

__R__ adar __D__ ata __F__ ormat
1.0

Generated by Doxygen 1.8.5

Mon Nov 25 2013 13:27:37

Contents

1 RDF Specs	1
1.1 The RDF Users Guide Version 0.0	1
1.2 Users Guide Organization	1
1.3 RDF File Format	1
1.3.1 Data File Components	1
1.3.2 Delimiters	2
1.3.3 Reserved Characters	2
1.3.4 Reserved Keywords	3
1.3.5 Prefix/Suffix	3
1.3.6 Include	3
1.3.7 UNIT	3
1.3.8 Continuation Lines	3
1.4 APPENDIX A	4
1.5 Python Implementation	4
1.5.1 Packages and Modules	4
1.5.2 How it works:	5
1.6 Flow Chart	6
2 Namespace Index	6
2.1 Packages	6
3 Hierarchical Index	7
3.1 Class Hierarchy	7
4 Class Index	11
4.1 Class List	11
5 File Index	16
5.1 File List	16
6 Namespace Documentation	17
6.1 rdf Namespace Reference	17
6.1.1 Detailed Description	18
6.1.2 Function Documentation	18
6.1.3 Variable Documentation	18
6.2 rdf.data Namespace Reference	18
6.2.1 Detailed Description	19
6.3 rdf.data.entries Namespace Reference	19
6.3.1 Detailed Description	19
6.3.2 Function Documentation	19

6.3.3	Variable Documentation	20
6.4	rdf.data.files Namespace Reference	20
6.4.1	Detailed Description	21
6.4.2	Function Documentation	21
6.4.3	Variable Documentation	21
6.5	rdf.eRDF Namespace Reference	21
6.5.1	Detailed Description	22
6.5.2	Function Documentation	22
6.6	rdf.iRDF Namespace Reference	22
6.6.1	Detailed Description	23
6.6.2	Function Documentation	23
6.6.3	Variable Documentation	23
6.7	rdf.language Namespace Reference	23
6.7.1	Detailed Description	24
6.8	rdf.language.errors Namespace Reference	24
6.8.1	Detailed Description	24
6.9	rdf.language.grammar Namespace Reference	24
6.9.1	Detailed Description	25
6.10	rdf.language.grammar.morpheme Namespace Reference	25
6.10.1	Detailed Description	25
6.11	rdf.language.grammar.punctuation Namespace Reference	25
6.11.1	Detailed Description	26
6.11.2	Function Documentation	26
6.11.3	Variable Documentation	28
6.12	rdf.language.grammar.syntax Namespace Reference	28
6.12.1	Detailed Description	29
6.12.2	Function Documentation	29
6.13	rdf.language.lexis Namespace Reference	29
6.13.1	Detailed Description	30
6.14	rdf.language.lexis.pragmatics Namespace Reference	30
6.14.1	Detailed Description	30
6.14.2	Variable Documentation	31
6.15	rdf.language.lexis.semantics Namespace Reference	31
6.15.1	Detailed Description	31
6.15.2	Variable Documentation	31
6.16	rdf.language.prosodic Namespace Reference	31
6.16.1	Detailed Description	32
6.16.2	Variable Documentation	32
6.17	rdf.parse Namespace Reference	32
6.17.1	Detailed Description	32

6.17.2 Variable Documentation	32
6.18 rdf.reserved Namespace Reference	33
6.18.1 Detailed Description	33
6.18.2 Variable Documentation	33
6.19 rdf.test Namespace Reference	34
6.19.1 Detailed Description	35
6.19.2 Function Documentation	35
6.19.3 Variable Documentation	36
6.20 rdf.units Namespace Reference	36
6.20.1 Detailed Description	37
6.20.2 Function Documentation	37
6.20.3 Variable Documentation	38
6.21 rdf.units.addendum Namespace Reference	38
6.21.1 Detailed Description	39
6.21.2 Variable Documentation	39
6.22 rdf.units.errors Namespace Reference	40
6.22.1 Detailed Description	40
6.23 rdf.units.physical_quantity Namespace Reference	40
6.23.1 Detailed Description	43
6.24 rdf.units.prefix Namespace Reference	43
6.24.1 Detailed Description	45
6.24.2 Function Documentation	45
6.24.3 Variable Documentation	46
6.25 rdf.uRDF Namespace Reference	50
6.25.1 Detailed Description	50
6.25.2 Function Documentation	50
6.26 rdf.utils Namespace Reference	52
6.26.1 Detailed Description	52
6.26.2 Function Documentation	53
7 Class Documentation	55
7.1 _Accepted Class Reference	55
7.1.1 Detailed Description	56
7.2 _AffixAdder Class Reference	56
7.2.1 Detailed Description	58
7.2.2 Member Function Documentation	58
7.2.3 Member Data Documentation	59
7.3 _Base Class Reference	59
7.3.1 Detailed Description	60
7.4 _Coherent Class Reference	60

7.4.1	Detailed Description	62
7.5	_Derived Class Reference	62
7.5.1	Detailed Description	64
7.6	_Noun Class Reference	64
7.6.1	Detailed Description	66
7.6.2	Member Function Documentation	66
7.6.3	Member Data Documentation	68
7.7	_RDFAccess Class Reference	68
7.7.1	Detailed Description	69
7.7.2	Constructor & Destructor Documentation	69
7.7.3	Member Function Documentation	70
7.7.4	Member Data Documentation	71
7.8	_RDFRecord Class Reference	71
7.8.1	Detailed Description	73
7.8.2	Constructor & Destructor Documentation	73
7.8.3	Member Data Documentation	73
7.9	_SI Class Reference	73
7.9.1	Detailed Description	75
7.10	_Special Class Reference	75
7.10.1	Detailed Description	77
7.11	_SymbolChanger Class Reference	77
7.11.1	Detailed Description	79
7.11.2	Member Function Documentation	79
7.11.3	Member Data Documentation	80
7.12	_UnAccepted Class Reference	80
7.12.1	Detailed Description	81
7.13	_Unit Class Reference	81
7.13.1	Detailed Description	83
7.13.2	Member Function Documentation	84
7.13.3	Member Data Documentation	86
7.14	_Verb Class Reference	87
7.14.1	Detailed Description	88
7.14.2	Member Function Documentation	88
7.14.3	Member Data Documentation	90
7.15	AbsorbedDose Class Reference	90
7.15.1	Detailed Description	92
7.15.2	Member Data Documentation	92
7.16	Acceleration Class Reference	93
7.16.1	Detailed Description	94
7.16.2	Member Data Documentation	94

7.17	Activity Class Reference	95
7.17.1	Detailed Description	96
7.17.2	Member Data Documentation	96
7.18	Affix Class Reference	97
7.18.1	Detailed Description	98
7.18.2	Member Function Documentation	98
7.18.3	Member Data Documentation	100
7.19	AmountOfSubstance Class Reference	101
7.19.1	Detailed Description	102
7.19.2	Member Data Documentation	102
7.20	Area Class Reference	103
7.20.1	Detailed Description	104
7.20.2	Member Data Documentation	104
7.21	BackwardBracketsError Class Reference	105
7.21.1	Detailed Description	106
7.22	BinaryPrefix Class Reference	106
7.22.1	Detailed Description	108
7.22.2	Member Function Documentation	108
7.22.3	Member Data Documentation	108
7.23	Bit Class Reference	108
7.23.1	Detailed Description	110
7.23.2	Member Data Documentation	110
7.24	BitPerSecond Class Reference	110
7.24.1	Detailed Description	111
7.24.2	Member Data Documentation	111
7.25	Brackets Class Reference	111
7.25.1	Detailed Description	113
7.25.2	Member Function Documentation	113
7.25.3	Member Data Documentation	116
7.26	Byte Class Reference	116
7.26.1	Detailed Description	118
7.26.2	Member Data Documentation	118
7.27	BytesPerSecond Class Reference	118
7.27.1	Detailed Description	119
7.27.2	Member Data Documentation	119
7.28	Capacitance Class Reference	119
7.28.1	Detailed Description	121
7.28.2	Member Data Documentation	121
7.29	CatalyticActivity Class Reference	122
7.29.1	Detailed Description	123

7.29.2 Member Data Documentation	123
7.30 CelsiusTemperattrue Class Reference	124
7.30.1 Detailed Description	125
7.30.2 Member Data Documentation	125
7.31 Charge Class Reference	126
7.31.1 Detailed Description	127
7.31.2 Member Data Documentation	127
7.32 Comment Class Reference	128
7.32.1 Detailed Description	129
7.33 Comment Class Reference	129
7.33.1 Detailed Description	131
7.33.2 Member Function Documentation	131
7.33.3 Member Data Documentation	131
7.34 Concentration Class Reference	131
7.34.1 Detailed Description	133
7.34.2 Member Data Documentation	133
7.35 CurrentDensity Class Reference	134
7.35.1 Detailed Description	135
7.35.2 Member Data Documentation	135
7.36 dBPower Class Reference	136
7.36.1 Detailed Description	137
7.36.2 Member Data Documentation	137
7.37 Density Class Reference	137
7.37.1 Detailed Description	139
7.37.2 Member Data Documentation	139
7.38 DoseEquivalent Class Reference	140
7.38.1 Detailed Description	141
7.38.2 Member Data Documentation	141
7.39 ElectricalConductance Class Reference	142
7.39.1 Detailed Description	143
7.39.2 Member Data Documentation	143
7.40 ElectricalResistance Class Reference	144
7.40.1 Detailed Description	145
7.40.2 Member Data Documentation	145
7.41 ElectricCurrent Class Reference	146
7.41.1 Detailed Description	147
7.41.2 Member Data Documentation	147
7.42 EMF Class Reference	148
7.42.1 Detailed Description	149
7.42.2 Member Data Documentation	149

7.43	Energy Class Reference	150
7.43.1	Detailed Description	151
7.43.2	Member Data Documentation	151
7.44	FatalUnitError Class Reference	152
7.44.1	Detailed Description	153
7.45	Force Class Reference	153
7.45.1	Detailed Description	154
7.45.2	Member Data Documentation	154
7.46	Frequency Class Reference	155
7.46.1	Detailed Description	156
7.46.2	Member Data Documentation	156
7.47	Glyph Class Reference	157
7.47.1	Detailed Description	158
7.47.2	Member Function Documentation	158
7.48	Grammar Class Reference	161
7.48.1	Detailed Description	163
7.48.2	Constructor & Destructor Documentation	164
7.48.3	Member Function Documentation	164
7.48.4	Member Data Documentation	170
7.49	Illuminance Class Reference	172
7.49.1	Detailed Description	174
7.49.2	Member Data Documentation	174
7.50	Include Class Reference	175
7.50.1	Detailed Description	176
7.50.2	Member Function Documentation	176
7.51	Inductance Class Reference	177
7.51.1	Detailed Description	178
7.51.2	Member Data Documentation	178
7.52	Length Class Reference	179
7.52.1	Detailed Description	180
7.52.2	Member Data Documentation	180
7.53	lexicon Class Reference	181
7.53.1	Detailed Description	181
7.53.2	Member Function Documentation	182
7.54	Luminance Class Reference	182
7.54.1	Detailed Description	184
7.54.2	Member Data Documentation	184
7.55	LuminouFlux Class Reference	185
7.55.1	Detailed Description	186
7.55.2	Member Data Documentation	186

7.56 LuminousIntensity Class Reference	187
7.56.1 Detailed Description	188
7.56.2 Member Data Documentation	188
7.57 MagneticFieldStrength Class Reference	189
7.57.1 Detailed Description	190
7.57.2 Member Data Documentation	190
7.58 MagneticFlux Class Reference	191
7.58.1 Detailed Description	192
7.58.2 Member Data Documentation	192
7.59 MagneticFluxDensity Class Reference	193
7.59.1 Detailed Description	194
7.59.2 Member Data Documentation	194
7.60 Mass Class Reference	195
7.60.1 Detailed Description	196
7.60.2 Member Data Documentation	196
7.61 MetricPrefix Class Reference	197
7.61.1 Detailed Description	198
7.61.2 Member Function Documentation	198
7.61.3 Member Data Documentation	198
7.62 MomentOfForce Class Reference	198
7.62.1 Detailed Description	200
7.62.2 Member Data Documentation	200
7.63 MorphemeExchangeError Class Reference	201
7.63.1 Detailed Description	201
7.64 NullCommandError Class Reference	202
7.64.1 Detailed Description	202
7.65 Operator Class Reference	203
7.65.1 Detailed Description	204
7.66 PlaneAngle Class Reference	204
7.66.1 Detailed Description	206
7.66.2 Member Data Documentation	206
7.67 Power Class Reference	207
7.67.1 Detailed Description	208
7.67.2 Member Data Documentation	208
7.68 Prefix Class Reference	209
7.68.1 Detailed Description	210
7.69 Prefix Class Reference	210
7.69.1 Detailed Description	211
7.69.2 Member Function Documentation	212
7.70 Prefix Class Reference	212

7.70.1	Detailed Description	213
7.70.2	Constructor & Destructor Documentation	214
7.70.3	Member Function Documentation	214
7.70.4	Member Data Documentation	216
7.71	Pressure Class Reference	216
7.71.1	Detailed Description	218
7.71.2	Member Data Documentation	218
7.72	Prosodic Class Reference	219
7.72.1	Detailed Description	220
7.72.2	Constructor & Destructor Documentation	221
7.72.3	Member Function Documentation	221
7.72.4	Member Data Documentation	229
7.73	Ratio Class Reference	229
7.73.1	Detailed Description	231
7.74	RDF Class Reference	231
7.74.1	Detailed Description	233
7.74.2	Constructor & Destructor Documentation	233
7.74.3	Member Function Documentation	233
7.74.4	Member Data Documentation	242
7.75	rdf_list Class Reference	242
7.75.1	Detailed Description	243
7.75.2	Member Function Documentation	243
7.76	RDFAccumulator Class Reference	244
7.76.1	Detailed Description	245
7.76.2	Constructor & Destructor Documentation	246
7.76.3	Member Function Documentation	247
7.76.4	Member Data Documentation	248
7.77	RDFAnalyzer Class Reference	248
7.77.1	Detailed Description	249
7.77.2	Member Function Documentation	249
7.78	RDFComment Class Reference	250
7.78.1	Detailed Description	251
7.78.2	Member Function Documentation	251
7.79	RDFError Class Reference	252
7.79.1	Detailed Description	253
7.80	RDFField Class Reference	253
7.80.1	Detailed Description	255
7.80.2	Member Function Documentation	255
7.80.3	Member Data Documentation	259
7.81	RDFPreRecord Class Reference	260

7.81.1	Detailed Description	261
7.81.2	Member Function Documentation	261
7.82	RDFRecord Class Reference	262
7.82.1	Detailed Description	263
7.82.2	Member Function Documentation	263
7.83	RDFWarning Class Reference	264
7.83.1	Detailed Description	264
7.84	RDFWrapper Class Reference	264
7.84.1	Detailed Description	266
7.84.2	Constructor & Destructor Documentation	266
7.84.3	Member Function Documentation	266
7.84.4	Member Data Documentation	267
7.85	Record Class Reference	267
7.85.1	Detailed Description	269
7.85.2	Member Function Documentation	269
7.85.3	Member Data Documentation	269
7.86	RedefiningUnitWarning Class Reference	269
7.86.1	Detailed Description	271
7.87	ReservedCharacterError Class Reference	271
7.87.1	Detailed Description	272
7.88	RunOnSentenceError Class Reference	272
7.88.1	Detailed Description	273
7.89	SolidAngle Class Reference	273
7.89.1	Detailed Description	275
7.89.2	Member Data Documentation	275
7.90	SpecificVolume Class Reference	276
7.90.1	Detailed Description	277
7.90.2	Member Data Documentation	277
7.91	Suffix Class Reference	278
7.91.1	Detailed Description	279
7.91.2	Member Function Documentation	279
7.92	Suffix Class Reference	279
7.92.1	Detailed Description	281
7.93	SymbolsMixIn Class Reference	281
7.93.1	Detailed Description	282
7.93.2	Member Data Documentation	282
7.94	Temperature Class Reference	283
7.94.1	Detailed Description	284
7.94.2	Member Data Documentation	284
7.95	Time Class Reference	285

7.95.1 Detailed Description	286
7.95.2 Member Data Documentation	286
7.96 Unit Class Reference	287
7.96.1 Detailed Description	288
7.96.2 Member Function Documentation	288
7.97 UnitsError Class Reference	288
7.97.1 Detailed Description	289
7.98 UnmatchedBracketsError Class Reference	290
7.98.1 Detailed Description	291
7.99 UnrecognizedUnitWarning Class Reference	291
7.99.1 Detailed Description	292
7.100 Velocity Class Reference	293
7.100.1 Detailed Description	294
7.100.2 Member Data Documentation	294
7.101 Volume Class Reference	295
7.101.1 Detailed Description	296
7.101.2 Member Data Documentation	296
7.102 WaveNumber Class Reference	297
7.102.1 Detailed Description	298
7.102.2 Member Data Documentation	298
7.103 Word Class Reference	299
7.103.1 Detailed Description	300
7.103.2 Member Function Documentation	300
7.103.3 Member Data Documentation	301
8 File Documentation	301
8.1 rdf/__init__.py File Reference	301
8.2 rdf/data/__init__.py File Reference	302
8.3 rdf/language/__init__.py File Reference	302
8.4 rdf/language/grammar/__init__.py File Reference	302
8.5 rdf/language/lexis/__init__.py File Reference	302
8.6 rdf/test/__init__.py File Reference	302
8.7 rdf/units/__init__.py File Reference	303
8.8 rdf/data/entries.py File Reference	303
8.9 rdf/data/files.py File Reference	304
8.10 rdf/DOCS/mainpage.txt File Reference	304
8.11 rdf/eRDF.py File Reference	304
8.12 rdf/iRDF.py File Reference	305
8.13 rdf/language/errors.py File Reference	305
8.14 rdf/units/errors.py File Reference	306

8.15 rdf/language/grammar/morpheme.py File Reference	306
8.16 rdf/language/grammar/punctuation.py File Reference	307
8.17 rdf/language/grammar/syntax.py File Reference	307
8.18 rdf/language/lexis/pragmatics.py File Reference	308
8.19 rdf/language/lexis/semantics.py File Reference	308
8.20 rdf/language/prosodic.py File Reference	309
8.21 rdf/parse.py File Reference	309
8.22 rdf/reserved.py File Reference	309
8.23 rdf/units/addendum.py File Reference	310
8.24 rdf/units/physical_quantity.py File Reference	311
8.25 rdf/units/prefix.py File Reference	313
8.26 rdf/uRDF.py File Reference	315
8.27 rdf/utils.py File Reference	315

1 RDF Specs

1.1 The RDF Users Guide Version 0.0

The Radar Definition Format (RDF) was developed as part of the Advanced Radar Technology Program (ART). It was a proposal to help simplify the exchange of radar system design information between various Section 334 design tools. Although funding for the task was canceled before completion, a fair amount of work was spent defining the file format and developing software to read and edit these files. Though the file format is neither new nor elaborate, it has a potential for simplifying data transfer between radar system elements.

1.2 Users Guide Organization

The Radar Definition Format (RDF) Users Guide is divided into three distinct chapters. The first chapter contains a description of the file format, its advantages, and limitations. The Second chapter describes the RDF Reader software available to read, write, and edit RDF files. The final chapter presents conventions, restrictions and guidelines that different projects have adopted when using the RDF files.

1.3 RDF File Format

The Radar Definition File is an ASCII file and is organized using a Key = Value method. In its simplest form, each line of the file contains a parameter name, an operator such as $\hat{O}=\hat{O}$, and the value of the parameter. Optional fields include the units of the parameter and any comments that one wishes to make. This method allows sufficient flexibility to handle complex multi-mode systems as well as single string low cost radars. Only the parameters that are necessary for a particular system must be specified, keeping the file size down and making the RDF much more readable. The ordering of parameters is not important, so they can appear in whatever sequence makes the most sense for a given system.

1.3.1 Data File Components

RDF files can contain two kinds of logical records: a data record and a comment record. A data record consists of the following components:

Keyword	(Units)	[dimensions]	{element}	Operator	Values	Comments
---------	---------	--------------	-----------	----------	--------	----------

Although they must occur in the order shown above, the fields indicated by Italics are considered optional. A comment record consists of any line which does not contain an operator field, including lines that are entirely blank.

1.3.1.1 Keyword

All data in an RDF file are identified by unique Keywords. The Keyword is composed of any arbitrary string of printable ascii characters except for the 11 reserved characters listed in section 2.3. Spaces (ascii #32) and tabs contained within a keyword are considered part of the keyword and must be present when making keyword searches. However, spaces and tabs are considered equivalent. Also, consecutive spaces and tabs within a keyword are equivalent to a single character. Spaces and tabs on the leading and trailing ends of keywords are ignored. Both upper and lower case letters are allowed, however, no distinction is made between them and they are equivalent in a keyword search.

1.3.1.2 Units

If present, the Units field must occur between the keyword and the operator. It is distinguish from the key word by open and close parenthesis (). The units are expressed as an ascii field. The RDF certified list of units can be found in appendix A. They include measures of length, velocity, mass, time, etc. If a logical record contains more than one data value, a separate unit can be specified for each value. If there are more values than units, the last unit specified applies to all the remaining values.

1.3.1.3 Operator

The operator separates the keyword from the list of values. The default value is the equals sign, =. The default operator can be change by the following:

```
OPERATOR = string
```

where string is the desired string of characters to be used as the operator. It must be present in every logical data record.

1.3.1.4 Values

The value field can consist of any printable ascii character except for the comment delimiter (see following section). The ascii representation of numeric data follows the FORTRAN conventions for integer, fix point, and floating point data formats. Floating point data uses an E to indicate the exponent value. Data values can be space or comma delimited. When character and numeric data occur within the same data field, and the character string contains spaces or commas, the string must be enclosed by double quotes, ".

1.3.1.5 Comments

Any characters following the comment delimiter are considered a comment. The default comment delimiter is an exclamation point, !. The comment delimiter can be change by specifying:

```
COMMENT = string
```

where string can be any ascii string. There is no restriction on the content of the comment following the comment delimiter.

1.3.1.6 Dimensions (Format Extension)

1.3.1.7 Elements (Unused)

1.3.2 Delimiters

When multiple values occur in the Units or Value fields, the entries can be separated from each other using: commas, tabs, or spaces.

1.3.3 Reserved Characters

The following characters are not allowed in keywords, units, dimensions, and elements: (,), [,], { , }, <, >, =, !, and ;. In addition, the semicolon, ;, is not allowed in the value field. If ascii data contains spaces, commas or tabs, it must be enclosed by double quotes. Finally, if the last character of a physical record is a backslash, \, the logical record is continued onto the following line (see section 2.5).

1.3.4 Reserved Keywords

The following keywords are reserved and cannot be used in a data file:

```
COMMENT, OPERATOR, PREFIX, SUFFIX, INCLUDE, and UNIT.
```

1.3.5 Prefix/Suffix

Often, a group of keywords will begin with similar character strings. An alternative to typing the repetitious portion of the keyword over and over for each record is to use the following Prefix command at the beginning of the group of records:

```
PREFIX = Repetitious portion at the START of a set of Keywords
```

This applies to all following keywords in the file until another prefix command is encountered. At the end of the group, the prefix can be turned off by putting the following record in the file:

```
PREFIX =
```

The PREFIX string is considered as if it were part of the key for all Keywords between the two above PREFIX entries. It is the combination of the PREFIX and explicit keyword that must satisfy the uniqueness requirement.

