properties
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

## 10.6 Class Executor

```python
class SCons.Executor.Executor
```

A class for controlling instances of executing an action.

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

### 10.6.1 Methods

- **`__init__`**: (self, action, env=False, overridelist=[], targets=[], sources=[], builder_kw={})
  
  x.__init__(...) initializes x; see x.__class__.__doc__ for signature
  
  Overrides: object.__init__(inherited documentation)

- **get_lvars(self)**

- **get_action_targets(self)**

- **set_action_list(self, action)**

- **get_action_list(self)**

- **get_all_targets(self)**
  
  Returns all targets for all batches of this Executor.

- **get_all_sources(self)**
  
  Returns all sources for all batches of this Executor.

- **get_all_children(self)**
  
  Returns all unique children (dependencies) for all batches of this Executor.

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

get_action_side_effects(self)
Returns all side effects for all batches of this Executor used by the underlying Action.

get_build_env(self)
Fetch or create the appropriate build Environment for this Executor.

get_build_scanner_path(self, scanner)
Fetch the scanner path for this executor’s targets and sources.

get_kw(self, kw={})

do_nothing(self, target, kw)

do_execute(self, target, kw)
Actually execute the action list.

__call__(self, target, **kw)

cleanup(self)

add_sources(self, sources)
Add source files to this Executor’s list. This is necessary for "multi" Builders that can be called repeatedly to build up a source file list for a given target.

get_sources(self)

add_batch(self, targets, sources)
Add pair of associated target and source to this Executor’s list. This is necessary for "batch" Builders that can be called repeatedly to build up a list of matching target and source files that will be used in order to update multiple target files at once from multiple corresponding source files, for tools like MSVC that support it.
prepare(self)
Preparatory checks for whether this Executor can go ahead
and (try to) build its targets.

add_pre_action(self, action)

add_post_action(self, action)

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

nullify(self)

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

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

scan_targets(self, scanner)

scan_sources(self, scanner)

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

get_unignored_sources(self, node, ignore=())

get_implicit_deps(self)
Return the executor’s implicit dependencies, i.e. the nodes of
the commands to be executed.

_delattr__(...)
x.__delattr__('name') <-> del x.name
Class Executor

Module SCons.Executor

```python
__getattribute__(...)
x.__getattribute__('name') == x.name
```

```python
__hash__(x)
hash(x)
```

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

Return Value

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

```python
__reduce__(...)
```

helper for pickle

```python
__reduce_ex__(...)
```

helper for pickle

```python
__repr__(x)
```

repr(x)

```python
__setattr__(...)
x.__setattr__('name', value) == x.name = value
```

10.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

10.6.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
</tbody>
</table>

10.6.4 Instance Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>my_str</td>
<td>Value: &lt;function my_str at 0x843187c&gt;</td>
</tr>
</tbody>
</table>

134
10.7 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.

10.7.1 Methods

_init__(self, *args, **kw)

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

Overrides: object._init_(inherited documentation)

get_build_env(self)

get_build_scanner_path(self)

cleanup(self)

prepare(self)

get_unignored_sources(self, *args, **kw)

get_action_targets(self)

get_action_list(self)

get_all_targets(self)

get_all_sources(self)

get_all_children(self)

get_all_prerequisites(self)

get_action_side_effects(self)

_call__(self, *args, **kw)

get_contents(self)
add_pre_action(self, action)

add_post_action(self, action)

set_action_list(self, action)

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

__getattribute__(...)  
x.__getattribute__('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)

10.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
11 Module 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>revision</td>
<td>Value: 'src/engine/SCons/Job.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
<tr>
<td>explicit_stack_size</td>
<td>Value: False</td>
</tr>
<tr>
<td>default_stack_size</td>
<td>Value: 256</td>
</tr>
<tr>
<td>interrupt_msg</td>
<td>Value: 'Build interrupted.'</td>
</tr>
</tbody>
</table>

11.2 Class InterruptState

object └── SCons.Job.InterruptState

11.2.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(self)</td>
<td>x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature</td>
</tr>
<tr>
<td>Overrides: object.<strong>init</strong> (inherited documentation)</td>
<td></td>
</tr>
<tr>
<td>set(self)</td>
<td></td>
</tr>
<tr>
<td><em>call</em>_(self)</td>
<td></td>
</tr>
<tr>
<td><strong>delattr</strong>(...)</td>
<td>x.<strong>delattr</strong>('name') &lt;==&gt; del x.name</td>
</tr>
<tr>
<td><em>getattribute</em>_(...)</td>
<td>x.<strong>getattribute</strong>('name') &lt;==&gt; x.name</td>
</tr>
<tr>
<td><em>hash</em> (x)</td>
<td>hash(x)</td>
</tr>
</tbody>
</table>
11.2.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

11.3 Class Jobs

```
object       \_ SCons.Job.Jobs
```

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

```
_init__(self, num, taskmaster)

create 'num' jobs using the given taskmaster.

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

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

Overrides: object._init_
```

```
run(self, postfunc=<function <lambda> at 0x87217d4>)

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.
```

```
__delattr__(...)

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

```
__getattr__(...)

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

```
__hash__(x)

hash(x)
```

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

Return Value

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

11.4 Class Serial

```
object SCons.Job.Serial
```

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

This class is not thread safe.

11.4.1 Methods

```
__init__(self, taskmaster)
```

Create a new serial job given a taskmaster.

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

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

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

__getattribute__(...)  
x.__getattribute__('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)

11.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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

```python
_init__(self, requestQueue, resultsQueue, interrupted)
Overrides: threading.Thread._init_

run(self)
Overrides: threading.Thread.run

delattr(...)
x.__delattr__('name') == del x.name

_getattribute__(...)
x.__getattribute__('name') == x.name

_hash__(x)
hash(x)

_new__(T, S, ...)
Return Value
   a new object with type S, a subtype of T

_reduce__(...)
helper for pickle

_reduce_ex__(...)
helper for pickle

_repr__(self)
repr(x)
Overrides: object.__repr__ (inherited documentation)
```
Class ThreadPool

Module SCons.Job

```python
setattr

```x.

```python
setattr('name', value)
```<

```python
str(x)
```

```python
getName(self)
```

```python
isAlive(self)
```

```python
isDaemon(self)
```

```python
join(self, timeout=False)
```

```python
setDaemon(self, daemonic)
```

```python
setName(self, name)
```

```python
start(self)
```

11.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

11.6 Class ThreadPool

```
object ▼
```

```
SCons.Job.ThreadPool
```

This class is responsible for spawning and managing worker threads.

11.6.1 Methods

```python
__init__(self, num, stack_size, interrupted)
```

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

One must specify the stack size of the worker threads. The stack size is specified in kilobytes.

Overrides: object.__init__

```python
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.

**__delattr__(...)**
\[x.__delattr__('name') \Rightarrow \text{del} x.name\]

**__getattribute__(...)**
\[x.__getattribute__('name') \Rightarrow x.name\]

**__hash__(x)**
\[\text{hash}(x)\]

**__new__(T, S, ...)**
Return Value
\[\text{a new object with type } S, \text{ a subtype of } T\]

**__reduce__(...)**
helper for pickle

**__reduce_ex__(...)**
helper for pickle

**__repr__(x)**
\[\text{repr}(x)\]

**__setattr__(...)**
\[x.__setattr__('name', value) \Rightarrow x.name = value\]

**__str__(x)**
\[\text{str}(x)\]
11.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

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

```python
__init__(self, taskmaster, num, stack_size)
```
Create a new parallel job given a taskmaster.

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

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

Overrides: `object.__init__`

```python
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.

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

```python
x.__delattr__('name') == del x.name
```

```python
__getattr__(...)
```

```python
x.__getattr__('name') == x.name
```
11.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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:

```python
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:

```python
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:
        memo_dict = self._memo['bar']  # Memoization
    except KeyError:
        memo_dict = {}  # Memoization
        self._memo['dict'] = memo_dict
    else:
        try:
            return memo_dict[memo_key]  # Memoization
        except KeyError:
            pass  # Memoization
    result = self.compute_bar_value(argument)
```

147
Functions

```python
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

```python
Dump(title=False)
```
EnableMemoization()

12.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Memoize.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
<tr>
<td><strong>doc</strong></td>
<td>Value: &quot;&quot;&quot;&quot;Memoi...</td>
</tr>
<tr>
<td>use_memoizer</td>
<td>Value: False</td>
</tr>
<tr>
<td>CounterList</td>
<td>Value: []</td>
</tr>
</tbody>
</table>

12.3 Class Counter

object

SCons.Memoize.Counter

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

Base class for counting memoization hits and misses.

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

12.3.1 Methods

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

Overrides: object.__init__

```python
display(self)
```

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

```python
__delattr__(...)  
x.__delattr__("name") <== del x.name
```

```python
__getattr__(...)  
x.__getattr__("name") <== x.name
```

```python
__hash__(x)
```

hash(x)
12.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

12.4 Class 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

```python
Callable methods:

_call__(self, *args, **kw)

_cmp__(self, other)

delattr__(...)

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

_getattribute__(...)

x.__getattribute__('name') == x.name

__hash__(x)

hash(x)

_init__(self, method_name)

Overrides: object.__init__

_new__(T, S, ...)

Return Value

a new object with type S, a subtype of T

_reduce__(...)

default helper for pickle

_reduce_ex__(...)

default helper for pickle

_repr__(x)

repr(x)

_setattr__(...)

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

_str__(x)

str(x)

display(self)
```
12.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_class_</code></td>
<td>Value: <code>&lt;attribute '_class_' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

12.5 Class 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

```python
def __init__(self, method_name, keymaker):
    overrides: SCons.Memoize.Counter.__init__

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

def __cmp__(self, other):

def __delattr__(self, other):
    x.__delattr__('name') <-> del x.name

def __getattribute__(self, other):
    x.__getattribute__('name') <-> x.name

def __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)

display(self)

### 12.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

### 12.6 Class Memoizer

object ▼

SCons.Memoize.Memoizer

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

### 12.6.1 Methods

_init__(self)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)
Class Memoized_Metaclass

Module SCons.Memoize

```python
__delattr__(...)  
\x0a\x0a\x0ax._\x00\x00\x00__delattr__('name') \x3d\x0a\x00\x00\x00del x.name

__getattribute__(...)  
\x0a\x0a\x0ax._\x00\x00\x00__getattribute__('name') \x3d\x0a\x00\x00\x00x.name

__hash__(x)  
\x0a\x0a\x0ahash(x)

__new__(T, S, ...)  

Return Value
\x0a\x0a\x0aa new object with type S, a subtype of T

__reduce__(...)  

helper for pickle

__reduce_ex__(...)  

helper for pickle

__repr__(x)  
\x0a\x0a\x0arepr(x)

__setattr__(...)  
\x0a\x0a\x0ax._\x00\x00\x00__setattr__('name', value) \x3d\x0a\x00\x00\x00x.name = value

__str__(x)  
\x0a\x0a\x0astr(x)

12.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

12.7 Class Memoized_Metaclass

object

    type

SCons.Memoize.Memoized_Metaclass
12.7.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__(cls, name, bases, cls_dict)</code></td>
<td><code>x.__init__(...)</code> initializes <code>x</code>; see <code>x.__class__.__doc__</code> for signature</td>
</tr>
<tr>
<td><code>__call__(...)</code></td>
<td><code>x(...)</code></td>
</tr>
<tr>
<td><code>__cmp__(x, y)</code></td>
<td><code>cmp(x,y)</code></td>
</tr>
<tr>
<td><code>__delattr__(...)</code></td>
<td><code>x.__delattr__('name')</code> &lt;-&gt; del <code>x.name</code></td>
</tr>
<tr>
<td><code>__getattribute__(...)</code></td>
<td><code>x.__getattribute__('name')</code> &lt;-&gt; <code>x.name</code></td>
</tr>
<tr>
<td><code>__hash__(x)</code></td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><code>__new__(T, S, ...)</code></td>
<td>Return Value&lt;br&gt;a new object with type <code>S</code>, a subtype of <code>T</code></td>
</tr>
<tr>
<td><code>__reduce__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__repr__(x)</code></td>
<td><code>repr(x)</code></td>
</tr>
</tbody>
</table>

Overrides for methods:
- `object.__init__`
- `object.__call__`
- `object.__cmp__`
- `object.__delattr__`
- `object.__getattribute__`
- `object.__hash__`
- `object.__new__`
- `object.__reduce__`
- `object.__reduce_ex__`
- `object.__repr__`
Class Memoized

__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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>base</strong></td>
<td>Value: &lt;member '<strong>base</strong>' of 'type' objects&gt;</td>
</tr>
<tr>
<td><strong>bases</strong></td>
<td>Value: &lt;attribute '<strong>bases</strong>' of 'type' objects&gt;</td>
</tr>
<tr>
<td><strong>basicsize</strong></td>
<td>Value: &lt;member '<strong>basicsize</strong>' of 'type' objects&gt;</td>
</tr>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td><strong>dictoffset</strong></td>
<td>Value: &lt;member '<strong>dictoffset</strong>' of 'type' objects&gt;</td>
</tr>
<tr>
<td><strong>flags</strong></td>
<td>Value: &lt;member '<strong>flags</strong>' of 'type' objects&gt;</td>
</tr>
<tr>
<td><strong>itemsize</strong></td>
<td>Value: &lt;member '<strong>itemsize</strong>' of 'type' objects&gt;</td>
</tr>
<tr>
<td><strong>mro</strong></td>
<td>Value: &lt;member '<strong>mro</strong>' of 'type' objects&gt;</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Value: &lt;attribute '<strong>name</strong>' of 'type' objects&gt;</td>
</tr>
<tr>
<td><strong>weakrefoffset</strong></td>
<td>Value: &lt;member '<strong>weakrefoffset</strong>' of 'type' objects&gt;</td>
</tr>
</tbody>
</table>
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. 160)*

- **FS**: scons.Node.FS
  
  File system nodes.  
  
  *(Section 15, p. 177)*

- **Python**: scons.Node.Python
  
  Python nodes.  
  
  *(Section 16, p. 273)*

13.2 Functions

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>classname(obj)</code></td>
</tr>
<tr>
<td><code>Annotate(node)</code></td>
</tr>
<tr>
<td><code>get_children(node, parent)</code></td>
</tr>
<tr>
<td><code>ignore_cycle(node, stack)</code></td>
</tr>
<tr>
<td><code>do_nothing(node, parent)</code></td>
</tr>
</tbody>
</table>

13.3 Variables
### Class NodeInfoBase

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

```
_delattr_(...)  
x._delattr_('name')  ==>  del x.name

_getattribute_(...)  
x._getattribute_('name')  ==>  x.name

_hash_(x)  
hash(x)

_init_(self, node=False)  
x._init_(...)  initializes x;  see x._class_.doc_  for signature
Overrides: object._init_.exit(inherited documentation)
```
### Class NodeInfoBase

**Return Value**

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

---

### 13.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: &lt;attribute '.<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

### 13.4.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
</tbody>
</table>
13.5 Class BuildInfoBase

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

```python
__delattr__(...)  
x.__delattr__('name') == del x.name

getattr__(...)  
x.__getattr__('name') == x.name

__hash__(x)  
hash(x)

__init__(self, node=False)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
 Overrides: object.__init__ (inherited documentation)

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

__reduce__(...)  
helper for pickle

__reduce_ex__(...)  
helper for pickle
```
Class Node

Package SCons.Node

```
__repr__(x)
repr(x)

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

__str__(x)
str(x)

merge(self, other)
```

13.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

13.5.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
</tbody>
</table>

13.6 Class Node

```
object

SCons.Node.Node
```


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

13.6.1 Methods

```
Decider(self, function)

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

__getattribute__(...)
x.__getattribute__('name') ==> x.name
```
### Class Node

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>hash</strong></td>
<td>hash(x)</td>
</tr>
</tbody>
</table>
| __init__ | \(x\) initializes \(x\); see \(x\).__class__.doc_ for signature
| Overrides: object.__init__ (inherited documentation) |
| 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 | repr(x) |
| setattr | \(x\).setattr_('name', value) \iff x.name = value |
| str | str(x) |

#### Methods

- **add_dependency**
  - `add_dependency(self, depend)`
  - Adds dependencies.

- **add_ignore**
  - `add_ignore(self, depend)`
  - Adds dependencies to ignore.

- **add_prerequisite**
  - `add_prerequisite(self, prerequisite)`
  - Adds prerequisites

- **add_source**
  - `add_source(self, source)`
  - Adds sources.

- **add_to_implicit**
  - `add_to_implicit(self, deps)`
### Class Node

#### add_to_waiting_parents(self, node)

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

#### add_to_waiting_s_e(self, node)

#### add_wkid(self, wkid)

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

#### all_children(self, scan=1)

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

#### alter_targets(self)

Return a list of alternate targets for this Node.

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

Actually build the node.

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

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

#### builder_set(self, builder)

#### built(self)

Called just after this node is successfully built.
### changed

```
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

```
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

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

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

### children_are_up_to_date

```
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.

### 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_info(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.
Class Node

```
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=1)

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

```
get_foundIncludes(self, env, scanner, path)

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

The default is no implicit dependencies. We expect this method
to be overridden by any subclass that can be scanned for
implicit dependencies.
```
get_implicit_deps(self, env, scanner, path)

Return a list of implicit dependencies for this node.

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

get_ninfo(self)

get_source_scanner(self, node)

Fetch the source scanner for the specified node

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

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

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

get_state(self)

get_stored_implicit(self)

Fetch the stored implicit dependencies

get_stored_info(self)

get_string(self, for_signature)

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

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

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

### get_suffix(self)

### get_target_scanner(self)

### has_builder(self)

Return whether this Node has a builder or not.

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

### has_explicit_builder(self)

Return whether this Node has an explicit builder.

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

### is_derived(self)

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

This should return true only for nodes whose path should be in the variant directory when duplicate=0 and should contribute their build signatures when they are used as source files to other derived files. For example: source with source builders are not derived in this sense, and hence should not return true.
### `is_literal(self)`
Always pass the string representation of a Node to the command interpreter literally.

### `is_up_to_date(self)`
Default check for whether the Node is current: unknown Node subtypes are always out of date, so they will always get built.

### `make_ready(self)`
Get a Node ready for evaluation.

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

### `missing(self)`

### `multiple_side_effect_has_builder(self)`
Return whether this Node has a builder or not.

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

### `new_binfo(self)`

### `new_ninfo(self)`

### `postprocess(self)`
Clean up anything we don’t need to hang onto after we’ve been built.
**prepare(self)**

Prepare for this Node to be built.

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

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

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

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

**push_to_cache(self)**

Try to push a node into a cache

**remove(self)**

Remove this Node: no-op by default.

**render_include_tree(self)**

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

**reset_executor(self)**

Remove cached executor; forces recompute when needed.

**retrieve_from_cache(self)**

Try to retrieve the node’s content from a cache

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

Returns true iff the node was successfully retrieved.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rexists(self)</code></td>
<td>Does this node exist locally or in a repository?</td>
</tr>
<tr>
<td><code>scan(self)</code></td>
<td>Scan this node's dependents for implicit dependencies.</td>
</tr>
<tr>
<td><code>scanner_key(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>select_scanner(self, scanner)</code></td>
<td>Selects a scanner for this Node.</td>
</tr>
<tr>
<td></td>
<td>This is a separate method so it can be overridden by Node subclasses (specifically, Node.FS.Dir) that <em>must</em> use their own Scanner and don’t select one the Scanner.Selector that’s configured for the target.</td>
</tr>
<tr>
<td><code>set_always_build(self, always_build=1)</code></td>
<td>Set the Node’s always_build value.</td>
</tr>
<tr>
<td><code>set_executor(self, executor)</code></td>
<td>Set the action executor for this node.</td>
</tr>
<tr>
<td><code>set_explicit(self, is_explicit)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_nocache(self, nocache=1)</code></td>
<td>Set the Node’s nocache value.</td>
</tr>
<tr>
<td><code>set_noctime(self, noctime=1)</code></td>
<td>Set the Node’s noctime value.</td>
</tr>
<tr>
<td><code>set_precious(self, precious=1)</code></td>
<td>Set the Node’s precious value.</td>
</tr>
<tr>
<td><code>set_specific_source(self, source)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_state(self, state)</code></td>
<td></td>
</tr>
<tr>
<td><code>state_has_changed(self, target, prev_ni)</code></td>
<td>Make the build signature permanent (that is, store it in the .sconsign file or equivalent).</td>
</tr>
</tbody>
</table>
visited(self)

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

### 13.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

### 13.6.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: SCons.Memoize.Memoized_Metaclass</td>
</tr>
<tr>
<td>memoizer_counts</td>
<td>Value: []</td>
</tr>
</tbody>
</table>

### 13.7 Class 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)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>iadd</strong></td>
<td>Add another list to this list</td>
</tr>
<tr>
<td><strong>imul</strong></td>
<td>Multiply this list by another list</td>
</tr>
<tr>
<td><strong>init</strong></td>
<td>Initialize the list with an optional list</td>
</tr>
<tr>
<td><strong>le</strong></td>
<td>Check if this list is less than or equal to another list</td>
</tr>
<tr>
<td><strong>len</strong></td>
<td>Get the length of the list</td>
</tr>
<tr>
<td><strong>lt</strong></td>
<td>Check if this list is less than another list</td>
</tr>
<tr>
<td><strong>mul</strong></td>
<td>Multiply this list by another list</td>
</tr>
<tr>
<td><strong>ne</strong></td>
<td>Check if this list is not equal to another list</td>
</tr>
<tr>
<td><strong>radd</strong></td>
<td>Add another list to this list</td>
</tr>
<tr>
<td><strong>repr</strong></td>
<td>Get the string representation of the list</td>
</tr>
<tr>
<td><strong>rmul</strong></td>
<td>Multiply this list by another list</td>
</tr>
<tr>
<td><strong>setitem</strong></td>
<td>Set the item at the given index</td>
</tr>
<tr>
<td><strong>setslice</strong></td>
<td>Set the slice of the list</td>
</tr>
<tr>
<td>append</td>
<td>Add an item to the list</td>
</tr>
<tr>
<td>count</td>
<td>Count the occurrences of an item</td>
</tr>
<tr>
<td>extend</td>
<td>Extend the list with another list</td>
</tr>
<tr>
<td>index</td>
<td>Get the index of an item</td>
</tr>
<tr>
<td>insert</td>
<td>Insert an item at the given index</td>
</tr>
<tr>
<td>pop</td>
<td>Remove and return the last item</td>
</tr>
<tr>
<td>remove</td>
<td>Remove an item from the list</td>
</tr>
<tr>
<td>reverse</td>
<td>Reverse the order of the items</td>
</tr>
<tr>
<td>sort</td>
<td>Sort the items</td>
</tr>
</tbody>
</table>
13.8 Class Walker

object

SCons.Node.Walker

An iterator for walking a Node tree.

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

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

13.8.1 Methods

```python
__init__(self, node, kids_func=<function get_children at 0x849502c>, cycle_func=<function ignore_cycle at 0x84964c4>, eval_func=<function do_nothing at 0x84964fc>)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__ (inherited documentation)
```

get_next(self)

Return the next node for this walk of the tree.

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

is_done(self)

```python
__delattr__(...)  
x.__delattr__('name') == del x.name
```

getattribute(...)

```python
x.__getattribute__('name') == x.name
```

_hash_(x)

```python
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)

13.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
14 Module SCons.Node.Alias

scons.Node.Alias

Alias nodes.

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

14.1 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Node/Alias.py 5023 2010/06/14 22:05:46 ...</td>
</tr>
<tr>
<td>default_ans</td>
<td>Value: {}</td>
</tr>
</tbody>
</table>

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)
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

```python
str_to_node(self, s)
```
__delattr__(...)  
x.__delattr__('name') ==> del x.name

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

__hash__(x)  
hash(x)

__init__(self, node=False)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: object.__init__ (inherited documentation)

__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)

convert(self, node, val)

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

merge(self, other)

update(self, node)
14.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

14.3.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
<tr>
<td>field_list</td>
<td>Value: ['csig']</td>
</tr>
</tbody>
</table>

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

```
def __delattr__(...)  
x.__delattr__('name') == del x.name

def __getattribute__(...)  
x.__getattribute__('name') == x.name

def __hash__(x)  
hash(x)

def __init__(self, node=False)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
```

Overrides: object.__init__ (inherited documentation)
__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)

merge(self, other)

14.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

14.4.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
</tbody>
</table>

14.5 Class Alias

```
object
SCons.Node.Node
```
### 14.5.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__(self, name)</code></td>
<td>Overrides: SCons.Node.Node.<strong>init</strong>.</td>
</tr>
<tr>
<td><code>str_for_display(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>__str__(self)</code></td>
<td>str(x)</td>
</tr>
<tr>
<td><code>make_ready(self)</code></td>
<td>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 (inherited documentation)</td>
</tr>
<tr>
<td><code>really_build(self, **kw)</code></td>
<td>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().</td>
</tr>
<tr>
<td><code>is_up_to_date(self)</code></td>
<td>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</td>
</tr>
<tr>
<td><code>is_under(self, dir)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_contents(self)</code></td>
<td>The contents of an alias is the concatenation of the content signatures of all its sources.</td>
</tr>
</tbody>
</table>
**Class Alias Module**

### `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 (inherited documentation)

### `build(self)`

A "builder" for aliases.


### `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


### `Decider(self, function)`

### `__delattr__(...)`

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

### `__getattr__(...)`

`x.__getattr__('name')` <=> `x.name`
### Class Alias

**Module** SCons.Node.Alias

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__hash__(x)</code></td>
<td>hash(x)</td>
</tr>
<tr>
<td><code>__new__(T, S, ...)</code></td>
<td>Return Value: a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>__reduce__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__repr__(x)</code></td>
<td>repr(x)</td>
</tr>
<tr>
<td><code>__setattr__(...)</code></td>
<td>x.<strong>setattr</strong>('name', value) == x.name = value</td>
</tr>
</tbody>
</table>

**Methods**

- `add_dependency(self, depend)`
  Adds dependencies.

- `add_ignore(self, depend)`
  Adds dependencies to ignore.

- `add_prerequisite(self, prerequisite)`
  Adds prerequisites.

- `add_source(self, source)`
  Adds sources.

- `add_to_implicit(self, deps)`
**add_to_waiting_parents**(self, node)

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

**add_to_waiting_s_e**(self, node)

**add_wkid**(self, wkid)

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

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

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

**alter_targets**(self)

Return a list of alternate targets for this Node.

**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=1)

Return a list of the node’s direct children, minus those
that are ignored by this node.
**Class Alias**

**Module SCons.Node.Alias**

---

**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.


---

**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(self)**

**get_env(self)**

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

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

Fetch the action executor for this node. Create one if there isn’t already one, and requested to do so.
get_found_includes(*self, env, scanner, path*)

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

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

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

Return a list of implicit dependencies for this node.

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

get_ninfo(*self*)

get_source_scanner(*self, node*)

Fetch the source scanner for the specified node

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

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

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

get_state(*self*)

get_stored_implicit(*self*)

Fetch the stored implicit dependencies

get_stored_info(*self*)
Class Alias Module SCons.Node.Alias

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.
Class Alias

Module SCons.Node.Alias

```python
def 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).
```

```python
def 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.
```

```python
def is_literal(self):
    """Always pass the string representation of a Node to
    the command interpreter literally.
```

```python
def missing(self):
```

```python
def 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.
```

```python
def new_binfo(self):
```

```python
def new_ninfo(self):
```

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

push_to_cache(self)

Try to push a node into a cache

remove(self)

Remove this Node: no-op by default.

render_include_tree(self)

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

reset_executor(self)

Remove cached executor; forces recompute when needed.

retrieve_from_cache(self)

Try to retrieve the node’s content from a cache

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

Returns true iff the node was successfully retrieved.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rexists(self)</code></td>
<td>Does this node exist locally or in a repository?</td>
</tr>
<tr>
<td><code>scan(self)</code></td>
<td>Scan this node's dependents for implicit dependencies.</td>
</tr>
<tr>
<td><code>scanner_key(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>select_scanner(self, scanner)</code></td>
<td>Selects a scanner for this Node.</td>
</tr>
<tr>
<td></td>
<td>This is a separate method so it can be overridden by Node subclasses (specifically, Node.FS.Dir) that <em>must</em> use their own Scanner and don't select one the Scanner.Selector that's configured for the target.</td>
</tr>
<tr>
<td><code>set_always_build(self, always_build=1)</code></td>
<td>Set the Node’s always_build value.</td>
</tr>
<tr>
<td><code>set_executor(self, executor)</code></td>
<td>Set the action executor for this node.</td>
</tr>
<tr>
<td><code>set_explicit(self, is_explicit)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_nocache(self, nocache=1)</code></td>
<td>Set the Node’s nocache value.</td>
</tr>
<tr>
<td><code>set_noclean(self, noclean=1)</code></td>
<td>Set the Node’s noclean value.</td>
</tr>
<tr>
<td><code>set_precious(self, precious=1)</code></td>
<td>Set the Node’s precious value.</td>
</tr>
<tr>
<td><code>set_specific_source(self, source)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_state(self, state)</code></td>
<td></td>
</tr>
<tr>
<td><code>state_has_changed(self, target, prev ni)</code></td>
<td></td>
</tr>
<tr>
<td><code>store_info(self)</code></td>
<td>Make the build signature permanent (that is, store it in the .sconsign file or equivalent).</td>
</tr>
</tbody>
</table>
visited(self)

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

### 14.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_class_</code></td>
<td>Value: <code>&lt;attribute </code>.<em>class</em><code>of</code>object<code> objects&gt;</code></td>
</tr>
</tbody>
</table>

### 14.5.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__metaclass__</code></td>
<td>Value: <code>SCons.Memoize.Memoized_Metacls</code></td>
</tr>
<tr>
<td><code>memoizer_counters</code></td>
<td>Value: <code>[]</code></td>
</tr>
</tbody>
</table>
Module SCons.Node.FS

15 Module SCons.Node.FS

scons.Node.FS

File system nodes.

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

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

15.1 Functions

save_strings(val)

initialize_do_splitdrive()

initialize_normpath_check()

Initialize the normpath_check regular expression.

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

set_duplicate(duplicate)

LinkFunc(target, source, env)

LocalString(target, source, env)

UnlinkFunc(target, source, env)

MkdirFunc(target, source, env)

get_MkdirBuilder()

get_DefaultSCCSBuilder()

get_DefaultRCSBuilder()

do_diskcheck_match(node, predicate, errorfmt)

ignore_diskcheck_match(node, predicate, errorfmt)

do_diskcheck_rcs(node, name)
ignore_diskcheck_rcs(node, name)

do_diskcheck_sccs(node, name)

ignore_diskcheck_sccs(node, name)

set_diskcheck(list)

diskcheck_types()

has_glob_magic(s)

get_default_fs()

find_file(filename, paths, verbose=False)

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

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

returns - the node created from the found file.

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

Only the first file found is returned, and none is returned if no file is found.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.revision</td>
<td>Value: 'src/engine/SCons/Node/FS.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
</tbody>
</table>

continued on next page
### Class EntryProxyAttributeError

object

exceptions.BaseException

    exceptions.Exception

    exceptions.StandardError

    exceptions.AttributeError

    SCons.Node.FS.EntryProxyAttributeError

An AttributeError subclass for recording and displaying the name of the underlying Entry involved in an AttributeError exception.

#### 15.3.1 Methods

```python
__init__(self, entry_proxy, attribute)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: exceptions.AttributeError.__init__ (inherited documentation)
```
Class EntryProxyAttributeError

Overrides: exceptions.BaseException

str '(self)
str(x)

Overrides: exceptions.BaseException

_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)

__new__ (T, S, ...)

Return Value

a new object with type S, a subtype of T

Overrides: exceptions.StandardError

__reduce__ (...

helper for pickle

Overrides: object._reduce__

__reduce_ex__ (...

helper for pickle

__repr__ (x)

repr(x)

Overrides: object._repr__
15.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>

15.4 Class DiskChecker

[Diagram]

15.4.1 Methods

```python
__init__(self, type, do, ignore)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
overrides: object.__init__(inherited documentation)
```

```python
__call__(self, *args, **kw)
set(self, list)
```

```python
__delattr__(...)  
x.__delattr__('name') == del x.name
```

```python
__getattribute__(...)  
x.__getattribute__('name') == x.name
```

```python
__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)

15.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute ‘<strong>class</strong>’ of ‘object’ objects&gt;</td>
</tr>
</tbody>
</table>

15.5 Class EntryProxy

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

15.5.1 Methods

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

class Foo(Proxy):
    __str__ = Delegate(’__str__’)
Overrides: object.__str__
Class EntryProxy

Module SCons.Node.FS

```python
__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__ (inherited documentation)

__cmp__(self, other)

__delattr__(...
```delattr_('name') <-> del x.name

```python
__getattribute__(...
```getattribute_('name') <-> x.name

__hash__(x)
```hash(x)
```

```python
__init__(self, subject)
Wrap an object as a Proxy object
Overrides: object.__init__
```

```python
__new__(T, S, ...)  
Return Value  
a new object with type S, a subtype of T
```

```python
__reduce__(...
helper for pickle
```

```python
__reduce_ex__(...
helper for pickle
```

```python
__repr__(x)
repr(x)
```

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

```python
get(self)
Retrieve the entire wrapped object
```
15.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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)

rfile(self)

__str__(self)
A Node.FS.Base object’s string representation is its path
name.
Overrides: object.__str__

rstr(self)
A Node.FS.Base object’s string representation is its path
name.

stat(self)

exists(self)
Does this node exists?

reexists(self)
Does this node exist locally or in a repository?

gmtime(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.


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_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 0x8415c34>)

Generates a target entry that corresponds to this entry (usually a source file) with the specified prefix and suffix.

Note that this method can be overridden dynamically for generated files that need different behavior. See Tool/swig.py for an example.

Rfindalldirs(self, pathlist)

Return all of the directories for a given path list, including corresponding "backing" directories in any repositories.

The Node lookups are relative to this Node (typically a directory), so memoizing result saves cycles from looking up the same path for each target in a given directory.

RDirs(self, pathlist)

Search for a list of directories in the Repository list.

rentry(self)

Decider(self, function)

\_delattr\_(...)

x\_delattr\_('name') \leftrightarrow del x.name

\_getattr\_(...)  

x\_getattr\_('name') \leftrightarrow x.name
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__hash__(x)</code></td>
<td>Returns the hash of <code>x</code>.</td>
</tr>
<tr>
<td><code>hash(x)</code></td>
<td></td>
</tr>
<tr>
<td><code>__new__(T, S, ...)</code></td>
<td>Creates a new object with type <code>S</code>, a subtype of <code>T</code>.</td>
</tr>
<tr>
<td><code>__reduce__(...)</code></td>
<td>Helper for pickle.</td>
</tr>
<tr>
<td><code>__reduce_ex__(...)</code></td>
<td>Helper for pickle.</td>
</tr>
<tr>
<td><code>__repr__(x)</code></td>
<td>Returns the string representation of <code>x</code>.</td>
</tr>
<tr>
<td><code>setattr(...)</code></td>
<td>Sets the attribute of <code>x</code>.</td>
</tr>
<tr>
<td><code>xsetattr_\('name', value)\</code></td>
<td>Assigns <code>value</code> to the <code>name</code> attribute of <code>x</code>.</td>
</tr>
<tr>
<td><code>add_dependency(self, depend)</code></td>
<td>Adds dependencies.</td>
</tr>
<tr>
<td><code>add_ignore(self, depend)</code></td>
<td>Adds dependencies to ignore.</td>
</tr>
<tr>
<td><code>add_prerequisite(self, prerequisite)</code></td>
<td>Adds prerequisites.</td>
</tr>
<tr>
<td><code>add_source(self, source)</code></td>
<td>Adds sources.</td>
</tr>
<tr>
<td><code>add_to_implicit(self, deps)</code></td>
<td>Adds dependencies to implicit.</td>
</tr>
</tbody>
</table>
add_to_waiting_parents(self, node)

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

add_to_waiting_s_e(self, node)

add_wkid(self, wkid)
Add a node to the list of kids waiting to be evaluated

all_children(self, scan=1)
Return a list of all the node’s direct children.

alter_targets(self)
Return a list of alternate targets for this Node.

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

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

builder_set(self, builder)

built(self)
Called just after this node is successfully built.
changed(self, node=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=1)

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

children_are_up_to_date(self)

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

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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_cachedir_csig(self)</td>
<td></td>
</tr>
<tr>
<td>get_csig(self)</td>
<td></td>
</tr>
<tr>
<td>get_env(self)</td>
<td></td>
</tr>
<tr>
<td>get_env_scanner(self, env, kw={})</td>
<td></td>
</tr>
<tr>
<td>get_executor(self, create=1)</td>
<td>Fetch the action executor for this node. Create one if there isn’t already one, and requested to do so.</td>
</tr>
<tr>
<td>get_found_includes(self, env, scanner, path)</td>
<td>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.</td>
</tr>
<tr>
<td>get_implicit_deps(self, env, scanner, path)</td>
<td>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.</td>
</tr>
<tr>
<td>get_ninfo(self)</td>
<td></td>
</tr>
<tr>
<td>get_source_scanner(self, node)</td>
<td>Fetch the source scanner for the specified node.</td>
</tr>
<tr>
<td></td>
<td>NOTE: &quot;self&quot; is the target being built, &quot;node&quot; 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.</td>
</tr>
<tr>
<td></td>
<td>This function may be called very often; it attempts to cache the scanner found to improve performance.</td>
</tr>
<tr>
<td>get_state(self)</td>
<td></td>
</tr>
<tr>
<td>get_stored_implicit(self)</td>
<td>Fetch the stored implicit dependencies</td>
</tr>
</tbody>
</table>
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.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>prepare(self)</code></td>
<td>Prepare for this Node to be built. This is called after the Taskmaster has</td>
</tr>
<tr>
<td></td>
<td>decided that the Node is out-of-date and must be rebuilt, but before actually</td>
</tr>
<tr>
<td></td>
<td>calling the method to build the Node. This default implementation checks</td>
</tr>
<tr>
<td></td>
<td>that explicit or implicit dependencies either exist or are derived, and</td>
</tr>
<tr>
<td></td>
<td>initializes the BuildInfo structure that will hold the information about</td>
</tr>
<tr>
<td></td>
<td>how this node is, uh, built. (The existence of source files is checked</td>
</tr>
<tr>
<td></td>
<td>separately by the Executor, which aggregates checks for all of the targets</td>
</tr>
<tr>
<td></td>
<td>built by a specific action.) Overriding this method allows for a Node</td>
</tr>
<tr>
<td></td>
<td>subclass to remove the underlying file from the file system. Note that</td>
</tr>
<tr>
<td></td>
<td>subclass methods should call this base class method to get the child check</td>
</tr>
<tr>
<td></td>
<td>and the BuildInfo structure.</td>
</tr>
<tr>
<td><code>push_to_cache(self)</code></td>
<td>Try to push a node into a cache</td>
</tr>
<tr>
<td><code>remove(self)</code></td>
<td>Remove this Node: no-op by default.</td>
</tr>
<tr>
<td><code>render_include_tree(self)</code></td>
<td>Return a text representation, suitable for displaying to the</td>
</tr>
<tr>
<td></td>
<td>user, of the include tree for the sources of this node.</td>
</tr>
<tr>
<td><code>reset_executor(self)</code></td>
<td>Remove cached executor; forces recompute when needed.</td>
</tr>
<tr>
<td><code>retrieve_from_cache(self)</code></td>
<td>Try to retrieve the node’s content from a cache</td>
</tr>
<tr>
<td></td>
<td>This method is called from multiple threads in a parallel build, so only</td>
</tr>
<tr>
<td></td>
<td>do thread safe stuff here. Do thread unsafe stuff in built().</td>
</tr>
<tr>
<td></td>
<td>Returns true iff the node was successfully retrieved.</td>
</tr>
</tbody>
</table>
**Class Base Module SCons.Node.FS**

### scan

```python
scan(self)
```

Scan this node’s dependents for implicit dependencies.

### scanner_key

```python
scanner_key(self)
```

### select_scanner

```python
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

```python
set_always_build(self, always_build=1)
```

Set the Node’s always_build value.

### set_executor

```python
set_executor(self, executor)
```

Set the action executor for this node.

### set_explicit

```python
set_explicit(self, is_explicit)
```

### set_nocache

```python
set_nocache(self, nocache=1)
```

Set the Node’s nocache value.

### set_noclean

```python
set_noclean(self, noclean=1)
```

Set the Node’s noclean value.

### set_precious

```python
set_precious(self, precious=1)
```

Set the Node’s precious value.

### set_specific_source

```python
set_specific_source(self, source)
```

### set_state

```python
set_state(self, state)
```

### state_has_changed

```python
state_has_changed(self, target, prev_mtime)
```

### store_info

```python
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).

15.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

15.6.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoizer_counters</td>
<td>Value: <code>[]</code></td>
</tr>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: <code>SCons.Memoize.MemoizedMeta</code></td>
</tr>
</tbody>
</table>

15.7 Class Entry

object

SCons.Node.Node

SCons.Node.FS.Base

SCons.Node.FS.Entry

This is the class for generic Node.FS entries—that is, things that could be a File or a Dir, but we’re just not sure yet. Consequently, the methods in this class really exist just to transform their associated object into the right class when the time comes, and then call the same-named method in the transformed class.

15.7.1 Methods

diskcheck_match(self)

disambiguate(self, must_exist=False)


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
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_contents(self)</code></td>
<td>Fetch the contents of the entry. Returns the exact binary contents of the file.</td>
</tr>
<tr>
<td><code>get_text_contents(self)</code></td>
<td>Fetch the decoded text contents of a Unicode encoded Entry.</td>
</tr>
<tr>
<td><code>must_be_same(self, klass)</code></td>
<td>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</td>
</tr>
<tr>
<td><code>exists(self)</code></td>
<td>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</td>
</tr>
<tr>
<td><code>rel_path(self, other)</code></td>
<td></td>
</tr>
</tbody>
</table>
changed_since_last_build(self, target, prev_ni)

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

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

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

get_subst_proxy(self)

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

Overrides: SCons.Node.FS.Base.get_subst_proxy

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.
Class Entry

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

__getattribute__(...)  
x.__getattribute__('name') == x.name

__hash__(x)  
hash(x)

__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.


__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__(self)  
A Node.FS.Base object’s string representation is its path name.

Overrides: object.__str__
```
### add_dependency

```
def add_dependency(self, depend):
    Adds dependencies.
```

### add_ignore

```
def add_ignore(self, depend):
    Adds dependencies to ignore.
```

### add_prerequisite

```
def add_prerequisite(self, prerequisite):
    Adds prerequisites
```

### add_source

```
def add_source(self, source):
    Adds sources.
```

### add_to_implicit

```
def add_to_implicit(self, deps):
    Add a node to the list of kids waiting to be evaluated
```

### add_to_waiting_parents

```
def 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

```
def add_to_waiting_s_e(self, node):
    Add a node to the list of kids waiting to be evaluated
```

### add_wkid

```
def add_wkid(self, wkid):
    Add a node to the list of kids waiting to be evaluated
```

### all_children

```
def all_children(self, scan=1):
    Return a list of all the node's direct children.
```

### alter_targets

```
def 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=1)

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

### children_are_up_to_date(self)

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


### 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.

get_abspath(self)
Get the absolute path of the file.

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(self)

### get_csig(self)

### get_dir(self)

### get_env(self)

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

### get_executor(self, create=1)
Fetch the action executor for this node. Create one if there isn’t already one, and requested to do so.

### get_found_includes(self, env, scanner, path)
Return the scanned include lines (implicit dependencies) found in this node.

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

### get_implicit_deps(self, env, scanner, path)
Return a list of implicit dependencies for this node.

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

### get_ninfo(self)
get_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_suffix(self)


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 a Node subclass to remove the underlying file from the file system. Note that subclass methods should call this base class method to get the child check and the BuildInfo structure.
push_to_cache(self)
Try to push a node into a cache

remove(self)
Remove this Node: no-op by default.

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

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?

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=1)
Set the Node’s always build value.

set executor(self, executor)
Set the action executor for this node.

set explicit(self, is_explicit)

set local(self)

set nocache(self, nocache=1)
Set the Node’s nocache value.

set noclean(self, noclean=1)
Set the Node’s noclean value.

set precious(self, precious=1)
Set the Node’s precious value.

set specific source(self, source)

set 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 0x8415c34>)
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).

15.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

15.7.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: SCons.Memoize.MemoizedMetaClass</td>
</tr>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
</tbody>
</table>
15.8 Class LocalFS

object

SCons.Node.FS.LocalFS

Known Subclasses: SCons.Node.FS.FS

15.8.1 Methods

chmod(self, path, mode)

copy(self, src, dst)

copy2(self, src, dst)

exists(self, path)

getmtime(self, path)

getsize(self, path)

isdir(self, path)

isfile(self, path)

link(self, src, dst)

lstat(self, path)

listdir(self, path)

makedirs(self, path)

mkdir(self, path)

rename(self, old, new)

stat(self, path)

symlink(self, src, dst)

open(self, path)

unlink(self, path)

islink(self, path)
readlink(self, file)

_delattr_(__)

x._delattr_('_name') == del x._name

_getattribute_(__)

x._getattribute_('name') == x._name

_hash_(x)

hash(x)

_init__(...)

x._init__(...) initializes x; see x._class_.doc for signature

_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)

15.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
15.8.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

15.9 Class FS

object
SCons.Node.FS.LocalFS  
SCons.Node.FS.FS

15.9.1 Methods

```python
__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.
Overrides: object.__init__
```

```python
set_SConstruct_dir(self, dir)
```

```python
get_max_drift(self)
```

```python
set_max_drift(self, max_drift)
```

```python
getcwd(self)
```

```python
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.
```

```python
get_root(self, drive)
Returns the root directory for the specified drive, creating it if necessary.
```
Entry\((self, name, directory=False, create=1)\)

Look up or create a generic Entry node with the specified name. If the name is a relative path (begins with `.`, `../`, or a file name), then it is looked up relative to the supplied directory node, or to the top level directory of the FS (supplied at construction time) if no directory is supplied.

File\((self, name, directory=False, create=1)\)

Look up or create a File node with the specified name. If the name is a relative path (begins with `.`, `../`, or a file name), then it is looked up relative to the supplied directory node, or to the top level directory of the FS (supplied at construction time) if no directory is supplied.

This method will raise TypeError if a directory is found at the specified path.

Dir\((self, name, directory=False, create=True)\)

Look up or create a Dir node with the specified name. If the name is a relative path (begins with `.`, `../`, or a file name), then it is looked up relative to the supplied directory node, or to the top level directory of the FS (supplied at construction time) if no directory is supplied.

This method will raise TypeError if a normal file is found at the specified path.

VariantDir\((self, variant_dir, src_dir, duplicate=1)\)

Link the supplied variant directory to the source directory for purposes of building files.

Repository\((self,*dirs)\)

Specify Repository directories to search.

variant_dir_target_climb\((self, orig, dir, tail)\)

Create targets in corresponding variant directories

Climb the directory tree, and look up path names relative to any linked variant directories we find.

Even though this loops and walks up the tree, we don’t memoize the return value because this is really only used to process the command-line targets.
Glob(self, pathname, ondisk=True, source=True, strings=False, cwd=False)

Globs

This is mainly a shim layer

__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)

chmod(self, path, mode)

copy(self, src, dst)

copy2(self, src, dst)
exists(self, path)

gmtime(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)

## 15.9.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

## 15.9.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: SCons.Memoize.Memoized_Metaclass</td>
</tr>
</tbody>
</table>
## 15.10 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.

### 15.10.1 Methods

```plaintext
str_to_node(self, s)

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

__getattribute__(...)  
x.__getattribute__('name') == x.name

__hash__(x)  
hash(x)

__init__(self, node=False)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__ (inherited documentation)

__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)
```
Class DirBuildInfo

Module SCons.Node.FS

```python
setattr(x, 'name', value) == x.name = value

str(x)

convert(self, node, val)

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

merge(self, other)

update(self, node)
```

### 15.10.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

### 15.10.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
<tr>
<td><code>fs</code></td>
<td>Value: False</td>
</tr>
</tbody>
</table>

### 15.11 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.
## 15.11.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__delattr__(...)</code></td>
<td><code>x.__delattr__('name') &lt;=&gt; del x.name</code></td>
</tr>
<tr>
<td><code>__getattribute__(...)</code></td>
<td><code>x.__getattribute__('name') &lt;=&gt; x.name</code></td>
</tr>
<tr>
<td><code>__hash__(x)</code></td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><code>__init__(self, node=False)</code></td>
<td><code>x.__init__(...)</code> initializes x; see <code>x.__class__.__doc__</code> for signature</td>
</tr>
<tr>
<td><code>__new__(T, S, ...)</code></td>
<td>Return Value a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>__reduce__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__repr__(x)</code></td>
<td><code>repr(x)</code></td>
</tr>
<tr>
<td><code>__setattr__(...)</code></td>
<td><code>x.__setattr__('name', value) &lt;=&gt; x.name = value</code></td>
</tr>
<tr>
<td><code>__str__(x)</code></td>
<td><code>str(x)</code></td>
</tr>
</tbody>
</table>

## 15.11.2 Properties

*continued on next page*
15.11.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
</tbody>
</table>

15.12 Class Dir

A class for directories in a file system.

Known Subclasses: SCons.Node.FS.RootDir

15.12.1 Methods

```python
__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__ (inherited documentation)

```python
diskcheck_match(self)
```

```python
Entry(self, name)
```
Looks up or creates an entry node named 'name' relative to this directory.

```python
Dir(self, name, create=True)
```
Looks up or creates a directory node named 'name' relative to this directory.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>File(self, name)</code></td>
<td>Looks up or creates a file node named 'name' relative to this directory.</td>
</tr>
<tr>
<td><code>link(self, srcdir, duplicate)</code></td>
<td>Set this directory as the variant directory for the supplied source directory.</td>
</tr>
<tr>
<td><code>getRepositories(self)</code></td>
<td>Returns a list of repositories for this directory.</td>
</tr>
<tr>
<td><code>get_all_rdirs(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>addRepository(self, dir)</code></td>
<td></td>
</tr>
<tr>
<td><code>up(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>rel_path(self, other)</code></td>
<td>Return a path to &quot;other&quot; relative to this directory.</td>
</tr>
<tr>
<td><code>get_found_includes(self, env, scanner, path)</code></td>
<td>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</td>
</tr>
</tbody>
</table>
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 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 (inherited documentation)

build(self, **kw)
A null "builder" for directories.


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 (inherited documentation)

alter_targets(self)
Return any corresponding targets in a variant directory.


scanner_key(self)
A directory does not get scanned.

### get_text_contents(self)

We already emit things in text, so just return the binary version.

```python
get_text_contents(self)
```

### get_contents(self)

Return content signatures and names of all our children separated by new-lines. Ensure that the nodes are sorted.

```python
get_contents(self)
```

### 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.


```python
get_csig(self)
```

### do_duplicate(self, src)

### changed_since_last_build(self, target, prev ni)

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

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

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

```python
changed_since_last_build(self, target, prev ni)
```

### 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

```python
is_up_to_date(self)
```

### rdr(self)

```python
rdir(self)
```
### Class Dir

#### 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.

```python
x.__delattr__(‘name’) == del x.name
```

```python
x.__getattribute__(‘name’) == x.name
```

```python
hash(x)
```

```python
new(T, S, ...)
Return Value
    a new object with type S, a subtype of T
```

```python
reduce(...)
helper for pickle
```

```python
reduce_ex(...)
helper for pickle
```

```python
repr(x)
```

```python
setattr(x, ‘name’, value) == x.name = value
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_str_</code>(self)</td>
<td>A Node.FS.Base object’s string representation is its path name. Overrides: object.<em>str</em>.</td>
</tr>
<tr>
<td>add_dependency(self, depend)</td>
<td>Adds dependencies.</td>
</tr>
<tr>
<td>add_ignore(self, depend)</td>
<td>Adds dependencies to ignore.</td>
</tr>
<tr>
<td>add_prerequisite(self, prerequisite)</td>
<td>Adds prerequisites</td>
</tr>
<tr>
<td>add_source(self, source)</td>
<td>Adds sources.</td>
</tr>
<tr>
<td>add_to_implicit(self, deps)</td>
<td></td>
</tr>
<tr>
<td>add_to_waiting_parents(self, node)</td>
<td>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 &quot;clean up&quot; this function by using True and False instead...)</td>
</tr>
<tr>
<td>add_to_waiting_s_e(self, node)</td>
<td></td>
</tr>
<tr>
<td>add_wkid(self, wkid)</td>
<td>Add a node to the list of kids waiting to be evaluated</td>
</tr>
<tr>
<td>all_children(self, scan=1)</td>
<td>Return a list of all the node’s direct children.</td>
</tr>
<tr>
<td>builder_set(self, builder)</td>
<td></td>
</tr>
<tr>
<td>built(self)</td>
<td>Called just after this node is successfully built.</td>
</tr>
</tbody>
</table>
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=1)

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

children_are_up_to_date(self)

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


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 (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.

get_abspath(self)
Get the absolute path of the file.

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=1)
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 (inherited documentation)

### `get_suffix(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)

**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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new_binfo(self)</td>
<td></td>
</tr>
<tr>
<td>new_ninfo(self)</td>
<td></td>
</tr>
<tr>
<td>postprocess(self)</td>
<td>Clean up anything we don’t need to hang onto after we’ve been built.</td>
</tr>
<tr>
<td>push_to_cache(self)</td>
<td>Try to push a node into a cache</td>
</tr>
<tr>
<td>remove(self)</td>
<td>Remove this Node: no-op by default.</td>
</tr>
<tr>
<td>render_include_tree(self)</td>
<td>Return a text representation, suitable for displaying to the user, of the include tree for the sources of this node.</td>
</tr>
<tr>
<td>rentry(self)</td>
<td></td>
</tr>
<tr>
<td>reset_executor(self)</td>
<td>Remove cached executor; forces recompute when needed.</td>
</tr>
<tr>
<td>retrieve_from_cache(self)</td>
<td>Try to retrieve the node’s content from a cache</td>
</tr>
<tr>
<td></td>
<td>This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in built().</td>
</tr>
<tr>
<td></td>
<td>Returns true iff the node was successfully retrieved.</td>
</tr>
<tr>
<td>rexists(self)</td>
<td>Does this node exist locally or in a repository?</td>
</tr>
<tr>
<td>rfile(self)</td>
<td></td>
</tr>
<tr>
<td>rstr(self)</td>
<td>A Node.FS.Base object’s string representation is its path name.</td>
</tr>
</tbody>
</table>
### Class Dir

#### Module SCons.Node.FS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scan(self)</code></td>
<td>Scan this node’s dependents for implicit dependencies.</td>
</tr>
<tr>
<td><code>select_scanner(self, scanner)</code></td>
<td>Selects a scanner for this Node. This is a separate method so it can be overridden by Node subclasses (specifically, Node.FS.Dir) that <em>must</em> use their own Scanner and don't select one the Scanner.Selector that’s configured for the target.</td>
</tr>
<tr>
<td><code>set_always_build(self, always_build=1)</code></td>
<td>Set the Node’s always_build value.</td>
</tr>
<tr>
<td><code>set_executor(self, executor)</code></td>
<td>Set the action executor for this node.</td>
</tr>
<tr>
<td><code>set_explicit(self, is_explicit)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_local(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_nocache(self, nocache=1)</code></td>
<td>Set the Node’s nocache value.</td>
</tr>
<tr>
<td><code>set_noclean(self, noclean=1)</code></td>
<td>Set the Node’s noclean value.</td>
</tr>
<tr>
<td><code>set_precious(self, precious=1)</code></td>
<td>Set the Node’s precious value.</td>
</tr>
<tr>
<td><code>set_specific_source(self, source)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_src_builder(self, builder)</code></td>
<td>Set the source code builder for this node.</td>
</tr>
<tr>
<td><code>set_state(self, state)</code></td>
<td></td>
</tr>
</tbody>
</table>
.. currentmodule:: SCons.Node.FS

.. py:method:: 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).

.. py:method:: stat(self)

.. py:method:: state_has_changed(self, target, prev_mtime)

.. py:method:: store_info(self)

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

.. py:method:: str_for_display(self)

.. py:method:: target_from_source(self, prefix, suffix, splitext={!function splitext at 0x8415c34}):

   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.

.. py:method:: visited(self)

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

## 15.12.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

## 15.12.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: SCons.Memoize.Memoized_Metaclass</td>
</tr>
</tbody>
</table>
15.13 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.

15.13.1 Methods

_init__(self, name, fs)
Initialize a generic Node.FS.Base object.

Call the superclass initialization, take care of setting up our relative and absolute paths, identify our parent directory, and indicate that this node should use signatures.

Overrides: 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 extit(inherited documentation)

__str__(self)
A Node.FS.Base object’s string representation is its path name.

Overrides: SCons.Node.FS.Base.__str__ extit(inherited documentation)

element_abspath(self, name)
Overides: SCons.Node.FS.Dir.element_abspath

element_labspath(self, name)
Overides: SCons.Node.FS.Dir.element_labspath

element_path(self, name)
Overides: SCons.Node.FS.Dir.element_path
### Class RootDir

#### 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

#### 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.

```python
_x.__delattr__("name") == del x.name
```

```python
_x.__getattribute__("name") == x.name
```

```python
_x.__hash__(x)
```

```python
_x.__new__(T, S, ...)  
Return Value  
a new object with type S, a subtype of T
```

```python
_x.__reduce__(...)  
helper for pickle
```

```python
_x.__reduce_ex__(...)  
helper for pickle
```

```python
_x.__repr__(x)
```

```python
_x.__setattr__("name", value) == x.name = value
```

**addRepository**(self, dir)

**add_dependency**(self, depend)

Adds dependencies.
<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_ignore</td>
<td>(self, depend)</td>
<td>Adds dependencies to ignore.</td>
</tr>
<tr>
<td>add_prerequisite</td>
<td>(self, prerequisite)</td>
<td>Adds prerequisites</td>
</tr>
<tr>
<td>add_source</td>
<td>(self, source)</td>
<td>Adds sources.</td>
</tr>
<tr>
<td>add_to_implicit</td>
<td>(self, deps)</td>
<td></td>
</tr>
<tr>
<td>add_to_waiting_parents</td>
<td>(self, node)</td>
<td>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 &quot;clean up&quot; this function by using True and False instead...)</td>
</tr>
<tr>
<td>add_to_waiting_s_e</td>
<td>(self, node)</td>
<td></td>
</tr>
<tr>
<td>add_wkid</td>
<td>(self, wkid)</td>
<td>Add a node to the list of kids waiting to be evaluated</td>
</tr>
<tr>
<td>all_children</td>
<td>(self, scan=1)</td>
<td>Return a list of all the node’s direct children.</td>
</tr>
<tr>
<td>alter_targets</td>
<td>(self)</td>
<td>Return any corresponding targets in a variant directory.</td>
</tr>
<tr>
<td>build</td>
<td>(self, **kw)</td>
<td>A null &quot;builder&quot; for directories.</td>
</tr>
<tr>
<td>builder_set</td>
<td>(self, builder)</td>
<td></td>
</tr>
<tr>
<td>buildt</td>
<td>(self)</td>
<td>Called just after this node is successfully built.</td>
</tr>
</tbody>
</table>
changed\( (self, \text{node}=:False) \)

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\( (self, \text{target}, \text{prev}_n) \)

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\_n 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 (inherited documentation)

children\( (self, \text{scan}=1) \)

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

children\_are\_up\_to\_date\( (self) \)

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

### 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_info(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?


### 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.


getRepositories(self)

Returns a list of repositories for this directory.

get_abspath(self)

Get the absolute path of the file.


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_csigt(self)

### get_contents(self)

Return content signatures and names of all our children separated by new-lines. Ensure that the nodes are sorted.

### get_csigt(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.


### get_env(self)

### get_env_scantt(self, env, kw={})


### get_executor(self, create=1)

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

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

Return 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.


get suffix(self)


get target scanner(self)


get text contents(self)

We already emit things in text, so just return the binary version.

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

new_binfo(self)

new_ninfo(self)

postprocess(self)
Clean up anything we don’t need to hang onto after we’ve been built.

prepare(self)
Prepare for this Node to be built.

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

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

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

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


push_to_cache(self)
Try to push a node into a cache
### Class RootDir

#### `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?


#### `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

A directory does not get scanned.


#### sconsign

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

#### select_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

Set the Node's always_build value.

#### set_executor

Set the action executor for this node.

#### set_explicit

#### set_local

#### set_nocache

Set the Node’s nocache value.

#### set_noclean

Set the Node’s noclean value.

#### set_precious

Set the Node’s precious value.

#### set_specific_source

#### set_src_builder

Set the source code builder for this node.

#### set_state
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>srcdir_duplicate()</code></td>
<td></td>
</tr>
<tr>
<td><code>srcdir_find_file()</code></td>
<td></td>
</tr>
<tr>
<td><code>srcdir_list()</code></td>
<td></td>
</tr>
<tr>
<td><code>srcnode()</code></td>
<td>Dir has a special need for srcnode()...if we have a srcdir attribute set, then that <em>is</em> our srcnode.</td>
</tr>
<tr>
<td></td>
<td>Overrides: SCons.Node.FS.Base.srcnode</td>
</tr>
<tr>
<td><code>stat()</code></td>
<td></td>
</tr>
<tr>
<td><code>state_has_changed()</code></td>
<td></td>
</tr>
<tr>
<td><code>store_info()</code></td>
<td>Make the build signature permanent (that is, store it in the .sconsign file or equivalent).</td>
</tr>
<tr>
<td><code>str_for_display()</code></td>
<td></td>
</tr>
<tr>
<td><code>target_from_source()</code></td>
<td>Generates a target entry that corresponds to this entry (usually a source file) with the specified prefix and suffix.</td>
</tr>
<tr>
<td></td>
<td>Note that this method can be overridden dynamically for generated files that need different behavior. See Tool/swig.py for an example.</td>
</tr>
<tr>
<td><code>visited()</code></td>
<td>Called just after this node has been visited (with or without a build).</td>
</tr>
</tbody>
</table>
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).

15.13.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

15.13.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: SCons.Memoize.Memoized_Metaclass</td>
</tr>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
</tbody>
</table>

15.14 Class FileNodeInfo

Object diagram:

```
```

The generic base class for signature information for a Node.

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

15.14.1 Methods

str_to_node(self, s)
Class FileNodeInfo

_delattr__(...)  
x._delattr_('name')  <=>  del x.name

_getattribute__(...)  
x._getattribute_('name')  <=>  x.name

_hash__(x)  
hash(x)

_init__(self, node=False)  
x._init__(...) initializes x; see x._class__.__doc__ for signature  
Overrrides: object._init__ (inherited documentation)

_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)

convert(self, node, val)

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

merge(self, other)

update(self, node)
15.14.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: '&lt;attribute <strong>class</strong> of 'object' objects&gt;'</td>
</tr>
</tbody>
</table>

15.14.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
<tr>
<td>field_list</td>
<td>Value: ['csig', 'timestamp', 'size']</td>
</tr>
<tr>
<td>fs</td>
<td>Value: False</td>
</tr>
</tbody>
</table>

15.15 Class FileBuildInfo

```
oBJECT
SCons.Node.BuildInfoBase
SCons.Node.FS.FileBuildInfo
```

**Known Subclasses:** SCons.SConf.SConfBuildInfo

The generic base class for build information for a Node.

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

15.15.1 Methods

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

__getattribute__(...)  
x.__getattribute__('name') == x.name

__hash__(x)  
hash(x)

__init__(self, node=False)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: object.__init__.__exit__(inherited documentation)
```

270
Class FileBuildInfo

Module SCons.Node.FS

__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)

convert_from_sconsign(self, dir, name)

Converts a newly-read FileBuildInfo object for in-SCons use

For normal up-to-date checking, we don’t have any conversion to perform—but we’re leaving this method here to make that clear.

convert_to_sconsign(self)

Converts this FileBuildInfo object for writing to a .sconsign file

This replaces each Node in our various dependency lists with its usual string representation: relative to the top-level SConstruct directory, or an absolute path if it’s outside.

format(self, names=0)

merge(self, other)
prepare_dependencies(self)

Prepares a FileBuildInfo object for explaining what changed

The bsources, bdepends and bimplicit lists have all been
stored on disk as paths relative to the top-level SConstruct
directory. Convert the strings to actual Nodes (for use by the
--debug=explain code and --implicit-cache).

15.15.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

15.15.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

15.16 Class File

```
object
SCons.Node.Node
   SCons.Node.FS.File
```

A class for files in a file system.

15.16.1 Methods

diskcheck_match(self)

__init__(self, name, directory, fs)

Initialize a generic Node.FS.Base object.

Call the superclass initialization, take care of setting up
our relative and absolute paths, identify our parent
directory, and indicate that this node should use
signatures.

Overrides: SCons.Node.FS.Base.__init__(inherited documentation)
### Entry(self, name)
Create an entry node named 'name' relative to the directory of this file.

### Dir(self, name, create=True)
Create a directory node named 'name' relative to the directory of this file.

### Dirs(self, pathlist)
Create a list of directories relative to the SConscript directory of this file.

### File(self, name)
Create a file node named 'name' relative to the directory of this file.

### scanner_key(self)

### get_contents(self)

### get_text_contents(self)

### get_content_hash(self)
Compute and return the MD5 hash for this file.

### get_size(self)

### get_timestamp(self)

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

### convert_old_entry(self, old_entry)

### get_stored_info(self)

### get_stored_implicit(self)
Fetch the stored implicit dependencies
rel_path(self, other)

get_found_includes(self, env, scanner, path)
Return the included implicit dependencies in this file. Cache results so we only scan the file once per path regardless of how many times this information is requested.
Overrides: SCons.Node.Node.get_found_includes

push_to_cache(self)
Try to push the node into a cache

retrieve_from_cache(self)
Try to retrieve the node’s content from a cache
This method is called from multiple threads in a parallel build, so only do thread safe stuff here. Do thread unsafe stuff in built().
Returns true iff the node was successfully retrieved.
Overrides: SCons.Node.Node.retrieve_from_cache

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

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.
**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.


**prepare**(*self*)

Prepare for this file to be created.


**remove**(*self*)

Remove this file.

Overrides: SCons.Node.Node.remove

**do_duplicate**(*self, src*)

**exists**(*self*)

Does this node exist?

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


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


**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 (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 (inherited documentation)
### rfile

```python
rfile(self)
```

Overrides: `SCons.Node.FS.Base.rfile`

### rstr

```python
rstr(self)
```

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

Overrides: `SCons.Node.FS.Base.rstr` (inherited documentation)

### get_cachedir_csig

```python
def 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.


### get_cachedir_bsig

```python
def get_cachedir_bsig(self):
```

### Decider

```python
Decider(self, function)
```

### RDirs

```python
RDirs(self, pathlist)
```

Search for a list of directories in the Repository list.

### Rfindalldirs

```python
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.