Similarly, if the end of a group of keywords is repetitious, the following entry can be used:

```
SUFFIX = Repetitious portion at the END of a set of Keywords
```

1.3.6 Include

The Include command provides a convenient way of combining RDF files without actually merging them. The syntax is as follows

```
INCLUDE = filename
```

The result is equivalent to inserting all the records of the included file into the original RDF file with exception of the PREFIX and SUFFIX commands. If a prefix or suffix is used in the original file, it is applied to the included file as well. If the include file also has a prefix or suffix, they do not overwrite the original suffix/prefix. Instead, the new prefix/suffix gets combined with the original for all applicable records in the included file. Should the included file also have an include statement, the process continues. The maximum depth of include files open at any one time is 1000 (the python default, which can be changed).

1.3.7 UNIT

The UNIT command is new to RDF3000. It allows a user to specify a unit that is subsequently used in the (unit) field of an RDF record. The protocol is:

```
UNIT = (*args, **kwargs)
```

with the (python) function signature:

```
(abbreviation="", multiplier=1, adder=0, si_unit=None)
```

Because it subs a built-in (str), it uses **new** instead of the more familiar **init**, which leads to some nuances in the signatures:

- (1) You cannot use the abbreviation keyword, so that the abbreviation argument must be positiona;

1.3.8 Continuation Lines

If the last non-space character in a physical record is backslash, \, the logical record is continued onto the following line. The last character before the backslash is immediately followed by the first non-space character of the next line. If the first non-space character on the following line is also a backslash, then it is skipped and the line continues with the character immediately after the backslash. This construction allows the line breaks to be made even in the middle of a string of spaces without any ambiguity. For Instance:

```
Example_Key = The rain in Spain falls mainly on the plain
               is equivalent to all of the following:
```

```
Example_Key = The rain in Spa\
               in falls mainly on the plain
```

```
Example_Key = The rain in Spain \
               falls mainly on the plain
```

```
Example_Key = The rain in Spain \
               \falls mainly on the plain
```

```
Example_Key = The rain in Spain\
               \ falls mainly on the plain
```

1.4 APPENDIX A

The original RDF unit spec, based on a FORTRAN-77 platform, was inadequate. The python implementation is described within.

1.5 Python Implementation

1.5.1 Packages and Modules

[uRDF.py](#) basic **u**-users

[iRDF.py](#) advanced **i**-teractive stuff

[eRDF.py](#) uncut **e**-xperiment stuff

[parse.py](#) A script for processing an rdf file to stdout

[read.py](#) generators that read rdf files

[utils.py](#) non-rdf specific utilites

data External object for data

[entries.py](#) Data structure for file lines

[files.py](#) Mapping object (RDF) for files

reserved Constants as primitives.

[glyphs.py](#) Constant character glyphs

[words.py](#) Command WORDS

language Grammar and Lexicon

[prosodic.py](#) The interaction between text (clitics) and the host grammar.

language/ grammar Symbols and Parsing

[morpheme.py](#) Self-aware morphemes (affixes)

[punctuation.py](#) Self-aware punctuations

[syntax.py](#) the Grammar object that knows how to read RDF

language/ lexis Words

[semantics.py](#) Words that become things (Nouns)

[pragmatics.py](#) Words that will an can change Grammar (Verbs)

units/ Units

[physical_quantity.py](#) SI Units, Prefixes and all the basics

[addendum.py](#) Non SI and user units.

1.5.2 How it works:

RDF input is ASCII strings- usually from and RDF file. They are converted to an `rdf.data.files.RDF` mapping objects as follows:

A src file name is passed to `rdf.uRDF.rdf_include` -which is the mothership. Before processing begins, an `rdf.language.grammar.syntax.Grammar` object is created- it knows the default RDF spec.

Input lines are ingested, with continued lines unwrapped, and yielded one-by-one by the likes of `rdf.read.unwrap_file`

The lines are passed to the grammar -which knows how to interpret them. There are procesed in `rdf.language.grammar.syntax.Grammar.process` which loops through the possible input types in a chain of the iterable constants:

[rdf.language.lexis.pragmatics.VERBS](#)

[rdf.language.lexis.semantics.NOUNS](#)

until the line is ID's as and instance of one of them. That instance knows what to do:

Verbs change the grammar ans possibly recursivey include files

Nouns are Records or Comments.

In either case the grammar object does not know what to do, the Verb or Noun object doesn't knnow what to do, but it does know who to ask, and it asks

[rdf.language.prosodic.Prosodic](#)

All verb except [rdf.language.lexis.pragmatics.Include](#) return and empty tuple, while the later returns a list of the conetents of the included file.

There are to Nouns:

[rdf.language.lexis.semantics.Record](#)

[rdf.language.lexis.semantics.Comment](#)

These become:

[rdf.data.entries.RDFPreRecord](#)

[rdf.data.entries.RDFComment](#)

Both of these are iterable. The former is a single object, and when iterated, yields a `rdf.data.entries.RDFRecord` and is exhausted.

The later iterates like an empty tuple: it just disappears. The point being: no matter what you get back from the `rdf_include` generator: you can iterate it and it will yield 0, 1, or many `rdf.data.entries.RDFRecord` objects (which is the fundamental data representation of an RDF Record). Note: you really can't yield 0, so what happens is it moves to the next rdf line, until a finite number of records is kicked put. Thus the generator chains them all together seemlessly, no matter the depth of recursion- and that can be unpacked into the `rdf.data.files.RDF` constructor.

Note: the `RDFRecord` is a fancy (Key, "value") pair, where the key is the rdf-dictionary key, and the value is an `rdf.data.entries.RDFField` tuple. Hence the dict (really collection.OrderedDict) constructor unpacks that pair to make the rdf mapping.

The `rdf.data.entries.RDFField` is a tuple of:

```
value (units) {dimensions} [element] !comments
```

with some extras- for instance, when it is created, the units (if present) are check against the `rdf.units.units_.UNITS`

dictionary and converted to base units. It also has an eval() method, which will do what it takes to take the value from a string into its native type.

Finally the `rdf.data.files.RDF` object— which is what users really put into their programs— has some features:

(1) failed attribute requests are passed to the underline mapping class- so you can perform OrderDict method calls on them.

(2) getitem is special: `rdf[key]` returns `eval(value)`— so the numeric (or whatever) version of value, while `rdf.get(key)` returns the whole rdf field. Note: it may be smarter to call eval on construction— that is TBD.

1.6 Flow Chart

The flow chart:

`and it's high resolution version`

2 Namespace Index

2.1 Packages

Here are the packages with brief descriptions (if available):

rdf	The rdf package	??
rdf.data	RDF The final form for lines and files	??
rdf.data.entries	Usable data objects for lines (records)	??
rdf.data.files	Usable data object for files	??
rdf.eRDF	e xperimental RDF objects	??
rdf.iRDF	i nteractive RDF tools for 'perts	??
rdf.language	The Language: grammar + lexis = language mod prosodic	??
rdf.language.errors	RDF Language Exceptions	??
rdf.language.grammar	Grammar (language - lexis)	??
rdf.language.grammar.morpheme	Key Changing Morphemes	??
rdf.language.grammar.punctuation	Language's Punctuation Marks	??
rdf.language.grammar.syntax	Syntax glues it all together	??

rdf.language.lexis	The Lexis comprises the words in the language	??
rdf.language.lexis.pragmatics	Words with (reflexive) meaning (Verb)	??
rdf.language.lexis.semantics	References to Things (Noun)	??
rdf.language.prosodic	Prosodics define a Dependency Grammar	??
rdf.parse	RDF Parsing script	??
rdf.reserved	Reserved Symbols	??
rdf.test	A brief test suite	??
rdf.units	Units according to http://physics.nist.gov/cuu/pdf/sp811.pdf	??
rdf.units.addendum	Non metric and user units	??
rdf.units.errors	RDF Unit Errors	??
rdf.units.physical_quantity	Classes for Physical Quantities	??
rdf.units.prefix	Auto Prefix Handlers	??
rdf.uRDF	u sers' inteface to language.py and data.py from rdf import read	??
rdf.utils	Non-RDF specific utilities	??

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<u>RDFField</u>		??
ABCMeta		
lexicon		??
Exception		
RDFError		??
MorphemeExchangeError		??

FatalUnitError	??
list	
rdf_list	??
Affix	??
Prefix	??
Suffix	??
object	
_RDFRecord	??
RDFPreRecord	??
RDFRecord	??
RDF	??
RDFWrapper	??
_RDFAccess	??
RDFAccumulator	??
RDFAnalyzer	??
Prosodic	??
SymbolsMixin	??
RDFField	??
Grammar	??
Prefix	??
BinaryPrefix	??
MetricPrefix	??
str	
RDFComment	??
Brackets	??
Glyph	??
Word	??
_Verb	??
_AffixAdder	??
Prefix	??
Suffix	??
_SymbolChanger	??
Comment	??

Operator	??
Include	??
Unit	??
_Noun	??
Comment	??
Record	??
_Unit	??
_Accepted	??
Bit	??
BitPerSecond	??
Byte	??
BytesPerSecond	??
dBPower	??
Ratio	??
_SI	??
_Base	??
AmountOfSubstance	??
ElectricCurrent	??
Length	??
LuminousIntensity	??
Mass	??
Temperature	??
Time	??
_Derived	??
_Coherent	??
_Special	??
AbsorbedDose	??
Activity	??
Capacitance	??
CatalyticActivity	??
CelsiusTemperattrue	??
Charge	??

DoseEquivalent	??
ElectricalConductance	??
ElectricalResistance	??
EMF	??
Energy	??
Force	??
Frequency	??
Illuminance	??
Inductance	??
LuminousFlux	??
MagneticFlux	??
MagneticFluxDensity	??
PlaneAngle	??
Power	??
Pressure	??
SolidAngle	??
Acceleration	??
Area	??
Concentration	??
CurrentDensity	??
Density	??
Luminance	??
MagneticFieldStrength	??
SpecificVolume	??
Velocity	??
Volume	??
WaveNumber	??
MomentOfForce	??
<u>_UnAccepted</u>	??
UserWarning	
RDFWarning	??
NullCommandError	??

ReservedCharacterError	??
BackwardBracketsError	??
RunOnSentenceError	??
UnmatchedBracketsError	??
UnitsError	??
RedefiningUnitWarning	??
UnrecognizedUnitWarning	??

4 Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

_Accepted	Accepted non SI units	??
_AffixAdder	ABC sends sub's class name to <code>rdf.language.prosodic.set_affix</code>	??
_Base	Base SI units	??
_Coherent	Coherent Derived SI Units	??
_Derived	Derived DI units	??
_Noun	Nouns are <code>rdf.language.lexis.Word</code> objects that "concrete" as their <code>sin_qua_non</code>	??
_RDFAccess	Base class	??
_RDFRecord	Base for iterable (key, field) pair	??
_SI	SI units	??
_Special	Special Coehert Derived SI Units	??
_SymbolChanger	ABC sends sub's class. <code>__name__.lower()</code> to <code>rdf.language.prosodic.change_symbol</code>	??
_UnAccepted	Unaccepted units (If you must)	??
_Unit	Memoizes its instances	??

<u>Verb</u>	Verbs are <code>rdf.language.lexis.Word</code> objects that "act" as their <code>sin_qua_non</code>	??
<u>AbsorbedDose</u>	Louis Harold Gray (10 November 1905 – 9 July 1965)	??
<u>Acceleration</u>	Meters per seconds squared	??
<u>Activity</u>	Antoine Henri Becquerel (15 December 1852 – 25 August 1908)	??
<u>Affix</u>	Abstract Base Class for Pre/Suf behavior	??
<u>AmountOfSubstance</u>	Lorenzo Romano Amedeo Carlo Avogadro di Quaregna e di Cerreto, Count of Quaregna and Cerreto (9 August 1776, Turin, Piedmont – 9 July 1856)	??
<u>Area</u>	Meters squared	??
<u>BackwardBracketsError</u>	Unmatched or un parsable pairs	??
<u>BinaryPrefix</u>	Binary Prefix Note: limits/differences of/between JEDEC and IEC	??
<u>Bit</u>	Data <code>Volume</code> conversion to bits	??
<u>BitPerSecond</u>	Data rate conversion to bps	??
<u>Brackets</u>	Brackets that know thyselfes	??
<u>Byte</u>	Data <code>Volume</code> conversion to bits	??
<u>BytesPerSecond</u>	Data rate conversion to bytes per second	??
<u>Capacitance</u>	Michael Faraday, FRS (22 September 1791 – 25 August 1867)	??
<u>CatalyticActivity</u>	Katal	??
<u>CelsiusTemperatue</u>	Anders Celsius (27 November 1701 – 25 April 1744)	??
<u>Charge</u>	Charles-Augustin de Coulomb (14 June 1736 – 23 August 1806)	??
<u>Comment</u>	Change <code>rdf.language.grammar.syntax.Grammar.comment</code> via <code>_SymbolChangrr.act</code>	??
<u>Comment</u>	The <code>Comment</code> Noun remembers passive comment lines	??

Concentration		??
Concentration		
CurrentDensity		??
Amps per square meter		
dBPower		??
Decibel Power -is not power- it's just a number: (Alexander Graham Bell (March 3, 1847 – August 2, 1922))		
Density		??
ρ		
DoseEquivalent		??
Professor Rolf Maximilian Sievert (6 May 1896 – 3 October 1966)		
ElectricalConductance		??
Ernst Werner Siemens, von Siemens since 1888, (13 December 1816 – 6 December 1892)		
ElectricalResistance		??
Georg Simon Ohm (16 March 1789 – 6 July 1854)		
ElectricCurrent		??
Andre-Marie Ampère (22 January 1775 – 10 June 1836)		
EMF		??
Alessandro Giuseppe Antonio Anastasio Volta (18 February 1745 – 5 March 1827)		
Energy		??
James Prescott Joule FRS (24 December 1818 – 11 October 1889)		
FatalUnitError		??
Fatal error for unknown units (not really an acceptable idea)		
Force		??
Sir Isaac Newton PRS MP (25 December 1642 – 20 March 1727)		
Frequency		??
Heinrich Rudolf Hertz (22 February 1857 – 1 January 1894)		
Glyph		??
A character that knows how to find itself in strings		
Grammar		??
Grammar tracks the state of grammar, and uses it to dispatch work when it is called		
Illuminance		??
Lux		
Include		??
Open an Include File		
Inductance		??
Joseph Henry (December 17, 1797 – May 13, 1878)		
Length		??
Length conversion to meters		

lexicon	Metaclass for Grammar gets defines the pragmatics and semantics at load-time" pragmatics.- Verb instances are assigned according to the <code>rdf.reserved.words.KEYWORDS</code> , and the <code>symantics.Noun</code> instances are created- these are needed by <code>Grammar.process</code>	??
Luminance	<code>Cd/m**2</code>	??
LuminouFlux	<code>Lumen</code>	??
LuminousIntensity	<code>Brightness</code>	??
MagneticFieldStrength	<code>A/m</code>	??
MagneticFlux	<code>Wilhelm Eduard Weber (24 October 1804 – 23 June 1891)</code>	??
MagneticFluxDensity	<code>Nikola Tesla (10 July 1856 – 7 January 1943)</code>	??
Mass	<code>Conversion to kilograms</code>	??
MetricPrefix	<code>Metric Prefix</code>	??
MomentOfForce	<code>Torque</code>	??
MorphemeExchangeError	<code>Morphe Exchange Currents?</code>	??
NullCommandError	<code>Should be thrown in constructor?</code>	??
Operator	<code>Change <code>rdf.language.grammar.syntax.Grammar.operator</code> via <code>_SymbolChangrr.act</code></code>	??
PlaneAngle	<code>Angle conversion to degrees</code>	??
Power	<code>James Watts, FRS, FRSE (19 January 1736 – 25 August 1819)</code>	??
Prefix	<code>Add to <code>rdf.language.grammar.syntax.Grammar.prefix</code> via <code>_AffixAdder.act</code></code>	??
Prefix	<code>Appears Before the stem:</code>	??
Prefix	<code>Abstract Base class for class decorator factory that makes an instance of the decorated <code>_Unit</code> subclass according to the prefix and the power</code>	??
Pressure	<code>Blaise Pascal (19 June 1623 – 19 August 1662)</code>	??

Prosodic

In Dependency Grammar, the clitic (an RDF line) depends on its host (the current grammar)—together they are a **Prosodic**—this is important because and RDF line on its own has no definite meaning— it must be interpreted in terms of its host Grammar, this tuple exists because the pair is passed all over the place

??

Ratio

TBD

??

RDFAn **RDF** Mothership: A fully interpreted **RDF** File

??

rdf_list

A list of rdf records that can filter itself and make an RDF

??

RDFAccumulator

A TBD rdf accumulator - probably slower than **RDFAnalyzer** as it rebuilds dictionary all the time—see **RDF.__iadd__**

??

RDFAnalyzer

RDFAnalyzer is created with an `rdf.language.syntax.Grammar` object and then emulates a function that converts single line inputs into a pre-RDF output -note: it is overly complicated, and its sole purpose is to provide an interactive RDF reader

??

RDFComment

The RDF Comment is a comment string, endowed with False RDF-ness and null response to iteration

??

RDFError

Fatal attempt to CODE badly

??

RDFFieldAdd methods and constants to `_RDFField` so that it lives up to its name

??

RDFPreRecord

The pre Record is built from data and is a len=1 iterator: iterating builds the final product: **RDFRecord**— thus line reads or include file reads yield the same (polymorphic) result: iterators that yield Records

??

RDFRecordThis is a fully parsed RDF record, and is an `_RDFRecord` with a formattable string

??

RDFWarning

RDF Warning of INPUT problems

??

RDFWrapper

A generic base class for RDF wrapped data structures -clients should use this when they have an object with RDF dependency injection and then further behavior as defined by the subclasses methods

??

RecordThe **Record** Noun processes the basic input: An RDF line

??

RedefiningUnitWarning

Warning when you overwrite a unit

??

ReservedCharacterError

Error for using a character in RESERVED

??

RunOnSentenceError

Unmatched or un parsable pairs

??

SolidAngle		
Steradians		??
SpecificVolume		
ρ^{-1}		??
Suffix		
Appears After the stem		??
Suffix		
Add to rdf.language.grammar.syntax.Grammar.suffix via _AffixAdder.act		??
SymbolsMixin		
For classes that need to know about OPERATOR and COMMENT (statically)		??
Temperature		
William Thomson, 1st Baron Kelvin OM, GCVO, PC, PRS, PRSE, (26 June 1824 – 17 December 1907)		??
Time		
Time conversion to seconds		??
Unit		
Add to an rdf.units.physical_quantity.Unit to the rdf.units.GLOSSARY via rdf.prosodic.-Prosodic.set_unit		??
UnitsError		
RDF Error for a unit problem (not sure what kind of error this is)		??
UnmatchedBracketsError		
Unmatched or un parsable pairs		??
UnrecognizedUnitWarning		
Error for input w/ unknown units		??
Velocity		
Meters per second		??
Volume		
Meters cubed		??
WaveNumber		
Inverse meter		??
Word		
The Pragamtic's are RDF lines meaning		??

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

rdf/__init__.py		??
rdf/eRDF.py		??
rdf/iRDF.py		??

rdf/ parse.py	??
rdf/ reserved.py	??
rdf/ uRDF.py	??
rdf/ utils.py	??
rdf/data/ __init__.py	??
rdf/data/ entries.py	??
rdf/data/ files.py	??
rdf/language/ __init__.py	??
rdf/language/ errors.py	??
rdf/language/ prosodic.py	??
rdf/language/grammar/ __init__.py	??
rdf/language/grammar/ morpheme.py	??
rdf/language/grammar/ punctuation.py	??
rdf/language/grammar/ syntax.py	??
rdf/language/lexis/ __init__.py	??
rdf/language/lexis/ pragmatics.py	??
rdf/language/lexis/ semantics.py	??
rdf/test/ __init__.py	??
rdf/units/ __init__.py	??
rdf/units/ addendum.py	??
rdf/units/ errors.py	??
rdf/units/ physical_quantity.py	??
rdf/units/ prefix.py	??

6 Namespace Documentation

6.1 rdf Namespace Reference

The rdf package.

Namespaces

- [data](#)
RDF The final form for lines and files.
- [eRDF](#)
e xperimental RDF objects
- [iRDF](#)

- *i*nteractive RDF tools for 'parts.
 - **language**
The Language: grammar + lexis = language mod prosodic.
 - **parse**
RDF Parsing script.
 - **reserved**
Reserved Symbols.
 - **test**
A brief test suite.
 - **units**
Units according to <http://physics.nist.gov/cuu/pdf/sp811.pdf>.
 - **uRDF**
users' interface to language.py and data.py from rdf import read
 - **utils**
Non-RDF specific utilities.

Functions

- def **test**

Variables

- **rdfparse** = **rdf_reader**
Backwards compatible rdf readers.
- **parse** = **rdf_reader**
less redundant parser

6.1.1 Detailed Description

The rdf package.

6.1.2 Function Documentation

6.1.2.1 def rdf.test()

test() function – run from rdf/test

Definition at line 55 of file __init__.py.

References rdfparse.

```
55
56 def test():
57     """test() function - run from rdf/test"""
58     import os
59     rdf_ = rdfparse('rdf.txt')
60     with open('new.rdf', 'w') as fdst:
61         fdst.write(str(rdf_))
62     if os.system("xdiff old.rdf new.rdf"):
63         os.system("diff old.rdf new.rdf")
```

6.1.3 Variable Documentation

6.1.3.1 parse = rdf_reader

less redundant parser

Definition at line 52 of file __init__.py.

6.1.3.2 rdfparse = rdf_reader

Backwards compatible rdf readers.

Definition at line 50 of file `__init__.py`.

Referenced by `RDF.fromfile()`, and `test()`.

6.2 rdf.data Namespace Reference

RDF The final form for lines and files.

Namespaces

- [entries](#)

Usable data objects for lines (records).

- [files](#)

Usable data object for files.

6.2.1 Detailed Description

RDF The final form for lines and files.

Structures to hold information in fields and files

6.3 rdf.data.entries Namespace Reference

Usable data objects for lines (records).

Classes

- class [RDFField](#)

Add methods and constants to `_RDFField` so that it lives up to its name.

- class [_RDFRecord](#)

Base for iterable (key, field) pair.

- class [RDFPreRecord](#)

The pre Record is built from data and is a len=1 iterator: iterating builds the final product: [RDFRecord](#)— thus line reads or include file reads yield the same (polymorphic) result: iterators that yield Records.

- class [RDFRecord](#)

This is a fully parsed RDF record, and is an [_RDFRecord](#) with a formatable string.

- class [RDFComment](#)

The RDF Comment is a comment string, endowed with False RDF-ness and null response to iteration.

Functions

- def [_cast](#)

Decorator to cast values.

Variables

- string `SPACE = " "`
- tuple `_RDFField`
Base RDF Field named tuple -it's all in here -note, it's assigned (public) name differs from its variable (private) name, so that users never need to know about this private assignemnt.
- tuple `_RDFRecord = collections.namedtuple("RDFRecord", "key field")`
This assignment is a bit deeper: Just a key and a field.

6.3.1 Detailed Description

Usable data objects for lines (records).

Define the RDF Entries as:
`RDFRecord = (key, RDFField)`

6.3.2 Function Documentation

6.3.2.1 `def rdf.data.entries._cast(magicmethodbinding) [private]`

Decorator to cast values.