### __delattr__

```python
__delattr__(...)
x.__delattr__("name") <=> del x.name
```

### __getattribute__

```python
__getattribute__(...)
x.__getattribute__("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__(self)
A Node.FS.Base object’s string representation is its path name.
Overrides: object.__str__

add_dependency(self, depend)
Adds dependencies.

add_ignore(self, depend)
Adds dependencies to ignore.

add_prerequisite(self, prerequisite)
Adds prerequisites

add_source(self, source)
Adds sources.

add_to_implicit(self, deps)
add_to_waiting_parents(self, node)

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

add_to_waiting_s_e(self, node)

add_wkid(self, wkid)
Add a node to the list of kids waiting to be evaluated

all_children(self, scan=1)
Return a list of all the node’s direct children.

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().

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=1)

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

children_are_up_to_date(self)

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


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.

### get_abspath(self)

Get the absolute path of the file.


### 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=1)

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.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_ninfo(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_path(self, dir=False)</code></td>
<td>Return path relative to the current working directory of the Node.FS.Base object that owns us.</td>
</tr>
<tr>
<td><code>get_source_scanner(self, node)</code></td>
<td>Fetch the source scanner for the specified node. NOTE: &quot;self&quot; is the target being built, &quot;node&quot; 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.</td>
</tr>
<tr>
<td><code>get_state(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>get_string(self, for_signature)</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>get_subst_proxy(self)</code></td>
<td>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 <em>getattr</em>() 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)</td>
</tr>
</tbody>
</table>
get_suffix(self)

get_target_scanner(self)

gmtime(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?
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>scan(self)</code></td>
<td>Scan this node’s dependents for implicit dependencies.</td>
</tr>
<tr>
<td><code>select_scaner(self, scanner)</code></td>
<td>Selects a scanner for this Node.</td>
</tr>
<tr>
<td></td>
<td>This is a separate method so it can be overridden by Node subclasses (specifically, Node.FS.Dir) that <em>must</em> use their own Scanner and don’t select one the Scanner.Selector that’s configured for the target.</td>
</tr>
<tr>
<td><code>set_always_build(self, always_build=1)</code></td>
<td>Set the Node’s always_build value.</td>
</tr>
<tr>
<td><code>set_executor(self, executor)</code></td>
<td>Set the action executor for this node.</td>
</tr>
<tr>
<td><code>set_explicit(self, is_explicit)</code></td>
<td>Set the Node’s explicit value.</td>
</tr>
<tr>
<td><code>set_local(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>set_nocache(self, nocache=1)</code></td>
<td>Set the Node’s nocache value.</td>
</tr>
<tr>
<td><code>set_noclean(self, noclean=1)</code></td>
<td>Set the Node’s noclean value.</td>
</tr>
<tr>
<td><code>set_precious(self, precious=1)</code></td>
<td>Set the Node’s precious value.</td>
</tr>
<tr>
<td><code>set_specific_source(self, source)</code></td>
<td>Set the specific source for this node.</td>
</tr>
<tr>
<td><code>set_src_builder(self, builder)</code></td>
<td>Set the source code builder for this node.</td>
</tr>
<tr>
<td><code>set_state(self, state)</code></td>
<td></td>
</tr>
</tbody>
</table>
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).

csrcnode(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 0x8415c34>)

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.

15.16.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

15.16.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
<tr>
<td>md5_chunksize</td>
<td>Value: 64</td>
</tr>
<tr>
<td>convert_copy_attrs</td>
<td>Value: ['bsources', 'bimplicit', 'bdepends', 'bact', 'bactsig', ...]</td>
</tr>
<tr>
<td>convert_sig_attrs</td>
<td>Value: ['bsourcesigs', 'bimplicitssigs', 'bdependssigs']</td>
</tr>
<tr>
<td><strong>metaclass</strong></td>
<td>Value: SCons.Memoize.Memoized_Metaclass</td>
</tr>
</tbody>
</table>
15.17 Class FileFinder

```
object
    SCons.Node.FS.FileFinder
```

15.17.1 Methods

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

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

A helper method for find_file() that looks up a directory for
a file we’re trying to find. This only creates the Dir Node if
it exists on-disk, since if the directory doesn’t exist we know
we won’t find any files in it... :-)

It would be more compact to just use this as a nested function
with a default keyword argument (see the commented-out version
below), but that doesn’t work unless you have nested scopes,
so we define it here just so this work under Python 1.5.2.

```
find_file(self, filename, paths, verbose=False)
```

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

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

returns - the node created from the found file.

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

Only the first file found is returned, and none is returned
if no file is found.

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

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

15.17.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

15.17.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoizer_counters</td>
<td>Value: []</td>
</tr>
</tbody>
</table>
```
16 Module SCons.Node.Python

scons.Node.Python

Python nodes.

16.1 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Node/Python.py 5023 2010/06/14 22:05:46...</td>
</tr>
</tbody>
</table>

16.2 Class ValueNodeInfo

object ⊄

SCons.Node.NodeInfoBase ⊄


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

```python
str_to_node(self, s)

__delattr__(...)  
x.__delattr__('name') ↔ del x.name

__getattribute__(...)  
x.__getattribute__('name') ↔ x.name

__hash__(x)  
hash(x)

__init__(self, node=False)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: object.__init__ (inherited documentation)
```
Return Value
   a new object with type S, a subtype of T

(helper for pickle)

(helper for pickle)


repr(x)

setattr('name', value) <== x.name = value

str(x)

convert(self, node, val)

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

merge(self, other)

update(self, node)

16.2.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

16.2.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
<tr>
<td>field_list</td>
<td>Value: ['csig']</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>delattr</strong>(...)</td>
<td>x.<strong>delattr</strong>('name') =&gt; del x.name</td>
</tr>
<tr>
<td><strong>getattr</strong>(...)</td>
<td>x.<strong>getattr</strong>('name') =&gt; x.name</td>
</tr>
<tr>
<td><strong>hash</strong>(...)</td>
<td>hash(x)</td>
</tr>
<tr>
<td><strong>init</strong>(...)</td>
<td>x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature</td>
</tr>
</tbody>
</table>
| __new__(...)    | Return Value
|                 | a new object with type S, a subtype of T                                      |
| __reduce__(...) | helper for pickle                                                            |
| __reduce_ex__(...) | helper for pickle                                                        |
| __repr__(...)   | repr(x)                                                                     |
### Class Value

Object diagram:

```
  object
    SCons.Node.Node
```

A class for Python variables, typically passed on the command line or generated by a script, but not from a file or some other source.

#### 16.4.1 Methods

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

- `x._init__(...)` initializes `x`; see `x._class__._doc__` for signature

**str_for_display**(self)

**str**(self)

- `str(x)`
- Overrides: `object.__str__` (inherited documentation)

---

### 16.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

### 16.3.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
</tbody>
</table>
### Class Value

#### 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.


#### 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.


Overrides: SCons.Node.Node.is_up_to_date

#### is_under(self, dir)

#### write(self, built_value)
Set the value of the node.

#### read(self)
Return the value. If necessary, the value is built.

#### get_text_contents(self)
By the assumption that the node.built_value is a deterministic product of the sources, the contents of a Value are the concatenation of all the contents of its sources. As the value need not be built when get_contents() is called, we cannot use the actual node.built_value.
get_contents(self)

By the assumption that the node.built_value is a
deterministic product of the sources, the contents of a Value
are the concatenation of all the contents of its sources. As
the value need not be built when get_contents() is called, we
cannot use the actual node.built_value.

changed_since_last_build(self, target, prev_ni)

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

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

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

get_csig(self, calc=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.


Decider(self, function)

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

__getattribute__(...) x.__getattribute__('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

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...)
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add wkid(self, wkid)</td>
<td>Add a node to the list of kids waiting to be evaluated</td>
</tr>
<tr>
<td>all_children(self, scan=1)</td>
<td>Return a list of all the node’s direct children.</td>
</tr>
<tr>
<td>alter_targets(self)</td>
<td>Return a list of alternate targets for this Node.</td>
</tr>
<tr>
<td>builder_set(self, builder)</td>
<td></td>
</tr>
<tr>
<td>built(self)</td>
<td>Called just after this node is successfully built.</td>
</tr>
<tr>
<td>changed(self, node=False)</td>
<td>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 <em>always</em> 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.</td>
</tr>
<tr>
<td>children(self, scan=1)</td>
<td>Return a list of the node’s direct children, minus those that are ignored by this node.</td>
</tr>
<tr>
<td>children_are_up_to_date(self)</td>
<td>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.</td>
</tr>
<tr>
<td>clear(self)</td>
<td>Completely clear a Node of all its cached state (so that it can be re-evaluated by interfaces that do continuous integration builds).</td>
</tr>
</tbody>
</table>
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.
### Class Value Module `SCons.Node.Python`

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `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=1)` | Fetch the action executor for this node. Create one if there isn’t already one, and requested to do so. |
| `get_found_includes(self, env, scanner, path)` | Return the scanned include lines (implicit dependencies) found in this node.  
  The default is no implicit dependencies. We expect this method to be overridden by any subclass that can be scanned for implicit dependencies. |
get_implicit_deps(self, env, scanner, path)

Return a list of implicit dependencies for this node.

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

get_ninfo(self)

get_source_scanner(self, node)

Fetch the source scanner for the specified node

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

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

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

get_state(self)

get_stored_implicit(self)

Fetch the stored implicit dependencies

get_stored_info(self)

get_string(self, for_signature)

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

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

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

get_suffix(self)

get_target.Scanner(self)

has_builder(self)

Return whether this Node has a builder or not.

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

has_explicit_builder(self)

Return whether this Node has an explicit builder

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

is_derived(self)

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

This should return true only for nodes whose path should be in the variant directory when duplicate=0 and should contribute their build signatures when they are used as source files to other derived files. For example: source with source builders are not derived in this sense, and hence should not return true.
is_literal(self)
Always pass the string representation of a Node to the command interpreter literally.

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 a Node subclass to remove the underlying file from the file system. Note that subclass methods should call this base class method to get the child check and the BuildInfo structure.
### push_to_cache(self)
Try to push a node into a cache

### remove(self)
Remove this Node: no-op by default.

### render_include_tree(self)
Return a text representation, suitable for displaying to the user, of the include tree for the sources of this node.

### reset_executor(self)
Remove cached executor; forces recompute when needed.

### retrieve_from_cache(self)
Try to retrieve the node’s content from a cache

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

Returns true iff the node was successfully retrieved.

### rexists(self)
Does this node exist locally or in a 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=1)
Set the Node’s always_build value.
set_executor(self, executor)
Set the action executor for this node.

set_explicit(self, is_explicit)

set_nocache(self, nocache=1)
Set the Node's nocache value.

set_noclean(self, noclean=1)
Set the Node's noclean value.

set_precious(self, precious=1)
Set the Node's precious value.

set_specific_source(self, source)

set_state(self, state)

state_has_changed(self, target, prev_ni)

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

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

16.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

16.4.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>memoizer_counts</td>
<td>Value: []</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>revision</code></td>
<td>Value: <code>src/engine/SCons/PathList.py 5023 2010/06/14 22:05:46 sc...</code></td>
</tr>
<tr>
<td><code>.doc</code></td>
<td>Value: <code>SCons.PathL...</code></td>
</tr>
<tr>
<td>TYPE_STRING_NO_SUBST</td>
<td>Value: 0</td>
</tr>
<tr>
<td>TYPE_STRING_SUBST</td>
<td>Value: 1</td>
</tr>
<tr>
<td>TYPE_OBJECT</td>
<td>Value: 2</td>
</tr>
</tbody>
</table>
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)
CheckCHandler(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=1)

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

CheckLibWithHeader(context, libs, header, language, call=False, autoadd=1)

Another (more sophisticated) test for a library. Checks, if library and header is available for language (may be 'C' or 'CXX'). Call maybe be a valid expression with a trailing ';'. As in CheckLib, we support library=None, to test if the call compiles without extra link flags.

18.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/SConf.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
<tr>
<td>build_type</td>
<td>Value: False</td>
</tr>
<tr>
<td>build_types</td>
<td>Value: ['clean', 'help']</td>
</tr>
<tr>
<td>dryrun</td>
<td>Value: 0</td>
</tr>
<tr>
<td>AUTO</td>
<td>Value: 0</td>
</tr>
<tr>
<td>FORCE</td>
<td>Value: 1</td>
</tr>
<tr>
<td>CACHE</td>
<td>Value: 2</td>
</tr>
<tr>
<td>cache_mode</td>
<td>Value: 0</td>
</tr>
<tr>
<td>progress_display</td>
<td>Value: SCons.Util.display</td>
</tr>
<tr>
<td>SConfFS</td>
<td>Value: False</td>
</tr>
<tr>
<td>sconf_global</td>
<td>Value: False</td>
</tr>
</tbody>
</table>
### 18.3 Class SConfWarning

```
object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.SConf.SConfWarning
```

#### 18.3.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>delattr</strong></td>
<td>x.<strong>delattr</strong>(‘name’) &gt;= del x.name</td>
<td>x.<strong>delattr</strong>(‘name’)</td>
</tr>
<tr>
<td><strong>getattribute</strong></td>
<td>x.<strong>getattribute</strong>(‘name’) &gt;= x.name</td>
<td>x.<strong>getattribute</strong>(‘name’)</td>
</tr>
<tr>
<td><strong>getitem</strong></td>
<td>x[y]</td>
<td>x[y]</td>
</tr>
<tr>
<td><strong>getslice</strong></td>
<td>x[i:j]</td>
<td>x[i:j]</td>
</tr>
<tr>
<td><strong>hash</strong></td>
<td>hash(x)</td>
<td>hash(x)</td>
</tr>
<tr>
<td><strong>init</strong></td>
<td>x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature</td>
<td>x.<strong>init</strong>(...)</td>
</tr>
<tr>
<td><strong>new</strong></td>
<td>Return Value. A new object with type S, a subtype of T</td>
<td>S, ...</td>
</tr>
</tbody>
</table>
Class `SConfError` Module `SCons.SConf`

```
_reduce_(...)  
helper for pickle  
Overrides: object._reduce_ (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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
<tr>
<td><code>args</code></td>
<td>Value: <code>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
<tr>
<td><code>message</code></td>
<td>Value: <code>&lt;member 'message' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
</tbody>
</table>

### 18.4 Class `SConfError`

```
object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.SConf.SConfError
```

**Known Subclasses:** `SCons.SConf.ConfigureCacheError`, `SCons.SConf.ConfigureDryRunError`
18.4.1 Methods

```python
__init__(self, msg)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.Exception.__init__(inherited documentation)
```

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

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

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

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

```python
__hash__(x)
hash(x)
```

```python
__new__(T, S, ...)
Return Value
    a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__
```

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

```python
__reduce_ex__(...)
helper for pickle
```

```python
__repr__(x)
repr(x)
Overrides: object.__repr__
```
Class ConfigureDryRunError Module SCons.SConf

```python
__setattr__(...)  
x.__setattr__(name, value) ==> x.name = value
Overrides: object.__setattr__
```

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

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

### 18.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>

### 18.5 Class ConfigureDryRunError

```
object
```

```
exceptions.BaseException
```

```
exceptions.Exception
```

```
SCons.Errors.UserError
```

```
SCons.SConf.SConfError
```

```
SCons.SConf.ConfigureDryRunError
```

Raised when a file or directory needs to be updated during a Configure process, but the user requested a dry-run

### 18.5.1 Methods

```python
__init__(self, target)
Overrides: SCons.SConf.SConfError.__init__
```

```python
__delattr__(...)  
x.__delattr__(name) ==> del x.name
Overrides: object.__delattr__
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `__getattr__` | \( x \ldots \) `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__` | \( \text{hash}(x) \) |
| `__new__` | \( T, S, \ldots \)  
Return Value:  
a new object with type \( S \), a subtype of \( T \)  
Overrides: `exceptions.BaseException.__new__` |
| `__reduce__` | \( \ldots \)  
helper for pickle  
Overrides: `object.__reduce__` (inherited documentation) |
| `__reduce_ex__` | \( \ldots \)  
helper for pickle |
| `__repr__` | \( \text{repr}(x) \)  
\( \text{repr}(x) \)  
Overrides: `object.__repr__` |
| `__setattr__` | \( x \ldots \) `name`, value <==> \( x.name = \text{value} \)  
Overrides: `object.__setattr__` |
| `__setstate__` | \( \ldots \)  
Overrides: `object.__setstate__` |
| `__str__` | \( \text{str}(x) \)  
\( \text{str}(x) \)  
Overrides: `object.__str__` |
18.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
<tr>
<td>args</td>
<td>Value: <code>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
<tr>
<td>message</td>
<td>Value: <code>&lt;member 'message' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
</tbody>
</table>

18.6 Class ConfigureCacheError

object

exceptions.BaseException

    exceptions.Exception

        SCons.Errors.UserError

            SCons.SConf.SConfError

SCons.SConf.ConfigureCacheError

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

18.6.1 Methods

```python
__init__ (self, target)
Overrides: SCons.SConf.SConfError.__init__
```

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

```python
__getattribute__(...)
x.__getattribute__('name') ==> x.name
Overrides: object.__getattribute__
```

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

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

Module SCons.SConf

__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__ (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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
18.7 Class SConfBuildInfo

object

SCons.Node.BuildInfoBase

SCons.Node.FS.FileBuildInfo

SCons.SConf.SConfBuildInfo

Special build info for targets of configure tests. Additional members are result (did the builder succeed last time?) and string, which contains messages of the original build phase.

18.7.1 Methods

```
set_build_result(self, result, string)
```

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

```
x.__delattr__(‘name’) ==> del x.name
```

```
__getattr__(...)
```

```
x.__getattr__(‘name’) ==> x.name
```

```
__hash__(x)
```

```
hash(x)
```

```
__init__(self, node=False)
```

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

Overrides: object.__init__(inherited documentation)

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

**Return Value**

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

```
__reduce__(...)
```

**helper for pickle**

```
__reduce_ex__(...)
```

**helper for pickle**
```python
repr(x)
```

```python
setattr(x, 'name', value)
```

```python
str(x)
```

**convert_from_sconsign** *(self, dir, name)*

Converts a newly-read FileBuildInfo object for in-SCons use

For normal up-to-date checking, we don’t have any conversion to perform—but we’re leaving this method here to make that clear.

**convert_to_sconsign** *(self)*

Converts this FileBuildInfo object for writing to a .sconsign file

This replaces each Node in our various dependency lists with its usual string representation: relative to the top-level SConstruct directory, or an absolute path if it’s outside.

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

**merge** *(self, other)*

**prepare_dependencies** *(self)*

Prepares a FileBuildInfo object for explaining what changed

The bsources, bdepends and bimplicit lists have all been stored on disk as paths relative to the top-level SConstruct directory. Convert the strings to actual Nodes (for use by the --debug=explain code and --implicit-cache).

### 18.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

### 18.7.3 Class Variables
18.8 Class Streamer

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

18.8.1 Methods

```python
__init__(self, orig)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)

write(self, str)
writelines(self, lines)

getvalue(self)
Return everything written to orig since the Streamer was created.

flush(self)

__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
```
Class SConfBuildTask

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_class__</td>
<td>Value: &lt;attribute _class_ of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

18.9 Class SConfBuildTask

object
  SCons.Taskmaster.Task
  SCons.Taskmaster.AlwaysTask
  SCons.SConf.SConfBuildTask

This is almost the same as SCons.Script.BuildTask. Handles SConfErrors correctly and knows about the current cache_mode.
18.9.1 Methods

**display**(self, message)

Hook to allow the calling interface to display a message.

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

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

**display_cached_string**(self, bi)

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

**failed**(self)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

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

**__delattr__**(...)

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

**__getattr__**(...)

```
x.__getattr__('name') == x.name
```
__hash__(x)

hash(x)

__init__(self, tm, targets, top, node)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)

__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)

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

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.

called when the task has been successfully executed and the Taskmaster instance doesn't want 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.
fail_continue(self)

Explicit continue-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

fail_stop(self)

Explicit stop-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

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)

Always returns True (indicating this Task should always be executed).

Subclasses that need this behavior (as opposed to the default of only executing Nodes that are out of date w.r.t. their dependencies) can use this as follows:

```python
class MyTaskSubclass(SCons.Taskmaster.Task):
    needs_execute = SCons.Taskmaster.Task.execute_always
```

Overrides: SCons.Taskmaster.Task.needs_execute

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.

trace_message(self, method, node, description='node')

18.9.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
18.10 Class SConfBase

This is simply a class to represent a configure context. After creating a SConf object, you can call any tests. After finished with your tests, be sure to call the Finish() method, which returns the modified environment.

Some words about caching: In most cases, it is not necessary to cache Test results explicitly. Instead, we use the scons dependency checking mechanism. For example, if one wants to compile a test program (SConf.TryLink), the compiler is only called, if the program dependencies have changed. However, if the program could not be compiled in a former SConf run, we need to explicitly cache this error.

18.10.1 Methods

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

Constructor. Pass additional tests in the custom_tests-dictionary, e.g. custom_tests={'CheckPrivate':MyPrivateTest}, where MyPrivateTest defines a custom test.

Note also the conf_dir and log_file arguments (you may want to build tests in the VariantDir, not in the SourceDir)

Overrides: object._init_

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

```python
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

```python
TryBuild(self, builder, text=True, extension='')
```

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

### TryAction

```python
TryAction(self, action, text=True, 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

```python
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

```python
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

```python
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

```python
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

```python
x.__delattr__('name') == del x.name
```

```python
x.__getattr__('name') == x.name
```

```python
hash(x)
```

```python
T, S, ...
Return Value
a new object with type S, a subtype of T
```

```python
helper for pickle
```

```python
helper for pickle
```

```python
repr(x)
```

```python
x.__setattr__('name', value) == x.name = value
```

```python
str(x)
```

### 18.10.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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:

```python
def CheckCustom(context, ...)
    context.Message('Checking my weird test ... ')
    ret = myWeirdTestFunction(...)  
    context.Result(ret)
```

Often, myWeirdTestFunction will be one of context.TryCompile/context.TryLink/context.TryRun. The results of those are cached, for they are only rebuild, if the dependencies have changed.

### 18.11.1 Methods

- **init**(self, sconf)
  Constructor. Pass the corresponding SConf instance.
  Overrides: object._init_

- **Message**(self, text)
  Inform about what we are doing right now, e.g. ‘Checking for SOMETHING ... ’

- **Result**(self, res)
  Inform about the result of the test. res may be an integer or a string. In case of an integer, the written text will be ‘yes’ or ‘no’. The result is only displayed when self.did_show_result is not set.

- **TryBuild**(self, *args, **kw)
- **TryAction**(self, *args, **kw)
- **TryCompile**(self, *args, **kw)
- **TryLink**(self, *args, **kw)
- **TryRun**(self, *args, **kw)
__getattr__(self, attr)

BuildProg(self, text, ext)

CompileProg(self, text, ext)

CompileSharedObject(self, text, ext)

RunProg(self, text, ext)

AppendLIBS(self, lib_name_list)

PrependLIBS(self, lib_name_list)

SetLIBS(self, val)

Display(self, msg)

Log(self, msg)

__delattr__(...)

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

__getattribute__(...)

x.__getattribute__('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)
Class CheckContext

_module: SCons.SConf

*setattr*

```
xsetattr_('name', value) => x.name = value
```

*str*

```
str(x)
```

### 18.11.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute 'class_' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/SConsign.py 5023 2010/06/14 22:05:46 sc...</td>
</tr>
<tr>
<td>sig_files</td>
<td>Value: []</td>
</tr>
<tr>
<td>DataBase</td>
<td>Value: {}</td>
</tr>
<tr>
<td>DB_Name</td>
<td>Value: '.sconsign'</td>
</tr>
<tr>
<td>DB_sync_list</td>
<td>Value: []</td>
</tr>
</tbody>
</table>

19.3 Class SConsignEntry

SCons.SConsign.SConsignEntry

Wrapper class for the generic entry in a .sconsign file. The Node subclass populates it with attributes as it pleases.

XXX As coded below, we do expect a '.binfo' attribute to be added, but we’ll probably generalize this in the next refactorings.
### 19.3.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(self)</td>
<td>x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature</td>
</tr>
<tr>
<td>convert_to_sconsign(self)</td>
<td></td>
</tr>
<tr>
<td>convert_from_sconsign(self, dir, name)</td>
<td></td>
</tr>
<tr>
<td><strong>delattr</strong>(...)</td>
<td>x.<strong>delattr</strong>('name') ==&gt; del x.name</td>
</tr>
<tr>
<td><strong>getattr</strong>(...)</td>
<td>x.<strong>getattr</strong>('name') ==&gt; x.name</td>
</tr>
<tr>
<td><strong>hash</strong>(x)</td>
<td>hash(x)</td>
</tr>
<tr>
<td><strong>new</strong>(T, S, ...)</td>
<td>Return Value</td>
</tr>
<tr>
<td></td>
<td>a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><strong>reduce</strong>(...)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><strong>reduce_ex</strong>(...)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><strong>repr</strong>(x)</td>
<td>repr(x)</td>
</tr>
<tr>
<td><strong>setattr</strong>(...)</td>
<td>x.<strong>setattr</strong>('name', value) ==&gt; x.name = value</td>
</tr>
<tr>
<td><strong>str</strong>(x)</td>
<td>str(x)</td>
</tr>
</tbody>
</table>

### 19.3.2 Properties
19.3.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_version_id</td>
<td>Value: 1</td>
</tr>
</tbody>
</table>

19.4 Class Base

object ▶
SCons.SConsign.Base

**Known Subclasses:** SCons.SConsign.DB, SCons.SConsign.Dir

This is the controlling class for the signatures for the collection of entries associated with a specific directory. The actual directory association will be maintained by a subclass that is specific to the underlying storage method. This class provides a common set of methods for fetching and storing the individual bits of information that make up signature entry.

19.4.1 Methods

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

```python
get_entry(self, filename)
```
Fetch the specified entry attribute.

```python
set_entry(self, filename, obj)
```
Set the entry.

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

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

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

```python
merge(self)
```
Class DB Module SCons.SConsign

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

__getattribute__(...)  
x.__getattribute__('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)

19.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '.<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

19.5 Class DB

object  
\[ SCons.SConsign.Base \]  
\[ SCons.SConsign.DB \]
A Base subclass that reads and writes signature information from a global `.sconsign.db*` file—the actual file suffix is determined by the database module.

19.5.1 Methods

```python
__init__(self, dir)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: SCons.SConsign.Base.__init__
```

```python
write(self, sync=1)
```

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

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

```python
_hash__(x)
hash(x)
```

```python
_new__(T, S, ...)
Return Value
a new object with type S, a subtype of T
```

```python
_reduce__(...)
helper for pickle
```

```python
_reduce_ex__(...)
helper for pickle
```

```python
_repr__(x)
repr(x)
```

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

```python
_str__(x)
str(x)
```
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)

19.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

19.6 Class Dir

object └── SCons.SConsign.Base └── SCons.SConsign.Dir

Known Subclasses: SCons.SConsign.DirFile

19.6.1 Methods

__init__ (self, fp=False, dir=False)
fp - file pointer to read entries from
Overrides: SCons.SConsign.Base.__init__

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

__getattribute__(...)
x.__getattribute__('name') <=> x.name
Class Dir

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)

do_not_set_entry(self, filename, obj)

do_not_store_info(self, filename, node)

get_entry(self, filename)

Fetch the specified entry attribute.

merge(self)

set_entry(self, filename, obj)

Set the entry.

store_info(self, filename, node)

19.6.2 Properties
### Class DirFile

Object diagram:

```
object
  |-- SCons.SConsign.Base
      |-- SCons.SConsign.Dir
      |       |-- SCons.SConsign.DirFile
```

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

#### 19.7.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `__init__(self, dir)` | dir - the directory for the file
| | Overrides: SCons.SConsign.Dir.__init__ |
| `write(self, sync=1)` | Write the `.sconsign` file to disk. |
| | Try to write to a temporary file first, and rename it if we succeed. If we can’t write to the temporary file, it’s probably because the directory isn’t writable (and if so, how did we build anything in this directory, anyway?), so try to write directly to the `.sconsign` file as a backup. If we can’t rename, try to copy the temporary contents back to the `.sconsign` file. Either way, always try to remove the temporary file at the end. |
| `__delattr__(...)` | x.__delattr__('name') == del x.name |
| `__getattr__(...)` | x.__getattr__('name') == x.name |
| `__hash__(x)` | hash(x) |
Class DirFile  

Module SCons.SConsign

```python
_new__(T, S, ...)

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

```python
_reduce__(...)

helper for pickle
```

```python
_reduce_ex__(...)

helper for pickle
```

```python
_repr__(x)

repr(x)
```

```python
_setattr__(...)

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

```python
_str__(x)

str(x)
```

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

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

```python
get_entry(self, filename)

Fetch the specified entry attribute.
```

```python
merge(self)
```

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

Set the entry.
```

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

19.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.<em>class</em>_</td>
<td>Value: &lt;attribute '<em>class</em>_' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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

```python
__init__(self, dir)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: SCons.SConsign.Base.__init__

write(self, sync=1)

__delattr__(...)

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

__getattribute__(...)

x.__getattribute__('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)
```
```python
__setattr__(...)
x.__setattr__(‘name’, value) <=> x.name = value
```

```python
__str__(x)
str(x)
```

```python
do_not_set_entry(self, filename, obj)
do_not_store_info(self, filename, node)
```

```python
def_entry(self, filename)
Fetch the specified entry attribute.
```

```python
merge(self)
```

```python
set_entry(self, filename, obj)
Set the entry.
```

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

### 19.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>
20 Package 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. 342)
- D: SCons.Scanner.D
  Scanner for the Digital Mars "D" programming language.
  (Section 22, p. 349)
- Dir (Section 23, p. 352)
- Fortran: SCons.Scanner.Fortran
  This module implements the dependency scanner for Fortran code.
  (Section 24, p. 353)
- IDL: SCons.Scanner.IDL
  This module implements the dependency scanner for IDL (Interface Definition Language) files.
  (Section 25, p. 356)
- LaTeX: SCons.Scanner.LaTeX
  This module implements the dependency scanner for LaTeX code.
  (Section 26, p. 357)
- Prog (Section 27, p. 364)
- RC: SCons.Scanner.RC
  This module implements the dependency scanner for RC (Interface Definition Language) files.
  (Section 28, p. 365)

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: <code>'src/engine/SCons/Scanner/__init__.py 5023 2010/06/14 22:</code>...</td>
</tr>
</tbody>
</table>

continued on next page
### Class FindPathDirs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCons.Scanner.FindPathDirs</td>
<td>A class to bind a specific <em>PATH</em> variable name to a function that will return all of the <em>path</em> directories.</td>
</tr>
</tbody>
</table>

#### 20.4.1 Methods

- **init**(self, variable)
  x.__init__(...) initializes x; see x.__class__.doc_ for signature
  Overrides: object.__init__ (inherited documentation)

- **call**(self, env, dir=False, target=False, source=False, argument=False)

- **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)
20.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

20.5 Class Base

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)

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

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

__hash__(self)  
  hash(x)  
  Overrides: object.__hash__ (inherited documentation)
Construct a new scanner object given a scanner function.

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

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

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

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

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

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

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

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

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

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

Examples:

s = Scanner(my_scanner_function)

s = Scanner(function = my_scanner_function)
Class Base

Package SCons.Scanner

_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__(self)_  

str(x)

 Overrides: object._str_  extit(inherited documentation)

_add_scanner(self, skey, scanner)_

_add_skey(self, skey)_  

_Add a skey to the list of skeys_

_get_skeys(self, env=False)_

_path(self, env, dir=False, target=False, source=False)_

_recuse_nodes(self, nodes)_

_select(self, node)_

20.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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

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

Construct a new scanner object given a scanner function.

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

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

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

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

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

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

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

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

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

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

Examples:

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

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


**select**(self, node)
Overrides: SCons.Scanner.Base.select

**add_scanner**(self, skey, scanner)
Overrides: SCons.Scanner.Base.add_scanner

**_cmp_**(self, other)

**_delattr_**(...)  
x._delattr_('name') == del x.name

**_getattribute_**(...)  
x._getattribute_('name') == x.name

**_hash_**(self)
hash(x)
Overrides: object._hash_.

**_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_(self)
str(x)
Overrides: object._str_ (inherited documentation)

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)

20.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

20.7 Class Current

object

SCons.Scanner.Base

SCons.Scanner.Current

Known Subclasses: SCons.Scanner.Classic

A class for scanning files that are source files (have no builder)
or are derived files and are current (which implies that they exist,
either locally or in a repository).
20.7.1 Methods

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

Construct a new scanner object given a scanner function.

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

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

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

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

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

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

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

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

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

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

Examples:

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__call__(self, node, env, path=())</td>
<td>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.</td>
</tr>
<tr>
<td>__cmp__(self, other)</td>
<td></td>
</tr>
<tr>
<td>__delattr__(_)</td>
<td>x.__delattr__('name') \implies del x.name</td>
</tr>
<tr>
<td>__getattribute__(_)</td>
<td>x.__getattribute__('name') \implies x.name</td>
</tr>
<tr>
<td>__hash__(self)</td>
<td>hash(x) \n Overrides: object.__hash_ extit(inherited documentation)</td>
</tr>
<tr>
<td>__new__(T, S, ...)</td>
<td>Return Value \n a new object with type S, a subtype of T</td>
</tr>
<tr>
<td>__reduce__(_)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td>__reduce_ex__(_)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td>__repr__(x)</td>
<td>repr(x)</td>
</tr>
<tr>
<td>__setattr__(_)</td>
<td>x.__setattr__('name', value) \implies x.name = value</td>
</tr>
<tr>
<td>__str__(self)</td>
<td>str(x) \n Overrides: object.__str_ extit(inherited documentation)</td>
</tr>
<tr>
<td>add_scanner_(self, skey, scanner)</td>
<td></td>
</tr>
</tbody>
</table>
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)

20.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

20.8 Class Classic

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)

__delattr__(...)

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

__getattribute__(...)

x.__getattribute__('name') == x.name

__hash__(self)

hash(x)

Overrides: object.__hash__ (inherited documentation)

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

Overrides: SCons.Scanner.Current.__init__

__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__(self)

str(x)

Overrides: object.__str__ (inherited documentation)

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)

20.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

20.9 Class ClassicCPP


A Classic Scanner subclass which takes into account the type of bracketing used to include the file, and uses classic CPP rules for searching for the files based on the bracketing.

Note that in order for this to work, the regular expression passed to the constructor must return the leading bracket in group 0, and the contained filename in group 1.

20.9.1 Methods

find_include(self, include, source_dir, path)
Overrides: SCons.Scanner.Classic.find_include

sort_key(self, include)
Overrides: SCons.Scanner.Classic.sort_key
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.

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

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

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

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

```
__hash__(self)  
hash(x)  
Overrides: object.__hash__ extit(inherited documentation)
```

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

```
__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__(self)  
str(x)  
Overrides: object.__str__ extit(inherited documentation)
```
add_scanner(self, skey, scanner)

add_skey(self, skey)
Add a skey to the list of skeys

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)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
21 Module SCons.Scanner.C

SCons.Scanner.C

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

21.1 Functions

dictify_CPPDEFINES(env)

CScanner()

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

21.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Scanner/C.py 5023 2010/06/14 22:05:46 s...</td>
</tr>
</tbody>
</table>

21.3 Class SConsCPPScanner

object __________

SCons.cpp.PreProcessor __________

SCons.Scanner.C.SConsCPPScanner

SCons-specific subclass of the cpp.py module’s processing.

We subclass this so that: 1) we can deal with files represented by Nodes, not strings; 2) we can keep track of the files that are missing.

21.3.1 Methods

__init__(self, *args, **kw)

Overrides: SCons.cpp.PreProcessor.__init__

initialize_result(self, fname)

Overrides: SCons.cpp.PreProcessor.initialize_result

finalize_result(self, fname)

Overrides: SCons.cpp.PreProcessor.finalize_result
### Class SConsCPPScanner

#### find_include_file
```python
find_include_file(self, t)
```
Finds the `#include` file for a given preprocessor tuple.
Overrides: SCons.cpp.PreProcessor.find_include_file

#### read_file
```python
read_file(self, file)
```
Overrides: SCons.cpp.PreProcessor.read_file

#### __call__
```python
__call__(self, file)
```
Pre-processes a file.
This is the main public entry point.

#### __delattr__
```python
__delattr__(...)  
x.__delattr__('name') <-> del x.name
```

#### __getattribute__
```python
__getattribute__(...)  
x.__getattribute__('name') <-> x.name
```

#### __hash__
```python
__hash__(x)
```

#### __new__
```python
__new__(T, S, ...)  
Return Value
  a new object with type S, a subtype of T
```

#### __reduce__
```python
__reduce__(...)  
helper for pickle
```

#### __reduce_ex__
```python
__reduce_ex__(...)  
helper for pickle
```

#### __repr__
```python
__repr__(x)
```

#### __setattr__
```python
__setattr__(...)  
x.__setattr__('name', value) <-> x.name = value
```
```python
__str__(x)
str(x)

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__ifndef__(self, t)
Default handling of a #ifndef 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.
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>do_include(self, t)</code></td>
<td>Default handling of a #include line.</td>
</tr>
<tr>
<td><code>do_include_next(self, t)</code></td>
<td>Default handling of a #include line.</td>
</tr>
<tr>
<td><code>do_nothing(self, t)</code></td>
<td>Null method for when we explicitly want the action for a specific preprocessor directive to do nothing.</td>
</tr>
<tr>
<td><code>do_UNDEF(self, t)</code></td>
<td>Default handling of a #undef line.</td>
</tr>
<tr>
<td><code>eval_expression(self, t)</code></td>
<td>Evaluates a C preprocessor expression.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td><code>process_contents(self, contents, fname=False)</code></td>
<td>Pre-processes a file contents.</td>
</tr>
<tr>
<td></td>
<td>This is the main internal entry point.</td>
</tr>
<tr>
<td><code>resolve_include(self, t)</code></td>
<td>Resolve a tuple-ized #include line.</td>
</tr>
<tr>
<td></td>
<td>This handles recursive expansion of values without &quot;&quot; or &lt;&gt; surrounding the name until an initial &quot; or &lt; is found, to handle #include FILE where FILE is a #define somewhere else.</td>
</tr>
<tr>
<td><code>restore(self)</code></td>
<td>Pops the previous dispatch table off the stack and makes it the current one.</td>
</tr>
</tbody>
</table>
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.

21.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
## 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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__(self, name, variable)</code></td>
<td>x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature Overrides: object.<strong>init</strong>(inherited documentation)</td>
</tr>
<tr>
<td><code>__call__(self, node, env, path=())</code></td>
<td>recurse_nodes(self, nodes) select(self, node) delattr(...) getattribute(...) hash(x) new(T, S, ...) reduce(...) reduce_ex(...)</td>
</tr>
<tr>
<td><code>x.__delattr__('name')</code></td>
<td>del x.name</td>
</tr>
<tr>
<td><code>x.__getattribute__('name')</code></td>
<td>x.name</td>
</tr>
<tr>
<td><code>hash(x)</code></td>
<td></td>
</tr>
<tr>
<td><code>__new__(T, S, ...)</code></td>
<td>Return Value</td>
</tr>
<tr>
<td></td>
<td>a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>__reduce__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__(...)</code></td>
<td>helper for pickle</td>
</tr>
</tbody>
</table>
```python
__repr__(x)
repr(x)

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

__str__(x)
str(x)
```

### 21.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: <code>src/engine/SCons/Scanner/D.py 5023 2010/06/14 22:05:46 s...</code></td>
</tr>
</tbody>
</table>

22.3 Class D

object

SCons.Scanner.Base

SCons.Scanner.Current

SCons.Scanner.Classic

SCons.Scanner.D.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
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.

```
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)

22.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
23 Module SCons.Scanner.Dir

23.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>only_dirs(nodes)</td>
<td></td>
</tr>
<tr>
<td>DirScanner(**kw)</td>
<td>Return a prototype Scanner instance for scanning directories for on-disk files</td>
</tr>
<tr>
<td>DirEntryScanner(**kw)</td>
<td>Return a prototype Scanner instance for &quot;scanning&quot; directory Nodes for their in-memory entries</td>
</tr>
<tr>
<td>do_not_scan(k)</td>
<td></td>
</tr>
<tr>
<td>scan_on_disk(node, env, path=())</td>
<td>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.</td>
</tr>
<tr>
<td>scan_in_memory(node, env, path=())</td>
<td>&quot;Scans&quot; a Node.FS.Dir for its in-memory entries.</td>
</tr>
</tbody>
</table>

23.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Scanner/Dir.py 5023 2010/06/14 22:05:46...'</td>
</tr>
<tr>
<td>skip_entry</td>
<td>Value: {'.': 1, '..': 1, '.sconsign': 1, '.sconsign.bak': 1, '.s...</td>
</tr>
<tr>
<td>skip_entry_list</td>
<td>Value: ['.', '..', '.sconsign', '.sconsign.dblite', '.sconsign.d...</td>
</tr>
<tr>
<td>skip</td>
<td>Value: '.sconsign.db'</td>
</tr>
</tbody>
</table>
24  Module 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Scanner/Fortran.py 5023 2010/06/14 22:0...</td>
</tr>
</tbody>
</table>

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

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)

### __delattr__(...)__

```
x.__delattr__(‘name’) ==> del x.name
```

### __getattr__(...)__

```
x.__getattribute__(‘name’) ==> x.name
```

### __hash__(self)

```
hash(x)
```

Overrides: object.__hash__ (inherited documentation)

### __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__(...)__

```
repr(x)
```

### __setattr__(...)__

```
x.__setattr__(‘name’, value) ==> x.name = value
```

### __str__(...)__

```
str(x)
```

Overrides: object.__str__ (inherited documentation)
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)

24.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
25 Module SCons.Scanner.IDL

SCons.Scanner.IDL

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

25.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLScan()</td>
<td>Return a prototype Scanner instance for scanning IDL source files</td>
</tr>
</tbody>
</table>

25.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Scanner/IDL.py 5023 2010/06/14 22:05:46...'</td>
</tr>
</tbody>
</table>
26 Module SCons.Scanner.LaTeX

This module implements the dependency scanner for LaTeX code.

26.1 Functions

```python
modify_env_var(env, var, abspath)
```

LaTeXScanner()

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

PDFLaTeXScanner()

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

26.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Scanner/LaTeX.py 5023</td>
</tr>
<tr>
<td></td>
<td>2010/06/14 22:05:...</td>
</tr>
<tr>
<td>TexGraphics</td>
<td>Value: ['.eps', '.ps']</td>
</tr>
<tr>
<td>LatexGraphics</td>
<td>Value: ['.pdf', '.png', '.jpg', '.gif', '.tif']</td>
</tr>
</tbody>
</table>

26.3 Class FindENVPathDirs

A class to bind a specific *PATH variable name to a function that will return all of the *path directories.

26.3.1 Methods

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

26.4 Class LaTeX

```
object                   
SCons.Scanner.Base     
SCons.Scanner.LaTeX.LaTeX
```
Class for scanning LaTeX files for included files.

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

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

The actual subset and search order may be altered by DeclareGraphicsExtensions command. This complication is ignored.

The default order corresponds to experimentation with teTeX

```bash
$ latex --version
pdfeTeX 3.141592-1.21a-2.2 (Web2C 7.5.4)
kpathsea version 3.5.4
```

The order is:

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

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

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

FIXME: also look for the class or style in document{class|style}{}
FIXME: also look for the argument of bibliographystyle{}`
26.4.1 Methods

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

Construct a new scanner object given a scanner function.

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

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

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

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

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

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

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

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

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

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

Examples:

```python
s = Scanner(my_scanner_function)
s = Scanner(function = my_scanner_function)
```
Class LaTeX  

Module SCons.Scanner.LaTeX

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

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

```python
scan(self, node)
```

```python
scan_recursive(self, node, path=())
```

This do a recursive scan of the top level target file  
This lets us search for included files based on the  
directory of the main file just as latex does

```python
__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.

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

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

```python
x.__delattr__('name') == del x.name
```

```python
__setattr__(...)
```

```python
x.__setattr__('name') == x.name
```

```python
_hash__(self)
```

```python
hash(x)
```

Overrides: object._hash_ (inherited documentation)

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

Return Value  
a new object with type S, a subtype of T

```python
__reduce__(...)
```

helper for pickle

```python
__reduce_ex__(...)
```

helper for pickle
Class LaTeX

Module SCons.Scanner.LaTeX

```python
__repr__(x)
repr(x)

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

addScanner(self, skey, scanner)

addSkey(self, skey)
Add a skey to the list of skeys

getSkeys(self, env=False)

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

recurseNodes(self, nodes)

select(self, node)
```

### 26.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

### 26.4.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyword_paths</td>
<td>Value: <code>{‘bibliography’: ’BIBINPUTS’, ‘bibliographystyle’: ’BSTIN...</code></td>
</tr>
<tr>
<td>env_variables</td>
<td>Value: <code>[’BIBINPUTS’, ’TEXINPUTS’, ’BSTINPUTS’]</code></td>
</tr>
</tbody>
</table>

379
27 Module SCons.Scanner.Prog

27.1 Functions

<table>
<thead>
<tr>
<th>ProgramScanner(**kw)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return a prototype Scanner instance for scanning executable</td>
</tr>
<tr>
<td>files for static-lib dependencies</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>scan(node, env, libpath=())</th>
</tr>
</thead>
<tbody>
<tr>
<td>This scanner scans program files for static-library</td>
</tr>
<tr>
<td>dependencies. It will search the LIBPATH environment variable</td>
</tr>
<tr>
<td>for libraries specified in the LIBS variable, returning any</td>
</tr>
<tr>
<td>files it finds as dependencies.</td>
</tr>
</tbody>
</table>

27.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>..revision..</td>
<td>Value: 'src/engine/SCons/Scanner/Prog.py 5023 2010/06/14 22:05:4...</td>
</tr>
<tr>
<td>print_find_libs</td>
<td>Value: False</td>
</tr>
</tbody>
</table>
28 Module SCons.Scanner.RC

SCons.Scanner.RC

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

28.1 Functions

RCScan()

Return a prototype Scanner instance for scanning RC source files

28.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: <code>src/engine/SCons/Scanner/RC.py 5023 2010/06/14 22:05:46 ...</code></td>
</tr>
</tbody>
</table>
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. 373)*
- **Main**: SCons.Script
  This file implements the main() function used by the scons script.
  *(Section 31, p. 377)*
- **SConscript**: SCons.Script.SConscript
  This module defines the Python API provided to SConscript and SConstruct files.
  *(Section 32, p. 404)*

29.2  Functions

<table>
<thead>
<tr>
<th>HelpFunction(text)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Variables(files=[], args={})</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Options(files=[], args={})</th>
</tr>
</thead>
</table>

29.3  Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Script/<strong>init</strong>.py 5023 2010/06/14 22:0...</td>
</tr>
<tr>
<td>start_time</td>
<td>Value: 1276578478.69</td>
</tr>
<tr>
<td>call_stack</td>
<td>Value: []</td>
</tr>
<tr>
<td>PathVariable</td>
<td>Value: &lt;SCons.Variables.PathVariable._PathVariableClass object a...</td>
</tr>
<tr>
<td>PathOption</td>
<td>Value: &lt;SCons.Options.PathOption._PathOptionClass object at 0x8b...</td>
</tr>
<tr>
<td>Chmod</td>
<td>Value: ActionFactory(chmod_func, chmod_strfunc)</td>
</tr>
<tr>
<td>Copy</td>
<td>Value: ActionFactory(copy_func, lambda dest, src: 'Copy(&quot;%s&quot;, &quot;%...</td>
</tr>
<tr>
<td>Delete</td>
<td>Value: ActionFactory(delete_func, delete_strfunc)</td>
</tr>
</tbody>
</table>

*continued on next page*
### Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mkdir</td>
<td>Value: <code>ActionFactory(mkdir_func, lambda dir: 'Mkdir(%s)' % get_p...</code></td>
</tr>
<tr>
<td>Move</td>
<td>Value: <code>ActionFactory(move_func, lambda dest, src: 'Move(&quot;%s&quot;, &quot;%s&quot;)' % get_p...</code></td>
</tr>
<tr>
<td>Touch</td>
<td>Value: <code>ActionFactory(touch_func, lambda file: 'Touch(%s)' % get_p...</code></td>
</tr>
<tr>
<td>CScanner</td>
<td>Value: <code>SCons.Tool.CScanner</code></td>
</tr>
<tr>
<td>DScanner</td>
<td>Value: <code>SCons.Tool.DScanner</code></td>
</tr>
<tr>
<td>DirScanner</td>
<td>Value: <code>SCons.Scanner.Dir.DirScanner()</code></td>
</tr>
<tr>
<td>ProgramScanner</td>
<td>Value: <code>SCons.Tool.ProgramScanner</code></td>
</tr>
<tr>
<td>SourceFileScanner</td>
<td>Value: <code>SCons.Tool.SourceFileScanner</code></td>
</tr>
<tr>
<td>CScan</td>
<td>Value: <code>SCons.Tool.CScanner</code></td>
</tr>
<tr>
<td>ARGUMENTS</td>
<td>Value: <code>{}</code></td>
</tr>
<tr>
<td>ARGLIST</td>
<td>Value: <code>[]</code></td>
</tr>
<tr>
<td>BUILD_TARGETS</td>
<td>Value: <code>[]</code></td>
</tr>
<tr>
<td>COMMAND_LINE_TARGETS</td>
<td>Value: <code>[]</code></td>
</tr>
<tr>
<td>DEFAULT_TARGETS</td>
<td>Value: <code>[]</code></td>
</tr>
<tr>
<td>help_text</td>
<td>Value: <code>False</code></td>
</tr>
<tr>
<td>sconscript_reading</td>
<td>Value: <code>0</code></td>
</tr>
<tr>
<td>GlobalDefaultEnviron-</td>
<td>Value: <code>['Default', 'EnsurePythonVersion', 'EnsureSConsVersion', ...</code></td>
</tr>
<tr>
<td>ents</td>
<td>GlobalDefaultBuilders</td>
</tr>
<tr>
<td>SConscript</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Command</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>AddPostAction</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>AddPreAction</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Alias</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>AlwaysBuild</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>BuildDir</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>CFile</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>CXXFile</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>CacheDir</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Clean</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>DVI</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Decider</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
</tbody>
</table>

*continued on next page*
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Depends</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Dir</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>EnsurePythonVersion</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>EnsureSConsVersion</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Entry</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Execute</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Exit</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Export</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>File</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>FindFile</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>FindInstalledFiles</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>FindSourceFiles</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Flatten</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>GetBuildPath</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>GetLaunchDir</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Glob</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Help</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Ignore</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Import</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Install</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>InstallAs</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Jar</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Java</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>JavaH</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
</tbody>
</table>

continued on next page
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Literal</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Local</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>M4</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>MSVSProject</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>NoCache</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>NoClean</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Object</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>PCH</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>PDF</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Package</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>ParseDepends</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>PostScript</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Precious</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Program</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>RES</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>RMIC</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Repository</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Requires</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>SConscriptChdir</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>SConsignFile</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>SharedLibrary</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>SharedObject</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>SideEffect</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>SourceCode</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SourceSignatures</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Split</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>StaticLibrary</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>StaticObject</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Tag</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Tar</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>TargetSignatures</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>TypeLibrary</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Value</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>VariantDir</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
<tr>
<td>Zip</td>
<td>Value: <code>&lt;SCons.Script.SConscript.DefaultEnvironmentCall object at...</code></td>
</tr>
</tbody>
</table>

### 29.4 Class TargetList

```
UserList.UserList └── SCons.Script.TargetList
```

#### 29.4.1 Methods

```
__add__(self, other)
__cmp__(self, other)
__contains__(self, item)
__delitem__(self, i)
__delslice__(self, i, j)
__eq__(self, other)
__ge__(self, other)
__getitem__(self, i)
```
.. method:: __getslice__(self, i, j)

.. method:: __gt__(self, other)

.. method:: __iadd__(self, other)

.. method:: __imul__(self, n)

.. method:: __init__(self, initlist=False)

.. method:: __le__(self, other)

.. method:: __len__(self)

.. method:: __lt__(self, other)

.. method:: __mul__(self, n)

.. method:: __ne__(self, other)

.. method:: __radd__(self, other)

.. method:: __repr__(self)

.. method:: __rmul__(self, n)

.. method:: __setitem__(self, i, item)

.. method:: __setslice__(self, i, j, other)

.. method:: append(self, item)

.. method:: count(self, item)

.. method:: extend(self, other)

.. method:: index(self, item, *args)

.. method:: insert(self, i, item)

.. method:: pop(self, i=-1)

.. method:: remove(self, item)

.. method:: reverse(self)
sort(self, *args, **kwds)
30 Module SCons.Script.Interactive

SCons interactive mode

30.1 Functions

```python
interact(fs, parser, options, targets, target_top)
```

30.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>revision</td>
<td>Value: 'src/engine/SCons/Script/Interactive.py 5023 2010/06/14 2...</td>
</tr>
<tr>
<td>doc</td>
<td>Value: ...</td>
</tr>
</tbody>
</table>

30.3 Class SConsInteractiveCmd

```python
cmd.Cmd

SCons.Script.Interactive.SConsInteractiveCmd
```

- **build [TARGETS]**: Build the specified TARGETS and their dependencies. 'b' is a synonym.
- **clean [TARGETS]**: Clean (remove) the specified TARGETS and their dependencies. 'c' is a synonym.
- **exit**: Exit SCons interactive mode.
- **help [COMMAND]**: Prints help for the specified COMMAND. 'h' and '?' are synonyms.
- **shell [COMMANDLINE]**: Execute COMMANDLINE in a subshell. 'sh' and '!' are synonyms.
- **version**: Prints SCons version information.

30.3.1 Methods

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

Instantiate a line-oriented interpreter framework.

The optional argument 'completekey' is the readline name of a completion key; it defaults to the Tab key. If completekey is not None and the readline module is available, command completion is done automatically. The optional arguments stdin and stdout specify alternate input and output file objects; if not specified, sys.stdin and sys.stdout are used.

Overrides: cmd.Cmd.__init__, exit (inherited documentation)
**default**(\texttt{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: \texttt{cmd.Cmd.default} (inherited documentation)

**onecmd**(\texttt{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 \texttt{precmd()} and \texttt{postcmd()} methods for useful execution hooks. The return value is a flag indicating whether interpretation of commands by the interpreter should stop.

Overrides: \texttt{cmd.Cmd.onecmd} (inherited documentation)

**do_build**(\texttt{self, argv})

\texttt{build [TARGETS]} Build the specified TARGETS and their dependencies. 'b' is a synonym.

**do_clean**(\texttt{self, argv})

\texttt{clean [TARGETS]} Clean (remove) the specified TARGETS and their dependencies. 'c' is a synonym.

**do_EOF**(\texttt{self, argv})

**do_exit**(\texttt{self, argv})

\texttt{exit} Exit SCons interactive mode.

**do_help**(\texttt{self, argv})

\texttt{help [COMMAND]} Prints help for the specified COMMAND. 'h' and '?' are synonyms.

Overrides: \texttt{cmd.Cmd.do_help}

**do_shell**(\texttt{self, argv})

\texttt{shell [COMMANDLINE]} Execute COMMANDLINE in a subshell. 'sh' and '!' are synonyms.

**do_version**(\texttt{self, argv})

\texttt{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)

completdescriptor(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

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>synonyms</td>
<td>Value: {'b': 'build', 'c': 'clean', 'h': 'help', 'scons': 'build...}</td>
<td></td>
</tr>
<tr>
<td>doc_header</td>
<td>Value: 'Documented commands (type help &lt;topic&gt;):'</td>
<td></td>
</tr>
<tr>
<td>doc_leader</td>
<td>Value: ''</td>
<td></td>
</tr>
<tr>
<td>identchars</td>
<td>Value: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123...</td>
<td></td>
</tr>
<tr>
<td>intro</td>
<td>Value: False</td>
<td></td>
</tr>
<tr>
<td>lastcmd</td>
<td>Value: ''</td>
<td></td>
</tr>
<tr>
<td>misc_header</td>
<td>Value: 'Miscellaneous help topics:'</td>
<td></td>
</tr>
<tr>
<td>nohelp</td>
<td>Value: '*** No help on %s'</td>
<td></td>
</tr>
<tr>
<td>prompt</td>
<td>Value: '(Cmd) '</td>
<td></td>
</tr>
<tr>
<td>ruler</td>
<td>Value: '='</td>
<td></td>
</tr>
<tr>
<td>undoc_header</td>
<td>Value: 'Undocumented commands:'</td>
<td></td>
</tr>
<tr>
<td>use_rawinput</td>
<td>Value: 1</td>
<td></td>
</tr>
</tbody>
</table>
31 Module SCons.Script.Main

SCons.Script

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

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

31.1 Functions

`fetch_win32_parallel_msg()`

`Progress(*args, **kw)`

`GetBuildFailures()`

`python_version_string()`

`python_version_unsupported(version=(2, 5, 2, 'final', 0))`

`python_version_deprecated(version=(2, 5, 2, 'final', 0))`

`AddOption(*args, **kw)`

`GetOption(name)`

`SetOption(name, value)`

`find_deepest_user_frame(tb)`

Find the deepest stack frame that is not part of SCons.

Input is a "pre-processed" stack trace in the form returned by `traceback.extract_tb()` or `traceback.extract_stack()`

`version_string(label, module)`

`main()`

31.2 Variables
### Class SConsPrintHelpException

#### 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]
```

31.3.1 Methods
__getslice__(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__ (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
# Class Progressor

Object \[ SCons.Script.Main.Progressor \]

## 31.4.1 Methods

- `__init__(self, obj, interval=1, file=False, overwrite=False)`
  - `x.__init__(...)` initializes `x`; see `x.__class__.__doc__` for signature
  - Overrides: `object.__init__` (inherited documentation)

- `write(self, s)`

- `erase_previous(self)`

- `spinner(self, node)`

- `string(self, node)`

- `replace_string(self, node)`

- `__call__(self, node)`

- `__delattr__(...)`
  - `x.__delattr__(\'name\')` $\Rightarrow$ del `x.name`

- `__getattr__(...)`
  - `x.__getattr__(\'name\')` $\Rightarrow$ `x.name`

- `__hash__(x)`
  - `hash(x)`

- `__new__(T, S, ...)`
  - Return Value
    - a new object with type `S`, a subtype of `T`
31.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

31.4.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>prev</code></td>
<td>Value: ''</td>
</tr>
<tr>
<td><code>count</code></td>
<td>Value: 0</td>
</tr>
<tr>
<td><code>target_string</code></td>
<td>Value: '$TARGET'</td>
</tr>
</tbody>
</table>

31.5 Class 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 (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 (inherited documentation)

**needs_execute(self)**

Returns True (indicating this Task should be executed) if this Task’s target state indicates it needs executing, which has already been determined by an earlier up-to-date check.

Overrides: SCons.Taskmaster.OutOfDateTask.needs_execute (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 (inherited documentation)

**do_failed(self, status=2)**
### executed(self)

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

This may have been a do-nothing operation (to preserve build order), so we must check the node's state before deciding whether it was "built", in which case we call the appropriate Node method.

In any event, we always call "visited()", which will handle any post-visit actions that must take place regardless of whether or not the target was an actual built target or a source Node.

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

### failed(self)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

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

### postprocess(self)

Post-processes a task after it's been executed.

This examines all the targets just built (or not, we don’t care if the build was successful, or even if there was no build because everything was up-to-date) to see if they have any waiting parent Nodes, or Nodes waiting on a common side effect, that can be put back on the candidates list.

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

### make_ready(self)

Make a task ready for execution

Overrides: SCons.Taskmaster.Task.make_ready

```python
_delattr (...)  
x._delattr_('name') == del x.name
```

```python
__getattribute__ (...)  
x.__getattribute__('name') == x.name
```

```python
__hash__ (x)  
hash(x)
```
__init__(self, tm, targets, top, node)
_x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)

__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)

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.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

fail_stop(self)

Explicit stop-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

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.

trace_message(self, method, node, description='node')

31.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

31.5.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>progress</td>
<td>Value: Null(0x08BD80CC)</td>
</tr>
</tbody>
</table>

31.6 Class CleanTask

object

SCons.Taskmaster.Task

SCons.Taskmaster.AlwaysTask

SCons.Script.Main.CleanTask

An SCons clean task.

31.6.1 Methods

fs_delete(self, path, pathstr, remove=1)

show(self)
remove(self)

Called to execute the task.

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

execute(self)

Called to execute the task.

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

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

```
def __delattr__(self, name):
    del self.name
```

```
def __getattr__(self, name):
    return self.name
```
Class CleanTask

Module SCons.Script.Main

```python
__hash__(x)
hash(x)

__init__(self, tm, targets, top, node)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)

__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)

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.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

fail_stop(self)

Explicit stop-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

failed(self)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

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)

Always returns True (indicating this Task should always be executed).

Subclasses that need this behavior (as opposed to the default of only executing Nodes that are out of date w.r.t. their dependencies) can use this as follows:

```python
class MyTaskSubclass(SCons.Taskmaster.Task):
    needs_execute = SCons.Taskmaster.Task.execute_always
```

Overrides: SCons.Taskmaster.Task.needs_execute

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.

trace_message(self, method, node, description='node')

31.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

31.7 Class QuestionTask

object

SCons.Taskmaster.Task

SCons.Taskmaster.AlwaysTask

SCons.Script.Main.QuestionTask
An SCons task for the -q (question) option.

### 31.7.1 Methods

**prepare**(self)

Called just before the task is executed.

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

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

```
x.__delattr__(‘name’) <=> del x.name
```

```
x.__getattr__(‘name’) <=> x.name
```

```
hash(x)
```
Class QuestionTask

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

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

__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)

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.
Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().
### fail_stop(self)

Explicit stop-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

### failed(self)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

### 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)

Always returns True (indicating this Task should always be executed).

Subclasses that need this behavior (as opposed to the default of only executing Nodes that are out of date w.r.t. their dependencies) can use this as follows:

```python
class MyTaskSubclass(SCons.Taskmaster.Task):
    needs_execute = SCons.Taskmaster.Task.execute_always
```

Overrides: SCons.Taskmaster.Task.needs_execute

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.

trace_message(self, method, node, description='node')

31.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

31.8 Class TreePrinter

object

SCons.Script.Main.TreePrinter

31.8.1 Methods

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

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

Overrides: object.__init__(inherited documentation)

get_all_children(self, node)

get-derived_children(self, node)
display(self, t)

_delattr_(...)

x._delattr_('name') ==> del x.name

_getattribute_(...)

x._getattribute_('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)

### 31.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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

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

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

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

_hash_(x)  
hash(x)

_init_(...)  
x._init_(...) initializes x; see x._class_._doc_ for signature

_new_(T, S, ...)  
Return Value  
a new object with type S, a subtype of T

_reduce_(...)  
helper for pickle

_reduce_ex_(...)  
helper for pickle
```
31.9.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

31.9.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>values</td>
<td>Value: FakeOptionValues()</td>
</tr>
</tbody>
</table>

31.10 Class Stats

```
object
    SCons.Script.Main.Stats
```


31.10.1 Methods

```
__init__(self)
```

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

Overrides: object.__init__ (inherited documentation)

```
enable(self, outfp)
```

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

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

31.11 Class CountStats

```
object
  SCons.Script.Main.Stats
      SCons.Script.Main.CountStats
```

31.11.1 Methods

```python
do_append(self, label)
do_print(self)
```
Class CountStats

Module SCons.Script.Main

```python
x.__delattr__('name')  # del x.name
```

```python
x.__getattribute__('name')  # x.name
```

```python
__hash__(x)
```

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

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

Return Value

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

```python
__reduce__(...)
```

helper for pickle

```python
__reduce_ex__(...)
```

helper for pickle

```python
__repr__(x)
```

repr(x)

```python
__setattr__(...)
```

x.__setattr__('name', value)  # x.name = value

```python
__str__(x)
```

str(x)

```python
do_nothing(self, *args, **kw)
```

```python
enable(self, outfp)
```

31.11.2 Properties

417
31.12 Class MemStats

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

31.12.1 Methods

```python
do_append(self, label)
do_print(self)

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

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

__hash__(x)
hash(x)

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

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

__reduce__(...)
helper for pickle

__reduce_ex__(...)
helper for pickle
```
```python
__repr__(x)
repr(x)

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

__str__(x)  
str(x)

do_nothing(self, *args, **kw)

enable(self, outfp)

31.12.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
```
32 Module SCons.Script.SConscript’

SCons.Script.SConscript

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

32.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_calling_namespaces()</td>
<td>Return the locals and globals for the function that called into this module in the current call stack.</td>
</tr>
<tr>
<td>compute_exports(exports)</td>
<td>Compute a dictionary of exports given one of the parameters to the Export() function or the exports argument to SConscript().</td>
</tr>
<tr>
<td>SConscript_exception(file=sys.stdout)</td>
<td>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.</td>
</tr>
<tr>
<td>annotate(node)</td>
<td>Annotate a node with the stack frame describing the SConscript file and line number that created it.</td>
</tr>
<tr>
<td>Configure(*args, **kw)</td>
<td>Configure(*args, **kw)</td>
</tr>
<tr>
<td>get_DefaultEnvironmentProxy()</td>
<td>Get the default environment proxy.</td>
</tr>
<tr>
<td>BuildDefaultGlobals()</td>
<td>Create a dictionary containing all the default globals for SConstruct and SConscript files.</td>
</tr>
</tbody>
</table>

32.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>revision</em></td>
<td>Value: 'src/engine/SCons/Script/SConscript.py 5023 2010/06/14 22...</td>
</tr>
</tbody>
</table>

continued on next page
### 32.3 Class SConscriptReturn

```python
class SConscriptReturn:
    # Class SConscriptReturn
```

#### 32.3.1 Methods

- **delattr**
  ```python
def _delattr_(self, name):
    del self.name
    
    # Overrides: object._delattr_
  ```

- **getattribute**
  ```python
def _getattribute_(self, name):
    return self.name
    
    # Overrides: object._getattribute_
  ```

- **getitem**
  ```python
def _getitem_(self, x, y):
    return self[x]
```

- **getslice**
  ```python
def _getslice_(self, x, i, j):
    return self[x[i:j]]
    
    # Use of negative indices is not supported.
```

- **hash**
  ```python
def _hash_(self):
    return hash(self)
```

- **init**
  ```python
def _init_(self, *args, **kwargs):
    # _init_(...) initializes self; see _class_.__doc__ for signature
    
    # Overrides: exceptions.BaseException._init_
  ```
Class Frame

Module SCons.Script.SConscript

__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__ (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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>

32.4 Class Frame

```
object

SCons.Script.SConscript'.Frame
```

A frame on the SConstruct/SConscript call stack
32.4.1 Methods

```
__init__(self, fs, exports, sconscript)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)
```

```
__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)
```

32.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
Class SConsEnvironment

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)

Default(self, *targets)

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

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

Exit(self, value=0)

Export(self, *vars, **kw)

GetLaunchDir(self)

GetOption(self, name)

Help(self, text)

Import(self, *vars)

SConscript(self, *ls, **kw)

SConscriptChdir(self, flag)

SetOption(self, name, value)
<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Action</code></td>
<td><code>self(*args,**kw)</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>AddMethod</code></td>
<td><code>self(function, name=False)</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>AddPostAction</code></td>
<td><code>self(files, action)</code></td>
<td></td>
</tr>
<tr>
<td><code>AddPreAction</code></td>
<td><code>self(files, action)</code></td>
<td></td>
</tr>
<tr>
<td><code>Alias</code></td>
<td><code>self(target, source=[], action=False,**kw)</code></td>
<td></td>
</tr>
<tr>
<td><code>AlwaysBuild</code></td>
<td><code>self(*targets)</code></td>
<td></td>
</tr>
<tr>
<td><code>Append</code></td>
<td><code>self(**kw)</code></td>
<td>Append values to existing construction variables in an Environment.</td>
</tr>
<tr>
<td><code>AppendENVPath</code></td>
<td><code>self(name, newpath, envname='ENV', sep=':?', delete_existing=1)</code></td>
<td>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).</td>
</tr>
<tr>
<td><code>AppendUnique</code></td>
<td><code>self(delete_existing=0,**kw)</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>BuildDir</code></td>
<td><code>self(*args,**kw)</code></td>
<td></td>
</tr>
<tr>
<td><code>Builder</code></td>
<td><code>self(**kw)</code></td>
<td></td>
</tr>
<tr>
<td><code>CacheDir</code></td>
<td><code>self(path)</code></td>
<td></td>
</tr>
<tr>
<td><code>Clean</code></td>
<td><code>self(targets, files)</code></td>
<td></td>
</tr>
</tbody>
</table>
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)
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment(self, **kw)</td>
<td></td>
</tr>
<tr>
<td>Execute(self, action, *args, **kw)</td>
<td>Directly execute an action through an Environment</td>
</tr>
<tr>
<td>File(self, name, *args, **kw)</td>
<td></td>
</tr>
<tr>
<td>FindFile(self, file, dirs)</td>
<td></td>
</tr>
<tr>
<td>FindInstalledFiles(self)</td>
<td>returns the list of all targets of the Install and InstallAs Builder.</td>
</tr>
<tr>
<td>FindIxes(self, paths, prefix, suffix)</td>
<td>Search a list of paths for something that matches the prefix and suffix.</td>
</tr>
<tr>
<td>FindSourceFiles(self, node='.’)</td>
<td>returns a list of all source files.</td>
</tr>
<tr>
<td>Flatten(self, sequence)</td>
<td></td>
</tr>
<tr>
<td>GetBuildPath(self, files)</td>
<td></td>
</tr>
<tr>
<td>Glob(self, pattern, ondisk=True, source=False, strings=False)</td>
<td></td>
</tr>
<tr>
<td>Ignore(self, target, dependency)</td>
<td>Ignore a dependency.</td>
</tr>
<tr>
<td>Literal(self, string)</td>
<td></td>
</tr>
<tr>
<td>Local(self, *targets)</td>
<td></td>
</tr>
</tbody>
</table>
**Class SConsEnvironment**

**Module SCons.Script.SConscript**

### MergeFlags(self, args, unique=1, 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=1)

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

### ParseDepends(self, filename, must_exist=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

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

Precious

Prepend

Prepend values to existing construction variables in an Environment.

PrependENVPath

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

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

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

Replace

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

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

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

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.
### Class SConsEnvironment

Initialize 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__
### Class `SConsEnvironment`

**Module `SCons.Script.SConscript`**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `_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` |
| `_setitem_(self, key, value)` |  |
| `_str_(x)` | `str(x)` |
| `arg2nodes(self, args, node_factory=<class 'SCons.Environment.Null'>, lookup_list=<class 'SCons.Environment.Null'>, **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. |
Class SConsEnvironment

Module SCons.Script.SConscript'

get.Scanner(self, key)
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, executor=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, executor=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.
class DefaultEnvironmentCall(object):

    __init__(self, method_name, subst=0)

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

    Overrides: object.__init__(inherited documentation)

    __call__(self, *args, **kw)

        subst_target_source(self, string, raw=0, target=False, source=False, conv=False, executor=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.

    | Name           | Description                                           |
    |---------------|-------------------------------------------------------|
    | __class__     | Value: <attribute '__class__' of 'object' objects>    |

    32.5.3 Class Variables

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

    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)

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

    Overrides: object.__init__(inherited documentation)
### 32.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: &lt;attribute <code>__class__</code> of <code>object</code> objects&gt;</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/Sig.py 5023 2010/06/14 22:05:46 scons'</td>
</tr>
<tr>
<td><strong>doc</strong></td>
<td>Value: &quot;Place-holder for the old SCons.Sig module hierarchy...&quot;</td>
</tr>
<tr>
<td>msg</td>
<td>Value: 'The SCons.Sig module no longer exists. Remove the f...</td>
</tr>
<tr>
<td>default_calc</td>
<td>Value: False</td>
</tr>
<tr>
<td>default_module</td>
<td>Value: False</td>
</tr>
<tr>
<td>MD5</td>
<td>Value: MD5Null()</td>
</tr>
<tr>
<td>TimeStamp</td>
<td>Value: TimeStampNull()</td>
</tr>
</tbody>
</table>

33.2 Class MD5Null

object     └── SCons.Util.Null
            └── SCons.Sig.MD5Null

33.2.1 Methods