Parameters

<code>magicmethod-binding</code>	A function that binds to a magic method and casts instances (for example: float)
----------------------------------	--

Return values

<code>cast_method</code>	An instance method that cast ala magicmethodbinding decorator for magic method/function casting
--------------------------	--

Definition at line 19 of file entries.py.

```

19
20 def _cast(magicmethodbinding):
21     """decorator for magic method/function casting"""
22     def cast_method(self):
23         """__int__ --> int(self.value) --for example"""
24         return magicmethodbinding(self.value)
25     return cast_method

```

6.3.3 Variable Documentation

6.3.3.1 `tuple _RDFField`

Initial value:

```

1 = collections.namedtuple('RDFField',
                           'value units dimensions element comments')

```

Base RDF Field named tuple -it's all in here -note, it's assigned (public) name differs from its variable (private) name, so that users never need to know about this private assignemnt.

Definition at line 29 of file entries.py.

6.3.3.2 `tuple _RDFRecord = collections.namedtuple("RDFRecord", "key field")`

This assignment is a bit deeper: Just a key and a field.

Definition at line 151 of file entries.py.

6.3.3.3 string SPACE = " "

Definition at line 12 of file entries.py.

6.4 rdf.data.files Namespace Reference

Usable data object for files.

Classes

- class [RDF](#)

An RDF Mothership: A fully interpresed RDF File.

Functions

- def [stripper](#)

Decorator to cast and strip arguments.

Variables

- [DICT](#) = collections.OrderedDict
- [WARN](#) = False

6.4.1 Detailed Description

Usable data object for files.

data-->RDF is THE RDF OBJECT

6.4.2 Function Documentation

6.4.2.1 def rdf.data.files.stripper (*method*)

Decorator to cast and strip arguments.

Definition at line 19 of file files.py.

```
19
20 def stripper(method):
21     @functools.wraps(method)
22     def stripped_method(self, key, *args):
23         return method(self, str(key).strip(), *args)
24     return stripped_method
25
```

6.4.3 Variable Documentation

6.4.3.1 DICT = collections.OrderedDict

Definition at line 9 of file files.py.

6.4.3.2 WARN = False

Definition at line 15 of file files.py.

6.5 rdf.eRDF Namespace Reference

e xperimental RDF objects

Classes

- class [RDFWrapper](#)

A generic base class for RDF wrapped data structures -clients should use this when they have an object with RDF dependency injection and then further behavior as defined by the sub-classes methods.

Functions

- def [debug](#)

Use to find problem lines when developing.
- def [factor](#)

Experimental function to factor keys and rdf in least form.

6.5.1 Detailed Description

e xperimental RDF objects

eRdf Experimental RDF stuff- no warranty

6.5.2 Function Documentation

6.5.2.1 def rdf.eRDF.debug (*src* = 'rdf.txt', *sleep* = 0.1)

Use to find problem lines when developing.

Definition at line 43 of file eRDF.py.

```
43
44 def debug(src='rdf.txt', sleep=0.1):
45     import time
46     from rdf import utils
47     from rdf import iRDF
48     lines = utils.unwrap_file(src)
49
50     A = iRDF.RDFAnalyzer()
51     B = iRDF.RDFAccumulator()
52
53     for line in lines:
54         print "INPUT :", line
55         print "OUTPUT:", A(line)
56         print B(line)
57         time.sleep(sleep)
58     return B
59
```

6.5.2.2 def rdf.eRDF.factor (*rdf_*)

Experimental function to factor keys and rdf in least form.

Definition at line 61 of file eRDF.py.

```
61
62 def factor(rdf_):
63     import numpy as np
64     _k = rdf_.keys()
65     _k.sort()
66     k = _k[:]
67     longest = max(map(len, k))
68     m = np.zeros((len(k), 27), dtype=int)
```

```

69     for jdx, key in enumerate(k):
70         for idx, cc in enumerate(key):
71             m[jdx, idx] = ord(cc)
72     base = [2**__ for __ in map(long, range(len(m[0])))]
73     return NotImplemented

```

6.6 rdf.iRDF Namespace Reference

Interactive RDF tools for 'parts'.

Classes

- class [rdf_list](#)

A list of rdf records that can filter itself and make an RDF.

- class [_RDFAccess](#)

Base class.

- class [RDFAnalyzer](#)

RDFAnalyzer is created with an rdf.language.syntax.Grammar object and then emulates a function that converts single line inputs into a pre-RDF output -note: it is overly complicated, and its sole purpose is to provide an interactive RDF reader.

- class [RDFAccumulator](#)

A TBD rdf accumulator - prolly slower than RDFAnalyzer as it rebuild dictionary all the time- see RDF._iadd_.

Functions

- def [test](#)

Variables

- tuple [__all__](#)

The RDF Toybox.

6.6.1 Detailed Description

Interactive RDF tools for 'parts'.

iRdf are expert-usage interactive tools:

Analyzer
Accumulator

6.6.2 Function Documentation

6.6.2.1 def rdf.iRDF.test()

Definition at line 140 of file iRDF.py.

```

140
141 def test():
142     accum = RDFAccumulator()
143     accum("INCLUDE = rdf.txt")
144     return accum

```

6.6.3 Variable Documentation

6.6.3.1 tuple __all__

Initial value:

```
1 = ('Grammar', 'RDF', 'RDFField', 'RDFRecord', 'RDFAccumulator',
2      'RDFAnalyzer', 'SI', 'rdf_list')
```

The RDF Toybox.

Definition at line 16 of file iRDF.py.

6.7 rdf.language Namespace Reference

The Language: grammar + lexis = language mod prosodic.

Namespaces

- [errors](#)
RDF Language Exceptions.
- [grammar](#)
Grammar (language - lexis)
- [lexis](#)
The Lexis comprises the words in the language.
- [prosodic](#)
Prosidics define a Dependency Grammar.

6.7.1 Detailed Description

The Language: grammar + lexis = language mod prosodic.

Language sub-package

6.8 rdf.language.errors Namespace Reference

RDF Language Exceptions.

Classes

- class [RDFError](#)
Fatal attempt to CODE badly.
- class [RDFWarning](#)
RDF Warning of INPUT problems.
- class [MorphemeExchangeError](#)
Morphere Exchange Currents?
- class [ReservedCharacterError](#)
Error for using a character in RESERVED.
- class [UnmatchedBracketsError](#)
Unmatched or un parsable pairs.
- class [RunOnSentenceError](#)
Unmatched or un parsable pairs.

- class [BackwardBracketsError](#)
Unmatched or un parsable pairs.
- class [NullCommandError](#)
Should be thrown in constructor?

6.8.1 Detailed Description

RDF Language Exceptions.

RDF Language Exceptions

6.9 rdf.language.grammar Namespace Reference

Grammar (language - lexis)

Namespaces

- [morpheme](#)
Key Changing Morphemes.
- [punctuation](#)
Language's Punctuation Marks.
- [syntax](#)
Syntax glues it all together.

6.9.1 Detailed Description

Grammar (language - lexis)

Why stuff in a `__init__`?

6.10 rdf.language.grammar.morpheme Namespace Reference

Key Changing Morphemes.

Classes

- class [Affix](#)
Abstract Base Class for Pre/Suf behavior.
- class [Prefix](#)
Appears Before the stem:
- class [Suffix](#)
Appears After the stem.

6.10.1 Detailed Description

Key Changing Morphemes.

Supported morphemes are affixes. The `Affix` ABC (and subclass of the list built-in) has two concrete subs:

`Prefix`
`Suffix`

They know how to apply themselves, and they know what to do to Grammar as it traverses the IFT.

6.11 rdf.language.grammar.punctuation Namespace Reference

Language's Punctuation Marks.

Classes

- class **Glyph**
A character that knows how to find itself in strings.
- class **Brackets**
Brackets that know thyelves.

Functions

- def **parse_left**
and RDF line (that is, left of OPERATOR)
- def **read_rdfkey**
Remove brackets from left line.
- def **read_brackets**
Read list of brackets from left line.
- def **write_brackets**
Pack bracket values into a string.

Variables

- tuple **UNITS** = **Brackets**(reserved.UNITS)
(units) Brackets
- tuple **DIMENSIONS** = **Brackets**(reserved.DIMENSIONS)
{dim} Brackets
- tuple **ELEMENT** = **Brackets**(reserved.ELEMENT)
[elements] Brackets

6.11.1 Detailed Description

Language's Punctuation Marks.

Brackets: This is where glyphs take on meaning-- the Bracket class defines them, while the punctuation module serves as a static-class like object with the 3 RDF brackets defined:

```
(UNITS)
{DIMENSIONS}
[ELEMENT]
```

Along with reading and writing functions:

<code>parse_left</code>	Parse left side of an rdf line key, units, ..., element
<code>read_rdfkey</code>	Parse the key the left side of an rdf line
<code>read_brackets</code>	GET the 3 bracketed values
<code>write_brackets</code>	Write a nice formatted string with units, ..., element

Note: Function accept left side of an rdf line to preclude awkward parsing of potentially silly comment fields.

6.11.2 Function Documentation

6.11.2.1 def rdf.language.grammar.punctuation.parse_left(*leftline*)

and RDF line (that is, left of OPERATOR)

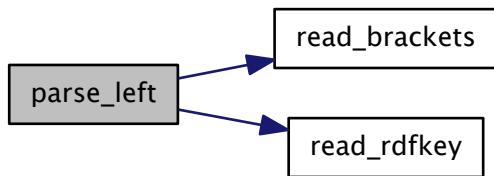
```
'key (unit) {dim} [el]' --> ['key', 'unit', 'dim', 'el']
```

Definition at line 207 of file punctuation.py.

References `rdf.language.grammar.punctuation.read_brackets()`, and `rdf.language.grammar.punctuation.read_rdfkey()`.

```
207
208 def parse_left(leftline):
209     """'key (unit) {dim} [el]' --> ['key', 'unit', 'dim', 'el']"""
210     return [read_rdfkey(leftline)] + read_brackets(leftline)
211
```

Here is the call graph for this function:



6.11.2.2 def rdf.language.grammar.punctuation.read_brackets(*leftline*)

Read list of brackets from left line.

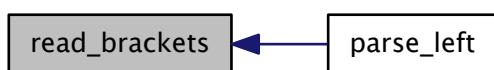
```
'key (unit) {dim} [el]' --> ['unit', 'dim', 'el']
```

Definition at line 222 of file punctuation.py.

Referenced by `rdf.language.grammar.punctuation.parse_left()`.

```
222
223 def read_brackets(leftline):
224     """'key (unit) {dim} [el]' --> ['unit', 'dim', 'el']"""
225     return [leftline << item for item in (UNITS, DIMENSIONS, ELEMENT)]
226
```

Here is the caller graph for this function:



6.11.2.3 def rdf.language.grammar.punctuation.read_rdfkey (*leftline*)

Remove brackets from left line.

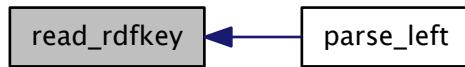
```
'key (unit) {dim} [el]' --> 'key'
```

Definition at line 213 of file punctuation.py.

Referenced by rdf.language.grammar.punctuation.parse_left().

```
213
214 def read_rdfkey(leftline):
215     """key (unit) {dim} [el]' --> 'key'"""
216     return (leftline -
217             UNITS -
218             DIMENSIONS -
219             ELEMENT).strip()
220
```

Here is the caller graph for this function:



6.11.2.4 def rdf.language.grammar.punctuation.write_brackets (*units*, *dimensions*, *element*)

Pack bracket values into a string.

```
'unit', 'dim', 'el' --> '(unit) {dim} [el]'
```

Definition at line 228 of file punctuation.py.

```
228
229 def write_brackets(units, dimensions, element):
230     """unit', 'dim', 'el' --> '(unit) {dim} [el]'"""
231     return ((units >> UNITS) +
232             (dimensions >> DIMENSIONS) +
233             (element >> ELEMENT))
```

6.11.3 Variable Documentation

6.11.3.1 tuple DIMENSIONS = Brackets(reserved.DIMENSIONS)

{dim} Brackets

Definition at line 199 of file punctuation.py.

6.11.3.2 tuple ELEMENT = Brackets(reserved.ELEMENT)

[elements] Brackets

Definition at line 203 of file punctuation.py.

6.11.3.3 tuple UNITS = Brackets(reserved.UNITS)

(units) [Brackets](#)

Definition at line 195 of file punctuation.py.

6.12 rdf.language.grammar.syntax Namespace Reference

Syntax glues it all together.

Classes

- class [lexicon](#)

Metaclass for Grammar gets defines the pragamatics and semantics at load-time" pragmatics. Verb instances are assigned according to the rdf.reserved.words.KEYWORDS, and the symantics.Noun instances are created- these are needed by Grammar.process.

- class [Grammar](#)

Grammar tracks the state of grammar, and uses it to dispatch work when it is called.

Functions

- def [null_command_watch](#)
decorate setters to prevent setting required commands to NULL
- def [unit_change](#)
decorate incremental ops to only allow +1 change

6.12.1 Detailed Description

Syntax glues it all together.

syntax handles syntax via the Grammar class. It handles syntax but farms out some work to cooperative classes

6.12.2 Function Documentation

6.12.2.1 def rdf.language.grammar.syntax.null_command_watch (*method*)

decorate setters to prevent setting required commands to NULL

Definition at line 14 of file syntax.py.

```
14
15 def null_command_watch(method):
16     def setter(self, value):
17         """if bool(value): setter, else raise NullCommandError"""
18         if not value: # guard
19             raise errors.NullCommandError
20         return method(self, value)
21     return setter
22
```

6.12.2.2 def rdf.language.grammar.syntax.unit_change (*method*)

decorate incremental ops to only allow +1 change

Definition at line 24 of file syntax.py.

```

24
25 def unit_change(method):
26     def setter(self, value):
27         """if value != 1: raise error"""
28         if value != 1: # guard
29             raise ValueError("can only in/decrement by +1")
30         return method(self, value)
31     return setter
32

```

6.13 rdf.language.lexis Namespace Reference

The Lexis comprises the words in the language.

Namespaces

- [pragmatics](#)
Words with (reflexive) meaning (Verb)
- [semantics](#)
References to Things (Noun)

Classes

- class [Word](#)
The Pragmatic's are RDF lines meaning.

6.13.1 Detailed Description

The Lexis comprises the words in the language.

6.14 rdf.language.lexis.pragmatics Namespace Reference

Words with (reflexive) meaning (Verb)

Classes

- class [_Verb](#)
Verbs are [rdf.language.lexis.Word](#) objects that "act" as their sin_qua_non.
- class [Include](#)
Open an [Include File](#).
- class [_SymbolChanger](#)
ABC sends sub's class.__name__.lower() to [rdf.language.prosodic.change_symbol](#).
- class [Operator](#)
Change [rdf.language.grammar.syntax.Grammar.operator](#) via [_SymbolChangrr.act](#).
- class [Comment](#)
Change [rdf.language.grammar.syntax.Grammar.comment](#) via [_SymbolChangrr.act](#).
- class [_AffixAdder](#)
ABC sends sub's class name to [rdf.language.prosodic.set_affix](#).
- class [Prefix](#)
Add to [rdf.language.grammar.syntax.Grammar.prefix](#) via [_AffixAdder.act](#).
- class [Suffix](#)
Add to [rdf.language.grammar.syntax.Grammar.suffix](#) via [_AffixAdder.act](#).
- class [Unit](#)
Add to an [rdf.units.physical_quantity.Unit](#) to the [rdf.units.GLOSSARY](#) via [rdf.prosodic.Prosodic.set_unit](#).

Variables

- tuple **VERBS** = (**Include**, **Operator**, **Comment**, **Prefix**, **Suffix**, **Unit**)

Verb classes are auto instantiated by `rdf.language.grammar.syntax.lexicon` metaclass.

6.14.1 Detailed Description

Words with (reflexive) meaning (Verb)

Verbs may appear to be an antipattern: the methods go mutate another objects attributes (Grammar). But that is how RDF works: metawords change the grammar.

6.14.2 Variable Documentation

6.14.2.1 tuple VERBS = (**Include**, **Operator**, **Comment**, **Prefix**, **Suffix**, **Unit**)

Verb classes are auto instantiated by `rdf.language.grammar.syntax.lexicon` metaclass.

Definition at line 118 of file pragmatics.py.

6.15 rdf.language.lexis.semantics Namespace Reference

References to Things (Noun)

Classes

- class **_Noun**
Nouns are `rdf.language.lexis.Word` objects that "concrete" as their `sin_qua_non`.
- class **Record**
The `Record` Noun processes the basic input: An RDF line.
- class **Comment**
The `Comment` Noun remembers passive comment lines.

Variables

- tuple **NOUNS** = (**Record**, **Comment**)
Nouns.

6.15.1 Detailed Description

References to Things (Noun)

The nouns.NOUNS classes process Record or Comment lines. The class structure may seem odd, and on its own it is. Its structure is polymorphic to the more complex Verb.VERB classes, which do much more.

6.15.2 Variable Documentation

6.15.2.1 tuple NOUNS = (**Record**, **Comment**)

Nouns.

Definition at line 74 of file semantics.py.

6.16 rdf.language.prosodic Namespace Reference

Prosidics define a [Dependency Grammar](#).

Classes

- class [Prosodic](#)

In Dependency Grammar, the clitic (an RDF line) depends on its host (the current grammar)– together they are a Prosodic– this is important because and RDF line on its own has no definite meaning- it must be interpreted in terms of its host Grammar, this tuple exists because the pair is passed all over the place.

Variables

- [_27 = False](#)

6.16.1 Detailed Description

Prosidics define a [Dependency Grammar](#).

Grammar/Language interactions live here in the form of the Prosodic.

The Prosodic comprises a clitic (a sentence or line) that is interpreted in the context of host (grammar).

Their coupling occurs in the Prosodic class.

6.16.2 Variable Documentation

6.16.2.1 [_27 = False](#)

Definition at line 16 of file prosodic.py.

6.17 rdf.parse Namespace Reference

RDF Parsing script.

Variables

- [pipe = sys.stderr](#)
- tuple [message = getattr\(sys.modules\['__name__'\], '__doc__'\)](#)
- [EXIT = errno.EINVAL](#)
- list [argv = sys.argv\[1:\]](#)
- list [src = argv\[-1\]](#)

6.17.1 Detailed Description

RDF Parsing script.

6.17.2 Variable Documentation

6.17.2.1 [list argv = sys.argv\[1:\]](#)

Definition at line 25 of file parse.py.

6.17.2.2 int EXIT = errno.EINVAL

Definition at line 20 of file parse.py.

6.17.2.3 tuple message = getattr(sys.modules['__name__'], '__doc__')

Definition at line 19 of file parse.py.

6.17.2.4 pipe = sys.stderr

Definition at line 18 of file parse.py.

6.17.2.5 list src = argv[-1]

Definition at line 28 of file parse.py.

6.18 rdf.reserved Namespace Reference

Reserved Symbols.

Classes

- class **SymbolsMixin**

For classes that need to know about OPERATOR and COMMENT (statically).

Variables

- string **OPERATOR** = "="
Meta word defining operator symbol.
- string **COMMENT** = "#"
Meta word defining comment symbol (depracted !)
- string **SEPARATOR** = ","
SEPARATOR Symbol.
- string **WRAP** = '/'
Line Wrap Symbol.
- string **CR** = '\n'
Carriage Return.
- string **UNITS** = '()'
Unit Delimiter ordered glyphs.
- string **DIMENSIONS** = '{}'
Dimensions wrapper ordered glyphs.
- string **ELEMENT** = '[]'
Element wrapper ordered glyphs.
- **SINGULAR** = **OPERATOR**+**COMMENT**
Singular characters, who's repetition constitutes an [rdf.language.errors.ReservedCharacterError](#).

6.18.1 Detailed Description

Reserved Symbols.

Reserved Symbols

6.18.2 Variable Documentation

6.18.2.1 string COMMENT = "#"

Meta word defining comment symbol (depracted !)

Definition at line 7 of file reserved.py.

6.18.2.2 string CR = '\n'

Carriage Return.

Definition at line 14 of file reserved.py.

6.18.2.3 string DIMENSIONS = '{}'

Dimensions wrapper ordered glyphs.

Definition at line 19 of file reserved.py.

6.18.2.4 string ELEMENT = '[]'

Element wrapper ordered glyphs.

Definition at line 21 of file reserved.py.

6.18.2.5 string OPERATOR = "="

Meta word defining operator symbol.

Definition at line 5 of file reserved.py.

6.18.2.6 string SEPARATOR = ","

SEPARATOR Symbol.

Definition at line 10 of file reserved.py.

6.18.2.7 SINGULAR = OPERATOR+COMMENT

Singular characters, who's repetition constitutes an [rdf.language.errors.ReservedCharacterError](#).

Definition at line 25 of file reserved.py.

6.18.2.8 string UNITS = '()'

Unit Delimiter ordered glyphs.

Definition at line 17 of file reserved.py.

6.18.2.9 string WRAP = '/'

Line Wrap Symbol.

Definition at line 12 of file reserved.py.

6.19 rdf.test Namespace Reference

A brief test suite.

Functions

- def **diff**

Run diff on tested files.

- def **main**
 $\text{rdf.parse}(\text{SRC}) >> \text{DST}$

Variables

- string **SRC** = "rdf.txt"
Top of the nested desting chain.
- string **DST** = "new.rdf"
File made by script.
- string **STANDARD** = "old.rdf"
Result of a sucessfile run.

6.19.1 Detailed Description

A brief test suite.

6.19.2 Function Documentation

6.19.2.1 def rdf.test.diff()

Run diff on tested files.

Definition at line 25 of file `__init__.py`.

Referenced by `rdf.test.main()`.

```

25
26 def diff():
27     import subprocess
28     try:
29         result = subprocess.check_call(["diff", DST, STANDARD])
30     except subprocess.CalledProcessError:
31         import errno
32         result = errno.EPERM
33     return result
34

```

Here is the caller graph for this function:



6.19.2.2 def rdf.test.main()

$\text{rdf.parse}(\text{SRC}) >> \text{DST}$

RDF... (SRC)>>DST

Definition at line 36 of file `__init__.py`.

References `rdf.test.diff()`.

```

36 def main():
37     """RDF... (SRC)>>DST"""
38     data = rdf.parse(SRC)
39     dst = data >> DST
40
41     return diff()

```

Here is the call graph for this function:



6.19.3 Variable Documentation

6.19.3.1 string DST = "new.rdf"

File made by script.

Definition at line 19 of file `__init__.py`.

6.19.3.2 string SRC = "rdf.txt"

Top of the nested desting chain.

Definition at line 17 of file `__init__.py`.

6.19.3.3 string STANDARD = "old.rdf"

Result of a sucessfile run.

Definition at line 21 of file `__init__.py`.

6.20 rdf.units Namespace Reference

Units according to <http://physics.nist.gov/cuu/pdf/sp811.pdf>.

Namespaces

- [addendum](#)
Non metric and user units.
- [errors](#)
RDF Unit Errors.
- [physical_quantity](#)
Classes for Physical Quantities.
- [prefix](#)
Auto Prefix Handlers.

Functions

- [def SI](#)

Convert (value, units) to SI pair - this is the interface to RDField Search various places for units...(TBD).

Variables

- **GLOSSARY** = `_Unit.Glossary`

The global unit glossary dictionary:[symbol]->converter function.

6.20.1 Detailed Description

Units according to <http://physics.nist.gov/cuu/pdf/sp811.pdf>.

The unit module.

The `rdf.data.entries.RDFField.__new__` only needs access to the SI function-- which identifies units and converts them to nominal inputs.

See `SI.__doc__` on how Units are used.

6.20.2 Function Documentation

6.20.2.1 def rdf.units_SI(value, units)

Convert (value, units) to SI pair - this is the interface to RDField Search various places for units...(TBD).