```python
__repr__ (self)
Overrides: SCons.Util.Null.__repr__
```

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

```python
__delattr__ (self, name)
x.__delattr__('name') ⇔ del x.name
Overrides: object.__delattr__ (inherited documentation)
```

```python
__getattr__ (self, name)
```
Class MD5Null

Module SCons.Sig

__getattribute__(...)  
x.__getattribute__('name') == x.name

__hash__(x)  
hash(x)

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

__new__(cls, *args, **kwargs)  
Return Value  
a new object with type S, a subtype of T  
 Overrides: object.__new__(inherited documentation)

__nonzero__(self)  

__reduce__(...)  
helper for pickle

__reduce_ex__(...)  
helper for pickle

__setattr__(self, name, value)  
x.__setattr__('name', value) == x.name = value  
 Overrides: object.__setattr__(inherited documentation)

__str__(x)  
str(x)

33.2.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
### 33.3 Class TimeStampNull

object

SCons.Util.Null

SCons.Sig.TimeStampNull

#### 33.3.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call(self, *args, **kwargs)</td>
<td></td>
</tr>
</tbody>
</table>
| delattr(self, name) | x.delattr('name') <=> del x.name  
                             Overrides: object.delattr extended (inherited documentation) |
| getattr(self, name) |                                                                 |
| getattribute(...) | x.getattribute('name') <=> x.name                                           |
| hash(x)         |                                                                 |
| init(self, *args, **kwargs) | x.init(...) initializes x; see x.class_.doc for signature  
                             Overrides: object.init extended (inherited documentation) |
| new(cls, *args, **kwargs) | Return Value:  
                                a new object with type S, a subtype of T  
                             Overrides: object.new extended (inherited documentation) |
| nonzero(self)   |                                                                 |
| reduce(...)     | helper for pickle                                                            |
| reduce_ex(...)  | helper for pickle                                                            |
```python
setattr(self, name, value)
x.__setattr__(name, value) #=> x.name = value
Overrides: object.__setattr__(inherited documentation)
```

```python
str(x)
```

### 33.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

Module SCons.Subst

SCons.Subst

SCons string substitution.

34.1 Functions

SetAllowableExceptions(*excepts)

raise_exception(exception, target, s)

quote_spaces(arg)

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

escape_list(mylist, escape_func)

Escape a list of arguments by running the specified escape_func on every object in the list that has an escape() method.

subst_dict(target, source)

Create a dictionary for substitution of special construction variables.

This translates the following special arguments:

target - the target (object or array of objects), used to generate the TARGET and TARGETS construction variables

source - the source (object or array of objects), used to generate the SOURCES and SOURCE construction variables

scons_subst(strSubst, env, mode=1, target=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**

```python
scons_subst_list(strSubst, env, mode=1, 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**

```python
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:

```python
env2 = env.Clone(CCFLAGS = '$CCFLAGS -g')
```

We do this with some straightforward, brute-force code here...

### 34.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__revision__</code></td>
<td>Value: <code>src/engine/SCons/Subst.py 5023 2010/06/14 22:05:46 scons</code></td>
</tr>
<tr>
<td>AllowableExceptions</td>
<td>Value: <code>&lt;type 'exceptions.IndexError'&gt;, &lt;type 'exceptions.NameError'&gt;...</code></td>
</tr>
<tr>
<td>NullNodesList</td>
<td>Value: <code>Null(0x843A40C)</code></td>
</tr>
<tr>
<td>SUBST_CMD</td>
<td>Value: <code>0</code></td>
</tr>
<tr>
<td>SUBST_RAW</td>
<td>Value: <code>1</code></td>
</tr>
<tr>
<td>SUBST_SIG</td>
<td>Value: <code>2</code></td>
</tr>
</tbody>
</table>

### 34.3 Class Literal

**object**

```python
SCons.Subst.Literal
```

A wrapper for a string. If you use this object wrapped around a string, then it will be interpreted as literal. When passed to the command interpreter, all special characters will be escaped.
### 34.3.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code>(<code>self</code>, <code>lstr</code>)</td>
<td>x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.doc__ for signature</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<strong>init</strong>.extit(inherited documentation)</td>
</tr>
<tr>
<td><code>__str__</code>( <code>self</code>)</td>
<td>str(x)</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<strong>str</strong>.extit(inherited documentation)</td>
</tr>
<tr>
<td><code>escape</code>(<code>self</code>, <code>escape_func</code>)</td>
<td></td>
</tr>
<tr>
<td><code>for_signature</code>(<code>self</code>)</td>
<td></td>
</tr>
<tr>
<td><code>is_literal</code>(<code>self</code>)</td>
<td></td>
</tr>
<tr>
<td><code>__delattr__</code>( <code>...</code>)</td>
<td>x.<strong>delattr</strong>('name') == del x.name</td>
</tr>
<tr>
<td><code>__getattr__</code>( <code>...</code>)</td>
<td>x.<strong>getattr</strong>('name') == x.name</td>
</tr>
<tr>
<td><code>__hash__</code>( <code>x</code>)</td>
<td>hash(x)</td>
</tr>
<tr>
<td><code>__new__</code>( <code>T</code>, <code>S</code>, ...)</td>
<td>Return Value</td>
</tr>
<tr>
<td></td>
<td>a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>__reduce__</code>( <code>...</code>)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__</code>( <code>...</code>)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__repr__</code>( <code>x</code>)</td>
<td>repr(x)</td>
</tr>
<tr>
<td><code>__setattr__</code>( <code>...</code>)</td>
<td>x.<strong>setattr</strong>('name', value) == x.name = value</td>
</tr>
</tbody>
</table>
34.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_class_</code></td>
<td>Value: &lt;attribute <code>_class_</code> of <code>object</code> objects&gt;</td>
</tr>
</tbody>
</table>

34.4 Class SpecialAttrWrapper

```
object

SCons.Subst.SpecialAttrWrapper
```

This is a wrapper for what we call a 'Node special attribute.'

This is any of the attributes of a Node that we can reference from
Environment variable substitution, such as `$TARGET.abspath` or `$SOURCES[1].filebase`. We implement the same methods as `Literal`
so we can handle special characters, plus a `for_signature` method,
such that we can return some canonical string during signature
calculation to avoid unnecessary rebuilds.

34.4.1 Methods

```
__init__(self, lstr, for_signature=False)

The for_signature parameter, if supplied, will be the
canonical string we return from for_signature(). Else
we will simply return lstr.
Overrides: object.__init__
```

```
__str__(self)
str(x)

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

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

```
for_signature(self)
```

```
is_literal(self)
```

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

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

```
__hash__(x)
hash(x)
```

443
Class CmdStringHolder

Module SCons.Subst

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

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

```python
__reduce__(...)

helper for pickle
```

```python
__reduce_ex__(...)

helper for pickle
```

```python
__repr__(x)

repr(x)
```

```python
__setattr__(...)

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

### 34.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

### 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

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

Overrides: UserString.UserString.__init__
```

```python
is_literal(self)
```
**escape**(self, escape_func, quote_func=<function quote_spaces at 0x843448c>)

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)
isdigit(self)
isdecimal(self)
islower(self)
isnumeric(self)
isupper(self)
join(self, seq)
ljust(self, width, *args)
lower(self)
isspace(self)
istitle(self)
partition(self, sep)
replace(self, old, new, maxsplit=-1)
rfind(self, sub, start=0, end=2147483647)
### 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

```python
_init__(self, list, func)
x._init__(...) initializes x; see x._class__._doc_ for signature
Overrides: object._init_  (inherited documentation)
```

```python
_delattr__(...)  
x._delattr_('name')  = del x.name
```

```python
_getattribute__(...)  
x._getattribute_('name')  = x.name
```

```python
_hash__(x)  
hash(x)
```

```python
_new__(T, S, ...)  
Return Value
  a new object with type S, a subtype of T
```

```python
_reduce__(...)  
helper for pickle
```

```python
_reduce_ex__(...)  
helper for pickle
```

```python
_repr__(x)  
repr(x)
```

```python
_setattr__(...)  
x._setattr_('name', value)  = x.name = value
```

```python
_str__(x)  
str(x)
```

34.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '.<em>class</em>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
34.7 Class Targets_or_Sources

UserList.UserList → SCons.SubstTargets_or_Sources

A class that implements $TARGETS or $SOURCES expansions by in turn wrapping a NLWrapper. This class handles the different methods used to access the list, calling the NLWrapper to create proxies on demand.

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

34.7.1 Methods

_init_ (self, nl)
Overrides: UserList.UserList._init_

_getattr_ (self, attr)

_getitem_ (self, i)
Overrides: UserList.UserList._getitem_

_getslice_ (self, i, j)
Overrides: UserList.UserList._getslice_

_str_ (self)

_repr_ (self)
Overrides: UserList.UserList._repr_

_add_ (self, other)

_cmp_ (self, other)

_contains_ (self, item)

_delitem_ (self, i)

_delslice_ (self, i, j)

_eq_ (self, other)

_ge_ (self, other)

_gt_ (self, other)
34.8 Class Target_or_Source

```
object  └── SCons.Subst.Target_or_Source
```
A class that implements $TARGET or $SOURCE expansions by in turn wrapping a NLWrapper. This class handles the different methods used to access an individual proxy Node, calling the NLWrapper to create a proxy on demand.

### 34.8.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code></td>
<td>(self, nl) x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature. Overrides: object.<strong>init</strong>(inherited documentation)</td>
</tr>
<tr>
<td><code>__getattr__</code></td>
<td>(self, attr)</td>
</tr>
<tr>
<td><code>__str__</code></td>
<td>(self) str(x) Overrides: object.<strong>str</strong>(inherited documentation)</td>
</tr>
<tr>
<td><code>__repr__</code></td>
<td>(self) repr(x) Overrides: object.<strong>repr</strong>(inherited documentation)</td>
</tr>
<tr>
<td><code>__delattr__</code></td>
<td>(...) x.<strong>delattr</strong>('name') == del x.name</td>
</tr>
<tr>
<td><code>__getattribute__</code></td>
<td>(...) x.<strong>getattribute</strong>('name') == x.name</td>
</tr>
<tr>
<td><code>__hash__</code></td>
<td>(x) hash(x)</td>
</tr>
<tr>
<td><code>__new__</code></td>
<td>(T, S, ...) Return Value a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>__reduce__</code></td>
<td>(...) helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__</code></td>
<td>(...) helper for pickle</td>
</tr>
</tbody>
</table>
34.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

34.9 Class NullNodeList

object

SCons.Util.Null

SCons.Util.NullSeq

SCons.Subst.NullNodeList

34.9.1 Methods

__call__(self, *args, **kwargs)
Overrides: SCons.Util._call_

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

__delattr__(self, name)
x.__delattr__('name') #=> del x.name
Overrides: object.__delattr__ (inherited documentation)

__delitem__(self, i)

__getattr__(self, name)

__getattribute__(...)
x.__getattribute__('name') #=> x.name

__getitem__(self, i)

__hash__(x)
hash(x)
Class NullNodeList

Module SCons.Subst

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

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

Overrides: object.__init__ (inherited documentation)

```python
__iter__(self)
```

```python
__len__(self)
```

```python
__new__(cls, *args, **kwargs)
```

Return Value

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

Overrides: object.__new__ (inherited documentation)

```python
__nonzero__(self)
```

```python
__reduce__(...)
```

Helper for pickle

```python
__reduce_ex__(...)
```

Helper for pickle

```python
__repr__(self)
```

`repr(x)`

Overrides: object.__repr__ (inherited documentation)

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

```python
x.__setattr__('name', value) == x.name = value
```

Overrides: object.__setattr__ (inherited documentation)

```python
__setitem__(self, i, v)
```

34.9.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>
35 Module SCons.Taskmaster

Generic Taskmaster module for the SCons build engine.

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

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

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

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

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

35.1 Functions

dump_stats()

find_cycle(stack, visited)

35.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value: ...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>doc</strong></td>
<td>'src/engine/SCons/Taskmaster.py 5023 2010/06/14 22:05:46 ...</td>
</tr>
<tr>
<td><strong>revision</strong></td>
<td>{0: 'no_state', 1: 'pending', 2: 'executing', 3: 'up_to_d...'</td>
</tr>
<tr>
<td>NODE_NO_STATE</td>
<td>Value: 0</td>
</tr>
<tr>
<td>NODE_PENDING</td>
<td>Value: 1</td>
</tr>
<tr>
<td>NODE_EXECUTING</td>
<td>Value: 2</td>
</tr>
<tr>
<td>NODE_UP_TO_DATE</td>
<td>Value: 3</td>
</tr>
<tr>
<td>NODE_EXECUTED</td>
<td>Value: 4</td>
</tr>
<tr>
<td>NODE_FAILED</td>
<td>Value: 5</td>
</tr>
</tbody>
</table>

continued on next page
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

__init__(self)
Instantiates a Taskmaster.Stats object, initializing all appropriate counters to zero.
Overrides: object.__init__

__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
35.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td><code>Value: &lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

35.4 Class Task

```
object ->  SCons.Taskmaster.Task
```

**Known Subclasses:** SCons.Taskmaster.AlwaysTask, SCons.Taskmaster.OutOfDateTask

Default SCons build engine task.

This controls the interaction of the actual building of node and the rest of the engine.

This is expected to handle all of the normally-customizable aspects of controlling a build, so any given application *should* be able to do what it wants by sub-classing this class and overriding methods as appropriate. If an application needs to 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

```python
__init__(self, tm, targets, top, node)
```

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

```python
trace_message(self, method, node, description='node')
```

```python
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.

```python
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.

```python
get_target(self)
```

Fetch the target being built or updated by this task.

```python
needs_execute(self)
```

```python
execute(self)
```

Called to execute the task.

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

Called when the task has been successfully executed and the Taskmaster instance doesn’t want to call the Node’s callback methods.

executed_with_callbacks(self)

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

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

executed(self)

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

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

failed(self)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().
fail_stop(self)

Explicit stop-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

fail_continue(self)

Explicit continue-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

make_ready_all(self)

Marks all targets in a task ready for execution.

This is used when the interface needs every target Node to be visited--the canonical example being the "scons -c" option.

make_ready_current(self)

Marks all targets in a task ready for execution if any target is not current.

This is the default behavior for building only what’s necessary.

make_ready(self)

Marks all targets in a task ready for execution if any target is not current.

This is the default behavior for building only what’s necessary.
postprocess(self)

Post-processes a task after it’s been executed.

This examines all the targets just built (or not, we don’t care if the build was successful, or even if there was no build because everything was up-to-date) to see if they have any waiting parent Nodes, or Nodes waiting on a common side effect, that can be put back on the candidates list.

exc_info(self)

Returns info about a recorded exception.

exc_clear(self)

Clears any recorded exception.

This also changes the "exception_raise" attribute to point to the appropriate do-nothing method.

exception_set(self, exception=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

__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
Class AlwaysTask

Module SCons.Taskmaster

```
__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)
```

35.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

35.5 Class AlwaysTask

```
object
 SCons.Taskmaster.Task
    SCons.Taskmaster.AlwaysTask

```
35.5.1 Methods

needs_execute(self)

Always returns True (indicating this Task should always be executed).

Subclasses that need this behavior (as opposed to the default of only executing Nodes that are out of date w.r.t. their dependencies) can use this as follows:

```python
class MyTaskSubclass(SCons.Taskmaster.Task):
    needs_execute = SCons.Taskmaster.Task.execute_always
```

Overrides: SCons.Taskmaster.Task.needs_execute

__delattr__(...)  
```python
x.__delattr__('name') == del x.name
```__getattribute__(...)  
```python
x.__getattribute__('name') == x.name
```__hash__(x)  
```python
hash(x)
```__init__(self, tm, targets, top, node)  
```python
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
```

Overrides: object.__init__(inherited documentation)

__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)  
```python
repr(x)
```
Class AlwaysTask

Module SCons.Taskmaster

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

```python
__str__(x)  
str(x)
```

```python
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.
```

```python
exc_clear(self)  
Clears any recorded exception.  
This also changes the \"exception_raise\" attribute to point  
to the appropriate do-nothing method.
```

```python
exc_info(self)  
Returns info about a recorded exception.
```

```python
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
```

```python
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().
```

463
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.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().
### fail_stop(self)

Explicit stop-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

### failed(self)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().

### 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.
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.

trace_message(self, method, node, description='node')

35.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

35.6 Class OutOfDateTask

object

SCons.Taskmaster.Task

SCons.Taskmaster.OutOfDateTask

Known Subclasses: SCons.Script.Main.BuildTask

35.6.1 Methods

needs_execute(self)

Returns True (indicating this Task should be executed) if this Task’s target state indicates it needs executing, which has already been determined by an earlier up-to-date check.

Overrides: SCons.Taskmaster.Task.needs_execute
### Class OutOfDateTask

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>delattr</strong>(...)</td>
<td><code>x.__delattr__('name')</code> &lt;=&gt; del x.name</td>
</tr>
<tr>
<td><strong>getattr</strong>(...)</td>
<td><code>x.__getattr__('name')</code> &lt;=&gt; x.name</td>
</tr>
<tr>
<td><strong>hash</strong>(x)</td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><strong>init</strong>(self, tm, targets, top, node)</td>
<td><code>x.__init__(...)</code> initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature</td>
</tr>
<tr>
<td><strong>new</strong>(T, S, ...)</td>
<td>Return Value: a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><strong>reduce</strong>(...)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><strong>reduce_ex</strong>(...)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><strong>repr</strong>(x)</td>
<td><code>repr(x)</code></td>
</tr>
<tr>
<td><strong>setattr</strong>(...)</td>
<td><code>x.__setattr__('name', value)</code> &lt;=&gt; x.name = value</td>
</tr>
<tr>
<td><strong>str</strong>(x)</td>
<td><code>str(x)</code></td>
</tr>
</tbody>
</table>
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

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

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

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

Called when the task has been successfully executed and the Taskmaster instance doesn’t want to call the Node’s callback methods.

fail_continue

Explicit continue-the-build failure.

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

Note: Although this function is normally invoked on nodes in the executing state, it might also be invoked on up-to-date nodes when using Configure().
fail_stop(self)

Explicit stop-the-build failure.

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

Note: Although this function is normally invoked on nodes in
the executing state, it might also be invoked on up-to-date
nodes when using Configure().

failed(self)

Default action when a task fails: stop the build.

Note: Although this function is normally invoked on nodes in
the executing state, it might also be invoked on up-to-date
nodes when using Configure().

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.
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.

trace_message(self, method, node, description='node')

35.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

35.7 Class Taskmaster

SCons.Taskmaster.Taskmaster

The Taskmaster for walking the dependency DAG.

35.7.1 Methods

__init__(self, targets=[], tasker=False, order=False, trace=False)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrrides: object.__init__ (inherited documentation)
find_next_candidate(self)

Returns the next candidate Node for (potential) evaluation.

The candidate list (really a stack) initially consists of all of the top-level (command line) targets provided when the Taskmaster was initialized. While we walk the DAG, visiting Nodes, all the children that haven’t finished processing get pushed on to the candidate list. Each child can then be popped and examined in turn for whether *their* children are all up-to-date, in which case a Task will be created for their actual evaluation and potential building.

Here is where we also allow candidate Nodes to alter the list of Nodes that should be examined. This is used, for example, when invoking SCons in a source directory. A source directory Node can return its corresponding build directory Node, essentially saying, "Hey, you really need to build this thing over here instead."

no_next_candidate(self)

Stops Taskmaster processing by not returning a next candidate.

Note that we have to clean-up the Taskmaster candidate list because the cycle detection depends on the fact all nodes have been processed somehow.

trace_message(self, message)

trace_node(self, node)

next_task(self)

Returns the next task to be executed.

This simply asks for the next Node to be evaluated, and then wraps it in the specific Task subclass with which we were initialized.

will_not_build(self, nodes, node_func=<function <lambda> at 0x873a3e4>)

Perform clean-up about nodes that will never be built. Invokes a user defined function on all of these nodes (including all of their parents).
### stop(self)

Stops the current build completely.

### cleanup(self)

Check for dependency cycles.

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

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

```python
hash(x)
```

```python
_return__(T, S, ...)
```

**Return Value**

A new object with type S, a subtype of T

```python
_help__(...)
```

**Helper for pickle**

```python
_help__ex__(...)
```

**Helper for pickle**

```python
__repr__(x)
```

```python
__setattr__(...)
```

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

```python
__str__(x)
```

### 35.7.2 Properties
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>
Module SCons.Util

SCons.Util

Various utility functions go here.

36.1 Functions

**dictify(keys, values, result={})**

**rightmost_separator(path, sep)**

**containsAny(str, set)**
Check whether sequence str contains ANY of the items in set.

**containsAll(str, set)**
Check whether sequence str contains ALL of the items in set.

**containsOnly(str, set)**
Check whether sequence str contains ONLY items in set.

**splitext(path)**
Same as os.path.splitext() but faster.

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

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

**Module SCons.Util**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>render_tree</code></td>
<td>Render a tree of nodes into an ASCII tree view.</td>
</tr>
<tr>
<td><code>print_tree</code></td>
<td>Print a tree of nodes. This is like <code>render_tree</code>, except it prints lines</td>
</tr>
<tr>
<td></td>
<td>directly instead of creating a string representation in memory, so that</td>
</tr>
<tr>
<td></td>
<td>huge trees can be printed.</td>
</tr>
<tr>
<td><code>is_Dict</code></td>
<td>Check if an object is a dictionary.</td>
</tr>
<tr>
<td><code>is_List</code></td>
<td>Check if an object is a list.</td>
</tr>
<tr>
<td><code>is_Sequence</code></td>
<td>Check if an object is a sequence.</td>
</tr>
<tr>
<td><code>is_Tuple</code></td>
<td>Check if an object is a tuple.</td>
</tr>
<tr>
<td><code>is_String</code></td>
<td>Check if an object is a string.</td>
</tr>
<tr>
<td><code>is_Scalar</code></td>
<td>Check if an object is a scalar.</td>
</tr>
<tr>
<td><code>do_flatten</code></td>
<td>Flatten a sequence recursively.</td>
</tr>
</tbody>
</table>

### Code Examples

```python
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.

```python
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.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten</td>
<td>Flatten a sequence to a non-nested list.</td>
<td>flatten(obj, isinstance=&lt;built-in function isinstance&gt;, StringTypes=...</td>
</tr>
<tr>
<td>flatten_sequence</td>
<td>Flatten a sequence to a non-nested list.</td>
<td>flatten_sequence(sequence, isinstance=&lt;built-in function isinstance&gt;...</td>
</tr>
<tr>
<td>to_String</td>
<td>Convert to string.</td>
<td>to_String(s, isinstance=&lt;built-in function isinstance&gt;, str=&lt;type 'str'&gt;,...</td>
</tr>
<tr>
<td>to_String_for_subst</td>
<td>Convert to string for substitution.</td>
<td>to_String_for_subst(s, isinstance=&lt;built-in function isinstance&gt;, str=&lt;type 'str'&gt;,...</td>
</tr>
<tr>
<td>to_String_for_signature</td>
<td>Convert to string for signature.</td>
<td>to_String_for_signature(obj, to_String_for_subst=&lt;function to_String_for_subst at 0x842217c&gt;, AttributeError=&lt;type 'exceptions.AttributeError'&gt;)</td>
</tr>
<tr>
<td>semi_deepcopy</td>
<td>Deep copy of an object.</td>
<td>semi_deepcopy(x)</td>
</tr>
<tr>
<td>RegGetValue</td>
<td>Get a value from the registry.</td>
<td>RegGetValue(root, key)</td>
</tr>
<tr>
<td>RegOpenKeyEx</td>
<td>Open a key in the registry.</td>
<td>RegOpenKeyEx(root, key)</td>
</tr>
<tr>
<td>WhereIs</td>
<td>Find a file in a given path.</td>
<td>WhereIs(file, path=False, pathext=False, reject=[])</td>
</tr>
</tbody>
</table>
**PrependPath***(oldpath, newpath, sep=':’, delete_existing=1, canonicalize=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 os.path.normpath and os.path.normcase all paths to help assure this. This can also handle the case where the given old path variable is a list instead of a string, in which case a list will be returned instead of a string.

Example:
Old Path: "/foo/bar:/foo"
New Path: "/biz/boom:/foo"
Result: "/biz/boom:/foo:/foo/bar"

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

If canonicalize is not None, it is applied to each element of newpath before use.

**AppendPath***(oldpath, newpath, sep=':’, delete_existing=1, canonicalize=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 os.path.normpath and os.path.normcase all paths to help assure this. This can also handle the case where the given old path variable is a list instead of a string, in which case a list will be returned instead of a string.

Example:
Old Path: "/foo/bar:/foo"
New Path: "/biz/boom:/foo"
Result: "/foo/bar:/biz/boom:/foo"

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

If canonicalize is not None, it is applied to each element of newpath before use.

**get_native_path**(path)

Transforms an absolute path into a native path for the system. Non-Cygwin version, just leave the path alone.

**Split**(arg)

**case_sensitive_suffixes**(s1, s2)
adjustixes(fname, pre, suf, ensure_suffix=False)

unique(s)

Return a list of the elements in s, but without duplicates.

For example, unique([1,2,3,1,2,3]) is some permutation of [1,2,3],
unique("abcabc") some permutation of ["a", "b", "c"], and
unique(((1, 2), [2, 3], [1, 2])) some permutation of
[[2, 3], [1, 2]].

For best speed, all sequence elements should be hashable. Then
unique() will usually work in linear time.

If not possible, the sequence elements should enjoy a total
ordering, and if list(s).sort() doesn’t raise TypeError it’s
assumed that they do enjoy a total ordering. Then unique() will
usually work in $O(N\cdot\log_2(N))$ time.

If that’s not possible either, the sequence elements must support
equality-testing. Then unique() will usually work in quadratic
time.

uniquer(seq, idfun=False)

uniquer_hashables(seq)

make_path_relative(path)

makes an absolute path name to a relative pathname.

AddMethod(obj, function, name=False)

Adds either a bound method to an instance or an unbound method to
a class. If name is ommited the name of the specified function
is used by default.

Example:
    a = A()
def f(self, x, y):
        self.z = x + y
    AddMethod(f, A, "add")
a.add(2, 4)
print a.z
AddMethod(lambda self, i: self.l[i], a, "listIndex")
print a.listIndex(5)
**Variables**

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DictTypes</td>
<td>DictTypes</td>
</tr>
<tr>
<td>ListTypes</td>
<td>list, UserList</td>
</tr>
<tr>
<td>SequenceTypes</td>
<td>(&lt;type 'list'&gt;, &lt;type 'tuple'&gt;, &lt;class UserList.UserList ...</td>
</tr>
<tr>
<td>StringTypes</td>
<td>(&lt;type 'str'&gt;, &lt;type 'unicode'&gt;, &lt;class UserString.UserSt...</td>
</tr>
<tr>
<td>BaseStringTypes</td>
<td>(&lt;type 'str'&gt;, &lt;type 'unicode'&gt;)</td>
</tr>
<tr>
<td>d</td>
<td>{&lt;type 'instance'&gt;: &lt;function _semi_deepcopy_inst at 0x84...</td>
</tr>
<tr>
<td>can_read_reg</td>
<td>Value: 0</td>
</tr>
<tr>
<td>hkey_mod</td>
<td>win32con</td>
</tr>
<tr>
<td>RegEnumKey</td>
<td>win32api.RegEnumKey</td>
</tr>
<tr>
<td>RegEnumValue</td>
<td>win32api.RegEnumValue</td>
</tr>
<tr>
<td>RegQueryValueEx</td>
<td>win32api.RegQueryValueEx</td>
</tr>
<tr>
<td>HKEY_CLASSES_ROOT</td>
<td>False</td>
</tr>
<tr>
<td>HKEY_LOCAL_MACHINE</td>
<td>False</td>
</tr>
<tr>
<td>HKEY_CURRENT_USER</td>
<td>False</td>
</tr>
<tr>
<td>HKEY_USERS</td>
<td>False</td>
</tr>
<tr>
<td>display</td>
<td>SCons.Util.display</td>
</tr>
<tr>
<td>md5</td>
<td>True</td>
</tr>
</tbody>
</table>
36.3 Class 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:

```python
class someList = NodeList([ 'foo', 'bar'])
class someList.strip()
['foo', 'bar']
```

36.3.1 Methods

- `_nonzero_(self)`
- `_str_(self)`
- `_iter_(self)`
- `_call_(self, *args, **kwargs)`
- `_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)`
Class NodeList

- `__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, **kwds)`
### 36.4 Class DisplayEngine

```python
class DisplayEngine(object):
    def __init__(self, mode):
        pass

def call(self, text, append_newline=1):
    pass

def set_mode(self, mode):
    pass

def delattr(self, name):
    del self.name

def getattr(self, name):
    return self.name

def hash(self):
    return hash(self)

def init(self, ...)"
    Initialize x; see x.__class__.__doc__ for signature

def new(T, S, ...)"
    Return a new object with type S, a subtype of T

def reduce(self, ...)"
    Helper for pickle

def reduce_ex(self, ...)"
    Helper for pickle

def repr(x):
    return repr(x)

def setattr(self, name, value):
    self.name = value
```

---

483
36.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

36.4.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>print_it</td>
<td>Value: True</td>
</tr>
</tbody>
</table>

36.5 Class Proxy

```
object

SCons.Util.Proxy
```

**Known Subclasses:** SCons.Builder.CompositeBuilder, SCons.Node.FS.EntryProxy

A simple generic Proxy class, forwarding all calls to subject. So, for the benefit of the python newbie, what does this really mean? Well, it means that you can take an object, let’s call it 'objA', and wrap it in this Proxy class, with a statement like this

```
proxyObj = Proxy(objA),
```

Then, if in the future, you do something like this

```
x = proxyObj.var1,
```

since Proxy does not have a 'var1' attribute (but presumably objA does), the request actually is equivalent to saying

```
x = objA.var1
```

Inherit from this class to create a Proxy.