Parameters

<code>value</code>	A float in units
<code>units</code>	a string describing the units

Return values

<code>(converter(value), converter.-symbol)</code>	<p>The new value in the right units</p> <p>Using Units: Unit instance are instance of <str>-- hence you can compare them or use them as keys in a dictionary. Hence:</p> <pre>>>>km = physical_quantity.Length(1000, 'km')</pre> <p>is a string == 'km', and it is a function that multiplies by 1000.</p> <p>Thus: SI just looks in a dictionary of UNITS, c.f:</p> <pre>{km : km}['km']</pre> <p>which returns km, such that:</p> <pre>>>>print km(1) 1000.</pre> <p>Sweet.</p> <p>See <code>physical_quanity</code> on how to make your own units and how to put them in the GLOASSRY.</p>
--	---

Definition at line 27 of file `__init__.py`.

Referenced by `RDFField._handle_units()`.

```

27
28 def SI(value, units):
29     """
30     Using Units:
31     Unit instance are instance of <str>-- hence you can compare them or
32     use them as keys in a dictionary. Hence:
33
34     >>>km = physical_quantity.Length(1000, 'km')
35

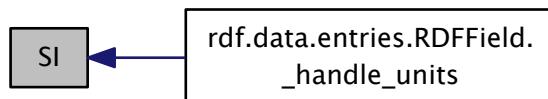
```

```

36     is a string == 'km', and it is a function that multiplies by 1000.
37
38     Thus: SI just looks in a dictionary of UNITS, c.f:
39
40     {km : km}['km']
41
42     which returns km, such that:
43
44     >>>print km(1)
45     1000.
46
47     Sweet.
48
49     See physical_quantity on how to make your own units and how to put
50     them in the GLOSSARY.
51     """
52     try:
53         converter = GLOSSARY[units]
54     except KeyError:
55         raise errors.UnrecognizedUnitWarning
56     return converter(value), converter.symbol

```

Here is the caller graph for this function:



6.20.3 Variable Documentation

6.20.3.1 GLOSSARY = _Unit.Glossary

The global unit glossary dictionary:[symbol]->converter function.

Definition at line 18 of file `__init__.py`.

6.21 rdf.units.addendum Namespace Reference

Non metric and user units.

Variables

- tuple `LENGTHS`
Supported_Length conversions.
- tuple `MASSES` = (`Mass('g', 0.001)`,)
- tuple `AREAS`
Supported_Area conversions.
- tuple `TIMES`
Supported_Time conversions.
- tuple `VELOCITES`
Supported_Velocity conversions.
- tuple `POWERS` = ()
- tuple `DBPOWERS` = (`dBPower('dBm', adder=-30)`,)
Suppoerted dB Power.

- tuple **ENERGIES** = ([Energy](#)('BTU', 1055.056),)
- tuple **FREQUENCIES** = ()

Supported Frequency conversions.

6.21.1 Detailed Description

Non metric and user units.

6.21.2 Variable Documentation

6.21.2.1 tuple AREAS

Initial value:

```
1 = (Area('ha', 10000),
2      Area('in**2', 6.4516e-4),
3      Area('ft**2', 9.290304e-2),
4      Area('mi**2', 2.58995511e6)
5      )
```

Supported _Area conversions.

Definition at line 23 of file addendum.py.

6.21.2.2 tuple DBPOWERS = (dBPower('dBm', adder=-30),)

Suppoerted dB Power.

Definition at line 48 of file addendum.py.

6.21.2.3 tuple ENERGIES = (Energy('BTU', 1055.056),)

Definition at line 50 of file addendum.py.

6.21.2.4 tuple FREQUENCIES = ()

Supported Frequency conversions.

Definition at line 57 of file addendum.py.

6.21.2.5 tuple LENGTHS

Initial value:

```
1 = (Length('in', 0.0254),
2      Length('ft', 0.3048),
3      Length('mi', 1.609344e3))
```

Supported _Length conversions.

Definition at line 15 of file addendum.py.

6.21.2.6 tuple MASSES = (Mass('g', 0.001),)

Definition at line 19 of file addendum.py.

6.21.2.7 tuple POWERS = ()

Definition at line 45 of file addendum.py.

6.21.2.8 tuple TIMES

Initial value:

```

1 = (Time('min', 60),
2      Time('hour', 3600),
3      Time('day', 86400)
4      )

```

Supported _Time conversions.

Definition at line 30 of file addendum.py.

6.21.2.9 tuple VELOCITES

Initial value:

```

1 = (Velocity('km/hr', operator.truediv(5, 18)),
2      Velocity('ft/s', 0.3048),
3      Velocity('mi/h', 0.44704),
4      Velocity('kn', operator.truediv(1852, 3600)),
5      Velocity('c', 299792458)
6      )

```

Supported _Velocity conversions.

Definition at line 37 of file addendum.py.

6.22 rdf.units.errors Namespace Reference

RDF Unit Errors.

Classes

- class [FatalUnitError](#)
Fatal error for unknown units (not really an acceptable idea)
- class [UnitsError](#)
RDF Error for a unit problem (not sure what kind of error this is)
- class [UnrecognizedUnitWarning](#)
Error for input w/ unknown units.
- class [RedefiningUnitWarning](#)
Warning when you overwrite a unit.

6.22.1 Detailed Description

RDF Unit Errors.

RDF Unit Exceptions

6.23 rdf.units.physical_quantity Namespace Reference

Classes for Physical Quantities.

Classes

- class [_Unit](#)
The [_Unit](#) class memoizes its instances.
- class [_SI](#)
SI units.
- class [_Accepted](#)
Accepted non SI units.
- class [_UnAccepted](#)
Unaccepted units (If you must)
- class [_Base](#)
Base SI units.
- class [_Derived](#)
Derived DI units.
- class [_Coherent](#)
Coherent Derived SI Units.
- class [_Special](#)
Special Coehert Derived SI Units.
- class [Length](#)
Length conversion to meters.
- class [Mass](#)
Conversion to kilograms.
- class [Time](#)
Time conversion to seconds.
- class [ElectricCurrent](#)
Andre-Marie Ampère (22 January 1775 – 10 June 1836)
- class [Temperature](#)
William Thomson, 1st Baron Kelvin OM, GCVO, PC, PRS, PRSE, (26 June 1824 – 17 December 1907)
- class [AmountOfSubstance](#)
Lorenzo Romano Amedeo Carlo Avogadro di Quaregna e di Cerreto, Count of Quaregna and Cerreto (9 August 1776, Turin, Piedmont – 9 July 1856)
- class [LuminousIntensity](#)
Brightness.
- class [PlaneAngle](#)
Angle conversion to degrees.
- class [SolidAngle](#)
Steradians.
- class [Pressure](#)
Blaise Pascal (19 June 1623 – 19 August 1662)
- class [Frequency](#)
Heinrich Rudolf Hertz (22 February 1857 – 1 January 1894)
- class [Force](#)
Sir Isaac Newton PRS MP (25 December 1642 – 20 March 1727)
- class [Energy](#)
James Prescott Joule FRS (24 December 1818 – 11 October 1889)
- class [Power](#)
James Watts, FRS, FRSE (19 January 1736 – 25 August 1819)
- class [Charge](#)
Charles-Augustin de Coulomb (14 June 1736 – 23 August 1806)
- class [EMF](#)
Alessandro Giuseppe Antonio Anastasio Volta (18 February 1745 – 5 March 1827)

- class [Capacitance](#)
Michael Faraday, FRS (22 September 1791 – 25 August 1867)
- class [ElectricalResistance](#)
Georg Simon Ohm (16 March 1789 – 6 July 1854)
- class [ElectricalConductance](#)
Ernst Werner Siemens, von Siemens since 1888, (13 December 1816 – 6 December 1892)
- class [MagneticFlux](#)
Wilhelm Eduard Weber (24 October 1804 – 23 June 1891)
- class [MagneticFluxDensity](#)
Nikola Tesla (10 July 1856 – 7 January 1943)
- class [Inductance](#)
Joseph Henry (December 17, 1797 – May 13, 1878)
- class [CelsiusTemperattrue](#)
Anders Celsius (27 November 1701 – 25 April 1744)
- class [LuminouFlux](#)
lumen
- class [Illuminance](#)
lux
- class [Activity](#)
Antoine Henri Becquerel (15 December 1852 – 25 August 1908)
- class [AbsorbedDose](#)
Louis Harold Gray (10 November 1905 – 9 July 1965)
- class [DoseEquivalent](#)
Professor Rolf Maximilian Sievert (6 May 1896 – 3 October 1966)
- class [CatalyticActivity](#)
katal
- class [Area](#)
meters squared
- class [Volume](#)
meters cubed
- class [Velocity](#)
meters per second
- class [MomentOfForce](#)
Torque.
- class [dBPower](#)
decibel Power -is not power- it's just a number: (Alexander Graham Bell (March 3, 1847 – August 2, 1922))
- class [WaveNumber](#)
Inverse meter.
- class [Acceleration](#)
meters per seconds squared
- class [Density](#)
 ρ
- class [SpecificVolume](#)
 ρ^{-1}
- class [CurrentDensity](#)
Amps per square meter.
- class [MagneticFieldStrength](#)
A/m.
- class [Luminance](#)
*cd/m**2*
- class [Concentration](#)

- class **Bit**
Data Volume conversion to bits.
- class **BitPerSecond**
Data rate conversion to bps.
- class **Byte**
Data Volume conversion to bits.
- class **BytesPerSecond**
Data rate conversion to bytes per second.
- class **Ratio**
TBD.

6.23.1 Detailed Description

Classes for Physical Quantities.

Physical Quantities are defined here. They are all sub classes of `_Unit` and that's what counts. Other private classes are only for organization: they impart no functionality.

So, to make you own unit that is called the "mine" and measure the amount of stuff:

```
@base
class Stuff(_Unit):
    _symbol = "Mine"
```

is all it takes. The symbol is defined by the string, and the `@base` decorator puts it in the GLOSSARY. To define your tithing, you could do:

```
@base
@deci
class Stuff(_Unit):
    _symbol = "Mine"
```

which also puts the "dMine" in the dictionary, such that 1 dMine is 10% of your stuff. Should you need a not standard version, say your 34% tax margin, then you go to the addendum module and add:

```
taxes = Stuff('tax', 0.34)
```

Then:

```
GLOSSARY['tax'] = .34
GLOSSARY['tax'].symbol = Mine
```

converts the value (100% of your stuff) to (34% of a Mine). (the taxes variable is irrelevant- it's just there to remind you what it was).

6.24 rdf.units.prefix Namespace Reference

Auto **Prefix** Handlers.

Classes

- class **Prefix**
Abstract Base class for class decorator factory that makes an instance of the decorated `_Unit` subclass according to the prefix and the power.
- class **MetricPrefix**

- Metric Prefix.
 - class **BinaryPrefix**
- Binary Prefix Note:** limits/differences of/between JEDEC and IEC

Functions

- def **metric**
Decorate All Metric Prefixes.
- def **jedec**
Decorate JEDEC prefixes.
- def **iec**
Decorate IEC prefixes.

Variables

- tuple **yotta** = MetricPrefix('Y', 24)
 10^{24}
- tuple **zetta** = MetricPrefix('Z', 21)
 10^{21}
- tuple **exa** = MetricPrefix('E', 18)
 10^{18}
- tuple **peta** = MetricPrefix('P', 15)
 10^{15}
- tuple **tera** = MetricPrefix('T', 12)
 10^{12}
- tuple **giga** = MetricPrefix('G', 9)
 10^9
- tuple **mega** = MetricPrefix('M', 6)
 10^6
- tuple **kilo** = MetricPrefix('k', 3)
 10^3
- tuple **hecto** = MetricPrefix('h', 2)
 10^2
- tuple **deca** = MetricPrefix('da', 1)
 10^1
- tuple **base** = MetricPrefix("", 0)
Trival (but it does create an instance and put it in [Unit.Glossary](#))
- tuple **deci** = MetricPrefix('d', -1)
 10^{-1}
- tuple **centi** = MetricPrefix('c', -2)
 10^{-2}
- tuple **milli** = MetricPrefix('m', -3)
 10^{-3}
- tuple **micro** = MetricPrefix('u', -6)
 10^{-6}
(NB: "u" is used instead of "μ" for typographical reasons)
- tuple **nano** = MetricPrefix('n', -9)
 10^{-9}
- tuple **pico** = MetricPrefix('p', -12)
 10^{-12}
- tuple **femto** = MetricPrefix('f', -15)

- tuple `atto = MetricPrefix('a', -18)`
 10^{-18}
- tuple `zepto = MetricPrefix('z', -21)`
 10^{-21}
- tuple `yocto = MetricPrefix('y', -24)`
 10^{-24}
- tuple `base2 = BinaryPrefix("", 0)`
Trival (integer measurement)
- tuple `kilo2 = BinaryPrefix('k', 1)`
 2^{10} , JEDEC
- tuple `mega2 = BinaryPrefix('M', 2)`
 $(2^{10})^2$, JEDEC
- tuple `giga2 = BinaryPrefix('G', 3)`
 $(2^{10})^3$, JEDEC
- tuple `kibi = BinaryPrefix('Ki', 1)`
 2^{10} , IEC
- tuple `mebi = BinaryPrefix('Mi', 2)`
 $(2^{10})^2$, IEC
- tuple `gibi = BinaryPrefix('Gi', 3)`
 $(2^{10})^3$, IEC
- tuple `tebi = BinaryPrefix('Ti', 4)`
 $(2^{10})^4$, IEC
- tuple `pebi = BinaryPrefix('Pi', 5)`
 $(2^{10})^5$, IEC
- tuple `exbi = BinaryPrefix('Ei', 6)`
 $(2^{10})^6$, IEC
- tuple `zebi = BinaryPrefix('Zi', 7)`
 $(2^{10})^7$, IEC
- tuple `yebi = BinaryPrefix('Yi', 8)`
 $(2^{10})^8$, IEC
- `yobi = yebi`

TBD: pick one.

6.24.1 Detailed Description

Auto Prefix Handlers.

Unit prefixes

6.24.2 Function Documentation

6.24.2.1 def rdf.units.prefix.iec (*cls*)

Decorate IEC prefixes.

Definition at line 213 of file prefix.py.

References `rdf.units.prefix.exbi`, `rdf.units.prefix.gibi`, `rdf.units.prefix.kibi`, `rdf.units.prefix.mebi`, `rdf.units.prefix.pebi`, `rdf.units.prefix.tebi`, `rdf.units.prefix.yobi`, and `rdf.units.prefix.zebi`.

```
213
214 def iec(cls):
215     return kibi() (mebi() (gibi() (tebi() (pebi() (exbi()
216         zebi() (yobi() (cls))))))))
```

6.24.2.2 def rdf.units.prefix.jedec (*cls*)

Decorate JEDEC prefixes.

Definition at line 208 of file prefix.py.

References rdf.units.prefix.giga2, rdf.units.prefix.kilo2, and rdf.units.prefix.mega2.

```
208
209 def jedec(cls):
210     return kilo2() (mega2() (giga2() (cls)))
211
```

6.24.2.3 def rdf.units.prefix.metric (*n* = 1)

Decorate All Metric Prefixes.

Definition at line 185 of file prefix.py.

References rdf.units.prefix.atto, rdf.units.prefix.base, rdf.units.prefix.centi, rdf.units.prefix.exa, rdf.units.prefix.femto, rdf.units.prefix.giga, rdf.units.prefix.kilo, rdf.units.prefix.mega, rdf.units.prefix.micro, rdf.units.prefix.milli, rdf.units.-prefix.nano, rdf.units.prefix.peta, rdf.units.prefix.pico, and rdf.units.prefix.tera.

```
185
186 def metric(n=1):
187     def pmetric(cls):
188         @exa(n)
189         @peta(n)
190         @tera(n)
191         @giga(n)
192         @mega(n)
193         @kilo(n)
194         @base(n)
195         @centi(n)
196         @milli(n)
197         @micro(n)
198         @nano(n)
199         @pico(n)
200         @femto(n)
201         @atto(n)
202         class _Dummy(cls):
203             pass
204         return cls
205     return pmetric
206
```

6.24.3 Variable Documentation

6.24.3.1 tuple atto = MetricPrefix('a', -18)

10^{-18}

Definition at line 148 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.2 tuple base = MetricPrefix("", 0)

Trival (but it does create an instance and put it in [_Unit.Glossary](#))

Definition at line 131 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.3 tuple base2 = BinaryPrefix("", 0)

Trival (integer measurement)

Definition at line 156 of file prefix.py.

6.24.3.4 tuple centi = MetricPrefix('c', -2) 10^{-2}

Definition at line 135 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.5 tuple deca = MetricPrefix('da', 1) 10^1

Definition at line 129 of file prefix.py.

6.24.3.6 tuple deci = MetricPrefix('d', -1) 10^{-1}

Definition at line 133 of file prefix.py.

6.24.3.7 tuple exa = MetricPrefix('E', 18) 10^{18}

Definition at line 115 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.8 tuple exbi = BinaryPrefix('Ei', 6) $(2^{10})^6$, IEC

Definition at line 175 of file prefix.py.

Referenced by rdf.units.prefix.iec().

6.24.3.9 tuple femto = MetricPrefix('f', -15) 10^{-15}

Definition at line 146 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.10 tuple gibi = BinaryPrefix('Gi', 3) $(2^{10})^3$, IEC

Definition at line 169 of file prefix.py.

Referenced by rdf.units.prefix.iec().

6.24.3.11 tuple giga = MetricPrefix('G', 9) 10^9

Definition at line 121 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.12 tuple giga2 = BinaryPrefix('G', 3) $(2^{10})^3$, JEDEC

Definition at line 162 of file prefix.py.

Referenced by rdf.units.prefix.jedec().

6.24.3.13 tuple **hecto** = **MetricPrefix**('h', 2)

10^2

Definition at line 127 of file prefix.py.

6.24.3.14 tuple **kibi** = **BinaryPrefix**('Ki', 1)

2^{10} , IEC

Definition at line 165 of file prefix.py.

Referenced by `rdf.units.prefix.iec()`.

6.24.3.15 tuple **kilo** = **MetricPrefix**('k', 3)

10^3

Definition at line 125 of file prefix.py.

Referenced by `rdf.units.prefix.metric()`.

6.24.3.16 tuple **kilo2** = **BinaryPrefix**('k', 1)

2^{10} , JEDEC

Definition at line 158 of file prefix.py.

Referenced by `rdf.units.prefix.jedec()`.

6.24.3.17 tuple **mebi** = **BinaryPrefix**('Mi', 2)

$(2^{10})^2$, IEC

Definition at line 167 of file prefix.py.

Referenced by `rdf.units.prefix.iec()`.

6.24.3.18 tuple **mega** = **MetricPrefix**('M', 6)

10^6

Definition at line 123 of file prefix.py.

Referenced by `rdf.units.prefix.metric()`.

6.24.3.19 tuple **mega2** = **BinaryPrefix**('M', 2)

$(2^{10})^2$, JEDEC

Definition at line 160 of file prefix.py.

Referenced by `rdf.units.prefix.jedec()`.

6.24.3.20 tuple **micro** = **MetricPrefix**('u', -6)

10^{-6}

(NB: "u" is used instead of "\u03bc" for typographical reasons)

Definition at line 140 of file prefix.py.

Referenced by `rdf.units.prefix.metric()`.

6.24.3.21 tuple **milli** = **MetricPrefix**('m', -3)

10^{-3}

Definition at line 137 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.22 tuple nano = MetricPrefix('n', -9)

10^{-9}

Definition at line 142 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.23 tuple pebi = BinaryPrefix('Pi', 5)

$(2^{10})^5$, IEC

Definition at line 173 of file prefix.py.

Referenced by rdf.units.prefix.iec().

6.24.3.24 tuple peta = MetricPrefix('P', 15)

10^{15}

Definition at line 117 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.25 tuple pico = MetricPrefix('p', -12)

10^{-12}

Definition at line 144 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.26 tuple tebi = BinaryPrefix('Ti', 4)

$(2^{10})^4$, IEC

Definition at line 171 of file prefix.py.

Referenced by rdf.units.prefix.iec().

6.24.3.27 tuple tera = MetricPrefix('T', 12)

10^{12}

Definition at line 119 of file prefix.py.

Referenced by rdf.units.prefix.metric().

6.24.3.28 tuple yebi = BinaryPrefix('Yi', 8)

$(2^{10})^8$, IEC

Definition at line 179 of file prefix.py.

6.24.3.29 yobi = yebi

TBD: pick one.

Definition at line 181 of file prefix.py.

Referenced by rdf.units.prefix.iec().

6.24.3.30 tuple yocto = MetricPrefix('y', -24)

10^{-24}

Definition at line 152 of file prefix.py.

6.24.3.31 tuple yotta = MetricPrefix('Y', 24)

10^{24}

Definition at line 111 of file prefix.py.

6.24.3.32 tuple zebi = BinaryPrefix('Zi', 7)

$(2^{10})^7$, IEC

Definition at line 177 of file prefix.py.

Referenced by rdf.units.prefix.iec().

6.24.3.33 tuple zepto = MetricPrefix('z', -21)

10^{-21}

Definition at line 150 of file prefix.py.

6.24.3.34 tuple zetta = MetricPrefix('Z', 21)

10^{21}

Definition at line 113 of file prefix.py.

6.25 rdf.uRDF Namespace Reference

u sers' inteface to language.py and data.py from rdf import read

Functions

- def `rdf_include`

The rdf_include function takes a src and rdf.language.syntax.Grammar object to go process the entirety of src- it is the sole controller of Grammar.depth, unpacking of _RDFRecord and lists of them- it deals with the recursion, etc.

- def `rdf_reader`

For src it's that simple.

6.25.1 Detailed Description

u sers' inteface to language.py and data.py from rdf import read

uRDF is the user's interface to rdf.

`rdf_include` is the key function- it reads rdf files recursivly.

`rdf_reader` unpacks the result into the RDF constructor.

6.25.2 Function Documentation

6.25.2.1 def rdf.uRDF.rdf_include(src, _kwargs)

The `rdf_include` function takes a src and `rdf.language.syntax.Grammar` object to go process the entirety of src- it is the sole controller of `Grammar.depth`, unpacking of `_RDFRecord` and lists of them- it deals with the recursion, etc.

Parameters

<code>src</code>	Is the source file name
------------------	-------------------------

Side Effect: None to external users (Grammar evolution internally)

Return values

<i>generator</i>	<i>Generator</i> that generates <code>rdf.data.entries.RDFRecord</code>
	<pre> rdf_include(src): src is an rdf file name. A generator is returned, and it yields RDFRecord objects one at time, in the order they come up.</pre>

Definition at line 23 of file uRDF.py.

References `rdf.utils.unwrap_file()`.

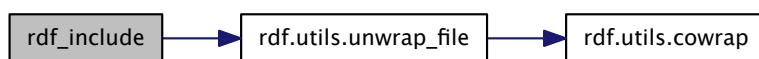
Referenced by `Prosodic.include_file()`, and `rdf.uRDF.rdf_reader()`.

```

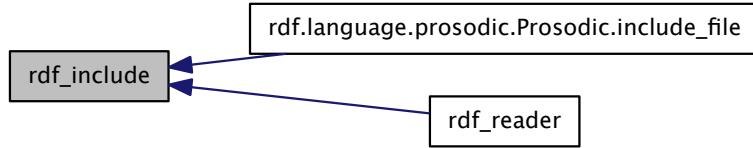
23
24 def rdf_include(src, **_kwargs):
25     """rdf_include(src):
26
27     src is an rdf file name. A generator is returned, and it yields
28     RDFRecord objects one at time, in the order they come up.
29     """
30     # There is one keyword allowed, and it is secret
31     # Get grammar passed in, or make a new one.
32     _grammar = _kwargs.get('_grammar') or syntax.Grammar()
33
34     # prepare grammar depth, or add on a recursive call
35     _grammar += 1
36     # read (full) line from src
37
38     for line in unwrap_file(src, wrap=_grammar.wrap):
39         # get the result as _grammar processes it.
40         result = _grammar(line)
41         # Polymorphic unpack:
42         # RdfPreRecord -> RDFRecord
43         # RDFComment --> [] --> break inner loop
44         # () from commands --> ditto
45         # INCLUDE --> a bunch of records
46         for item in result:
47             yield item
48     # to get here, you hit EOF, so you're moving up a level, or out for ever.
49     _grammar -= 1
50

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.25.2.2 def rdf.uRDF.rdf_reader(src=None)

For src it's that simple.

Parameters

<code>src</code>	Is the source file name
------------------	-------------------------

Return values

<code>rdf.data.files.RDF</code>	The RDF mapping object rdf = rdf_reader(src) src rdf filename rdf The RDF mapping object
---------------------------------	---

Definition at line 54 of file uRDF.py.

References `rdf.uRDF.rdf_include()`.