Note that, with new-style classes, this does *not* work transparently for Proxy subclasses that use special `__*__()` method names, because those names are now bound to the class, not the individual instances. You now need to know in advance which `__*__()` method names you want to pass on to the underlying Proxy object, and specifically delegate their calls like this:

```
class Foo(Proxy):
```
```
_class Proxy Module SCons.Util

__str__ = Delegate('__str__')

36.5.1 Methods

```python
__init__(self, subject)

Wrap an object as a Proxy object
 Overrides: object.__init__

__getattr__(self, name)

Retrieve an attribute from the wrapped object. If the named attribute doesn’t exist, AttributeError is raised

get(self)

Retrieve the entire wrapped object

__cmp__(self, other)

__delattr__(...)

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

__getattribute__(...)

x.__getattribute__('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)

485
36.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

36.6 Class Delegate

```
object
    SCons.Util.Delegate
```

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

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

36.6.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>init</strong>(self, attribute)</td>
<td>x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<strong>init</strong> (inherited documentation)</td>
</tr>
<tr>
<td><strong>get</strong>(self, obj, cls)</td>
<td></td>
</tr>
<tr>
<td><strong>delattr</strong>(...)</td>
<td>x.<strong>delattr</strong>('name') == del x.name</td>
</tr>
<tr>
<td><strong>getattribute</strong>(...)</td>
<td>x.<strong>getattribute</strong>('name') == x.name</td>
</tr>
<tr>
<td><strong>hash</strong>(x)</td>
<td>hash(x)</td>
</tr>
</tbody>
</table>
36.6.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

36.7 Class _NoError

object →

exceptions.BaseException →

exceptions.Exception →

SCons.Util._NoError

36.7.1 Methods

```python
__delattr__(...)  
x.__delattr__('name') ↔ del x.name
```

Overrides: object.__delattr__
Class `NoError`  

Module `SCons.Util`

```python
__getattribute__(...)  
x.__getattribute__('name') == x.name
Overrides: object.__getattribute__
```

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

```python
__getslice__(x, i, j)  
x[i:j]
```

Use of negative indices is not supported.

```python
__hash__(x)  
hash(x)
```

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.BaseException.__init__
```

```python
__new__(T, S, ...)  
Return Value  
a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__
```

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

```python
__reduce_ex__(...)  
helper for pickle
```

```python
__repr__(x)  
repr(x)
Overrides: object.__repr__
```

```python
__setattr__(...)  
x.__setattr__('name', value) == x.name = value
Overrides: object.__setattr__
```

```python
__setstate__(...)  
```
36.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.class_</code></td>
<td>Value: &lt;attribute <code>._class_</code> of <code>object</code> objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute <code>args</code> of <code>exceptions.BaseException</code> objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member <code>message</code> of <code>exceptions.BaseException</code> objects&gt;</td>
</tr>
</tbody>
</table>

36.8 Class WindowsError

object —
exceptions.BaseException —
exceptions.Exception —
exceptions.StandardError —
exceptions.EnvironmentError —
exceptions.OSError —
SCons.Util.WindowsError

36.8.1 Methods

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

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

```python
__getitem__(x, y)  
x[y]
```
内陆Error Module SCons.Util

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

```python
__hash__(x)
hash(x)
```

```python
__init__(...)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.EnvironmentError.__init__
```

```python
__new__(T, S, ...)
Return Value
    a new object with type S, a subtype of T
Overrides: exceptions.EnvironmentError.__new__
```

```python
__reduce__(...)
helper for pickle
Overrides: exceptions.BaseException.__reduce__
```

```python
__reduce_ex__(...)
helper for pickle
```

```python
__repr__(x)
repr(x)
Overrides: object.__repr__
```

```python
__setattr__(...)
x.__setattr__(‘name’, value) <==> x.name = value
Overrides: object.__setattr__
```

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

```python
__str__(x)
str(x)
Overrides: exceptions.BaseException.__str__
```

### 36.8.2 Properties
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__

__add__(self, other)  
Overrides: UserList.UserList.__add__

__radd__(self, other)  
Overrides: UserList.UserList.__radd__

__coerce__(self, other)

__str__(self)

__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)

_{-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, **kwds)

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
36.11 Class Selector

UserDict.UserDict

SCons.Util.OrderedDict

SCons.Util.Selector


A callable ordered dictionary that maps file suffixes to dictionary values. We preserve the order in which items are added so that get_suffix() calls always return the first suffix added.

36.11.1 Methods

_call__(self, env, source, ext=False)

__cmp__(self, dict)

__contains__(self, key)

__delitem__(self, key)

Overrides: UserDict.UserDict.__delitem__
```
_class = ClassSelector

_init__(self, dict=False)
Overrides: UserDict.UserDict._init_

_len_(self)

__repr__(self)

_setitem__(self, key, item)
Overrides: UserDict.UserDict._setitem_

clear(self)
Overrides: UserDict.UserDict.clear

copy(self)
Overrides: UserDict.UserDict.copy

fromkeys(cls, iterable, value=False)

get(self, key, failobj=False)

has_key(self, key)

items(self)
Overrides: UserDict.UserDict.items

iteritems(self)

iterkeys(self)

itervalues(self)

keys(self)
Overrides: UserDict.UserDict.keys

pop(self, key, *args)

popitem(self)
Overrides: UserDict.UserDict.popitem

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

update(self, dict)
Overrides: UserDict.UserDict.update
```
values(self)
Overrides: UserDict.UserDict.values

36.12 Class LogicalLines

object —

SCons.Util.LogicalLines

36.12.1 Methods

_init_(self, fileobj)
x._init_(...) initializes x; see x._class_._doc_ for signature
Overrides: object._init_ (inherited documentation)

readline(self)

readlines(self)

_delattr_(...)
x._delattr_('name') == del x.name

_getattribute_(...)
x._getattribute_('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)
Class UniqueList

Module SCons.Util

```python
setattr(...)
xsetattr_('name', value) <-> x.name = value

str(x)
```

### 36.12.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

### 36.13 Class UniqueList


#### 36.13.1 Methods

```python
__init__(self, seq=[])  
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__
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Overrides</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__getitem__</code>(self, i)</td>
<td>Get item at index <code>i</code></td>
<td>UserList.UserList.<strong>getitem</strong></td>
</tr>
<tr>
<td><code>__setitem__</code>(self, i, item)</td>
<td>Set item at index <code>i</code> to <code>item</code></td>
<td>UserList.UserList.<strong>setitem</strong></td>
</tr>
<tr>
<td><code>__getslice__</code>(self, i, j)</td>
<td>Get slice from index <code>i</code> to <code>j</code></td>
<td>UserList.UserList.<strong>getslice</strong></td>
</tr>
<tr>
<td><code>__setslice__</code>(self, i, j, other)</td>
<td>Set slice from index <code>i</code> to <code>j</code> to <code>other</code></td>
<td>UserList.UserList.<strong>setslice</strong></td>
</tr>
<tr>
<td><code>__add__</code>(self, other)</td>
<td>Add other to this object</td>
<td>UserList.UserList.<strong>add</strong></td>
</tr>
<tr>
<td><code>__radd__</code>(self, other)</td>
<td>Add this object to other</td>
<td>UserList.UserList.<strong>radd</strong></td>
</tr>
<tr>
<td><code>__iadd__</code>(self, other)</td>
<td>In-place addition</td>
<td>UserList.UserList.<strong>iadd</strong></td>
</tr>
<tr>
<td><code>__mul__</code>(self, other)</td>
<td>Multiply by <code>other</code></td>
<td>UserList.UserList.<strong>mul</strong></td>
</tr>
<tr>
<td><code>__rmul__</code>(self, other)</td>
<td>Multiply <code>other</code> by this object</td>
<td>UserList.UserList.<strong>rmul</strong></td>
</tr>
<tr>
<td><code>__imul__</code>(self, other)</td>
<td>In-place multiplication</td>
<td>UserList.UserList.<strong>imul</strong></td>
</tr>
<tr>
<td><code>append</code>(self, item)</td>
<td>Add <code>item</code> to this object</td>
<td>UserList.UserList.append</td>
</tr>
<tr>
<td><code>insert</code>(self, i)</td>
<td>Insert <code>item</code> at index <code>i</code></td>
<td>UserList.UserList.insert</td>
</tr>
<tr>
<td><code>count</code>(self, item)</td>
<td>Count <code>item</code> in this object</td>
<td>UserList.UserList.count</td>
</tr>
<tr>
<td><code>index</code>(self, item)</td>
<td>Get index of <code>item</code></td>
<td>UserList.UserList.index</td>
</tr>
<tr>
<td><code>reverse</code>(self)</td>
<td>Reverse this object</td>
<td>UserList.UserList.reverse</td>
</tr>
<tr>
<td><code>sort</code>(self, *args, **kwds)</td>
<td>Sort this object</td>
<td>UserList.UserList.sort</td>
</tr>
<tr>
<td><code>extend</code>(self, other)</td>
<td>Extend this object with <code>other</code></td>
<td>UserList.UserList.extend</td>
</tr>
</tbody>
</table>
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

```python
__contains__(self, item)
__delitem__(self, i)
__delslice__(self, i, j)
__repr__(self)
pop(self, i=-1)
remove(self, item)
```

```yaml
__init__(self, file)
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: object.__init__(inherited documentation)
write(self, arg)
__getattr__(self, attr)
__delattr__(...)
x.__delattr__('name') == del x.name
__getattribute__(...)
x.__getattribute__('name') == x.name
__hash__(x)
hash(x)
__new__(T, S, ...)
Return Value
a new object with type S, a subtype of T
```
36.14.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

36.15 Class Null

object     
|----------|
| SCons.Util.Null

**Known Subclasses:** SCons.Sig.MD5Null, SCons.Sig.TimeStampNull, SCons.Util.NullSeq

Null objects always and reliably "do nothing."

36.15.1 Methods

`__new__(cls, *args, **kwargs)`

**Return Value**

a new object with type S, a subtype of T

Overrides: object.__new__ (inherited documentation)

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

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

Overrides: object.__init__ (inherited documentation)

`__call__(self, *args, **kwargs)`
```
._repr__(self)
repr(x)
Overrides: object._repr__ (inherited documentation)

._nonzero__(self)

._getattr__(self, name)

._setattr__(self, name, value)
x._setattr__('name', value) ==> x.name = value
Overrides: object._setattr__ (inherited documentation)

._delattr__(self, name)
x._delattr__('name') ==> del x.name
Overrides: object._delattr__ (inherited documentation)

._getattribute__(...)
x._getattribute__('name') ==> x.name

._hash__(x)
hash(x)

._reduce__(...)
helper for pickle

._reduce_ex__(...)
helper for pickle

._str__(x)
str(x)
```

### 36.15.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em>_</td>
<td>Value: &lt;attribute '<em>class</em>_' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
36.16 Class NullSeq

object

SCons.Util.Null

SCons.Util.NullSeq

Known Subclasses: SCons.Subst.NullNodeList

36.16.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__len__</code>(self)</td>
<td></td>
</tr>
<tr>
<td><code>__iter__</code>(self)</td>
<td></td>
</tr>
<tr>
<td><code>__getitem__</code>(self, i)</td>
<td></td>
</tr>
<tr>
<td><code>__delitem__</code>(self, i)</td>
<td></td>
</tr>
<tr>
<td><code>__setitem__</code>(self, i, v)</td>
<td></td>
</tr>
<tr>
<td><code>__call__</code>(self, *args, **kwargs)</td>
<td></td>
</tr>
<tr>
<td><code>__delattr__</code>(self, name)</td>
<td>del x.name</td>
</tr>
<tr>
<td><code>__getattr__</code>(self, name)</td>
<td></td>
</tr>
<tr>
<td><code>__getattribute__</code>(...)</td>
<td></td>
</tr>
<tr>
<td><code>__hash__</code>(x)</td>
<td>hash(x)</td>
</tr>
<tr>
<td><code>__init__</code>(self, *args, **kwargs)</td>
<td>initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature</td>
</tr>
<tr>
<td><code>__new__</code>(cls, *args, **kwargs)</td>
<td></td>
</tr>
</tbody>
</table>

Return Value

a new object with type S, a subtype of T

Overrides: object.__new__, extit(inherited documentation)
Class NullSeq

Module SCons.Util

```python
_nonzero_(self)

_reduce_(...)
helper for pickle

_reduce_ex_(...)
helper for pickle

_repr_(self)
repr(x)
Overrides: object._repr_ (inherited documentation)

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

_str_(x)
str(x)
```

36.16.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
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. 492)

- **EnumVariable**: engine.SCons.Variables.EnumVariable
  This file defines the option type for SCons allowing only specified input-values.
  (Section 39, p. 493)

- **ListVariable**: engine.SCons.Variables.ListVariable
  This file defines the option type for SCons implementing 'lists'.
  (Section 40, p. 494)

- **PackageVariable**: engine.SCons.Variables.PackageVariable
  This file defines the option type for SCons implementing 'package activation'.
  (Section 41, p. 495)

- **PathVariable**: SCons.Variables.PathVariable
  This file defines an option type for SCons implementing path settings.
  (Section 42, p. 496)

37.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| ._revision__ | Value: `src/engine/SCons/Variables/__init__.py` 5023  
|       | 2010/06/14 2...                                                               |

37.3 Class Variables

    object SCons.Variables.Variables

37.3.1 Methods

    __init__(self, files=[], args={}, is_global=1)

    files - [optional] List of option configuration files to load
    (backward compatibility) If a single string is passed it is automatically placed in a file list

    Overrides: object.__init__

    504
**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:

```python
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=[])

_delattr__(...)  
x._delattr_('name') <==> del x.name

_getattribute__(...)  
x._getattribute_('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)
37.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_class_</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

37.3.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>Holds all the options, updates the environment with the variables, and renders the help text. Value: False</td>
</tr>
</tbody>
</table>
| format   | Value: '\n%s
 default: %s
 actual: %s
' |
| format_  | Value: '\n%s
 default: %s
 actual: %s
 aliases: ...' |
### Module SCons.Variables.BoolVariable

`engine.SCons.Variables.BoolVariable`

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

Usage example:

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

#### 38.1 Functions

**BoolVariable**(key, help, default)

The input parameters describe a boolean option, thus they are returned with the correct converter and validator appended. The 'help' text will by appended by '(yes|no) to show the valid valued. The result is usable for input to opts.Add().
Module SCons.Variables.EnumVariable

engine.SCons.Variables.EnumVariable

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

Usage example:

```python
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

<table>
<thead>
<tr>
<th>EnumVariable(key, help, default, allowed_values, map={}, ignorecase=0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The input parameters describe a option with only certain values</td>
</tr>
<tr>
<td>allowed. They are returned with an appropriate converter and</td>
</tr>
<tr>
<td>validator appended. The result is usable for input to</td>
</tr>
<tr>
<td>Variables.Add().</td>
</tr>
<tr>
<td>'key' and 'default' are the values to be passed on to Variables.Add().</td>
</tr>
<tr>
<td>'help' will be appended by the allowed values automatically</td>
</tr>
<tr>
<td>'allowed_values' is a list of strings, which are allowed as values</td>
</tr>
<tr>
<td>for this option.</td>
</tr>
<tr>
<td>The 'map'-dictionary may be used for converting the input value</td>
</tr>
<tr>
<td>into canonical values (eg. for aliases).</td>
</tr>
<tr>
<td>'ignorecase' defines the behaviour of the validator:</td>
</tr>
<tr>
<td>If ignorecase == 0, the validator/ converter are case-sensitive.</td>
</tr>
<tr>
<td>If ignorecase == 1, the validator/ converter are case-insensitive.</td>
</tr>
<tr>
<td>If ignorecase == 2, the validator/ converter is case-insensitive and the converted value will always be lower-case.</td>
</tr>
<tr>
<td>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).</td>
</tr>
</tbody>
</table>
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:

```python
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

```python
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).
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
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| PathVariable | Value: `<SCons.Variables.PathVariable._PathVariableClass
| object a...   |                                                 |
43 Module SCons.Warnings

SCons.Warnings

This file implements the warnings framework for SCons.

43.1 Functions

<table>
<thead>
<tr>
<th>suppressWarningClass(clazz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suppresses all warnings that are of type clazz or derived from clazz.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>enableWarningClass(clazz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enables all warnings that are of type clazz or derived from clazz.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>warningAsException(flag=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Turn warnings into exceptions. Returns the old value of the flag.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>warn(clazz, *args)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process string specifications of enabling/disabling warnings, as passed to the --warn option or the SetOption('warn') function.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_revision_</code></td>
<td>Value: <code>src/engine/SCons/Warnings.py 5023 2010/06/14 22:05:46 sc...</code></td>
</tr>
</tbody>
</table>

continued on next page
Class Warning

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>

43.3 Class Warning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning


43.3.1 Methods

```python
__.delattr__(...)
```

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

Overrids: object.__delattr__

```python
__.getattr__(...)
```

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

Overrids: object.__getattr__

```python
__.getitem__(x, y)
```

```
x[y]
```

```python
__.getslice__(x, i, j)
```

```
x[i:j]
```

Use of negative indices is not supported.

```python
__.hash__(x)
```

```
hash(x)
```

```python
__.init__(...)
```

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

Overrids: exceptions.BaseException.__init__
Class Warning Module SCons.Warnings

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

**Return Value**

A new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

```python
__reduce__(...)
```

Helper for pickle

Overrides: object.__reduce__

```python
__reduce_ex__(...)
```

Helper for pickle

```python
__repr__(x)
```

repr(x)

Overrides: object.__repr__

```python
__setattr__(...)
```

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

Overrides: object.__setattr__

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

```python
__str__(x)
```

str(x)

Overrides: object.__str__

### 43.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.4 Class WarningOnByDefault

```python
object
```

```
exceptions.BaseException
```

```
exceptions.Exception
```

```
SCons.Errors.UserError
```

```
SCons.Warnings.Warning
```

```
SCons.Warnings.WarningOnByDefault
```

**Known Subclasses:**
- SCons.Warnings.CorruptSConsignWarning
- SCons.Warnings.DuplicateEnvironmentWarning
- SCons.Warnings.LinkWarning
- SCons.Warnings.FutureReservedVariableWarning
- SCons.Warnings.MisleadingKeywordsWarning
- SCons.Warnings.MissingSConscriptWarning
- SCons.Warnings.NoMD5ModuleWarning
- SCons.Warnings.NoMetaclassSupportWarning
- SCons.Warnings.NoObjectCountWarning
- SCons.Warnings.NoParallelSupportWarning
- SCons.Warnings.ReservedVariableWarning
- SCons.WarningsStackSizeWarning
- SCons.Warnings.VisualCMissingWarning
- SCons.Warnings.VisualVersionMismatch

43.4.1 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__.__init__ for signature
```

Overrides: exceptions.BaseException.__init__

517
```python
new(T, S, ...)

Return Value
    a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__
```

```python
reduce(...)      
helper for pickle
Overrides: object.__reduce__, extit(inherited documentation)
```

```python
reduce_ex(...)    
helper for pickle
```

```python
repr(x)
repr(x)
Overrides: object.__repr__
```

```python
setattr(...)
    x.__setattr__('name', value) ::= x.name = value
Overrides: object.__setattr__
```

```python
setstate(...)    
```

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

### 43.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.5 Class CacheWriteErrorWarning

object  
exceptions.BaseException  
    exceptions.Exception  
        SCons.Errors.UserError  
            SCons.Warnings.Warning  
                SCons.Warnings.CacheWriteErrorWarning

43.5.1 Methods

```python
def __delattr__(self, name):
    del self.name
```
Overrides: object.__delattr__

```python
def __getattr__(self, name):
    return self.name
```
Overrides: object.__getattr__

```python
x[y]
```

```python
x[i:j]
```
Use of negative indices is not supported.

```python
hash(x)
```

```python
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
```
Overrides: exceptions.BaseException.__init__

```python
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_ (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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.6 Class CorruptSConsignWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.WarningOnByDefault

SCons.Warnings.CorruptSConsignWarning

43.6.1 Methods

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

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

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

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

```
__hash__(x)  
hash(x)
```

```
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.BaseException.__init__
```
Class CorruptSConsignWarning

__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__

__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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.7 Class DependencyWarning

```
object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DependencyWarning
```

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.
```

```_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_
```
Class DependencyWarning

Module SCons.Warnings

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

```python
__reduce_ex__(...)  
helper for pickle
```

```python
__repr__(x)  
repr(x)  
Overrides: object.__repr__
```

```python
__setattr__(...)  
x.__setattr__(‘name’, value) <-> x.name = value  
Overrides: object.__setattr__
```

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

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

### 43.7.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>&lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
<td>Description: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
<td>Description: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>&lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
<td>Description: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.8 Class DuplicateEnvironmentWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.WarningOnByDefault

SCons.Warnings.DuplicateEnvironmentWarning

43.8.1 Methods

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

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

__getitem__(x, y)  
x[y]

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

__hash__(x)  
hash(x)

__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.BaseException.__init__
### Class DuplicateEnvironmentWarning

#### Module SCons.Warnings

__new__\((T, S, \ldots)\)

**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

__reduce__(\ldots)

caller for pickle

Overrides: object.__reduce__ extit(inherited documentation)

__reduce_ex__(\ldots)

caller for pickle

__repr__(x)

repr(x)

Overrides: object.__repr__

__setattr__(\ldots)

x.__setattr__(\textquoteleft name\textquoteright, value) \iff x.name = value

Overrides: object.__setattr__

__setstate__(\ldots)

__str__(x)

str(x)

Overrides: object.__str__

43.8.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '.<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.9  **Class FutureReservedVariableWarning**

object   
exceptions.BaseException   
exceptions.Exception   
SCons.Errors.UserError   
SCons.Warnings.Warning   
SCons.Warnings.WarningOnByDefault   
SCons.Warnings.FutureReservedVariableWarning

43.9.1  **Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.delattr(...)</code></td>
<td><code>x.__delattr__('name')</code> &lt;==&gt; del x.name</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<strong>delattr</strong></td>
</tr>
<tr>
<td><code>.getattribute(...)</code></td>
<td><code>x.__getattribute__('name')</code> &lt;==&gt; x.name</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<strong>getattribute</strong></td>
</tr>
<tr>
<td><code>.getitem(x, y)</code></td>
<td><code>x[y]</code></td>
</tr>
<tr>
<td><code>.getslice(x, i, j)</code></td>
<td><code>x[i:j]</code></td>
</tr>
<tr>
<td></td>
<td>Use of negative indices is not supported.</td>
</tr>
<tr>
<td><code>.hash(x)</code></td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><code>.init(...)</code></td>
<td><code>x.__init__(...)</code> initializes x; see x.<strong>class</strong>.doc for signature</td>
</tr>
<tr>
<td></td>
<td>Overrides: exceptions.BaseException.<strong>init</strong></td>
</tr>
</tbody>
</table>
Class FutureReservedVariableWarning  
Module SCons.Warnings

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

**Return Value**

A new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

```python
__reduce__(...)
```

Helper for pickle

Overrides: object.__reduce__

```python
__reduce_ex__(...)
```

Helper for pickle

```python
__repr__(x)
```

repr(x)

Overrides: object.__repr__

```python
__setattr__(...)
```

x.__setattr__('name', value) \implies x.name = value

Overrides: object.__setattr__

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

```python
__str__(x)
```

Overrides: object.__str__

### 43.9.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.10 Class LinkWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.WarningOnByDefault

SCons.Warnings.LinkWarning

Known Subclasses: SCons.Warnings.FortranCxxMixWarning

43.10.1 Methods

__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__
```python
__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__

__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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.11 Class MisleadingKeywordsWarning

object
oxceptions.BaseException
  
  exceptions.Exception
    SCons.Errors.UserError
      SCons.Warnings.Warning
        SCons.Warnings.WarningOnByDefault
          SCons.Warnings.MisleadingKeywordsWarning

43.11.1 Methods

```python
_delattr__(...) 
x.__delattr__('name') == del x.name
Overrides: object.__delattr__
```

```python
_getattribute__(...) 
x.__getattribute__('name') == x.name
Overrides: object.__getattribute__
```

```python
_getitem__(x, y) 
x[y]
```

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

```python
_hash__(x) 
hash(x)
```

```python
_init__(...) 
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.BaseException.__init__
```
Class MisleadingKeywordsWarning

Module SCons.Warnings

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

Return Value
a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__
```

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

```python
__reduce_ex__(...)
helper for pickle
```

```python
__repr__(x)
repr(x)
Overrides: object.__repr__
```

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

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

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

### 43.11.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.12 Class MissingSConscriptWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.WarningOnByDefault

SCons.Warnings.MissingSConscriptWarning

43.12.1 Methods

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

```python
__getattribute__(...)  
x.__getattribute__('name') == x.name  
Overrides: object.__getattribute__
```

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

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

```python
__hash__(x)  
hash(x)
```

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: exceptions.BaseException.__init__
```
Class MissingSConscriptWarning

`.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__ ex(extended 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_class_</code></td>
<td>Value: <code>&lt;attribute '_class_' of 'object' objects&gt;</code></td>
</tr>
<tr>
<td>args</td>
<td>Value: <code>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
<tr>
<td>message</td>
<td>Value: <code>&lt;member 'message' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
</tbody>
</table>
43.13  Class `NoMD5ModuleWarning`

object

`exceptions.BaseException`

`exceptions.Exception`

`SCons.Errors.UserError`

`SCons.Warnings.Warning`

`SCons.Warnings.WarningOnByDefault`

`SCons.Warnings.NoMD5ModuleWarning`

43.13.1  Methods

```python
__delattr__(...)

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

```python
__getattribute__(...)

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

```python
__getitem__(x, y)

x[y]
```

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

x[i:j]
Use of negative indices is not supported.
```

```python
__hash__(x)

hash(x)
```

```python
__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_

### 43.13.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.14 Class NoMetaclassSupportWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.WarningOnByDefault

SCons.Warnings.NoMetaclassSupportWarning

43.14.1 Methods

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

__getattribute__(...)  
x.__getattribute__('name') <=> x.name
Overides: object.__getattribute__

__getitem__(x, y)  
x[y]

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

__hash__(x)  
hash(x)

__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overides: exceptions.BaseException.__init__
```
Class NoMetaclassSupportWarning

Module SCons.Warnings

```python
new(T, S, ...)

Return Value
  a new object with type S, a subtype of T
Overrides: exceptions.BaseException.new
```

```python
reduce(...)
helper for pickle
Overrides: object.reduce
```

```python
reduce_ex(...)
helper for pickle
```

```python
repr(x)
repr(x)
Overrides: object.repr
```

```python
setattr(...)
xsetattr_('name', value) ==> x.name = value
Overrides: object.setattr
```

```python
setstate(...)
```

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

### 43.14.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.15 Class NoObjectCountWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.WarningOnByDefault

SCons.Warnings.NoObjectCountWarning

43.15.1 Methods

```python
__delattr__(...)  
x.__delattr__(‘name’) == del x.name
Overrides: object.__delattr__
```

```python
__getattribute__(...)  
x.__getattribute__(‘name’) == x.name
Overrides: object.__getattribute__
```

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

```python
__getslice__(x, i, j)  
x[i:j]
```

Use of negative indices is not supported.

```python
__hash__(x)  
hash(x)
```

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.16 Class NoParallelSupportWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.NoParallelSupportWarning

43.16.1 Methods

```python
__delattr__(...)  
x.__delattr__(’name’) == del x.name  
Overrides: object.__delattr__
```

```python
__getattr__(...)  
x.__getattr__(’name’) == x.name  
Overrides: object.__getattr__
```

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

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

```python
__hash__(x)  
hash(x)
```

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: exceptions.BaseException.__init__
```
Class NoParallelSupportWarning

Module SCons.Warnings

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

**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

```python
__reduce__(...)
```

helper for pickle

Overrides: object.__reduce__

```python
__reduce_ex__(...)
```

helper for pickle

```python
__repr__(x)
```

repr(x)

Overrides: object.__repr__

```python
__setattr__(...)
```

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

Overrides: object.__setattr__

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

```python
__str__(x)
```

str(x)

Overrides: object.__str__

### 43.16.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>


43.17 Class ReservedVariableWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.WarningOnByDefault

SCons.Warnings.ReservedVariableWarning

43.17.1 Methods

__delattr__ (...)

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

Overrides: object.__delattr__

__getattribute__ (...)

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

Overrides: object.__getattribute__

__getitem__ (x, y)

x[y]

__getslice__ (x, i, j)

x[i:j]

Use of negative indices is not supported.

__hash__ (x)

hash(x)

__init__ (...)

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

Overrides: exceptions.BaseException.__init__
new(T, S, ...)

Return Value
a new object with type S, a subtype of T
Overrides: exceptions.BaseException.new

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

reduce_ex(...)
helper for pickle

repr_(x)
repr(x)
Overrides: object.repr_

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

setstate_(...)

str_(x)
str(x)
Overrides: object.str_

43.17.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.18 Class StackSizeWarning

object

exceptions.BaseException

    exceptions.Exception

        SCons.Errors.UserError

            SCons.Warnings.Warning

                SCons.Warnings.WarningOnByDefault

                    SCons.Warnings.StackSizeWarning

43.18.1 Methods

    __delattr__(...)

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

   Overrides: object.__delattr__

    __getattribute__(...)

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

   Overrides: object.__getattribute__

    __getitem__(x, y)

x[y]

    __getslice__(x, i, j)

x[i:j]

    Use of negative indices is not supported.

    __hash__(x)

    hash(x)

    __init__(...)

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

    Overrides: exceptions.BaseException.__init__
_new_(T, S, ...)

Return Value

d 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute ‘<em>class</em>’ of ‘object’ objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute ‘args’ of ‘exceptions.BaseException’ objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member ‘message’ of ‘exceptions.BaseException’ objects&gt;</td>
</tr>
</tbody>
</table>
43.19 Class VisualCMissingWarning

Object diagram:
```
object
  exceptions.BaseException
    exceptions.Exception
      SCons.Errors.UserError
        SCons.Warnings.Warning
          SCons.Warnings.WarningOnByDefault
            SCons.Warnings.VisualCMissingWarning
```

43.19.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>delattr</strong>(...)</td>
<td>x.<strong>delattr</strong>(‘name’) &lt;=&gt; del x.name</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<strong>delattr</strong></td>
</tr>
<tr>
<td><strong>getattribute</strong>(...)</td>
<td>x.<strong>getattribute</strong>(‘name’) &lt;=&gt; x.name</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<strong>getattribute</strong></td>
</tr>
<tr>
<td><strong>getitem</strong>(x, y)</td>
<td>x[y]</td>
</tr>
<tr>
<td><strong>getslice</strong>(x, i, j)</td>
<td>x[i:j]</td>
</tr>
<tr>
<td></td>
<td>Use of negative indices is not supported.</td>
</tr>
<tr>
<td><strong>hash</strong>(x)</td>
<td>hash(x)</td>
</tr>
<tr>
<td><strong>init</strong>(...)</td>
<td>x.<strong>init</strong>(...) initializes x; see x.<strong>class</strong>.<strong>doc</strong> for signature</td>
</tr>
<tr>
<td></td>
<td>Overrides: exceptions.BaseException.<strong>init</strong></td>
</tr>
</tbody>
</table>

547
Class VisualCMissingWarning

Module SCons.Warnings

```
__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__
```

### 43.19.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute ‘<strong>class</strong>’ of ‘object’ objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute ‘args’ of ‘exceptions.BaseException’ objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member ‘message’ of ‘exceptions.BaseException’ objects&gt;</td>
</tr>
</tbody>
</table>
43.20 Class VisualVersionMismatch

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.WarningOnByDefault
SCons.Warnings.VisualVersionMismatch

43.20.1 Methods

__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__
Class VisualVersionMismatch

Module SCons.Warnings

.. _new_\( (T, S, ...) \)

**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException._new_

.. _reduce_\( (\ldots) \)

helper for pickle

Overrides: object._reduce_ extit(inherited documentation)

.. _reduce_ex_\( (\ldots) \)

helper for pickle

.. _repr_\( (x) \)

repr(x)

Overrides: object._repr_

.. _setattr_\( (\ldots) \)

x._setattr_('name', value) \iff x.name = value

Overrides: object._setattr_

.. _setstate_\( (\ldots) \)

.. _str_\( (x) \)

str(x)

Overrides: object._str_

### 43.20.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '_class_' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
Class VisualStudioMissingWarning

object
exceptions.BaseException
exceptions.Exception
SCons.Errors.UserError
SCons.Warnings.Warning
SCons.Warnings.VisualStudioMissingWarning

43.21 Class VisualStudioMissingWarning

43.21.1 Methods

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

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

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

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

```python
__hash__(x)  
hash(x)
```

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: exceptions.BaseException.__init__
```

```python
__new__(T, S, ...)  
Return Value
   a new object with type S, a subtype of T  
Overrides: exceptions.BaseException.__new__
```
Class VisualStudioMissingWarning

Module SCons.Warnings

```python
_reduce_(...)  
helper for pickle
Overrides: object._reduce_ ext(inherited documentation)
```

```python
_reduce_ex_(...)  
helper for pickle
```

```python
_repr_(x)  
repr(x)
Overrides: object._repr_
```

```python
_setattr_(...)  
x._setattr_('name', value) <-> x.name = value
Overrides: object._setattr_
```

```python
_setstate_(...)  
```

```python
_str_(x)  
str(x)
Overrides: object._str_
```

43.21.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_<em>class</em></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.22 Class FortranCxxMixWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.WarningOnByDefault

SCons.Warnings.LinkWarning

SCons.Warnings.FortranCxxMixWarning

43.22.1 Methods

```python
__delattr__(...)

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

Overrides: object.__delattr__
```

```python
__getattribute__(...)

x.__getattribute__("name") <=> x.name

Overrides: object.__getattribute__
```

```python
__getitem__(x, y)

x[y]
```

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

x[i:j]

Use of negative indices is not supported.
```

```python
__hash__(x)

hash(x)
```

```python
__init__(...)

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

Overrides: exceptions.BaseException.__init__
```
Class FortranCxxMixWarning

Module SCons.Warnings

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

Return Value

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

```python
__reduce__(...)
```

helper for pickle

Overrides: object.__reduce__ ex(extended documentation)

```python
__reduce_ex__(...)
```

helper for pickle

```python
__repr__(x)
```

repr(x)

Overrides: object.__repr__

```python
__setattr__(...)
```

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

Overrides: object.__setattr__

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

```python
__str__(x)
```

str(x)

Overrides: object.__str__

### 43.22.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
Class FutureDeprecatedWarning

43.23 Class FutureDeprecatedWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.FutureDeprecatedWarning

Known Subclasses: SCons.Warnings.DeprecatedSourceCodeWarning

43.23.1 Methods

__delattr__(...)

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

Overrides: object.__delattr__

__getattribute__(...)

x.__getattribute__('name') == x.name

Overrides: object.__getattribute__

__getitem__(x, y)

x[y]

__getslice__(x, i, j)

x[i:j]

Use of negative indices is not supported.

__hash__(x)

hash(x)

__init__(...)

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

Overrides: exceptions.BaseException.__init__
Class FutureDeprecatedWarning  
Module SCons.Warnings

```
__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__
```

### 43.23.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.24 Class DeprecatedWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning


43.24.1 Methods

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

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

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

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

```ruby
__hash__(x)  
hash(x)
```

```ruby
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
Overrides: exceptions.BaseException.__init__
```
Class DeprecatedWarning

Module SCons.Warnings

___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__

43.24.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.25 Class MandatoryDeprecationWarning

```python
object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning

SCons.Warnings.MandatoryDeprecationWarning

```

43.25.1 Methods

```python
__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__.__init__ for signature  
Overrides: exceptions.BaseException.__init__
```
Class MandatoryDeprecatedWarning

```
__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__
```

### 43.25.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Description</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.26 Class `PythonVersionWarning`

object

```
exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning
```

SCons.Warnings.PythonVersionWarning

---

43.26.1 Methods

```
__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__
```
Class PythonVersionWarning

Module SCons.Warnings

```python
_new__(T, S, ...)

Return Value
    a new object with type S, a subtype of T
Overrides: exceptions.BaseException._new_
```

```python
_reduce__(...)
helper for pickle
Overrides: object._reduce__ extit(inherited documentation)
```

```python
_reduce_ex__(...)
helper for pickle
```

```python
_repr__(x)
repr(x)
Overrides: object._repr_
```

```python
_setattr__(...)
x._setattr__('name', value) <=> x.name = value
Overrides: object._setattr_
```

```python
_setstate__(...)
```

```python
_str__(x)
str(x)
Overrides: object._str_
```

### 43.26.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>class</em></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>

562
43.27 Class DeprecatedSourceCodeWarning

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

```python
x.__getattribute__('name') == x.name
```
Overrides: object.__getattribute__

```python
x[i:j]
```
Use of negative indices is not supported.

```python
hash(x)
```

```python
x.__init__(...) initializes x; see x.__class__.__doc__ for signature
```
Overrides: exceptions.BaseException.__init__
Class DeprecatedSourceCodeWarning

_.new_ \((T, S, \ldots)\)

Return Value

a new object with type \(S\), a subtype of \(T\)

Overrides: exceptions.BaseException_.new_

_.reduce_(\ldots)

helper for pickle

Overrides: object_.reduce_ extit(inherited documentation)

_.reduce_ex_(\ldots)

helper for pickle

_.repr_(\ldots)

repr(x)

Overrides: object_.repr_

_.setattr_(\ldots)

x_.setattr_\('name', value\) \(\iff\) x.name = value

Overrides: object_.setattr_

_.setstate_(\ldots)

_.str_(\ldots)

str(x)

Overrides: object_.str_

3.27.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.class_</td>
<td>Value: &lt;attribute '.class_' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.28 Class DeprecatedBuildDirWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning

SCons.Warnings.DeprecatedBuildDirWarning

43.28.1 Methods

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

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

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

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

```
__hash__(x)  
hash(x)
```

```
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: exceptions.BaseException.__init__
```
__new__(T, S, ...)

Return Value
a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__

__reduce__(...)

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

__reduce_ex__(...)

helper for pickle

__repr__(x)

repr(x)
Overrides: object.__repr__

__setattr__(...)

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

__setstate__(...)


__str__(x)

str(x)
Overrides: object.__str__

43.28.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute ‘<strong>class</strong>’ of ‘object’ objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute ‘args’ of ‘exceptions.BaseException’ objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member ‘message’ of ‘exceptions.BaseException’ objects&gt;</td>
</tr>
</tbody>
</table>
43.29 Class TaskmasterNeedsExecuteWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning

SCons.Warnings.TaskmasterNeedsExecuteWarning

43.29.1 Methods

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

```python
__getattribute__(...)  
x.__getattribute__("name") <=> x.name  
Overrides: object.__getattribute__
```

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

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

```python
__hash__(x)  
hash(x)
```

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
 Overrides: exceptions.BaseException.__init__
```
Class TaskmasterNeedsExecuteWarning

Method Summary

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

Return Value
    a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__
```

```python
__reduce__(...)
helper for pickle
Overrides: object.__reduce__
```

```python
__reduce_ex__(...)
helper for pickle
```

```python
__repr__(x)
repr(x)
Overrides: object.__repr__
```

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

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

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

### 43.29.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
## 43.30 Class DeprecatedCopyWarning

```python
object
```

```python
exceptions.BaseException
```

```python
exceptions.Exception
```

```python
SCons.Errors.UserError
```

```python
SCons.Warnings.Warning
```

```python
SCons.Warnings.DeprecatedWarning
```

```python
SCons.Warnings.MandatoryDeprecatedWarning
```

```python
SCons.Warnings.DeprecatedCopyWarning
```

### 43.30.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `__delattr__(...)`                         | \[x.__delattr__(‘name’) \]
|                                            | \[
|                                            | del x.name
|                                            | Overrides: object.__delattr__                                              |
|                                             |                                                                            |
| `__getattribute__(...)`                    | \[x.__getattribute__(‘name’) \]
|                                            | \[
|                                            | x.name
|                                            | Overrides: object.__getattribute__                                         |
|                                             |                                                                            |
| `__getitem__(x, y)`                       | \[x[y]
|                                             |                                                                            |
| `__getslice__(x, i, j)`                   | \[x[i:j]
|                                            | Use of negative indices is not supported.                                 |
|                                             |                                                                            |
| `__hash__(x)`                             | \[hash(x)
|                                             |                                                                            |
| `__init__(...)`                           | \[x.__init__(...) initializes x; see x.__class__.__doc__ for signature
|                                            | Overrides: exceptions.BaseException.__init__                              |
__new__(T, S, ...)  

Return Value  
a new object with type S, a subtype of T  
Overrides: exceptions.BaseException.__new__  

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

__reduce_ex__(...)  
helper for pickle  

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

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

__setstate__(...)  

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

### 43.30.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| __class__ | Value: `<attribute ‘__class__’ of ‘object’ objects>  
| args     | Value: `<attribute ‘args’ of ‘exceptions.BaseException’ objects>  
| message  | Value: `<member ‘message’ of ‘exceptions.BaseException’ objects> |
43.31 Class DeprecatedOptionsWarning

```python

43.31.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__delattr__(...)</code></td>
<td>Overrides: <code>object.__delattr__</code></td>
</tr>
<tr>
<td><code>x.__delattr__(‘name’)</code></td>
<td><code>del x.name</code></td>
</tr>
<tr>
<td><code>__getattr__(...)</code></td>
<td>Overrides: <code>object.__getattr__</code></td>
</tr>
<tr>
<td><code>x.__getattr__(‘name’)</code></td>
<td><code>x.name</code></td>
</tr>
<tr>
<td><code>__getitem__(x, y)</code></td>
<td><code>x[y]</code></td>
</tr>
<tr>
<td><code>__getslice__(x, i, j)</code></td>
<td><code>x[i:j]</code></td>
</tr>
<tr>
<td><code>Use of negative indices is not supported.</code></td>
<td></td>
</tr>
<tr>
<td><code>__hash__(x)</code></td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><code>__init__(...)</code></td>
<td><code>x.__init__(...) initializes x; see x.__class__.__doc__ for signature</code></td>
</tr>
<tr>
<td></td>
<td>Overrides: <code>exceptions.BaseException.__init__</code></td>
</tr>
</tbody>
</table>
```

Class DeprecatedOptionsWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning

SCons.Warnings.MandatoryDeprecatedWarning

SCons.Warnings.DeprecatedOptionsWarning

571
__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__ (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__

43.31.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.32 Class DeprecatedSourceSignaturesWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning

SCons.Warnings.MandatoryDeprecatedWarning

SCons.Warnings.DeprecatedSourceSignaturesWarning

43.32.1 Methods

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

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

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

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

```python
__hash__(x)  
hash(x)
```

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: exceptions.BaseException.__init__
```
```python
__new__(T, S, ...)

Return Value
  a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__
```

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

```python
__reduce_ex__(...)
helper for pickle
```

```python
__repr__(x)
repr(x)
Overrides: object.__repr__
```

```python
__setattr__(...)
x.__setattr__(name, value) <<= x.name = value
Overrides: object.__setattr__
```

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

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

### 43.32.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
### 43.33 Class DeprecatedTargetSignaturesWarning

```python
object

exceptions.BaseException

    exceptions.Exception

        SCons.Errors.UserError

            SCons.Warnings.Warning

                SCons.Warnings.DeprecatedWarning

                    SCons.Warnings.MandatoryDeprecatedWarning

                        SCons.Warnings.DeprecatedTargetSignaturesWarning
```

#### 43.33.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>delattr_</code></td>
<td><code>x._delattr_('name')</code></td>
<td><code>del x.name</code> Override: <code>object._delattr_</code></td>
</tr>
<tr>
<td><code>getattribute_</code></td>
<td><code>x._getattribute_('name')</code></td>
<td><code>x.name</code> Override: <code>object._getattribute_</code></td>
</tr>
<tr>
<td><code>getitem_</code></td>
<td><code>x[y]</code></td>
<td><code>x[y]</code></td>
</tr>
<tr>
<td><code>getslice_</code></td>
<td><code>x[i:j]</code></td>
<td><code>x[i:j]</code> Use of negative indices is not supported.</td>
</tr>
<tr>
<td><code>hash_</code></td>
<td><code>hash(x)</code></td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><code>init_</code></td>
<td><code>x._init_(...)</code></td>
<td><code>x._init_(...)</code> initializes x; see <code>x._class_._doc_</code> for signature Override: <code>exceptions.BaseException._init_</code></td>
</tr>
</tbody>
</table>
Class DeprecatedTargetSignaturesWarning

Module SCons.Warnings

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

**Return Value**

a new object with type S, a subtype of T

Overrides: exceptions.BaseException.__new__

```python
__reduce__(...)
```

helper for pickle

Overrides: object.__reduce__ extit(inherited documentation)

```python
__reduce_ex__(...)
```

helper for pickle

```python
__repr__(x)
```

repr(x)

Overrides: object.__repr__

```python
__setattr__(...)
```

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

Overrides: object.__setattr__

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

```python
__str__(x)
```

str(x)

Overrides: object.__str__

### 43.33.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>

576
43.34 Class DeprecatedDebugOptionsWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning

SCons.Warnings.MandatoryDeprecatedWarning

SCons.Warnings.DeprecatedDebugOptionsWarning

43.34.1 Methods

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

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

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

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

```python
__hash__(x)  
hash(x)
```

```python
__init__(...)  
x.__init__(...) initializes x; see x.__class__.__doc__ for signature  
Overrides: exceptions.BaseException.__init__
```
Class DeprecatedDebugOptionsWarning

Module SCons.Warnings

__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__(...)

default (inherited documentation)

__repr__(x)

repr(x)
Overrides: object.__repr__

__setattr__(...)

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

__str__(x)

str(x)
Overrides: object.__str__

43.34.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
43.35  **Class DeprecatedSigModuleWarning**

object

exceptions.BaseException

  exceptions.Exception

    SCons.Errors.UserError

    SCons.Warnings.Warning

    SCons.Warnings.DeprecatedWarning

    SCons.Warnings.MandatoryDeprecatedWarning

  SCons.Warnings.DeprecatedSigModuleWarning

43.35.1  **Methods**

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

\[ x.__delattr__('name') \Rightarrow \text{del } x.name \]

Overrides: object.__delattr__

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

\[ x.__getattribute__('name') \Rightarrow x.name \]

Overrides: object.__getattribute__

```python
__getitem__(x, y) 
```

\[ x[y] \]

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

\[ x[i:j] \]

Use of negative indices is not supported.

```python
__hash__(x) 
```

\[ \text{hash}(x) \]

```python
__init__(...) 
```

\[ x.__init__(...) \text{ initializes } x; \text{ see } x.__class__.__doc__ \text{ for signature} \]

Overrides: exceptions.BaseException.__init__
Class DeprecatedSigModuleWarning  

_ModuleWarning_  

Return Value  

.. _new__: *(T, S, ...)*  

a new object with type S, a subtype of T  

Overrides: exceptions.BaseException._new_  

.. _reduce__: *(...)*  

tool helper for pickle  

Overrides: object._reduce_  

.. _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)  

Overides: object._str_  

### 43.35.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__class__</code></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
<tr>
<td><code>args</code></td>
<td>Value: <code>&lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
<tr>
<td><code>message</code></td>
<td>Value: <code>&lt;member 'message' of 'exceptions.BaseException' objects&gt;</code></td>
</tr>
</tbody>
</table>
43.36 Class DeprecatedBuilderKeywordsWarning

object

exceptions.BaseException

exceptions.Exception

SCons.Errors.UserError

SCons.Warnings.Warning

SCons.Warnings.DeprecatedWarning

SCons.Warnings.MandatoryDeprecatedWarning

SCons.Warnings.DeprecatedBuilderKeywordsWarning

43.36.1 Methods

```python
_delattr_ (name)

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

Overrides: object._delattr_
```

```python
_getattribute_ (name)

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

Overrides: object._getattribute_
```

```python
_getitem_ (x, y)

x[y]
```

```python
_getslice_ (x, i, j)

x[i:j]

Use of negative indices is not supported.
```

```python
_hash_ (x)

hash(x)
```

```python
_init_ (...)

x._init_(...) initializes x; see x._class_.__doc__ for signature

Overrides: exceptions.BaseException._init_
```
Class DeprecatedBuilderKeywordsWarning

Module SCons.Warnings

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

Return Value
   a new object with type S, a subtype of T
Overrides: exceptions.BaseException.__new__
```

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

```python
__reduce_ex__(...)
helper for pickle
```

```python
__repr__(x)
repr(x)
Overrides: object.__repr__
```

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

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

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

### 43.36.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
44 Package SCons.compat

SCons compatibility package for old Python versions

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

Other code will not generally reference things in this package through the SCons.compat namespace. The modules included here add things to the builtins namespace or the global module list so that the rest of our code can use the objects and names imported here regardless of Python version.

Simply enough, things that go in the builtins name space come from our _scons_builtins module.

The rest of the things here will be in individual compatibility modules that are either: 1) suitably modified copies of the future modules that we want to use; or 2) backwards compatible re-implementations of the specific portions of a future module's API that we want to use.

GENERAL WARNINGS: Implementations of functions in the SCons.compat modules are *NOT* guaranteed to be fully compliant with these functions in later versions of Python. We are only concerned with adding functionality that we actually use in SCons, so be wary if you lift this code for other uses. (That said, making these more nearly the same as later, official versions is still a desirable goal, we just don't need to be obsessive about it.)

We name the compatibility modules with an initial '_scons_' (for example, _scons_subprocess.py is our compatibility module for subprocess) so that we can still try to import the real module name and fall back to our compatibility module if we get an ImportError. The import_as() function defined below loads the module as the "real" name (without the '_scons'), after which all of the "import {module}" statements in the rest of our code will find our pre-loaded compatibility module.

44.1 Modules

- _scons_builtins: Compatibility idioms for builtins names
  This module adds names to the builtins module for things that we want to use in SCons but which don’t show up until later Python versions than the earliest ones we support.
  (Section 45, p. 569)
- _scons_collections: collections compatibility module for older (pre-2.4) Python versions
  This does not not NOT (repeat, *NOT*!) provide complete collections functionality.
  (Section 46, p. 572)
- _scons_dbm: dbm compatibility module for Python versions that don’t have dbm.
  (Section 47, p. 573)
• **scons_hashlib**: hashlib backwards-compatibility module for older (pre-2.5) Python versions
  This does not not *NOT* provide complete hashlib functionality.
  *(Section 48, p. 576)*
• **scons_io**: io compatibility module for older (pre-2.6) Python versions
  This does not not *NOT* provide complete io functionality.
  *(Section 49, p. 580)*
• **scons_sets**: Classes to represent arbitrary sets (including sets of sets).
  *(Section 50, p. 581)*
• **scons_subprocess**: subprocess - Subprocesses with accessible I/O streams
  This module allows you to spawn processes, connect to their input/output/error pipes, and obtain their return codes.
  *(Section 51, p. 594)*

### 44.2 Functions

**import_as(module, name)**

Imports the specified module (from our local directory) as the specified name, returning the loaded module object.

**rename_module(new, old)**

Attempts to import the old module and load it under the new name. Used for purely cosmetic name changes in Python 3.x.

**lexists(path)**

### 44.3 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>doc</strong></td>
<td>Value: ...</td>
</tr>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/compat/<strong>init</strong>.py 5023 2010/06/14 22:0...</td>
</tr>
</tbody>
</table>
45 Module SCons.compat._scons_builtins

Compatibility idioms for builtins names

This module adds names to the builtins module for things that we want
to use in SCons but which don’t show up until later Python versions than
the earliest ones we support.

This module checks for the following builtins names:

    all()
    any()
    sorted()
    memoryview()

Implementations of functions are *NOT* guaranteed to be fully compliant
with these functions in later versions of Python. We are only concerned
with adding functionality that we actually use in SCons, so be wary
if you lift this code for other uses. (That said, making these more
nearly the same as later, official versions is still a desirable goal,
we just don’t need to be obsessive about it.)

If you’re looking at this with pydoc and various names don’t show up in
the FUNCTIONS or DATA output, that means those names are already built in
to this version of Python and we don’t need to add them from this module.

45.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all(iterable)</td>
<td>Returns True if all elements of the iterable are true.</td>
</tr>
<tr>
<td>any(iterable)</td>
<td>Returns True if any element of the iterable is true.</td>
</tr>
<tr>
<td>sorted(iterable, cmp=None, key=None, reverse=False)</td>
<td></td>
</tr>
</tbody>
</table>

45.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>doc</strong></td>
<td>Value: ...</td>
</tr>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/compat/_scons_builtins.py 5023 2010/06/...</td>
</tr>
</tbody>
</table>
### 45.3 Class memoryview

```python
class memoryview:
    def __init__(self, obj):
        # Initialization of memoryview
        # For signature: object.__init__(...) initializes x; see object.__class__.__doc__ for signature
        # Overrides: object.__init__(inherited documentation)

def __getitem__(self, indx):
    # Get item

def __delattr__(self, name):
    # Delete attribute
    del self.name

def __getattr__(self, name):
    # Get attribute
    return self.name

def __hash__(self):
    # Hash

def __new__(T, S, ...):
    # Return a new object with type S, a subtype of T

def __reduce__(self):
    # Helper for pickle

def __reduce_ex__(self):
    # Helper for pickle

def __repr__(self):
    # Representation
    return self

def __setattr__(self, name, value):
    # Set attribute
    self.name = value
```

586
```python
__str__(x)
str(x)
```

### 45.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>


46 Module `SCons.compat._scons_collections`

collections compatibility module for older (pre-2.4) Python versions

This does not not NOT (repeat, *NOT*) provide complete collections functionality. It only wraps the portions of collections functionality used by SCons, in an interface that looks enough like collections for our purposes.

46.1 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>.doc</code></td>
<td>Value: ...</td>
</tr>
<tr>
<td><code>.revision</code></td>
<td>Value: <code>src/engine/SCons/compat/_scons_collections.py</code> 5023 2010/...</td>
</tr>
</tbody>
</table>
Module SCons.compat._scons_dbm

dbm compatibility module for Python versions that don’t have dbm.

This does not not NOT (repeat, *NOT*) provide complete dbm functionality. It’s just a stub on which to hang just enough pieces of dbm functionality that the whichdb.whichdb() implementation in the various 2.X versions of Python won’t blow up even if dbm wasn’t compiled in.

47.1 Functions

open(*args, **kw)

47.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>doc</strong></td>
<td>Value: ...</td>
</tr>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/compat/_scons_dbm.py 5023 2010/06/14 22...</td>
</tr>
</tbody>
</table>

47.3 Class error

object ─

exceptions.BaseException ─

exceptions.Exception ─

SCons.compat._scons_dbm.error

47.3.1 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]
Class error

Module SCons.compat._scons_dbm

__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__

47.3.2 Properties
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
<tr>
<td>args</td>
<td>Value: &lt;attribute 'args' of 'exceptions.BaseException' objects&gt;</td>
</tr>
<tr>
<td>message</td>
<td>Value: &lt;member 'message' of 'exceptions.BaseException' objects&gt;</td>
</tr>
</tbody>
</table>
48 Module SCons.compat._scons_hashlib

hashlib backwards-compatibility module for older (pre-2.5) Python versions

This does not not NOT (repeat, *NOT*) provide complete hashlib functionality. It only wraps the portions of MD5 functionality used by SCons, in an interface that looks like hashlib (or enough for our purposes, anyway). In fact, this module will raise an ImportError if the underlying md5 module isn’t available.

48.1 Functions

\texttt{md5(string='')}  

48.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_doc_</td>
<td>Value: ...</td>
</tr>
<tr>
<td>_revision_</td>
<td>Value: 'src/engine/SCons/compat/_scons_hashlib.py 5023 2010/06/1...</td>
</tr>
</tbody>
</table>

48.3 Class md5obj

\texttt{object \_\_\_init\_\_(self, name, string='')}  
\texttt{x._\_\_init\_\_(...) initializes x; see x._\_class\_\_._\_doc\_\_ for signature}  
\texttt{Overrides: object._\_\_init\_\_ extit(inherited documentation)}

\texttt{._\_repr\_\_(self)}  
\texttt{repr(x)}  
\texttt{Overrides: object._\_\_repr\_\_ extit(inherited documentation)}

\texttt{copy(self)}

\texttt{digest(self)}

\texttt{update(self, arg)}

\texttt{hexdigest(self)}
Class md5obj

Module SCons.compat.scons.hashlib

```python
__delattr__(...)  
x...delattr_('name') <-- del x.name

__getattribute__(...)  
x...getattribute_('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

__str__(x)  
str(x)
```

48.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

48.3.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_module</td>
<td>Value: &lt;module 'md5' from '/usr/lib/python2.5/md5.pyc'&gt;</td>
</tr>
</tbody>
</table>
### 48.4 Class md5obj

```
object
   SCons.compat.scons_hashlib.md5obj
```

#### 48.4.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_init_</code>(self, name, string='')</td>
<td><code>x._init__(...)</code> initializes <code>x</code>; see <code>x._class__._doc_</code> for signature. Overrides: <code>object._init_</code> (inherited documentation).</td>
</tr>
<tr>
<td><code>_repr_</code>(self)</td>
<td><code>repr(x)</code>; Overrides: <code>object._repr_</code> (inherited documentation).</td>
</tr>
<tr>
<td><code>copy(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>digest(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>update(self, ary)</code></td>
<td></td>
</tr>
<tr>
<td><code>hexdigest(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>__delattr__(...)</code></td>
<td><code>x.__delattr__('name')</code> == del <code>x.name</code></td>
</tr>
<tr>
<td><code>__getattr__(...)</code></td>
<td><code>x.__getattr__('name')</code> == <code>x.name</code></td>
</tr>
<tr>
<td><code>__hash__(x)</code></td>
<td><code>hash(x)</code></td>
</tr>
<tr>
<td><code>__new__(T, S, ...)</code></td>
<td>Return Value: A new object with type <code>S</code>, a subtype of <code>T</code></td>
</tr>
<tr>
<td><code>__reduce__(...)</code></td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>__reduce_ex__(...)</code></td>
<td>helper for pickle</td>
</tr>
</tbody>
</table>
__setattr__(...)  
x.__setattr__(‘name’, value) <=> x.name = value

__str__(x)  
str(x)

48.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute ‘<strong>class</strong>’ of ‘object’ objects&gt;</td>
</tr>
</tbody>
</table>

48.4.3 Class Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>md5_module</td>
<td>Value: &lt;module ‘md5’ from ‘/usr/lib/python2.5/md5.pyc’&gt;</td>
</tr>
</tbody>
</table>
Module SCons.compat._scons_io

io compatibility module for older (pre-2.6) Python versions

This does not not NOT (repeat, *NOT*) provide complete io functionality. It only wraps the portions of io functionality used by SCons, in an interface that looks enough like io for our purposes.

49.1 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>doc</strong></td>
<td>Value: ...</td>
</tr>
<tr>
<td><strong>revision</strong></td>
<td>Value: 'src/engine/SCons/compat/_scons_io.py 5023 2010/06/14 22:...</td>
</tr>
</tbody>
</table>
50 Module SCons.compat._scons_sets

Classes to represent arbitrary sets (including sets of sets).

This module implements sets using dictionaries whose values are ignored. The usual operations (union, intersection, deletion, etc.) are provided as both methods and operators.

Important: sets are not sequences! While they support 'x in s', 'len(s)', and 'for x in s', none of those operations are unique for sequences; for example, mappings support all three as well. The characteristic operation for sequences is subscripting with small integers: s[i], for i in range(len(s)). Sets don't support subscripting at all. Also, sequences allow multiple occurrences and their elements have a definite order; sets on the other hand don't record multiple occurrences and don't remember the order of element insertion (which is why they don't support s[i]).

The following classes are provided:

BaseSet -- All the operations common to both mutable and immutable sets. This is an abstract class, not meant to be directly instantiated.

Set -- Mutable sets, subclass of BaseSet; not hashable.

ImmutableSet -- Immutable sets, subclass of BaseSet; hashable.

   An iterable argument is mandatory to create an ImmutableSet.

_TemporarilyImmutableSet -- A wrapper around a Set, hashable, giving the same hash value as the immutable set equivalent would have. Do not use this class directly.

Only hashable objects can be added to a Set. In particular, you cannot really add a Set as an element to another Set; if you try, what is actually added is an ImmutableSet built from it (it compares equal to the one you tried adding).

When you ask if 'x in y' where x is a Set and y is a Set or ImmutableSet, x is wrapped into a _TemporarilyImmutableSet z, and what's tested is actually 'z in y'.

50.1 Class BaseSet

    object __init__(self)

SCons.compat._scons_sets.BaseSet

Known Subclasses: SCons.compat._scons_sets.ImmutableSet, SCons.compat._scons_sets.Set, SCons.compat._scons_sets._TemporarilyImmutableSet

Common base class for mutable and immutable sets.
### 50.1.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__(self)</code></td>
<td>This is an abstract class. Override: object.<strong>init</strong></td>
</tr>
<tr>
<td><code>__len__(self)</code></td>
<td>Return the number of elements of a set.</td>
</tr>
<tr>
<td><code>__repr__(self)</code></td>
<td>Return string representation of a set. This looks like 'Set([&lt;list of elements&gt;])'. Override: object.<strong>repr</strong></td>
</tr>
<tr>
<td><code>__str__(self)</code></td>
<td>Return string representation of a set. This looks like 'Set([&lt;list of elements&gt;])'. Override: object.<strong>str</strong></td>
</tr>
<tr>
<td><code>__iter__(self)</code></td>
<td>Return an iterator over the elements or a set. This is the keys iterator for the underlying dict.</td>
</tr>
<tr>
<td><code>__cmp__(self, other)</code></td>
<td></td>
</tr>
<tr>
<td><code>__eq__(self, other)</code></td>
<td></td>
</tr>
<tr>
<td><code>__ne__(self, other)</code></td>
<td></td>
</tr>
<tr>
<td><code>copy(self)</code></td>
<td>Return a shallow copy of a set.</td>
</tr>
<tr>
<td><code>__copy__(self)</code></td>
<td>Return a shallow copy of a set.</td>
</tr>
<tr>
<td><code>deepcopy_(self, memo)</code></td>
<td>Return a deep copy of a set; used by copy module.</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>__or__(self, other)</code></td>
<td>Return the union of two sets as a new set. (I.e. all elements that are in either set.)</td>
</tr>
<tr>
<td><code>union(self, other)</code></td>
<td>Return the union of two sets as a new set. (I.e. all elements that are in either set.)</td>
</tr>
<tr>
<td><code>__and__(self, other)</code></td>
<td>Return the intersection of two sets as a new set. (I.e. all elements that are in both sets.)</td>
</tr>
<tr>
<td><code>intersection(self, other)</code></td>
<td>Return the intersection of two sets as a new set. (I.e. all elements that are in both sets.)</td>
</tr>
<tr>
<td><code>__xor__(self, other)</code></td>
<td>Return the symmetric difference of two sets as a new set. (I.e. all elements that are in exactly one of the sets.)</td>
</tr>
<tr>
<td><code>symmetric_difference(self, other)</code></td>
<td>Return the symmetric difference of two sets as a new set. (I.e. all elements that are in exactly one of the sets.)</td>
</tr>
<tr>
<td><code>__sub__(self, other)</code></td>
<td>Return the difference of two sets as a new Set. (I.e. all elements that are in this set and not in the other.)</td>
</tr>
<tr>
<td><code>difference(self, other)</code></td>
<td>Return the difference of two sets as a new Set. (I.e. all elements that are in this set and not in the other.)</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td><strong>contains</strong></td>
<td>Report whether an element is a member of a set.</td>
</tr>
<tr>
<td>issubset</td>
<td>Report whether another set contains this set.</td>
</tr>
<tr>
<td>issuperset</td>
<td>Report whether this set contains another set.</td>
</tr>
<tr>
<td><strong>le</strong></td>
<td>Report whether another set contains this set.</td>
</tr>
<tr>
<td><strong>ge</strong></td>
<td>Report whether this set contains another set.</td>
</tr>
<tr>
<td><strong>lt</strong></td>
<td>Report whether another set contains this set.</td>
</tr>
<tr>
<td><strong>gt</strong></td>
<td>Report whether this set contains another set.</td>
</tr>
<tr>
<td><strong>delattr</strong></td>
<td>delattr(x)</td>
</tr>
<tr>
<td><strong>getattribute</strong></td>
<td>getattribute(x)</td>
</tr>
<tr>
<td><strong>hash</strong></td>
<td>hash(x)</td>
</tr>
<tr>
<td><strong>new</strong></td>
<td>new(T, S, ...)</td>
</tr>
<tr>
<td><strong>reduce</strong></td>
<td>reduce</td>
</tr>
<tr>
<td><strong>reduce_ex</strong></td>
<td>reduce_ex</td>
</tr>
</tbody>
</table>
50.1.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

50.2 Class ImmutableSet

Object 

SCons.compat.scons_sets.BaseSet 

SCons.compat.scons_sets.ImmutableSet

Immutable set class.

50.2.1 Methods

_init__(self, iterable=False)

Construct an immutable set from an optional iterable.

Overrides: SCons.compat.scons_sets.BaseSet.__init__

_hash__(self)

hash(x)

Overrides: object.__hash__ (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’.)
Class ImmutableSet

- **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.

- **getattribute__(...)**
  x.__getattribute__('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
### Class ImmutableSet

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `.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.                                              |

This looks like 'Set([<list of elements>])'.  
Overrides: object._repr_
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.)

50.2.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.<em>class</em></td>
<td>Value: &lt;attribute '<em>class</em>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

50.3 Class Set

object SCons.compat.scons_sets.BaseSet
SCons.compat.scons_sets.Set

Mutable set class.
### 50.3.1 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__init__</code> <em>(self, iterable=False)</em></td>
<td>Construct a set from an optional iterable. Overrides: SCons.compat.scons_sets.BaseSet.<strong>init</strong></td>
</tr>
<tr>
<td><code>__getstate__</code> <em>(self)</em></td>
<td></td>
</tr>
<tr>
<td><code>__setstate__</code> <em>(self, data)</em></td>
<td></td>
</tr>
<tr>
<td><code>__hash__</code> <em>(self)</em></td>
<td>A Set cannot be hashed. Overrides: object.<strong>hash</strong></td>
</tr>
<tr>
<td><code>__ior__</code> <em>(self, other)</em></td>
<td>Update a set with the union of itself and another.</td>
</tr>
<tr>
<td><code>union_update</code> <em>(self, other)</em></td>
<td>Update a set with the union of itself and another.</td>
</tr>
<tr>
<td><code>__iand__</code> <em>(self, other)</em></td>
<td>Update a set with the intersection of itself and another.</td>
</tr>
<tr>
<td><code>intersection_update</code> <em>(self, other)</em></td>
<td>Update a set with the intersection of itself and another.</td>
</tr>
<tr>
<td><code>__ixor__</code> <em>(self, other)</em></td>
<td>Update a set with the symmetric difference of itself and another.</td>
</tr>
<tr>
<td><code>symmetric_difference_update</code> <em>(self, other)</em></td>
<td>Update a set with the symmetric difference of itself and another.</td>
</tr>
<tr>
<td><code>__isub__</code> <em>(self, other)</em></td>
<td>Remove all elements of another set from this set.</td>
</tr>
<tr>
<td><code>difference_update</code> <em>(self, other)</em></td>
<td>Remove all elements of another set from this set.</td>
</tr>
<tr>
<td><code>update</code> <em>(self, iterable)</em></td>
<td>Add all values from an iterable (such as a list or file).</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>clear(self)</code></td>
<td>Remove all elements from this set.</td>
</tr>
<tr>
<td><code>add(self, element)</code></td>
<td>Add an element to a set. This has no effect if the element is already present.</td>
</tr>
<tr>
<td><code>remove(self, element)</code></td>
<td>Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.</td>
</tr>
<tr>
<td><code>discard(self, element)</code></td>
<td>Remove an element from a set if it is a member. If the element is not a member, do nothing.</td>
</tr>
<tr>
<td><code>pop(self)</code></td>
<td>Remove and return an arbitrary set element.</td>
</tr>
<tr>
<td><code>_as_immutable_(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>_as_temporarily_immutable_(self)</code></td>
<td></td>
</tr>
<tr>
<td><code>_and_(self, other)</code></td>
<td>Return the intersection of two sets as a new set. (I.e. all elements that are in both sets.)</td>
</tr>
<tr>
<td><code>_cmp_(self, other)</code></td>
<td></td>
</tr>
<tr>
<td><code>_contains_(self, element)</code></td>
<td>Report whether an element is a member of a set. (Called in response to the expression ‘element in self’.)</td>
</tr>
<tr>
<td><code>_copy_(self)</code></td>
<td>Return a shallow copy of a set.</td>
</tr>
<tr>
<td><code>_deepcopy_(self, memo)</code></td>
<td>Return a deep copy of a set; used by copy module.</td>
</tr>
</tbody>
</table>
Class Set

Module SCons.compat_.scons_sets

```python
__delattr__(...)  
x.__delattr__('name') == del x.name

__eq__(self, other)

__ge__(self, other)
Report whether this set contains another set.

__gt__(self, other)

__getattribute__(...)  
x.__getattribute__('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
```
Class Set

Module SCons.compat_scons_sets

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_reduce_ex_</code>(...)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>_repr_</code>(self)</td>
<td>Return string representation of a set.</td>
</tr>
<tr>
<td></td>
<td>This looks like 'Set([&lt;list of elements&gt;])'.</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<em>repr</em></td>
</tr>
<tr>
<td><code>_setattr_</code>(...)</td>
<td>x.<em>setattr</em>('name', value) =&gt; x.name = value</td>
</tr>
<tr>
<td><code>_str_</code>(self)</td>
<td>Return string representation of a set.</td>
</tr>
<tr>
<td></td>
<td>This looks like 'Set([&lt;list of elements&gt;])'.</td>
</tr>
<tr>
<td></td>
<td>Overrides: object.<em>str</em></td>
</tr>
<tr>
<td><code>_sub_</code>(self, other)</td>
<td>Return the difference of two sets as a new Set.</td>
</tr>
<tr>
<td></td>
<td>(I.e. all elements that are in this set and not in the other.)</td>
</tr>
<tr>
<td><code>_xor_</code>(self, other)</td>
<td>Return the symmetric difference of two sets as a new set.</td>
</tr>
<tr>
<td></td>
<td>(I.e. all elements that are in exactly one of the sets.)</td>
</tr>
<tr>
<td><code>copy</code>(self)</td>
<td>Return a shallow copy of a set.</td>
</tr>
<tr>
<td><code>difference</code>(self, other)</td>
<td>Return the difference of two sets as a new Set.</td>
</tr>
<tr>
<td></td>
<td>(I.e. all elements that are in this set and not in the other.)</td>
</tr>
<tr>
<td><code>intersection</code>(self, other)</td>
<td>Return the intersection of two sets as a new set.</td>
</tr>
<tr>
<td></td>
<td>(I.e. all elements that are in both sets.)</td>
</tr>
</tbody>
</table>
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.)

50.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
51 Module SCons.compat._scons_subprocess

subprocess - Subprocesses with accessible I/O streams

This module allows you to spawn processes, connect to their
input/output/error pipes, and obtain their return codes. This module
intends to replace several other, older modules and functions, like:

os.system
os.spawn*
os.popen*
popen2.*
commands.*

Information about how the subprocess module can be used to replace these
modules and functions can be found below.