```

54
55 def rdf_reader(src=None):
56     """rdf = rdf_reader(src)
57
58     src      rdf filename
59     rdf      The RDF mapping object"""
60     from rdf import utils
61     return RDF(*rdf_include(src or utils.dialog_pickfile()))

```

Here is the call graph for this function:



6.26 rdf.utils Namespace Reference

Non-RDF specific utilities.

Functions

- def `coroutine`

- Coroutine Deocrator, returns it kick started.*
- def [cowrap](#)
Coroutine yields None, util a full line is ingested, then it's yielded.
 - def [unwrap_file](#)
Return unwrapped lines from a file.
 - def [parse_tuple_input](#)
A basic parsing of standard python 2x args and kwargs.
 - def [dialog_pickfile](#)
open a file picking GUI

6.26.1 Detailed Description

Non-RDF specific utilities.

Non RDF specific python helpers

6.26.2 Function Documentation

6.26.2.1 def rdf.utils.coroutine (generator)

Coroutine Deocrator, returns it kick started.

Definition at line 8 of file utils.py.

```

8
9 def coroutine(generator):
10     @functools.wraps(generator)
11     def starter(*args, **kwargs):
12         co_routine = generator(*args, **kwargs)
13         co_routine.send(None)
14         return co_routine
15     return starter
16

```

6.26.2.2 def rdf.utils.cowrap (wrap=reserved.WRAP)

Coroutine yields None, util a full line is ingested, then it's yielded.

Yield None, or a whole, unwrapped, line

Definition at line 19 of file utils.py.

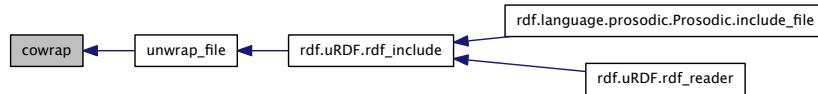
Referenced by [rdf.utils.unwrap_file\(\)](#).

```

19
20 def cowrap(wrap=reserved.WRAP):
21     """Yield None, or a whole, unwrapped, line"""
22     line = None
23     while True:
24         # take in a line while yielding the last one
25         line = yield line
26         # True is it's continued
27         while line.endswith(wrap):
28             # get continued while yielding None
29             dline = (yield None)
30             # concatenate w/o the wrap character
31             line = line.rstrip(wrap) + dline
32

```

Here is the caller graph for this function:



6.26.2.3 def rdf.utils.dialog_pickfile(args, kwargs)

open a file picking GUI

Use tkinter to make a dialog_pickfile GUI

Definition at line 84 of file utils.py.

```

84
85 def dialog_pickfile(*args, **kwargs):
86     """Use tkinter to make a dialog_pickfile GUI """
87 #     import tkMessageBox
88 #     from tkColorChooser import askcolor
89 #     from tkFileDialog import askopenfilename
90     return askopenfilename(*args, **kwargs) or raw_input("Last Chance:")
  
```

6.26.2.4 def rdf.utils.parse_tuple_input(raw_line)

A basic parsing of standard python 2x args and kwargs.

```

parse_tuple_input("1, 2, ..., n, a=1, b=2, ..., n=n")

returns [1,2,...,n], {'a':1, 'b':2, ..., 'n':n}
that is: args      ,   kwargs
  
```

Definition at line 45 of file utils.py.

```

45
46 def parse_tuple_input(raw_line):
47     """parse_tuple_input("1, 2, ..., n, a=1, b=2, ..., n=n")
48
49     returns [1,2,...,n], {'a':1, 'b':2, ..., 'n':n}
50     that is: args      ,   kwargs
51     """
52     # try to make a float (since this is for units)
53     def cast(x):
54         try:
55             z = float(x)
56         except ValueError:
57             z = str(x)
58         return z
59
60     # return [], kwarg
61     def get_kwarg(item):
62         key, value = map(str.strip, item.split("="))
63         return [], {key: cast(value)}
64
65     # return [arg], {}
66     def get_arg(item):
67         return [cast(item.strip())], {}
68
69     # get arg or kwarg based on inp.__contains__("=")
70     LOOK_UP_PARSER = {False: get_arg, True: get_kwarg}
71
72     # split signature up
73     inputs = raw_line.split(",")
74     args = []
75     kwargs = {}
76     for inp in inputs:
77         arg, kwarg = LOOK_UP_PARSER["=" in inp](inp)
  
```

```

78         args.extend(arg)
79         kwargs.update(kwarg)
80
81     return args, kwargs
82

```

6.26.2.5 def rdf.utils.unwrap_file(src, wrap = " / ")

Return unwrapped lines from a file.

```
Return lines unwraped realtive to wrap='/'
```

Definition at line 34 of file utils.py.

References rdf.utils.cowrap().

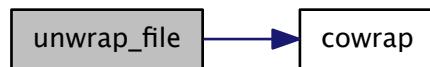
Referenced by rdf.uRDF.rdf_include().

```

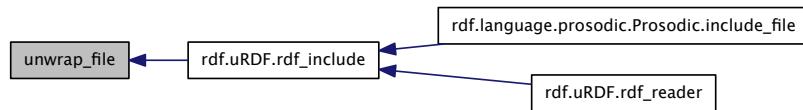
34
35 def unwrap_file(src, wrap="/"):
36     """Return lines unwraped realtive to wrap='/'"""
37     with open(src, 'r') as fsrc:
38         # strip the file's lines, unwrapped, chuck the Nones and blanks
39         return filter(bool,
40                       map(cowrap(wrap=wrap).send,
41                            map(str.strip,
42                                fsrc.readlines())))
43

```

Here is the call graph for this function:



Here is the caller graph for this function:

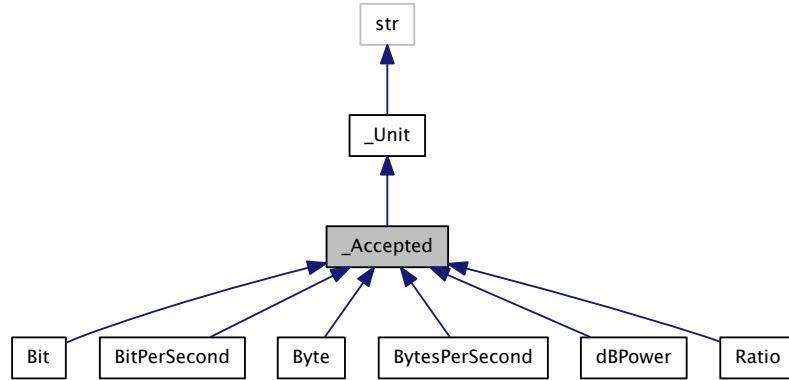


7 Class Documentation

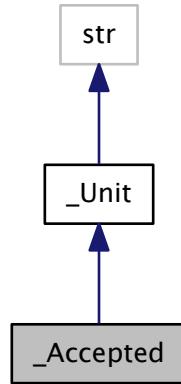
7.1 _Accepted Class Reference

Accepted non SI units.

Inheritance diagram for `_Accepted`:



Collaboration diagram for `_Accepted`:



Additional Inherited Members

7.1.1 Detailed Description

Accepted non SI units.

Definition at line 152 of file `physical_quantity.py`.

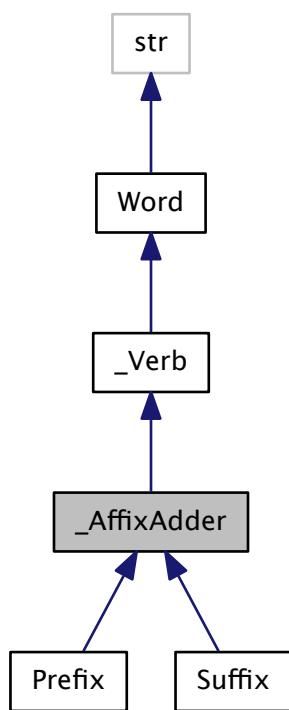
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

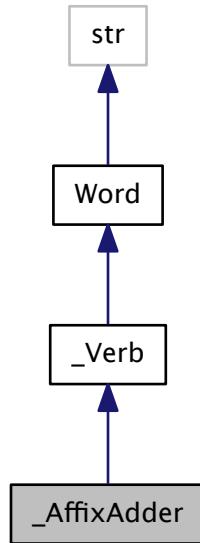
7.2 _AffixAdder Class Reference

ABC sends sub's class name to rdf.language.prosodic.set_affix.

Inheritance diagram for _AffixAdder:



Collaboration diagram for `_AffixAdder`:



Public Member Functions

- def `act`
*Act means set the affix, according to the sub's **name**.*

Static Private Attributes

- `__metaclass__` = `abc.ABCMeta`

7.2.1 Detailed Description

ABC sends sub's class name to `rdf.language.prosodic.set_affix`.

`_AffixAdder` is an ABC

Definition at line 81 of file `pragmatics.py`.

7.2.2 Member Function Documentation

7.2.2.1 def `act (self, prosodic)`

`Act` means set the affix, according to the sub's **name**.

add the affix

Definition at line 87 of file `pragmatics.py`.

```
87
88     def act(self, prosodic):
89         """add the affix"""
90         prosodic.set_affix(type(self).__name__.lower())
91
```

7.2.3 Member Data Documentation

7.2.3.1 __metaclass__ = abc.ABCMeta [static], [private]

Definition at line 84 of file pragmatics.py.

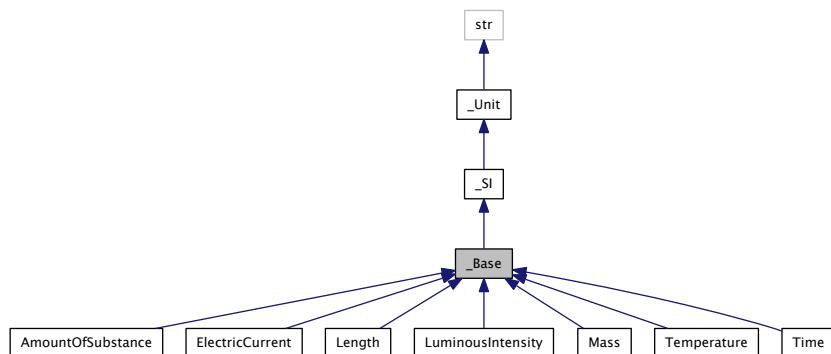
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

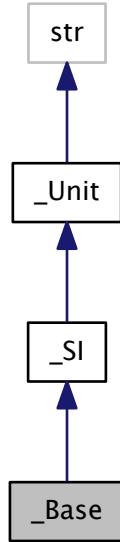
7.3 _Base Class Reference

Base SI units.

Inheritance diagram for _Base:



Collaboration diagram for `_Base`:



Additional Inherited Members

7.3.1 Detailed Description

Base SI units.

Definition at line 162 of file `physical_quantity.py`.

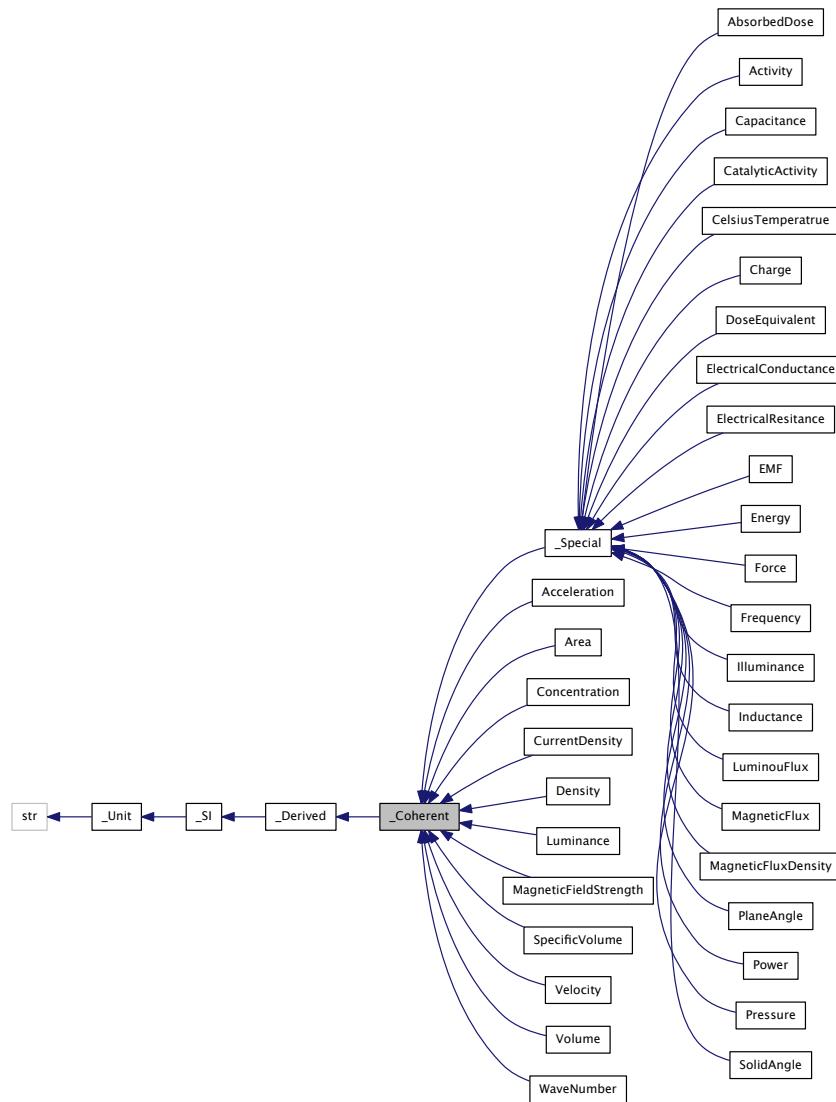
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

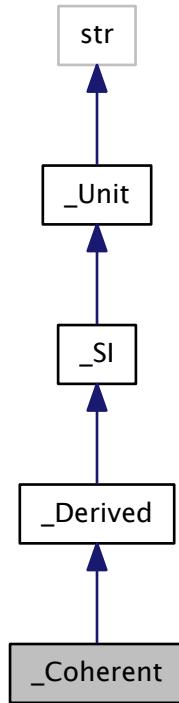
7.4 `_Coherent Class Reference`

Coherent Derived SI Units.

Inheritance diagram for _Coherent:



Collaboration diagram for `_Coherent`:



Additional Inherited Members

7.4.1 Detailed Description

Coherent Derived SI Units.

Definition at line 172 of file `physical_quantity.py`.

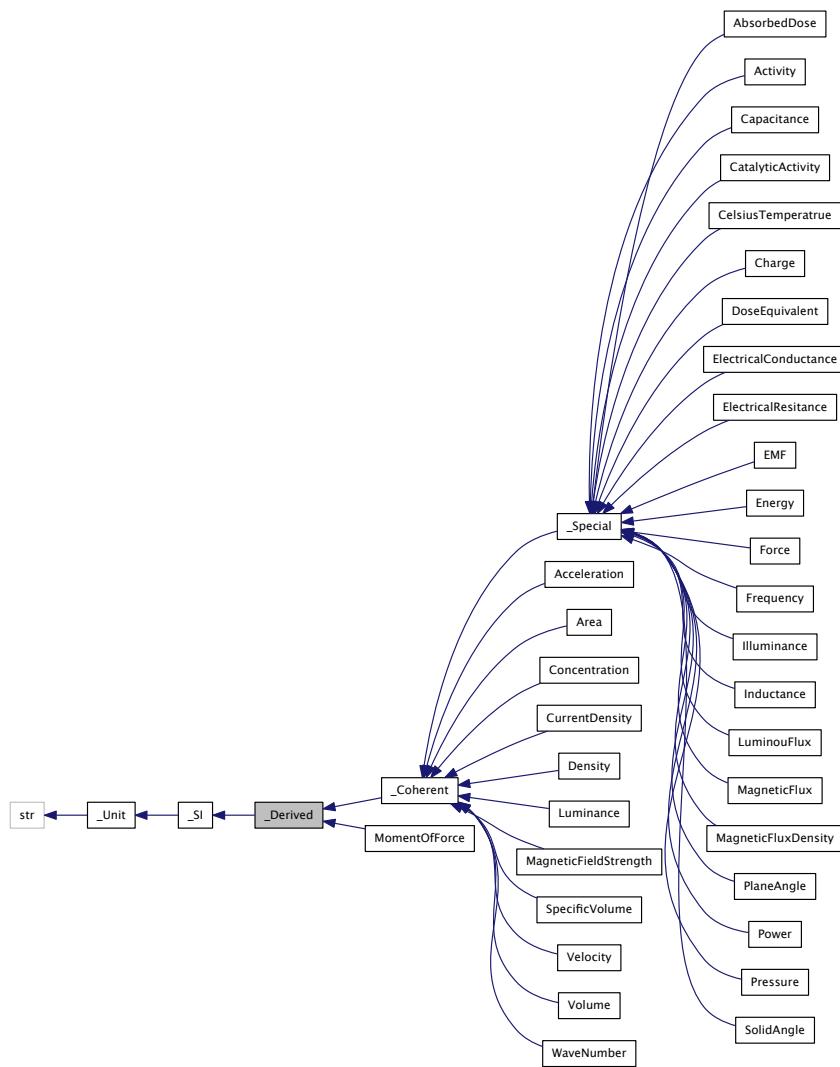
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

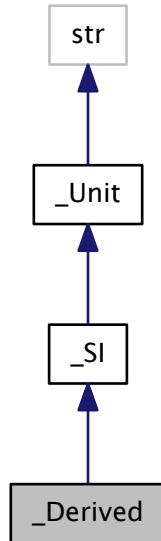
7.5 `_Derived` Class Reference

Derived DI units.

Inheritance diagram for _Derived:



Collaboration diagram for `_Derived`:



Additional Inherited Members

7.5.1 Detailed Description

Derived DI units.

Definition at line 167 of file `physical_quantity.py`.

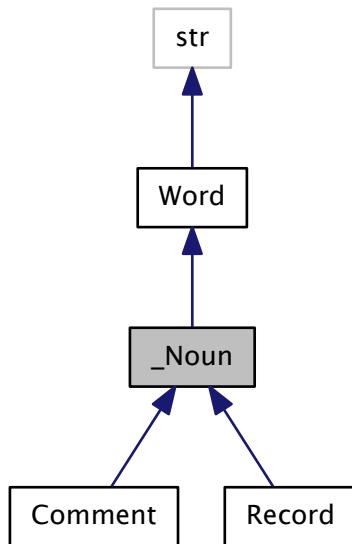
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

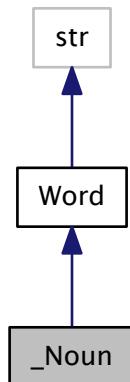
7.6 `_Noun` Class Reference

Nouns are [rdf.language.lexis.Word](#) objects that "concrete" as their `sin_qua_non`.

Inheritance diagram for _Noun:



Collaboration diagram for _Noun:



Public Member Functions

- def `__new__`
A Noun needs a name.
- def `is_`
ID Noun type – race hazard: don't send verbs to this...

- def `__call__`
Calling a Verb lets the agent act on the patient.
- def `concrete`
Calling a noun makes it's concrete person place or thing from line, according to grammar.

Static Public Attributes

- `sin_qua_non = concrete`
W/o a concrete rep, you're not a noun.

Static Private Attributes

- `__metaclass__ = abc.ABCMeta`

7.6.1 Detailed Description

Nouns are `rdf.language.lexis.Word` objects that "concrete" as their `sin_qua_non`.

Definition at line 12 of file `semantics.py`.

7.6.2 Member Function Documentation

7.6.2.1 def `__call__(self, prosodic)`

Calling a Verb lets the agent act on the patient.

Parameters

<code>line</code>	A complete RDF sentence (str)
<code>grammar</code>	An <code>rdf.language.grammar.syntax.Grammar</code> instance

Returns

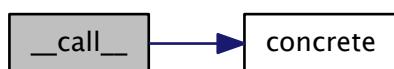
Whatever the noun's concrete method returns

Definition at line 32 of file `semantics.py`.

References `_Noun.concrete()`.

```
32
33     def __call__(self, prosodic):
34         return self.concrete(prosodic)
```

Here is the call graph for this function:



7.6.2.2 def __new__(cls)

A Noun needs a name.

Definition at line 17 of file semantics.py.

```
17
18     def __new__(cls):
19         return str.__new__(cls, cls.__name__.capitalize())
```

7.6.2.3 def concrete(self, prosodic)

Calling a noun makes it's concrete person place or thing from line, according to grammar.

Parameters

<i>line</i>	A complete RDF sentence (str)
<i>grammar</i>	An rdf.language.grammar.syntax.Grammar instance

Returns

N/A: this is an [abstractmethod](#)

Abstract method must be overridden in concrete subclasses

Definition at line 42 of file semantics.py.

Referenced by [_Noun.__call__\(\)](#).

```
42
43     def concrete(self, prosodic):
44         """Abstract method must be overridden in concrete subclasses"""
```

Here is the caller graph for this function:



7.6.2.4 def is_(self, prosodic)

ID Noun type– race hazard: don't send verbs to this...

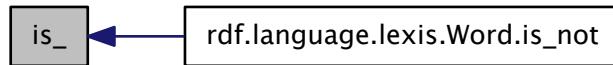
Definition at line 21 of file semantics.py.

References Record._operator_in_line, and Comment._operator_in_line.

Referenced by Word.is_not().

```
21
22     def is_(self, prosodic):
23         line, grammar = prosodic
24         return not (
25             (grammar.operator_in grammar.comment(line)[0]) ==
26             self._operator_in_line
27         )
```

Here is the caller graph for this function:



7.6.3 Member Data Documentation

7.6.3.1 __metaclass__ = abc.ABCMeta [static], [private]

Definition at line 14 of file semantics.py.

7.6.3.2 sin_qua_non = concrete [static]

W/o a concrete rep, you're not a noun.

Definition at line 46 of file semantics.py.

Referenced by Word.__call__().

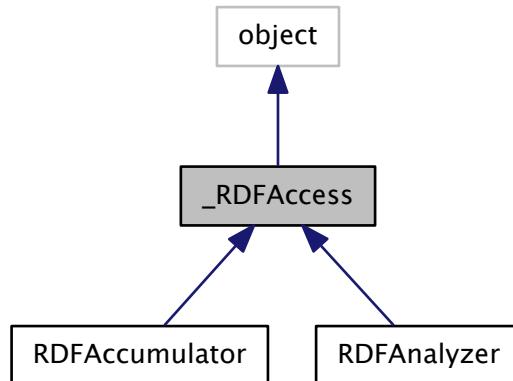
The documentation for this class was generated from the following file:

- [rdf/language/lexis/semantics.py](#)

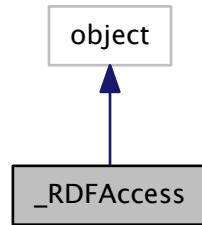
7.7 _RDFAccess Class Reference

Base class.

Inheritance diagram for _RDFAccess:



Collaboration diagram for _RDFAccess:



Public Member Functions

- def [__init__](#)
New instances get a private new `rdf.language.grammar.syntax.Grammar` instance.
- def [grammar](#)
Getter for grammar.
- def [grammar](#)
Protect the language!
- def [__call__](#)
Just call the grammar.

Private Attributes

- [_grammar](#)

Static Private Attributes

- [__metaclass__](#) = abc.ABCMeta

7.7.1 Detailed Description

Base class.

Base class for RDFAnalyzer and RDFAccumulator

Definition at line 38 of file iRDF.py.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 def __init__(self)

New instances get a private new `rdf.language.grammar.syntax.Grammar` instance.

Definition at line 45 of file iRDF.py.

```

45
46     def __init__(self):
47         self._grammar = Grammar()

```

7.7.3 Member Function Documentation

7.7.3.1 def __call__(self, line)

Just call the grammar.

Parameters

<i>line</i>	An full rdf line
-------------	------------------

Returns

Grammar's interpretation of line

Definition at line 61 of file iRDF.py.

References _RDFAccess.grammar().

```
61
62     def __call__(self, line):
63         return self.grammar(line)
64
```

Here is the call graph for this function:



7.7.3.2 def grammar(self)

Getter for grammar.

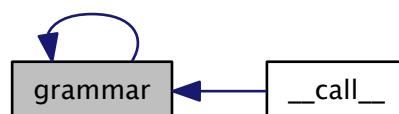
Definition at line 50 of file iRDF.py.

References _RDFAccess._grammar.

Referenced by _RDFAccess.__call__(), and _RDFAccess.grammar().

```
50
51     def grammar(self):
52         return self._grammar
```

Here is the caller graph for this function:



7.7.3.3 def grammar(self)

Protect the language!

Definition at line 55 of file iRDF.py.

References _RDFAccess.grammar().

```
55
56     def grammar(self):
57         raise TypeError("Cannot change grammar (like this)")
```

Here is the call graph for this function:



7.7.4 Member Data Documentation

7.7.4.1 __metaclass__ = abc.ABCMeta [static], [private]

Definition at line 41 of file iRDF.py.

7.7.4.2 _grammar [private]

Definition at line 46 of file iRDF.py.

Referenced by _RDFAccess.grammar().

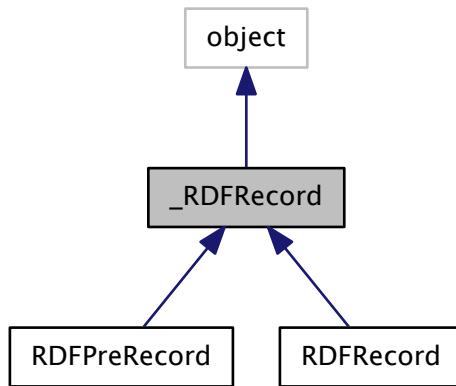
The documentation for this class was generated from the following file:

- [rdf/iRDF.py](#)

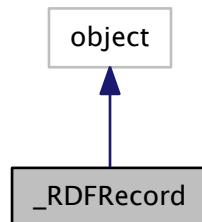
7.8 _RDFRecord Class Reference

Base for iterable (key, field) pair.

Inheritance diagram for `_RDFRecord`:



Collaboration diagram for `_RDFRecord`:



Public Member Functions

- def `__init__`
`(key, field)`

Public Attributes

- `key`
The RDF key (with affixes)
- `field`
The RDFField.

Static Private Attributes

- tuple `__slots__` = ('key', 'field')

7.8.1 Detailed Description

Base for iterable (key, field) pair.

See `__slots__`

Definition at line 155 of file `entries.py`.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 `def __init__(self, key, field)`

(key, field)

Definition at line 161 of file `entries.py`.

```
161
162     def __init__(self, key, field):
163         assert(isinstance(field, RDFField))
164         ## The RDF key (with affixes)
165         self.key = key
166         ## The RDFField
167         self.field = field
168
```

7.8.3 Member Data Documentation

7.8.3.1 `tuple __slots__ = ('key', 'field')` [static], [private]

Definition at line 158 of file `entries.py`.

Referenced by `Prosodic.__iter__()`, `RDFPreRecord.__iter__()`, and `RDFRecord.__iter__()`.

7.8.3.2 `field`

The [RDFField](#).

Definition at line 166 of file `entries.py`.

7.8.3.3 `key`

The RDF key (with affixes)

Definition at line 164 of file `entries.py`.

Referenced by `RDFRecord.__str__()`.

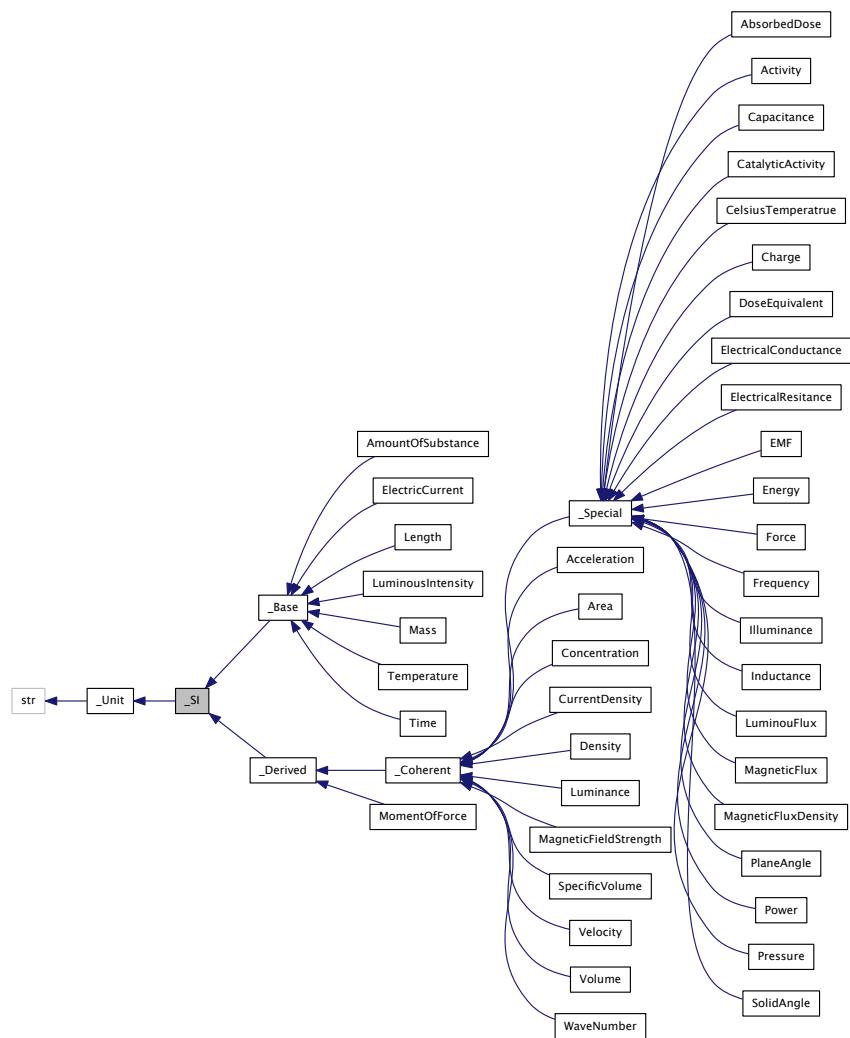
The documentation for this class was generated from the following file:

- [rdf/data/entries.py](#)

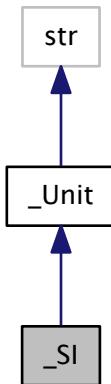
7.9 _SI Class Reference

SI units.

Inheritance diagram for _SI:



Collaboration diagram for _SI:



Additional Inherited Members

7.9.1 Detailed Description

SI units.

Definition at line 147 of file `physical_quantity.py`.

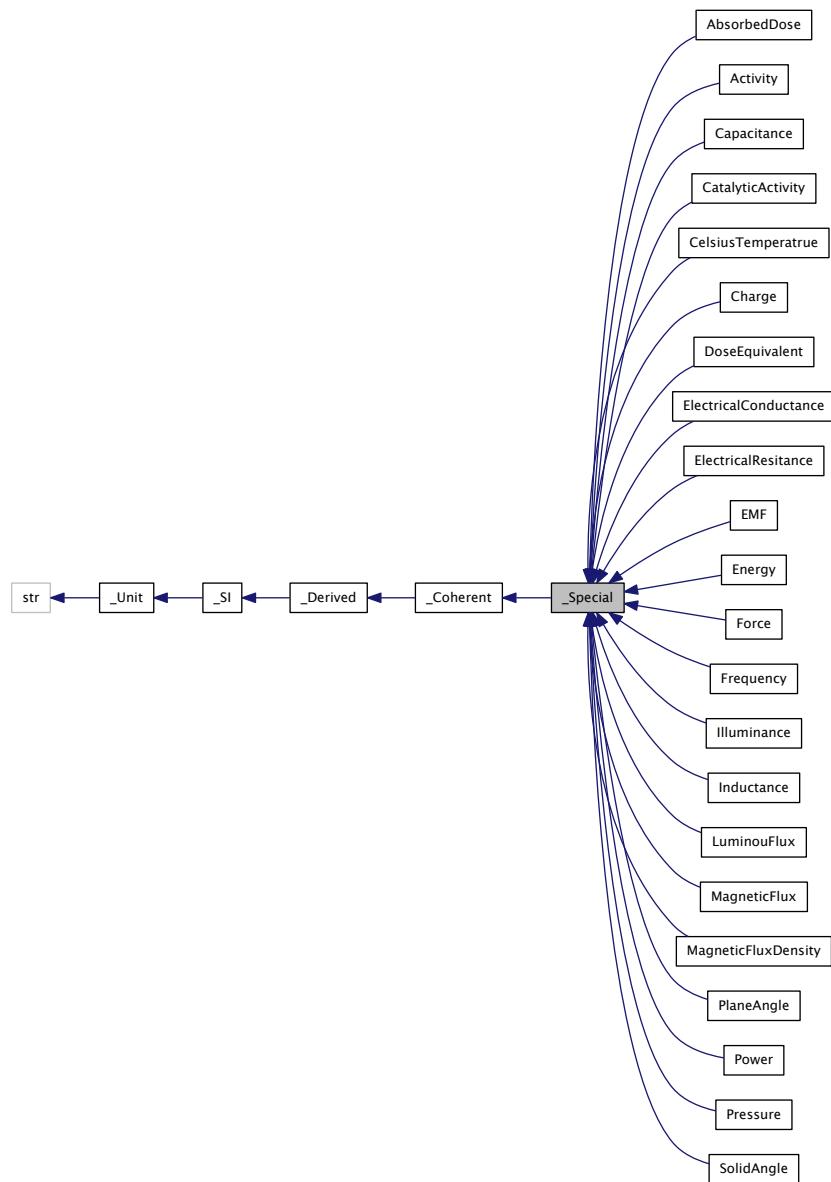
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

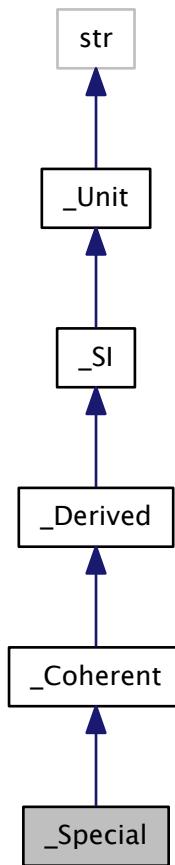
7.10 _Special Class Reference

Special Coehert Derived SI Units.

Inheritance diagram for _Special:



Collaboration diagram for _Special:



Additional Inherited Members

7.10.1 Detailed Description

Special Coehert Derived SI Units.

Definition at line 177 of file `physical_quantity.py`.

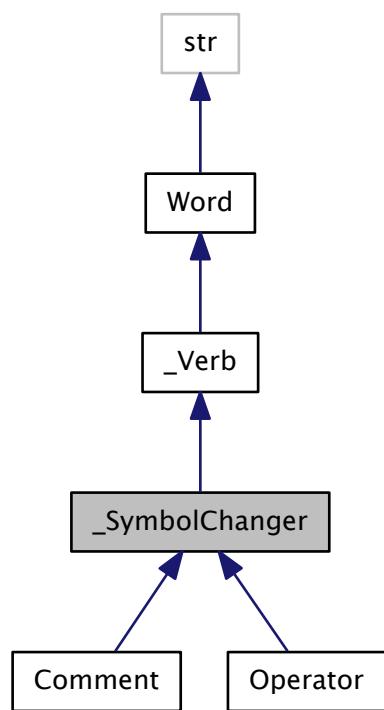
The documentation for this class was generated from the following file:

- `rdf/units/physical_quantity.py`

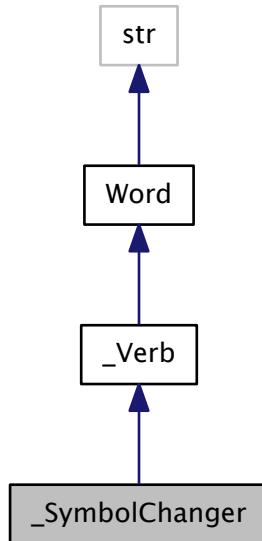
7.11 _SymbolChanger Class Reference

ABC sends sub's class.`__name__.lower()` to `rdf.language.prosodic.change_symbol`.

Inheritance diagram for `_SymbolChanger`:



Collaboration diagram for _SymbolChanger:



Public Member Functions

- def `act`
A concrete method for an abstract class– this changes grammar.

Static Private Attributes

- `__metaclass__` = `abc.ABCMeta`

7.11.1 Detailed Description

ABC sends sub's class.`__name__.lower()` to `rdf.language.prosodic.change_symbol`.

action is change symbol

Definition at line 57 of file `pragmatics.py`.

7.11.2 Member Function Documentation

7.11.2.1 def `act(self, prosodic)`

A concrete method for an abstract class– this changes grammar.

change the symbol

Definition at line 63 of file `pragmatics.py`.

```
63
64     def act(self, prosodic):
65         """change the symbol"""
66         prosodic.change_symbol(type(self).__name__.lower())
67
```

7.11.3 Member Data Documentation

7.11.3.1 __metaclass__ = abc.ABCMeta [static], [private]

Definition at line 60 of file pragmatics.py.

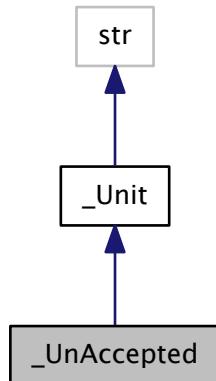
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

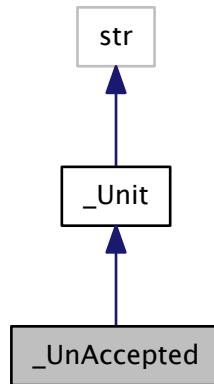
7.12 _UnAccepted Class Reference

Unaccepted units (If you must)

Inheritance diagram for _UnAccepted:



Collaboration diagram for _UnAccepted:



Additional Inherited Members

7.12.1 Detailed Description

Unaccepted units (If you must)

Definition at line 157 of file `physical_quantity.py`.

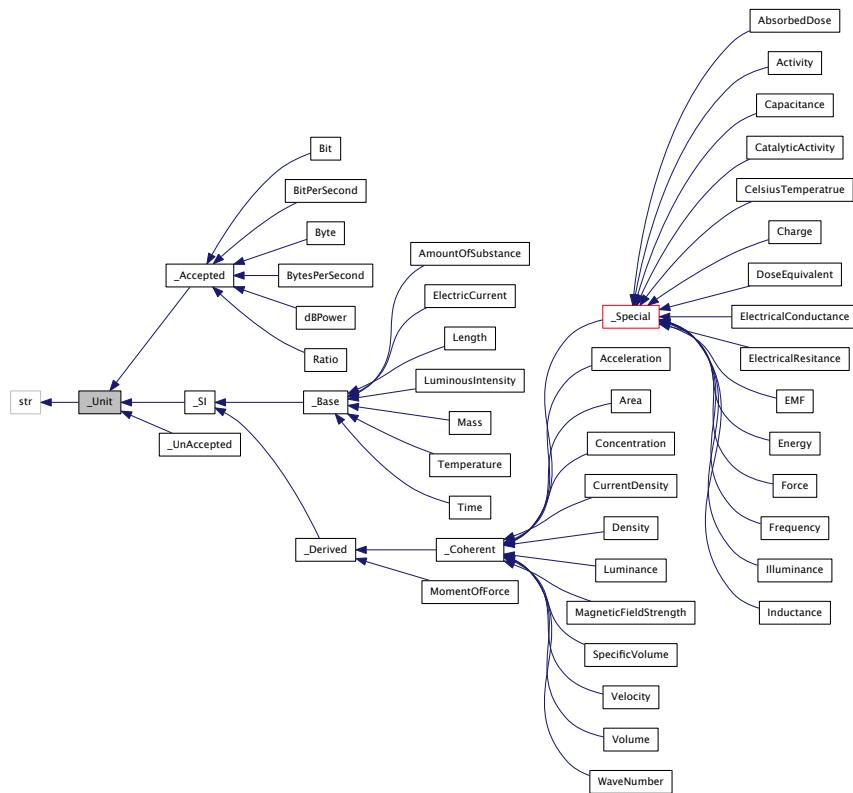
The documentation for this class was generated from the following file:

- [rdf/units/`physical_quantity.py`](#)

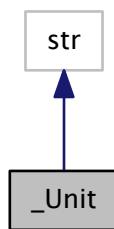
7.13 _Unit Class Reference

The [_Unit](#) class memoizes its instances.

Inheritance diagram for `_Unit`:



Collaboration diagram for `_Unit`:



Public Member Functions

- def `symbol`
getter for target unit
- def `symbol`
setter for target unit
- def `__new__`

The conversion function defined:

$$y = mx + b$$

- def `__call__`

The conversion function called:

$$y = mx + b$$

- def `__delitem__`
- def `__setitem__`

This is a `TypeError`: only `Prefix.init` can set `Unit.Glossary`.

Static Public Attributes

- dictionary `Glossary` = {}

When ever a unit is instantiated, it goes into here.

Private Member Functions

- def `_memoize`

Memoize into `_Unit.Glossary`.

Private Attributes

- `_symbol`

is this legit? what to do for units.

- `_multiplier`
- `_adder`

Static Private Attributes

- `__metaclass__` = abc.ABCMeta

The base class is for PRIVATE Units, hence the `_Unit`.

7.13.1 Detailed Description

The `_Unit` class memoizes its instances.

```
_Unit(value, multiplier=1, adder=0 [, symbol=None])
```

On `_Units` and `Prefixes`:

Instances of the `Prefix` class deocrate `_Unit` classes- and as such
create instances of:

`Sym = <Prefix> + <_Unit>` when the `_Unit` subclass is created (at import).

That instance is, of course, also a `<str>` and is memoized in

`_Unit.Glossary`

dictionary as:

```
{Sym: Sym}
```

At fist, that looks odd. The point is to do a hash-table search (not a list
search) in the `Glossary` with "Sym" as a key-- here "Sym" is the ordinary
string supplied by the RDF file's (unit) field.

the resulting Value converts units to <_Unit>'s symbol with <unit>.factor as a scaling.

Hence, of you're talking `float(x) "km"`, you get:

```
Glossary["km"] (x) --> 1000*x, "m"
```

Definition at line 44 of file `physical_quantity.py`.

7.13.2 Member Function Documentation

7.13.2.1 def __call__(self, x)

The conversion function called:

$$y = mx + b$$

Parameters

<code>x</code>	is the value in non-base/SI units, and must support <code>float()</code>
----------------	--

Return values

<code>y</code>	is the value in <code>self.symbol</code>
----------------	--

Definition at line 128 of file `physical_quantity.py`.

References `_Unit._adder`, and `_Unit._multiplier`.

```
128
129     def __call__(self, x):
130         # todo: case x? who has case?
131         return self._multiplier * float(x) + self._adder
```

7.13.2.2 def __delitem__(cls, index)

Parameters

<code>index</code>	Key to delete
--------------------	---------------

Side Effects:

deletes key from `rdf.units.GLOSSARY` for ever.

Definition at line 136 of file `physical_quantity.py`.

```
136
137     def __delitem__(cls, index):
138         del cls.Glossary[index]
```

7.13.2.3 def __new__(cls, abbreviation = "", multiplier = 1, adder = 0, symbol = None)

The conversion function defined:

$$y = mx + b$$

Parameters

<i>m</i>	is the multiplier for the conversion
<i>b</i>	is the adder (applied 1st).

Side Effects:

Instance is `_memoized()`'d.

Returns

A string that can be looked up with a str in a hash-table and can then do unit conversion.

```
abbrev="", multiplier=1, adder=0, symbol=None) :
```

Definition at line 98 of file `physical_quantity.py`.

```
98
99     def __new__(cls, abbreviation="", multiplier=1, adder=0, symbol=None):
100         """abbrev="", multiplier=1, adder=0, symbol=None)"""
101         self = str.__new__(cls, abbreviation or symbol or cls._symbol)
102         self._multiplier = multiplier
103         self._adder = adder
104         # is this legit? what to do for units..
105         if symbol is not None: # Guard on keyword option
106             self._symbol = str(symbol)
107
108         ## All new instances get memoized
109         self._memoize()
110
111     return self
```

7.13.2.4 def __setitem__(cls, index, value)

This is a `TypeError`: only `Prefix.init` can set `Unit.Glossary`.

Definition at line 141 of file `physical_quantity.py`.

```
141
142     def __setitem__(cls, index, value):
143         raise TypeError("Only Instaniation can set items for % class" %
144                         cls.__name__)
145
```

7.13.2.5 def _memoize(self, warn=True) [private]

Memoize into `_Unit.Glossary`.

```
save self into Glossary, w/ overwite warning option
```

Definition at line 113 of file `physical_quantity.py`.

References `_Unit.Glossary`.

```
113
114     def _memoize(self, warn=True):
115         """save self into Glossary, w/ overwite warning option"""
116         # check key or not?
117         if warn and self in self.Glossary: # Guard
118             from rdf.units.errors import RedefiningUnitWarning
119             raise RedefiningUnitWarning #you cannot raise this safely, why?
120             print >> sys.stderr, (
121                 'Warning: Overwriting Unit.Glossary[%s]' % self
122             )
123         self.Glossary.update({self: self})
```

7.13.2.6 def symbol (self)

getter for target unit

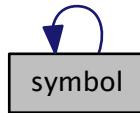
Definition at line 82 of file physical_quantity.py.

References `_Unit._symbol`.

Referenced by `_Unit.symbol()`.

```
82
83     def symbol(self):
84         return self._symbol
```

Here is the caller graph for this function:



7.13.2.7 def symbol (self, symbol)

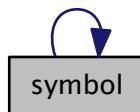
setter for target unit

Definition at line 87 of file physical_quantity.py.

References `_Unit.symbol()`.