Using the subprocess module
============================

This module defines one class called Popen:

class Popen(args, bufsize=0, executable=None,
            stdin=None, stdout=None, stderr=None,
            preexec_fn=None, close_fds=False, shell=False,
            cwd=None, env=None, universal_newlines=False,
            startupinfo=None, creationflags=0):

Arguments are:

args should be a string, or a sequence of program arguments. The
program to execute is normally the first item in the args sequence or
string, but can be explicitly set by using the executable argument.

On UNIX, with shell=False (default): In this case, the Popen class
uses os.execvp() to execute the child program. args should normally
be a sequence. A string will be treated as a sequence with the string
as the only item (the program to execute).

On UNIX, with shell=True: If args is a string, it specifies the
command string to execute through the shell. If args is a sequence,
the first item specifies the command string, and any additional items
will be treated as additional shell arguments.

On Windows: the Popen class uses CreateProcess() to execute the child
program, which operates on strings. If args is a sequence, it will be
converted to a string using the list2cmdline method. Please note that
not all MS Windows applications interpret the command line the same
way: The list2cmdline is designed for applications using the same
rules as the MS C runtime.

 bufsize, if given, has the same meaning as the corresponding argument to the built-in open() function: 0 means unbuffered, 1 means line buffered, any other positive value means use a buffer of (approximately) that size. A negative bufsize means to use the system default, which usually means fully buffered. The default value for bufsize is 0 (unbuffered).

 stdin, stdout and stderr specify the executed programs’ standard input, standard output and standard error file handles, respectively. Valid values are PIPE, an existing file descriptor (a positive integer), an existing file object, and None. PIPE indicates that a new pipe to the child should be created. With None, no redirection will occur; the child’s file handles will be inherited from the parent. Additionally, stderr can be STDOUT, which indicates that the stderr data from the applications should be captured into the same file handle as for stdout.

 If preexec_fn is set to a callable object, this object will be called in the child process just before the child is executed.

 If close_fds is true, all file descriptors except 0, 1 and 2 will be closed before the child process is executed.

 if shell is true, the specified command will be executed through the shell.

 If cwd is not None, the current directory will be changed to cwd before the child is executed.

 If env is not None, it defines the environment variables for the new process.

 If universal_newlines is true, the file objects stdout and stderr are opened as a text files, but lines may be terminated by any of 'n', the Unix end-of-line convention, 'r', the Macintosh convention or 'rn', 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"])

call(*popenargs, **kwargs):
    Run command with arguments. Wait for command to complete. If the exit code was zero then return, otherwise raise CalledProcessError. The CalledProcessError object will have the return code in the returncode attribute.

    The arguments are the same as for the Popen constructor. Example:

        check_call(["ls", "-l"])

Exceptions
-----------
Exceptions raised in the child process, before the new program has started to execute, will be re-raised in the parent. Additionally, the exception object will have one extra attribute called 'child_traceback’, which is a string containing traceback information from the childs point of view.

The most common exception raised is OSError. This occurs, for example, when trying to execute a non-existent file. Applications should prepare for OSErrors.

A ValueError will be raised if Popen is called with invalid arguments.

check_call() will raise CalledProcessError, if the called process returns a non-zero return code.

Security
--------
Unlike some other popen functions, this implementation will never call /bin/sh implicitly. This means that all characters, including shell metacharacters, can safely be passed to child processes.

Popen objects
-------------
Instances of the Popen class have the following methods:

poll()
    Check if child process has terminated. Returns returncode attribute.
wait()
    Wait for child process to terminate. Returns returncode attribute.

communicate(input=None)
    Interact with process: Send data to stdin. Read data from stdout and stderr, until end-of-file is reached. Wait for process to terminate. The optional stdin argument should be a string to be sent to the child process, or None, if no data should be sent to the child.

    communicate() returns a tuple (stdout, stderr).

    Note: The data read is buffered in memory, so do not use this method if the data size is large or unlimited.

The following attributes are also available:

stdin
    If the stdin argument is PIPE, this attribute is a file object that provides input to the child process. Otherwise, it is None.

stdout
    If the stdout argument is PIPE, this attribute is a file object that provides output from the child process. Otherwise, it is None.

stderr
    If the stderr argument is PIPE, this attribute is file object that provides error output from the child process. Otherwise, it is None.

pid
    The process ID of the child process.

returncode
    The child return code. A None value indicates that the process hasn't terminated yet. A negative value -N indicates that the child was terminated by signal N (UNIX only).

Replacing older functions with the subprocess module
===============================================

In this section, "a => b" means that b can be used as a replacement for a.

Note: All functions in this section fail (more or less) silently if the executed program cannot be found; this module raises an OSSError 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:

```python
retcode = os.spawnlp(os.P_WAIT, "/bin/mycmd", "mycmd", "myarg")
==>
retcode = call(
    ["/bin/mycmd", "myarg"])
```

Vector example:

```python
os.spawnvp(os.P_NOWAIT, path, args)
==>
Popen([path] + args[1:])
```

Environment example:

```python
os.spawnlpe(os.P_NOWAIT, "/bin/mycmd", "mycmd", "myarg", env)
==>
Popen(["/bin/mycmd", "myarg"], env="PATH": "/usr/bin")
```

Replacing os.popen*
-------------------

```python
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
```

```python
(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)
```

```python
(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)
```
Functions

Module SCons.compat._scons_subprocess

```
(child.stdin, child_stdout_and_stderr) = os.popen4(cmd, mode, bufsize)
==> p = Popen(cmd, shell=True, bufsize=bufsize,
              stdin=PIPE, stdout=PIPE, stderr=STDOUT, close_fds=True)
(child.stdin, child_stdout_and_stderr) = (p.stdin, p.stdout)
```

Replacing popen2.*

Note: If the cmd argument to popen2 functions is a string, the command
is executed through /bin/sh. If it is a list, the command is directly
executed.

```
(child_stdout, child_stdin) = popen2.popen2("somestring", bufsize, mode)
==> p = Popen(["somestring"], shell=True, bufsize=bufsize
              stdin=PIPE, stdout=PIPE, close_fds=True)
(child_stdout, child_stdin) = (p.stdout, p.stdin)
```

```
(child_stdout, child_stdin) = popen2.popen2(["mycmd", "myarg"], bufsize, mode)
==> p = Popen(["mycmd", "myarg"], bufsize=bufsize,
                stdin=PIPE, stdout=PIPE, close_fds=True)
(child_stdout, child_stdin) = (p.stdout, p.stdin)
```

The popen2.Popen3 and popen3.Popen4 basically works as subprocess.Popen,
except that:

* subprocess.Popen raises an exception if the execution fails
* the capturestderr argument is replaced with the stderr argument.
* stdin=PIPE and stdout=PIPE must be specified.
* popen2 closes all filedescriptors by default, but you have to specify
close_fds=True with subprocess.Popen.

### 51.1 Functions

```
call(*popenargs, **kwargs)
```

Run command with arguments. Wait for command to complete, then
return the returncode attribute.

The arguments are the same as for the Popen constructor. Example:

```
retcode = call(["ls", "-l"])
```
check_call(*popenargs, **kwargs)

Run command with arguments. Wait for command to complete. If the exit code was zero then return, otherwise raise CalledProcessError. The CalledProcessError object will have the return code in the returncode attribute.

The arguments are the same as for the Popen constructor. Example:

check_call(['ls', '-l'])

51.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIPE</td>
<td>Value: -1</td>
</tr>
<tr>
<td>STDOUT</td>
<td>Value: -2</td>
</tr>
</tbody>
</table>

51.3 Class CalledProcessError

object

exceptions.BaseException

exceptions.Exception

SCons.compat.scons_subprocess.CalledProcessError

This exception is raised when a process run by check_call() returns a non-zero exit status. The exit status will be stored in the returncode attribute.

51.3.1 Methods

_init__(self, returncode, cmd)

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

Overrides: exceptions.Exception.__init__(inherited documentation)

_str__(self)

str(x)

Overrides: exceptions.BaseException.__str__ (inherited documentation)

delattr(...)  

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

Overrides: object._delattr__
Class CalledProcessError

Module SCons.compat.scons_subprocess

```python
__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)

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

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

__reduce_ex__(...)  
helper for pickle

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

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

__setstate__(...)  

51.3.2 Properties  
```
## 51.4 Class Popen

```
class Popen

Name | Description
--- | ---
_class_ | Value: <attribute '._class_' of 'object' objects>
args | Value: <attribute 'args' of 'exceptions.BaseException' objects>
message | Value: <member 'message' of 'exceptions.BaseException' objects>
```

### 51.4.1 Methods

- **init**(self, args, bufsize=0, executable=False, stdin=False, stdout=False, stderr=False, preexec_fn=False, close_fds=False, shell=False, cwd=False, env=False, universal_newlines=False, startupinfo=False, creationflags=0)

Create new Popen instance.  
 Overrides: object.

- **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**(name)

  x._delattr_('name') <=> del x.name
51.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
52 Module SCons.cpp

SCons C Pre-Processor module

52.1 Functions

<table>
<thead>
<tr>
<th>CPP_to_Python_Ops_Sub(m)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CPP_to_Python(s)</th>
</tr>
</thead>
</table>

Converts a C pre-processor expression into an equivalent Python expression that can be evaluated.

52.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>doc</strong></td>
<td>Value: ...</td>
<td></td>
</tr>
<tr>
<td>cpp_lines_dict</td>
<td>Value: {('define'):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'\s+([A-Za-z][A-Za-z0-9]*)(\([^)]))?...</td>
<td></td>
</tr>
<tr>
<td>Table</td>
<td>Value: {('define'):</td>
<td></td>
</tr>
<tr>
<td></td>
<td>re.compile(r'&quot;s+([A-Za-z][A-Za-z0-9]*)(\([^)]))?...</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>Value: &quot;^[s*#\s*(elif</td>
<td>undef</td>
</tr>
<tr>
<td>CPP_Expression</td>
<td>Value: re.compile(r'(?m)^&quot;.<em>#\s</em>(elif</td>
<td>undef</td>
</tr>
<tr>
<td>CPP_to_Python_Ops_Dict</td>
<td>Value: {&quot;x&quot;: ', '!', 'not', '!=', '!', '!'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'and', '!'</td>
<td></td>
</tr>
<tr>
<td>CPP_to_Python_Ops_Expression</td>
<td>Value: re.compile(r'&quot;[^]</td>
<td>&amp;</td>
</tr>
<tr>
<td>CPP_to_Python_Eval_List</td>
<td>Value: re.compile(r'&quot;[^]</td>
<td>&amp;</td>
</tr>
<tr>
<td>line_continuations</td>
<td>Value: re.compile(r'&quot;[^]</td>
<td>\r?\n')</td>
</tr>
<tr>
<td>function_name</td>
<td>Value: re.compile(r'&quot;s+')</td>
<td></td>
</tr>
<tr>
<td>function_arg_separator</td>
<td>Value: re.compile(r'&quot;s+')</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>Value: 'if'</td>
<td></td>
</tr>
</tbody>
</table>

52.3 Class FunctionEvaluator

object SCons.cpp.FunctionEvaluator

Handles delayed evaluation of a #define function call.
52.3.1 Methods

```python
__init__(self, name, args, expansion)
```
Squirrels away the arguments and expansion value of a #define macro function for later evaluation when we must actually expand a value that uses it.
Overrides: object.__init__

```python
__call__(self, *values)
```
Evaluates the expansion of a #define macro function called with the specified values.

```python
__delattr__(...)  
x.__delattr__('name') == del x.name
```

```python
__getattribute__(...)  
x.__getattribute__('name') == x.name
```

```python
__hash__(x)
```
hash(x)

```python
__new__(T, S, ...)
```
Return Value  
a new object with type S, a subtype of T

```python
__reduce__(...)
```
helper for pickle

```python
__reduce_ex__(...)
```
helper for pickle

```python
__repr__(x)
```
repr(x)

```python
__setattr__(...)
```
x.__setattr__('name', value) == x.name = value
52.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: <code>&lt;attribute '__class__' of 'object' objects&gt;</code></td>
</tr>
</tbody>
</table>

52.4 Class PreProcessor

Object  

```
SCons.cpp.PreProcessor
```

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

The main workhorse class for handling C pre-processing.

52.4.1 Methods

```python
__call__(self, file)
```

Pre-processes a file.

This is the main public entry point.

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

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

```python
__hash__(x)
```

```python
hash(x)
```

```python
__init__(self, current='.', cpppath=(), dict={}, all=0)
```

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

Overrides: object.__init__ (inherited documentation)
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_new_</code>(<em>T, S, ...</em>)</td>
<td>Return Value</td>
</tr>
<tr>
<td></td>
<td>a new object with type S, a subtype of T</td>
</tr>
<tr>
<td><code>_reduce_</code>(...)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>_reduce_ex_</code>(...)</td>
<td>helper for pickle</td>
</tr>
<tr>
<td><code>_repr_</code>(<em>x</em>)</td>
<td><code>repr(x)</code></td>
</tr>
<tr>
<td><code>_setattr_</code>(...)</td>
<td><code>x._setattr_('name', value) &lt;= x.name = value</code></td>
</tr>
<tr>
<td><code>_str_</code>(<em>x</em>)</td>
<td><code>str(x)</code></td>
</tr>
<tr>
<td><code>all_include(*self, t)</code></td>
<td></td>
</tr>
<tr>
<td><code>do_define(*self, t)</code></td>
<td>Default handling of a <code>#define</code> line.</td>
</tr>
<tr>
<td><code>do_elif(*self, t)</code></td>
<td>Default handling of a <code>#elif</code> line.</td>
</tr>
<tr>
<td><code>do_else(*self, t)</code></td>
<td>Default handling of a <code>#else</code> line.</td>
</tr>
<tr>
<td><code>do_endif(*self, t)</code></td>
<td>Default handling of a <code>#endif</code> line.</td>
</tr>
</tbody>
</table>
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-ized #include line.
This handles recursive expansion of values without "" or <>
surrounding the name until an initial " or < is found, to handle
#include FILE
where FILE is a #define somewhere else.

restore(self)

Pops the previous dispatch table off the stack and makes it the
current one.

save(self)

Pushes the current dispatch table on the stack and re-initializes
the current dispatch table to the default.

scons_current_file(self, t)

start_handling_includes(self, t=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.

52.4.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>

52.5 Class DumbPreProcessor

A preprocessor that ignores all #if/#elif/#else/#endif directives and just reports back *all* of the #include files (like the classic SCons scanner did).

This is functionally equivalent to using a regular expression to find all of the #include lines, only slower. It exists mainly as an example of how the main PreProcessor class can be sub-classed to tailor its behavior.

52.5.1 Methods

__init__(self, *args, **kw)

Overrides: SCons.cpp.PreProcessor.__init__
Class DumbPreProcessor

Module SCons.cpp

```python
__call__(self, file)

Pre-processes a file.

This is the main public entry point.

__delattr__(...)

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

__getattribute__(...)

x.__getattribute__('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)

all_include(self, t)
```
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>do_define(self, t)</code></td>
<td>Default handling of a #define line.</td>
</tr>
<tr>
<td><code>do_else(self, t)</code></td>
<td>Default handling of a #else line.</td>
</tr>
<tr>
<td><code>do_elif(self, t)</code></td>
<td>Default handling of a #elif line.</td>
</tr>
<tr>
<td><code>do_if(self, t)</code></td>
<td>Default handling of a #if line.</td>
</tr>
<tr>
<td><code>doIFDEF(self, t)</code></td>
<td>Default handling of a #ifdef line.</td>
</tr>
<tr>
<td><code>do ifndef(self, t)</code></td>
<td>Default handling of a ifndef line.</td>
</tr>
<tr>
<td><code>do_import(self, t)</code></td>
<td>Default handling of a #import line.</td>
</tr>
<tr>
<td><code>do_include(self, t)</code></td>
<td>Default handling of a #include line.</td>
</tr>
<tr>
<td><code>do_include_next(self, t)</code></td>
<td>Default handling of a #include line.</td>
</tr>
</tbody>
</table>
### Class DumbPreProcessor

**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-ized `#include` line.

This handles recursive expansion of values without `"` or `<>` surrounding the name until an initial `"` or `<` is found, to handle `#include FILE` where FILE is a `#define` somewhere else.
```
restore(self)

Pops the previous dispatch table off the stack and makes it the current one.

save(self)

Pushes the current dispatch table on the stack and re-initializes the current dispatch table to the default.

scons_current_file(self, t)

start_handling_includes(self, t=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.

52.5.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '.<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
```
53 Module SCons.dblite

53.1 Functions

```python
corruption_warning(filename)
```

```python
is_string(s)
```

```python
unicode(s)
```

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

53.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep_all_files</td>
<td>Value: 0</td>
</tr>
<tr>
<td>ignore_corrupt_dbfiles</td>
<td>Value: 0</td>
</tr>
<tr>
<td>dblite_suffix</td>
<td>Value: '.dblite'</td>
</tr>
<tr>
<td>tmp_suffix</td>
<td>Value: '.tmp'</td>
</tr>
</tbody>
</table>

53.3 Class dblite

```python
object
    SCons.dblite.dblite
```

53.3.1 Methods

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

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

Overrides: object.__init__(inherited documentation)

```python
__del__(self)
```

```python
sync(self)
```

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

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

```python
keys(self)
```

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

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

632
iterkeys(self)

_iter__(self)

__len__(self)

__delattr__(...)

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

__getattribute__(...)

x.__getattribute__('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)

53.3.2 Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td>Value: &lt;attribute '<strong>class</strong>' of 'object' objects&gt;</td>
</tr>
</tbody>
</table>
54 Module SCons.exitfuncs

SCons.exitfuncs

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

54.1 Functions

```python
register(func, *targs, **kargs)
```

register a function to be executed upon normal program termination

*func* - function to be called at exit
*targs* - optional arguments to pass to func
*kargs* - optional keyword arguments to pass to func

54.2 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>...revision...</td>
<td>Value: 'src/engine/SCons/exitfuncs.py 5023 2010/06/14 22:05:46 s...</td>
</tr>
<tr>
<td>x</td>
<td>Value: sys.exitfunc</td>
</tr>
</tbody>
</table>
55  Module md5

55.1  Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blocksize</td>
<td>Value: 1</td>
</tr>
<tr>
<td>digest_size</td>
<td>Value: 16</td>
</tr>
</tbody>
</table>
INDEX

SCons.SConf (module), 289–312
  SCons.SConf.CheckCC (function), 289
  SCons.SConf.CheckCHeader (function), 289
  SCons.SConf.CheckContext (class), 310–312
  SCons.SConf.CheckCXX (function), 289
  SCons.SConf.CheckCXXHeader (function), 290
  SCons.SConf.CheckDeclaration (function), 289
  SCons.SConf.CheckFunc (function), 289
  SCons.SConf.CheckHeader (function), 289
  SCons.SConf.CheckLib (function), 290
  SCons.SConf.CheckLibWithHeader (function), 290
  SCons.SConf.CheckSHCC (function), 289
  SCons.SConf.CheckSHCXX (function), 289
  SCons.SConf.CheckType (function), 289
  SCons.SConf.CheckTypeSize (function), 289
  SCons.SConf.ConfigureCacheError (class), 296–298
  SCons.SConf.ConfigureDryRunError (class), 294–296
  SCons.SConf.CreateConfigHBuilder (function), 289
  SCons.SConf.createIncludesFromHeaders (function), 289
  SCons.SConf.SConf (function), 289
  SCons.SConf.SConfBase (class), 307–310
  SCons.SConf.SConfBuildInfo (class), 298–300
  SCons.SConf.SConfBuildTask (class), 301–307
  SCons.SConf.SConfError (class), 292–294
  SCons.SConf.SConfWarning (class), 291–292
  SCons.SConf.SetBuildType (function), 289
  SCons.SConf.SetCacheMode (function), 289
  SCons.SConf.SetProgressDisplay (function), 289
  SCons.SConf.Streamer (class), 300–301
SCons.SConsign (module), 313–323
  SCons.SConsign.Base (class), 315–316
  SCons.SConsign.corrupt_dblite_warning (function), 313
  SCons.SConsign.DB (class), 316–318, 322–323
  SCons.SConsign.Dir (class), 318–320
  SCons.SConsign.DirFile (class), 320–322
  SCons.SConsign.File (function), 313
  SCons.SConsign.GetDataBase (function), 313
  SCons.SConsign.Reset (function), 313
  SCons.SConsign.SConsignEntry (class), 313–315
  SCons.SConsign.write (function), 313
SCons.Script (module)
  SCons.Script.HelpFunction (function), 366
  SCons.Script.Options (function), 366
  SCons.Script.TargetList (class), 370–372
SCons.Script.Variables (function), 366
SCons.Script (package), 366–372
  SCons.Script.Interactive (module), 373–376
  SCons.Script.Main (module), 377–403
  SCons.Script.SConscript (module), 404–419
SCons.Sig (module), 420–423
  SCons.Sig.MD5Null (class), 420–422
  SCons.Sig.TimestampNull (class), 422–423
SCons.Subst (module), 424–437
SCons.Taskmaster (module), 438–458
  SCons.Taskmaster.AlwaysTask (class), 445–450
  SCons.Taskmaster.dump_stats (function), 438
  SCons.Taskmaster.find_cycle (function), 438
  SCons.Taskmaster.OutOfDateTask (class), 450–455
  SCons.Taskmaster.Stats (class), 439–440
  SCons.Taskmaster.Task (class), 440–445
  SCons.Taskmaster.Taskmaster (class), 455–458
SCons.Util (module), 459–487
SCons.Variables (package), 488–491
  SCons.Variables.BoolVariable (module), 492
  SCons.Variables.EmuVariable (module), 493
  SCons.Variables.ListVariable (module), 494
  SCons.Variables.PackageVariable (module), 495
  SCons.Variables.PathVariable (module), 496–497
  SCons.Variables.Variables (class), 488–491
SCons.Warnings (module), 498–566
SCons.Action._ActionAction (class)
  SCons.Action._ActionAction._call__ (method), 10, 17
  SCons.Action._ActionAction.print_cmd_line (method), 12, 16, 18
SCons.Action._Action (function), 7
SCons.Action._ActionBase (class), 8–10
  SCons.Action._ActionBase._add__ (method), 8, 10, 13, 15, 17, 19
  SCons.Action._ActionBase._cmp__ (method), 8, 11, 13, 15, 17, 20
  SCons.Action._ActionBase._radd__ (method), 8, 11, 13, 15, 18, 20
  SCons.Action._ActionBase.genstring (method), 8, 11, 18
  SCons.Action._ActionBase.get_contents (method), 8, 11, 14, 16, 18, 20
  SCons.Action._ActionBase.get_targets (method), 8, 11, 18, 20
  SCons.Action._ActionBase.get_varlist (method), 8, 12, 18
  SCons.Action._ActionBase.no_batch_key (method), 8, 11, 12, 14, 16, 18, 20

639
INDEX

SCons.Action.ActionBase.presub_lines (method), 8, 12, 14, 16, 18
SCons.Action.ActionCaller (class), 21–22
SCons.Action.ActionCaller._call__ (method), 21
SCons.Action.ActionCaller.get_contents (method), 21
SCons.Action.ActionCaller.strfunction (method), 21
SCons.Action.ActionCaller.subst (method), 21
SCons.Action.ActionCaller.subst_args (method), 21
SCons.Action.ActionCaller.subst_kw (method), 21
SCons.Action.ActionFactory (class), 22–24
SCons.Action.ActionFactory._call__ (method), 23
SCons.Action.CommandAction (class), 10–12
SCons.Action.CommandAction.execute (method), 10, 16
SCons.Action.CommandAction.get_implicit_deps (method), 10
SCons.Action.CommandAction.get_presig (method), 19
SCons.Action.CommandAction.process (method), 10, 16
SCons.Action.CommandAction.strfunction (method), 10, 16
SCons.Action.CommandGeneratorAction (class), 12–14
SCons.Action.CommandGeneratorAction._call__ (method), 12
SCons.Action.CommandGeneratorAction.get_implicit_deps (method), 13, 16
SCons.Action.CommandGeneratorAction.get_presig (method), 13
SCons.Action.default_exitstatfunc (function), 7
SCons.Action.FunctionAction (class), 17–19
SCons.Action.FunctionAction.execute (method), 17
SCons.Action.FunctionAction.function_name (method), 17
SCons.Action.FunctionAction.get_implicit_deps (method), 17
SCons.Action.FunctionAction.get_presig (method), 17
SCons.Action.FunctionAction.strfunction (method), 17
SCons.Action.get_default_ENV (function), 7
SCons.Action.LazyAction (class), 14–17
SCons.Action.LazyAction.get_parent_class (method), 14
SCons.Action.ListAction (class), 19–21
SCons.Action.ListAction._call__ (method), 19
SCons.Action.ListAction.get_implicit_deps (method), 19
SCons.Action.ListAction.get_presig (method), 19
SCons.Action.remove_set_lineno_codes (function), 7
SCons.Action.writeFile (function), 7
SCons.Conftest.CheckBuilder (function), 44
SCons.Conftest.CheckCC (function), 44
SCons.Conftest.CheckCXX (function), 44
SCons.Conftest.CheckDeclaration (function), 46
SCons.Conftest.CheckFunc (function), 45
SCons.Conftest.CheckHeader (function), 45
SCons.Conftest.CheckLib (function), 46
SCons.Conftest.CheckSHCC (function), 44
SCons.Conftest.CheckSHCXX (function), 44
SCons.Conftest.CheckType (function), 45
SCons.Conftest.CheckTypeSize (function), 45
SCons.cpp.CPP_to_Python (function), 605
SCons.cpp.CPP_to_Python_Ops_Sub (function), 605
SCons.cpp.DumbPreProcessor (class), 611–616
SCons.cpp.FunctionEvaluator (class), 605–607
SCons.cpp.FunctionEvaluator._call__ (method), 606
SCons.cpp.PreProcessor (class), 607–611
SCons.cpp.PreProcessor._call__ (method), 343, 607, 611
SCons.cpp.PreProcessor.all_include (method), 344, 608, 612
SCons.cpp.PreProcessor.do_define (method), 344, 608, 612
SCons.cpp.PreProcessor.do_elif (method), 344, 608, 613
SCons.cpp.PreProcessor.do_else (method), 344, 608, 613
SCons.cpp.PreProcessor.do_endif (method), 344, 608, 613
SCons.cpp.PreProcessor.do_if (method), 344, 608, 613
SCons.cpp.PreProcessor.do_ifdef (method), 344, 609, 613
SCons.cpp.PreProcessor.do_ifndef (method), 344, 609, 613
SCons.cpp.PreProcessor.do_import (method), 344, 609, 613
SCons.cpp.PreProcessor.do_include (method), 344, 345, 609, 613
SCons.cpp.PreProcessor.do_nothing (method), 345, 609, 613
SCons.cpp.PreProcessor.do_undef (method), 345, 609, 614
SCons.cpp.PreProcessor.eval_expression (method), 345, 609, 614
SCons.cpp.PreProcessor.finalize_result (method),
SCons.Node.Node
  .scan (method), 155, 175, 195, 208, 233, 249, 268, 286
  .Node.Node.scanner_key (method), 155, 175, 196, 286
SCons.Node.Node
  .select_scanner (method), 155, 175, 196, 208, 234, 250, 269, 286
SCons.Node.Node
  .set_always_build (method), 155, 175, 196, 209, 234, 250, 269, 286
SCons.Node.Node
  .set_executor (method), 155, 175, 196, 209, 234, 250, 269, 286
SCons.Node.Node
  .set_explicit (method), 155, 175, 196, 209, 234, 250, 269, 286
SCons.Node.Node
  .set_nocache (method), 155, 175, 196, 209, 234, 250, 269, 286
SCons.Node.Node
  .set_noclean (method), 155, 175, 196, 209, 234, 250, 269, 286
SCons.Node.Node
  .set_precious (method), 155, 175, 196, 209, 234, 250, 269, 286
SCons.Node.Node
  .set_specific_source (method), 155, 175, 196, 209, 234, 250, 269, 286
SCons.Node.Node
  .set_state (method), 155, 175, 196, 209, 234, 250, 269, 286
SCons.Node.Node
  .state_has_changed (method), 155, 175, 196, 210, 235, 251, 270, 287
SCons.Node.Node
  .store_info (method), 155, 175, 196, 210, 235, 251, 287
SCons.Node.Node
  .visited (method), 155, 175, 196, 210, 235, 251, 287
SCons.Node.NodeInfoBase (class), 142–144
  .NodeInfoBase.convert (method), 143, 162, 218, 253, 274
SCons.Node.NodeInfoBase.format (method), 143, 162, 218, 253, 274
SCons.Node.NodeInfoBase.merge (method), 143, 162, 218, 253, 274
SCons.Node.NodeList (class), 156–157
  .NodeList.__str__ (method), 156
SCons.Node.Walker (class), 157–159
SCons.Node.Walker.get_next (method), 158
SCons.Node.Walker.is_done (method), 158
SCons.Scanner.Dir.DirEntryScanner (function), 352
SCons.Scanner.Dir.DirScanner (function), 352
SCons.Scanner.Dir.do_not_scan (function), 352
SCons.Scanner.Dir.only_dirs (function), 352
SCons.Scanner.Dir.scan_in_memory (function), 352
SCons.Scanner.Dir.scan_on_disk (function), 352
SCons.Script.Interactive.interact (function), 373
SCons.Script.Interactive.SConsInteractiveCmd (class), 373–376
SCons.Script.Interactive.SConsInteractiveCmd.do_build (method), 374
SCons.Script.Interactive.SConsInteractiveCmd.do_clean (method), 374
SCons.Script.Interactive.SConsInteractiveCmd.do_EOF (method), 374
SCons.Script.Interactive.SConsInteractiveCmd.do_exit (method), 374
SCons.Script.Interactive.SConsInteractiveCmd.do_shell (method), 374
SCons.Script.Interactive.SConsInteractiveCmd.do_version (method), 374
SCons.Subst.CmdStringHolder (class), 428–431
SCons.Subst.CmdStringHolder.escape (method), 428
SCons.Subst.CmdStringHolder.is_literal (method), 428
SCons.Subst.escape_list (function), 424
SCons.Subst.Literal.escape (method), 426
SCons.Subst.Literal.for_signature (method), 426
SCons.Subst.Literal.is_literal (method), 426
SCons.Subst.scons_subst (function), 424
SCons.Subst.scons_subst_list (function), 424
SCons.Subst.scons_subst_once (function), 425
SCons.Subst.SetAllowableExceptions (function), 424
SCons.Subst.SpecialAttrWrapper (class), 427–428
SCons.Subst.SpecialAttrWrapper.escape (method), 427
SCons.Subst.SpecialAttrWrapper.is_literal (method), 427
SCons.Subst.SpecialAttrWrapper.escape (method), 427
SCons.Subst.specialdict (function), 424
SCons.Subst.Target_or_Source (class), 434–436
SCons.Subst.Target_or_Source.__getattr__ (method), 435
SCons.Subst.Targets_or_Sources (class), 433–434
SCons.Subst.Targets_or_Sources.__getattr__ (method), 433
SCons.Subst.Targets_or_Sources.__str__ (method), 433
SCons.Util._NoError (class), 471–473
SCons.Util.AddMethod (function), 463
SCons.Util.adjustizes (function), 462
SCons.Util.AppendPath (function), 462
SCons.Util.case_sensitive_suffixes (function), 462
SCons.Warnings.DeprecatedTargetSignaturesWarning (class), 559–561  threading.Thread.setDaemon (function), 127
SCons.Warnings.DeprecatedWarning (class), 541–543  threading.Thread.setName (function), 127
SCons.Warnings.DuplicateEnvironmentWarning (class) type.__call__ (function), 139
509–511 type.__cmp__ (function), 139
SCons.Warnings.enableWarningClass (function), 498 type.__subclasses__ (function), 140
SCons.Warnings.FortranCxxMixWarning (class), 537–  type.mro (function), 140
539
SCons.Warnings.FutureDeprecatedWarning (class), 539 UserDict.UserDict...__cmp__ (function), 27, 29, 30, 33, 541
SCons.Warnings.LinkWarning (class), 513–515 UserDict.UserDict...__delitem__ (function), 33, 160
SCons.Warnings.MisleadingKeywordsWarning (class) UserDict.UserDict...__init__ (function), 160
515–517 UserDict.UserDict...__len__ (function), 27, 29, 30, 34, 60, 160, 478, 479
SCons.Warnings.NoMD5ModuleWarning (class), 519– UserDict.UserDict...__setitem__ (function), 34, 160 521
SCons.Warnings.NoObjectCountWarning (class), 523– UserDict.UserDict.fromkeys (class method), 28, 29, 525 31, 34, 60, 161, 478, 479
SCons.Warnings.NoParallelSupportWarning (class), 527 UserDict.UserDict.get (function), 28, 29, 31, 34, 60, 161, 478, 479
SCons.Warnings.process_warn_strings (function), 498 UserDict.UserDict.has_key (function), 28, 29, 31, 34, 547 60, 161, 478, 479
SCons.Warnings.PythonVersionWarning (class), 545– UserDict.UserDict.items (function), 34, 60, 161 547
SCons.Warnings.ReservedVariableWarning (class), 527 UserDict.UserDict.iteritems (function), 28, 29, 31, 34, 529 60, 161, 478, 479
SCons.Warnings.StackSizeWarning (class), 529–531 UserDict.UserDict.iterkeys (function), 28, 29, 31, 34, 532 60, 161, 478, 479
SCons.Warnings.suppressWarningClass (function), 498 UserDict.UserDict.itervalues (function), 28, 29, 31, 548 34, 60, 161, 478, 479
SCons.Warnings.TaskmasterNeedsExecuteWarning (class); 551–553 UserDict.UserDict.keys (function), 34, 60, 161
SCons.Warnings.VisualStudioMissingWarning (class), 535–537 UserDict.UserDict.popitem (function), 34, 60, 161
SCons.Warnings.warn (function), 498 UserDict/UserDict.update (function), 34, 161
SCons.Warnings.Warning (class), 499–501 UserDict/UserDict.values (function), 34, 60, 161
threading.Thread.getName (function), 127 UserList/UserList...__contains__ (function), 32, 112, 156, 370, 433, 465, 475, 482
threading.Thread.isAlive (function), 127 UserList/UserList...__delitem__ (function), 32, 112, 156, 370, 433, 465, 475, 483
threading.Thread.isDaemon (function), 127
UserString.UserString.strip (function), 431
UserString.UserString.swapcase (function), 431
UserString.UserString.title (function), 431
UserString.UserString.translate (function), 431
UserString.UserString.upper (function), 431
UserString.UserString.zfill (function), 431