```
87
88     def symbol(self, symbol):
89         self._symbol = symbol
```

Here is the call graph for this function:



7.13.3 Member Data Documentation

7.13.3.1 __metaclass__ = abc.ABCMeta [static], [private]

The base class is for PRIVATE Units, hence the `_Unit`.

Definition at line 75 of file physical_quantity.py.

7.13.3.2 `_adder` [private]

Definition at line 102 of file `physical_quantity.py`.

Referenced by `_Unit.__call__()`.

7.13.3.3 `_multiplier` [private]

Definition at line 101 of file `physical_quantity.py`.

Referenced by `_Unit.__call__()`.

7.13.3.4 `_symbol` [private]

is this legit? what to do for units.

Definition at line 88 of file `physical_quantity.py`.

Referenced by `_Unit.symbol()`.

7.13.3.5 `dictionary Glossary = {}` [static]

When ever a unit is instantiated, it goes into here.

Definition at line 78 of file `physical_quantity.py`.

Referenced by `_Unit._memoize()`.

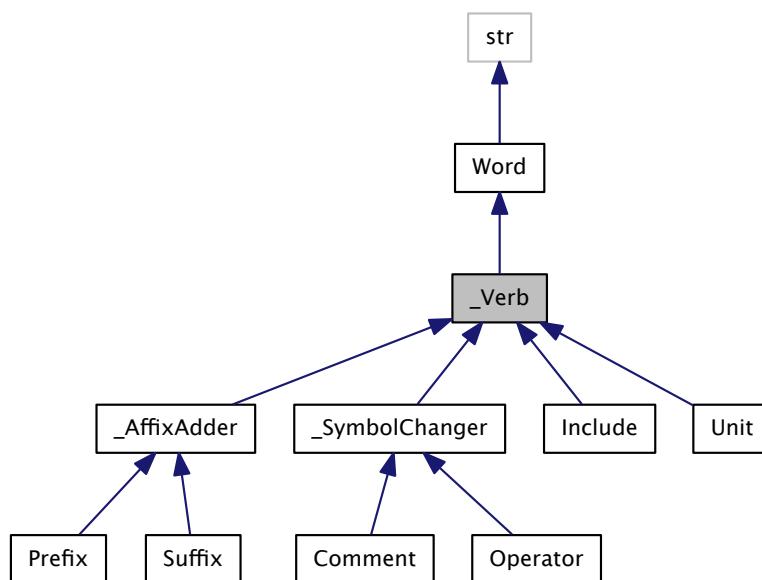
The documentation for this class was generated from the following file:

- `rdf/units/physical_quantity.py`

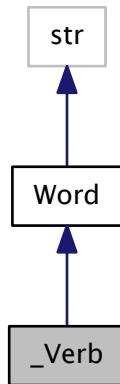
7.14 _Verb Class Reference

Verbs are `rdf.language.lexis.Word` objects that "act" as their `sin_qua_non`.

Inheritance diagram for `_Verb`:



Collaboration diagram for `_Verb`:



Public Member Functions

- def `__new__`
A Verb need a name, and it will be VERB.
- def `is_`
Allow class to identify itself in the context of an `rdf.language.prosodic`.
- def `act`
*Act is not action-> act tells this object to go do its thing
which is act on the grammar according to line.*
- def `sin_qua_non`
Verbs must act – or return an empty iterable.

Static Private Attributes

- `__metaclass__` = abc.ABCMeta
`_Verb` is to general to instantiated, it delegates `sin_qua_non` to `act` and decides `is_not()`

7.14.1 Detailed Description

Verbs are `rdf.language.lexis.Word` objects that "act" as their `sin_qua_non`.

`_Pragamtic` is an self identifying string

Definition at line 12 of file `pragmatics.py`.

7.14.2 Member Function Documentation

7.14.2.1 def `__new__(cls)`

A Verb need a name, and it *will* be VERB.

Definition at line 20 of file `pragmatics.py`.

```

20
21     def __new__(cls):
22         return str.__new__(cls, cls.__name__.upper())

```

7.14.2.2 def act(self, prosodic)

Act is not action-> act tells this object to go do its thing
which is act on the grammar according to line.

```
act(prosodic) --> do what the line says and return result
```

Definition at line 37 of file pragmatics.py.

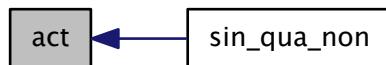
Referenced by `_Verb.sin_qua_non()`.

```

37
38     def act(self, prosodic):
39         """act(prosodic) --> do what the line says and return result"""

```

Here is the caller graph for this function:



7.14.2.3 def is_(self, prosodic)

Allow class to identify itself in the context of an [rdf.language.prosodic](#).

```
is_not(prosodic) IFF line is pragmatic
```

Definition at line 25 of file pragmatics.py.

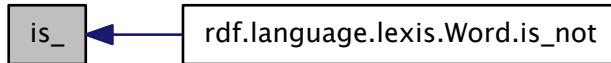
Referenced by `Word.is_not()`.

```

25
26     def is_(self, prosodic):
27         """is_not(prosodic) IFF line is pragmatic"""
28         line = prosodic.strip()
29         ## TODO: refactor this pos:
30         return not (
31             (line.startswith(self) and
32              (line.lstrip(self).strip().startswith(prosodic.operator)))
33         )

```

Here is the caller graph for this function:



7.14.2.4 def sin_qua_non (self, prosodic)

Verbs must act – or return an empty iterable.

see act

Definition at line 41 of file pragmatics.py.

References [_Verb.act\(\)](#).

```

41
42     def sin_qua_non(self, prosodic):
43         """see act"""
44         return self.act(prosodic) or ()
45
  
```

Here is the call graph for this function:



7.14.3 Member Data Documentation

7.14.3.1 __metaclass__ = abc.ABCMeta [static], [private]

[_Verb](#) is to general to instantiated, it delegates `sin_qua_non` to `act` and decides [is_not\(\)](#)

Definition at line 17 of file pragmatics.py.

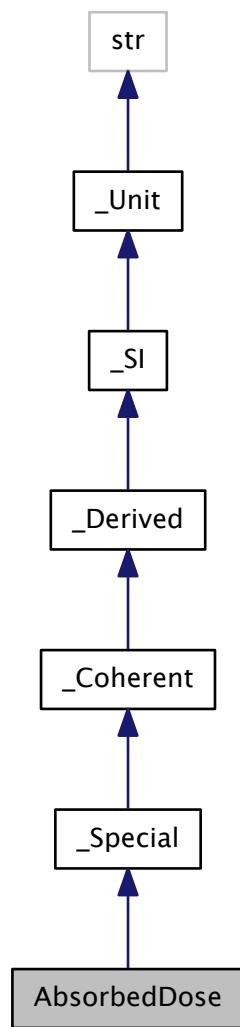
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

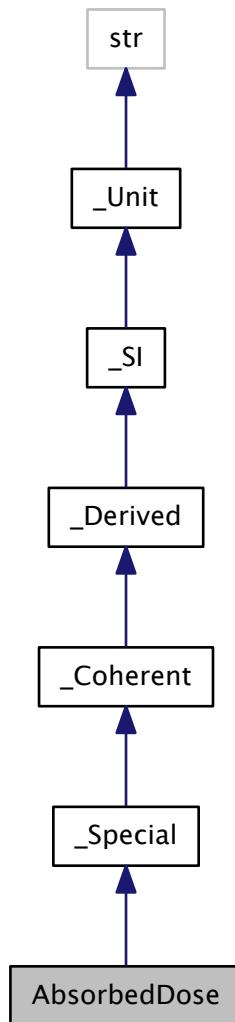
7.15 AbsorbedDose Class Reference

Louis Harold Gray (10 November 1905 – 9 July 1965)

Inheritance diagram for AbsorbedDose:



Collaboration diagram for AbsorbedDose:



Static Private Attributes

- string `_symbol` = 'Gy'

Additional Inherited Members

7.15.1 Detailed Description

Louis Harold Gray (10 November 1905 – 9 July 1965)

Definition at line 347 of file `physical_quantity.py`.

7.15.2 Member Data Documentation

7.15.2.1 `string _symbol = 'Gy' [static], [private]`

Definition at line 348 of file `physical_quantity.py`.

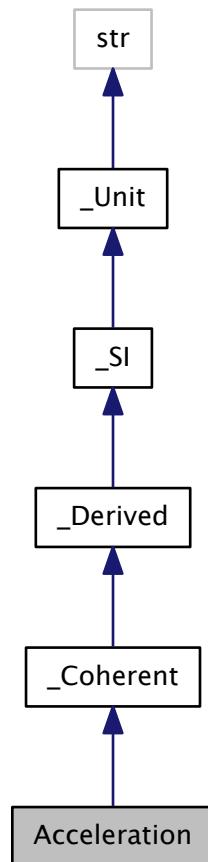
The documentation for this class was generated from the following file:

- `rdf/units/physical_quantity.py`

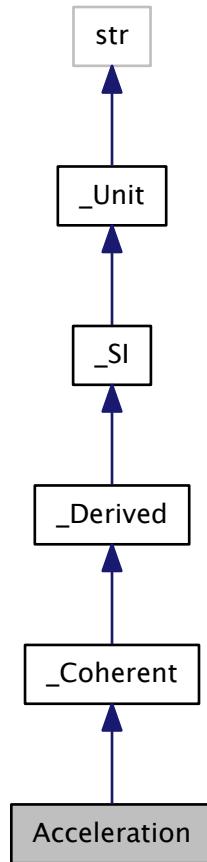
7.16 Acceleration Class Reference

meters per seconds squared

Inheritance diagram for Acceleration:



Collaboration diagram for Acceleration:



Static Private Attributes

- string `_symbol` = 'm/s**2'

Additional Inherited Members

7.16.1 Detailed Description

meters per seconds squared

Definition at line 402 of file physical_quantity.py.

7.16.2 Member Data Documentation

7.16.2.1 string `_symbol` = 'm/s**2' [static], [private]

Definition at line 403 of file physical_quantity.py.

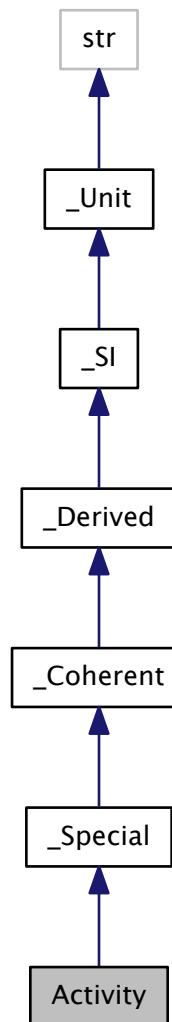
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

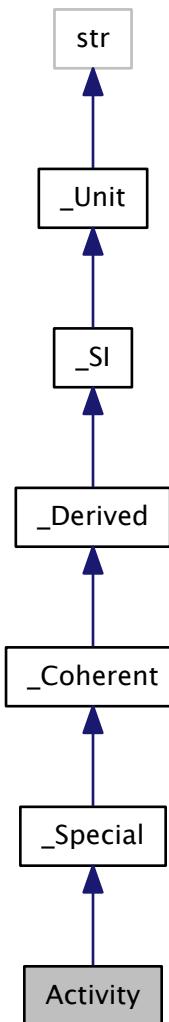
7.17 Activity Class Reference

Antoine Henri Becquerel (15 December 1852 – 25 August 1908)

Inheritance diagram for Activity:



Collaboration diagram for Activity:



Static Private Attributes

- string `_symbol` = 'Bq'

Additional Inherited Members

7.17.1 Detailed Description

Antoine Henri Becquerel (15 December 1852 – 25 August 1908)

Definition at line 341 of file `physical_quantity.py`.

7.17.2 Member Data Documentation

7.17.2.1 `string _symbol = 'Bq' [static], [private]`

Definition at line 342 of file `physical_quantity.py`.

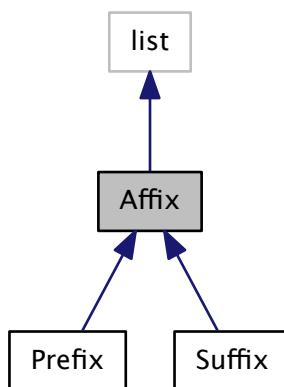
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

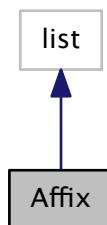
7.18 Affix Class Reference

Abstract Base Class for Pre/Suf behavior.

Inheritance diagram for Affix:



Collaboration diagram for Affix:



Public Member Functions

- def `descend`
Descend the IFT- add a null string to the affix list.
- def `ascend`

- Ascend the IFT- pop the affix off and forget it.
- def `__call__`
Call implements the construction of the affix (so IF you change the def you change this 1 line of code).
- def `__add__`
strictly for safety

Static Private Attributes

- `__metaclass__` = abc.ABCMeta
- `__radd__` = `__add__`

7.18.1 Detailed Description

Abstract Base Class for Pre/Suf behavior.

The Affix is an abstract base class.
It implements the:

descend/asend methods for traversing the IFT

It is callable: Given a key, it will do what morphemes do and make
as new key per the RDF spec.

Sub classes use operator overloads to do their thing

Definition at line 15 of file morpheme.py.

7.18.2 Member Function Documentation

7.18.2.1 def `__add__`(`self`, `other`)

strictly for safety

Definition at line 55 of file morpheme.py.

```
55
56     def __add__(self, other):
57         from rdf.language import errors
58         raise (
59             {True: errors.MorphemeExchangeError(
60                 "Cannot Pre/Append a Suf/Pre-fix"),
61              False: TypeError("Can only add strings to this list sub")})[
62                  isinstance(other, basestring)
63                  ]
64      )
```

7.18.2.2 def `__call__`(`self`)

Call implements the construction of the affix (so IF you change the def you change this 1 line of code).

Returns

Sum of self- the complete affix

call implements the nest affix protocol: add 'em up

Definition at line 50 of file morpheme.py.

```
50
51     def __call__(self):
52         """call implements the nest affix protocol: add 'em up"""
53         return "".join(self)
```

7.18.2.3 def ascend(*self*)

Ascend the IFT– pop the affix off and forget it.

Parameters

<i>None</i>	
-------------	--

Side Effects:

Pops last affix off of [Affix](#)

Returns

None

```
pop() from self
```

Definition at line 43 of file morpheme.py.

```
43     def ascend(self):
44         """pop() from self"""
45         return self.pop()
```

7.18.2.4 def descend(self)

Descend the IFT– add a null string to the affix list.

Parameters

<i>None</i>	
-------------	--

Side Effects:

Append null string to [Affix](#)

Returns

None

```
append null string to self
```

Definition at line 34 of file morpheme.py.

```
34     def descend(self):
35         """append null string to self"""
36         return self.append("")
```

7.18.3 Member Data Documentation**7.18.3.1 __metaclass__ = abc.ABCMeta [static], [private]**

Definition at line 27 of file morpheme.py.

7.18.3.2 __radd__ = __add__ [static], [private]

Definition at line 65 of file morpheme.py.

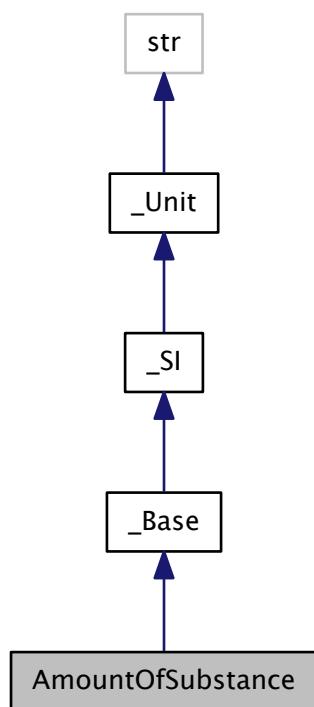
The documentation for this class was generated from the following file:

- rdf/language/grammar/[morpheme.py](#)

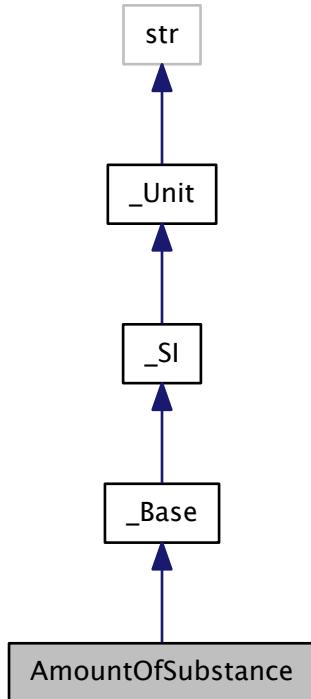
7.19 AmountOfSubstance Class Reference

Lorenzo Romano Amedeo Carlo Avogadro di Quaregna e di Cerreto, Count of Quaregna and Cerreto (9 August 1776, Turin, Piedmont – 9 July 1856)

Inheritance diagram for AmountOfSubstance:



Collaboration diagram for AmountOfSubstance:



Static Private Attributes

- string `_symbol` = "mol"

Additional Inherited Members

7.19.1 Detailed Description

Lorenzo Romano Amedeo Carlo Avogadro di Quaregna e di Cerreto, Count of Quaregna and Cerreto (9 August 1776, Turin, Piedmont – 9 July 1856)

Definition at line 219 of file `physical_quantity.py`.

7.19.2 Member Data Documentation

7.19.2.1 string `_symbol` = "mol" [static], [private]

Definition at line 220 of file `physical_quantity.py`.

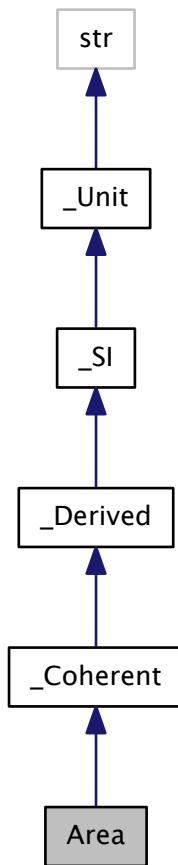
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

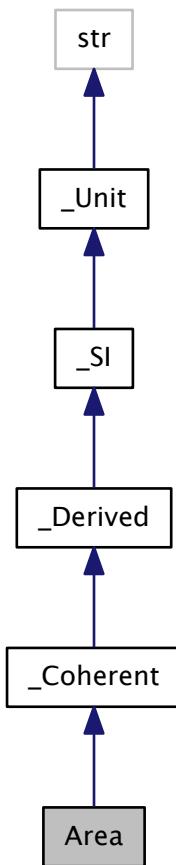
7.20 Area Class Reference

meters squared

Inheritance diagram for Area:



Collaboration diagram for Area:



Static Private Attributes

- string `_symbol` = 'm**2'

Additional Inherited Members

7.20.1 Detailed Description

meters squared

Definition at line 365 of file physical_quantity.py.

7.20.2 Member Data Documentation

7.20.2.1 string `_symbol` = 'm**2' [static], [private]

Definition at line 366 of file physical_quantity.py.

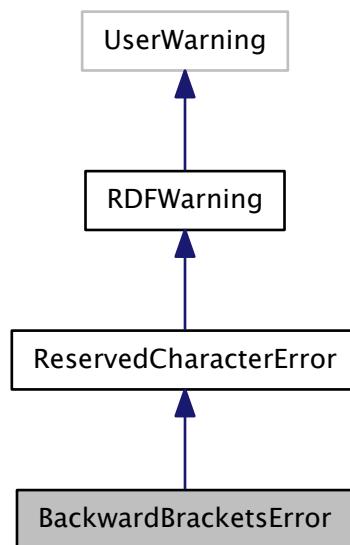
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

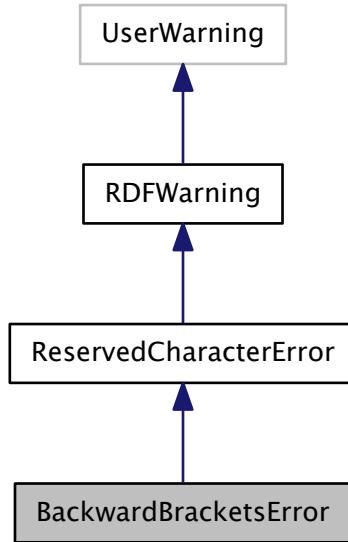
7.21 BackwardBracketsError Class Reference

Unmatched or un parseable pairs.

Inheritance diagram for BackwardBracketsError:



Collaboration diagram for BackwardBracketsError:



7.21.1 Detailed Description

Unmatched or un parsable pairs.

Inverted Punctuation

Definition at line 38 of file errors.py.

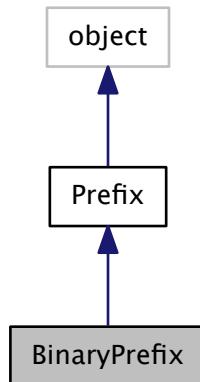
The documentation for this class was generated from the following file:

- rdf/language/[errors.py](#)

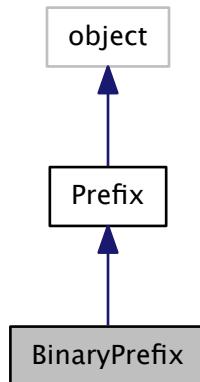
7.22 BinaryPrefix Class Reference

Binary Prefix Note: limits/differences of/between JEDEC and IEC

Inheritance diagram for BinaryPrefix:



Collaboration diagram for BinaryPrefix:



Public Member Functions

- def `__long__`
cast() returns a function that calls this
- def `cast`
long

Static Public Attributes

- int `base` = 1024
 2^{10}

Additional Inherited Members

7.22.1 Detailed Description

Binary Prefix Note: limits/differences of/between JEDEC and IEC

Prefix based on 1024

Definition at line 95 of file prefix.py.

7.22.2 Member Function Documentation

7.22.2.1 def __long__(self)

`cast()` returns a function that calls this

Definition at line 102 of file prefix.py.

References Prefix.factor.

```
102
103     def __long__(self):
104         return long(self.factor)
```

7.22.2.2 def cast(self)

long

Definition at line 106 of file prefix.py.

```
106
107     def cast(self):
108         return long
109
```

7.22.3 Member Data Documentation

7.22.3.1 int base = 1024 [static]

2^{10}

Definition at line 99 of file prefix.py.

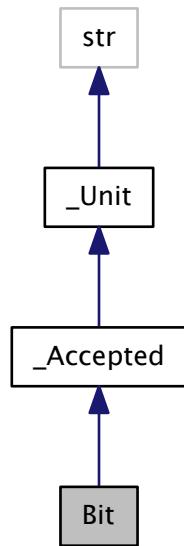
The documentation for this class was generated from the following file:

- rdf/units/[prefix.py](#)

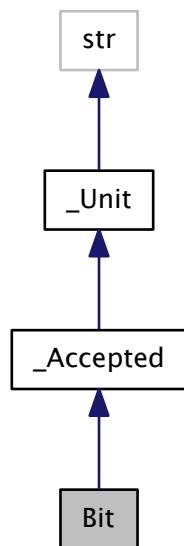
7.23 Bit Class Reference

Data [Volume](#) conversion to bits.

Inheritance diagram for Bit:



Collaboration diagram for Bit:



Static Private Attributes

- string `_symbol` = 'bits'

Additional Inherited Members

7.23.1 Detailed Description

Data `Volume` conversion to bits.

Definition at line 446 of file `physical_quantity.py`.

7.23.2 Member Data Documentation

7.23.2.1 string `_symbol` = 'bits' [static], [private]

Definition at line 447 of file `physical_quantity.py`.

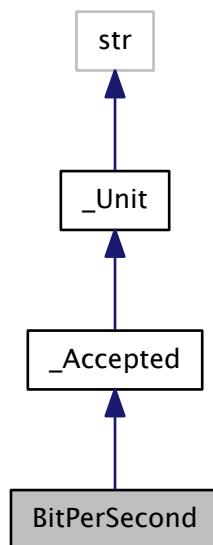
The documentation for this class was generated from the following file:

- `rdf/units/physical_quantity.py`

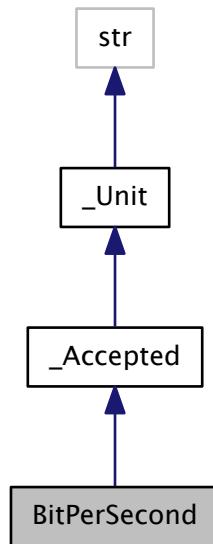
7.24 BitPerSecond Class Reference

Data rate conversion to bps.

Inheritance diagram for BitPerSecond:



Collaboration diagram for BitPerSecond:



Static Private Attributes

- string `_symbol` = 'bits/s'

Additional Inherited Members

7.24.1 Detailed Description

Data rate conversion to bps.

Definition at line 454 of file physical_quantity.py.

7.24.2 Member Data Documentation

7.24.2.1 string `_symbol` = 'bits/s' [static], [private]

Definition at line 455 of file physical_quantity.py.

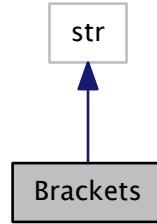
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

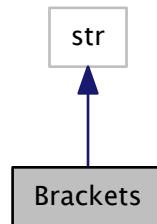
7.25 Brackets Class Reference

Brackets that know themselves.

Inheritance diagram for Brackets:



Collaboration diagram for Brackets:



Public Member Functions

- def `__pos__`
 $L, R \rightarrow -, +$
+ is right.
- def `__neg__`
 $L, R \rightarrow -, -$
- is left.
- def `__rlshift__`
extract enclosed: line<<pair
- def `__rrshift__`
Insert: line>>pair or go blank.
- def `__contains__`
(line in delimiter) IF the line has token in it legally
- def `__rsub__`
line - delimiter removes delimiter from line, with no IF

Private Member Functions

- def `_get_outer`

Extract part of line outside the glyph (stripped)

- def `_noop_on_inner`

Method "could be a function" -except it is called as an unbound method by `rsub` and the signature must match `_get_outer()` which could not be a function– so-to-speak.

Static Private Attributes

- `__lshift__ = __rrshift__`
- `__rshift__ = __rlshift__`
- dictionary `_rsub_choser = {True: _get_outer, False: _noop_on_inner}`

dict allows `rsub` to chose correct method to process a line with neither If/then no try/except.

7.25.1 Detailed Description

Brackets that know thyselfes.

```
_Delimeter('LR')

get it? Knows how to find itself in a line, par exemplio:

>>>line = "a line <with> a bracket"
>>>b = Brackets("<>")

You get:

OP          VALUE           Comments
-----
+b          '>'            + is the right side
-b          '<'            - is the left side

b >> line      'with'         / Extract bracket's contents for
line << b       'with'         \ a line with the bracket

'with' >> b     '<with>'       / Insert a value in the brackets
b << 'with'      '<with>'       \ and make a usable string

b - line        TypeError
line - b        'a line a bracket'  / get the bracket out of the line
                                         \ normal
b in line       False           / str.__contains__
b in '<>'       True            \

line in b       True            / True If line uses bracket
(line - b) in b False           \ False If it does not.
```

Definition at line 80 of file punctuation.py.

7.25.2 Member Function Documentation

7.25.2.1 def `__contains__`(`self, line`)

(line in delimiter) IF the line has token in it legally

Parameters

<code>line</code>	an RDF sentence
-------------------	-----------------

Return values

<code><bool></code>	IF Bracket is in the line
---------------------------	---------------------------

Definition at line 165 of file punctuation.py.

```
165     def __contains__(self, line):
166         return ((-self in line) and
167                 (+self in line) and
168                 line.index(-self) < line.index(+self))
```

7.25.2.2 def __neg__(self)

L, R → -, +

- is left.

Definition at line 115 of file punctuation.py.

```
115     def __neg__(self):
116         return self[:len(self)/2]
```

7.25.2.3 def __pos__(self)

L, R → -, +

+ is right.

Definition at line 111 of file punctuation.py.

```
111     def __pos__(self):
112         return self[-len(self)/2:]
```

7.25.2.4 def __rlshift__(self, line)

extract enclosed: line<<pair

Parameters

<code>line</code>	An RDF sentence
-------------------	-----------------

Side Effects:

raises RDFWarning on bad grammar

Return values

<code>contents</code>	The string inside the Bracket INPUTS: pair, line pair is 2 characters LR, this extracts part of line between L and R. Throws errors IF need be.
-----------------------	---

Definition at line 123 of file punctuation.py.

```
123
124     def __rlshift__(self, line):
125         """INPUTS: pair, line
126
127         pair is 2 characters LR, this extracts part of line
```

```

128      between L      and      R. Throws errors IF need be.
129      """
130      # 5 IF's are for error checking, not processing
131      from rdf.language import errors
132      ## Count start and stops
133      count = map(line.count, self)
134      ## Guard: early return.
135      if min(count) is 0:  # Guard
136          ## Check IF there is an oper/close error
137          if max(count):  # Guard
138              raise errors.UnmatchedBracketsError(self)
139          return None
140      ## Ensure 1 pair
141      if max(count) > 1:  # Guard
142          raise errors.RunOnSentenceError(self)
143      i_start = line.index(-self) + 1
144      i_stop = line.index(+self)
145      ## ensure order:
146      if i_stop <= i_start:  # Guard
147          raise errors.BackwardBracketsError(self)
148      contents = line[i_start: i_stop]
149      # finally check for nonesense
150      for single_char in contents:
151          if single_char in reserved.SINGULAR:  # Guard
152              raise errors.ReservedCharacterError(self)
153      return contents

```

7.25.2.5 def __rrshift__(self, line)

Insert: line>>pair or go blank.

Insert non-zero line in string, or nothing

Definition at line 155 of file punctuation.py.

```

155
156     def __rrshift__(self, line):
157         """Insert non-zero line in string, or nothing"""
158         return " %s%s " % (-self, str(line), +self) if line else ""

```

7.25.2.6 def __rsub__(self, line)

line - delimiter removes delimiter from line, with no IF

Get value surrounded

Definition at line 171 of file punctuation.py.

References Brackets._rsub_choser.

```

171
172     def __rsub__(self, line):
173         """Get value surrounded"""
174         return self._rsub_choser[line in self](self, line)

```

7.25.2.7 def _get_outer(self, line) [private]

Extract part of line outside the glyph (stripped)

_get_outer('a line <with> a glyph') --> 'a line a glyph'
so it's every thing but the glyph and its contents

Definition at line 176 of file punctuation.py.

```

176
177     def _get_outer(self, line):
178         """_get_outer('a line <with> a glyph') --> 'a line a glyph'
179         so it's every thing but the glyph and its contents"""
180         return (line[:line.index(-self)] +
181                 line[1+line.index(+self):]).strip()

```

7.25.2.8 `def _noop_on_inner(self, line) [private]`

Method "could be a function" -except it is called as an unbound method by `rsub` and the signature must match `_get_outer()` which could not be a function– so-to-speak.

If `glyph` is not in `line`, then do nothing

Definition at line 185 of file punctuation.py.

```
185     def _noop_on_inner(self, line):
186         """If glyph is not in line, then do nothing"""
187         return line
188
```

7.25.3 Member Data Documentation

7.25.3.1 `__lshift__ = __rrshift__ [static], [private]`

Definition at line 159 of file punctuation.py.

7.25.3.2 `__rshift__ = __rlshift__ [static], [private]`

Definition at line 160 of file punctuation.py.

7.25.3.3 `dictionary_rsub_choser = {True: _get_outer, False: _noop_on_inner} [static], [private]`

dict allows `rsub` to chose correct method to process a line with neither If/then no try/except.

Definition at line 191 of file punctuation.py.

Referenced by Brackets.`__rsub__()`.

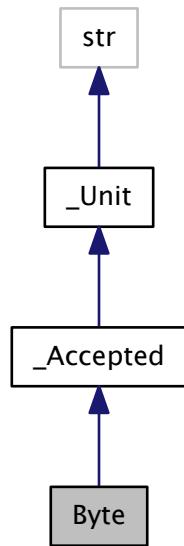
The documentation for this class was generated from the following file:

- rdf/language/grammar/punctuation.py

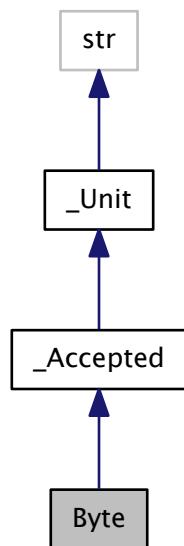
7.26 Byte Class Reference

Data [Volume](#) conversion to bits.

Inheritance diagram for Byte:



Collaboration diagram for Byte:



Static Private Attributes

- string `_symbol` = 'byte'

Additional Inherited Members

7.26.1 Detailed Description

Data [Volume](#) conversion to bits.

Definition at line 462 of file `physical_quantity.py`.

7.26.2 Member Data Documentation

7.26.2.1 string `_symbol` = 'byte' [static], [private]

Definition at line 463 of file `physical_quantity.py`.

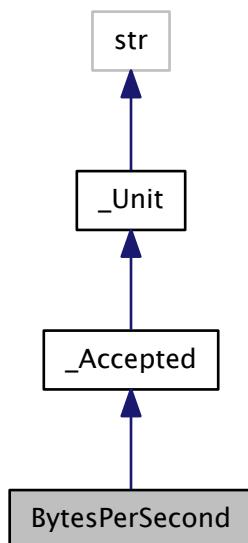
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

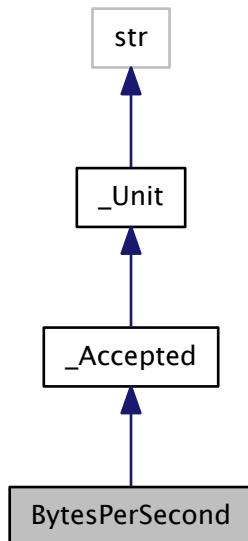
7.27 BytesPerSecond Class Reference

Data rate conversion to bytes per second.

Inheritance diagram for BytesPerSecond:



Collaboration diagram for BytesPerSecond:



Static Private Attributes

- string [_symbol](#) = 'byte/s'

Additional Inherited Members

7.27.1 Detailed Description

Data rate conversion to bytes per second.

Definition at line 470 of file `physical_quantity.py`.

7.27.2 Member Data Documentation

7.27.2.1 string [_symbol](#) = 'byte/s' [static], [private]

Definition at line 471 of file `physical_quantity.py`.

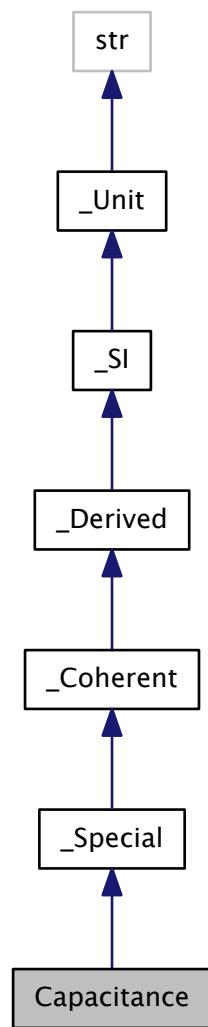
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

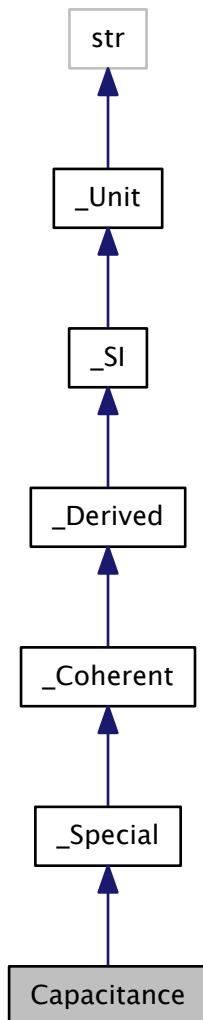
7.28 Capacitance Class Reference

Michael Faraday, FRS (22 September 1791 – 25 August 1867)

Inheritance diagram for Capacitance:



Collaboration diagram for Capacitance:



Static Private Attributes

- string `_symbol` = 'F'

Additional Inherited Members

7.28.1 Detailed Description

Michael Faraday, FRS (22 September 1791 – 25 August 1867)

Definition at line 286 of file physical_quantity.py.

7.28.2 Member Data Documentation

7.28.2.1 string _symbol = 'F' [static], [private]

Definition at line 287 of file physical_quantity.py.

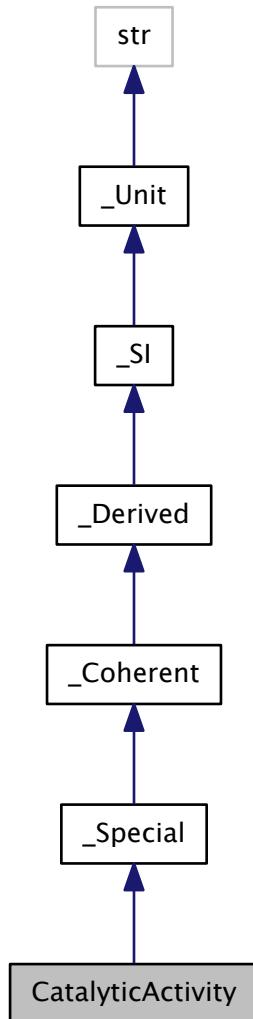
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

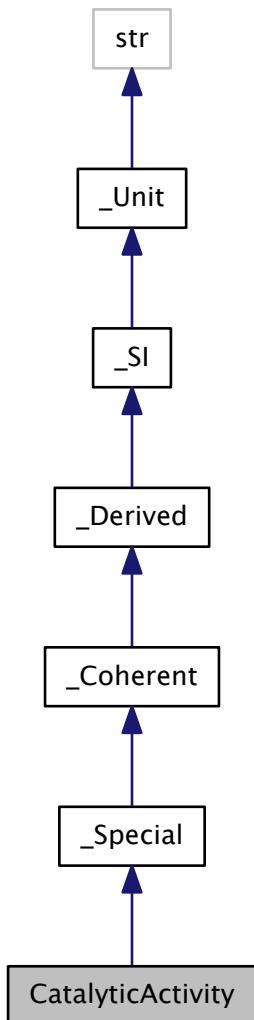
7.29 CatalyticActivity Class Reference

katal

Inheritance diagram for CatalyticActivity:



Collaboration diagram for CatalyticActivity:



Static Private Attributes

- string symbol = 'kat'

Additional Inherited Members

7.29.1 Detailed Description

katal

Definition at line 359 of file physical_quantity.py.

7.29.2 Member Data Documentation

7.29.2.1 `string _symbol = 'kat' [static], [private]`

Definition at line 360 of file `physical_quantity.py`.

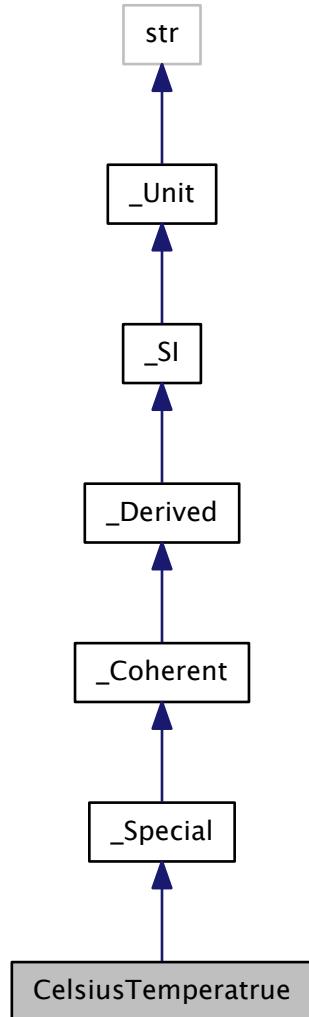
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

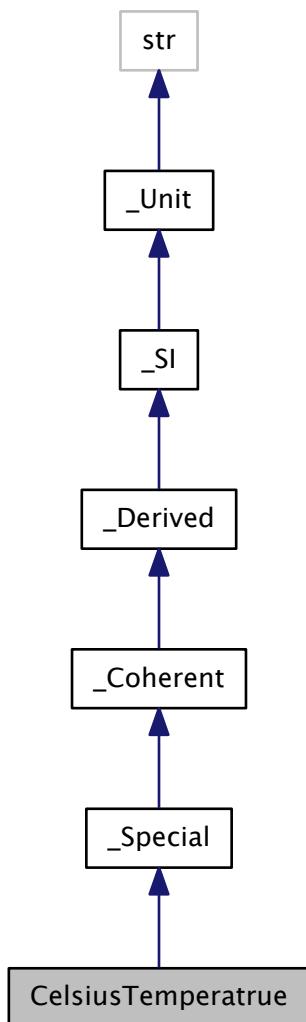
7.30 CelsiusTemperattrue Class Reference

Anders Celsius (27 November 1701 – 25 April 1744)

Inheritance diagram for CelsiusTemperattrue:



Collaboration diagram for CelsiusTemperatrue:



Static Private Attributes

- string symbol = 'degC'

Additional Inherited Members

7.30.1 Detailed Description

Anders Celsius (27 November 1701 – 25 April 1744)

Definition at line 323 of file physical_quantity.py.

7.30.2 Member Data Documentation

7.30.2.1 string _symbol = 'degC' [static], [private]

Definition at line 324 of file physical_quantity.py.

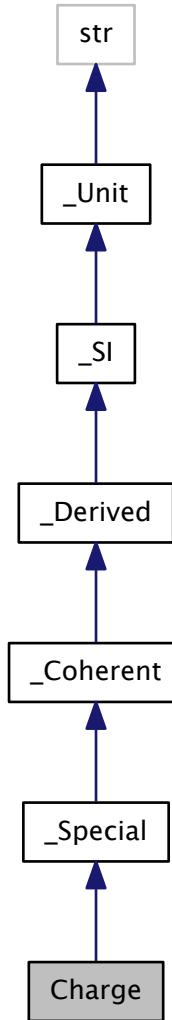
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

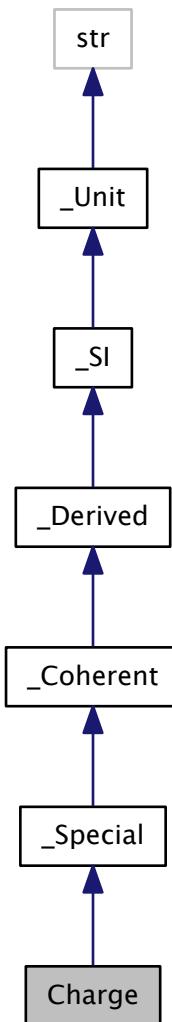
7.31 Charge Class Reference

Charles-Augustin de Coulomb (14 June 1736 – 23 August 1806)

Inheritance diagram for Charge:



Collaboration diagram for Charge:



Static Private Attributes

- string `_symbol` = 'C'

Additional Inherited Members

7.31.1 Detailed Description

Charles-Augustin de Coulomb (14 June 1736 – 23 August 1806)

Definition at line 273 of file physical_quantity.py.

7.31.2 Member Data Documentation

7.31.2.1 string _symbol = 'C' [static], [private]

Definition at line 274 of file physical_quantity.py.

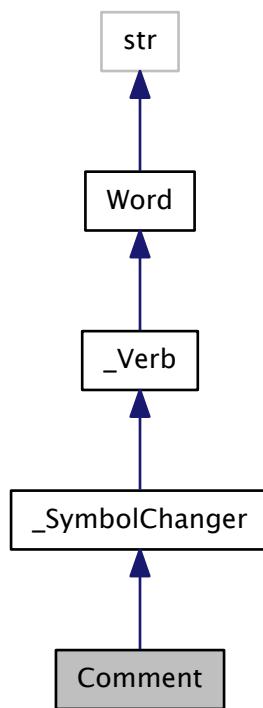
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

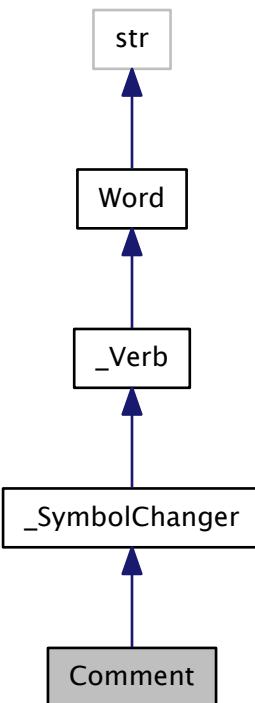
7.32 Comment Class Reference

Change [rdf.language.grammar.syntax.Grammar.comment](#) via `_SymbolChanger.act`.

Inheritance diagram for Comment:



Collaboration diagram for Comment:



Additional Inherited Members

7.32.1 Detailed Description

Change `rdf.language.grammar.syntax.Grammar.comment` via `_SymbolChangrr.act`.

Change grammar's comment attribute

Definition at line 76 of file `pragmatics.py`.

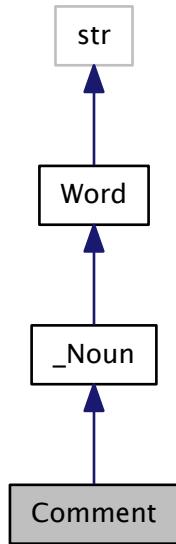
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

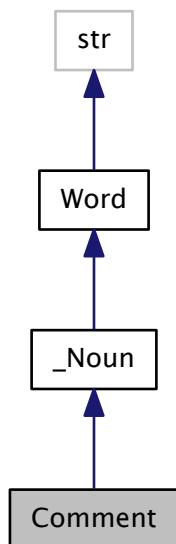
7.33 Comment Class Reference

The `Comment` Noun remembers passive comment lines.

Inheritance diagram for Comment:



Collaboration diagram for Comment:



Static Public Member Functions

- def [concrete](#)

Static Private Attributes

- [_operator_in_line](#) = False

Additional Inherited Members

7.33.1 Detailed Description

The [Comment](#) Noun remembers passive comment lines.

Definition at line 63 of file [semantics.py](#).

7.33.2 Member Function Documentation

7.33.2.1 def [concrete](#)([prosodic](#)) [static]

Return an [RDFComment](#)

Definition at line 68 of file [semantics.py](#).

```
68
69      def concrete(prosodic):
70          """Return an RDFComment"""
71          return prosodic.extract_comment()
72
```

7.33.3 Member Data Documentation

7.33.3.1 [_operator_in_line](#) = False [static], [private]

Definition at line 65 of file [semantics.py](#).

Referenced by [_Noun.is_\(\)](#).

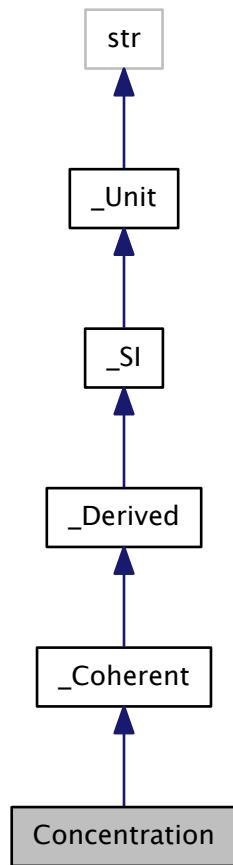
The documentation for this class was generated from the following file:

- [rdf/language/lexis/semantics.py](#)

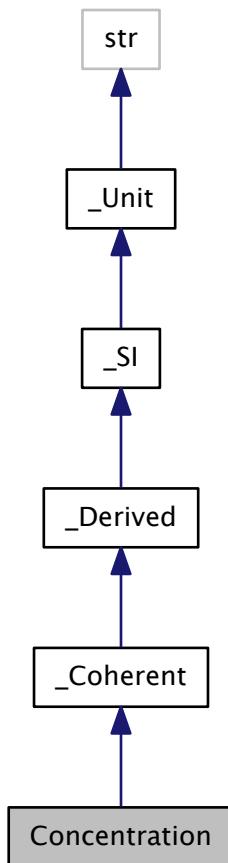
7.34 Concentration Class Reference

[concentration](#)

Inheritance diagram for Concentration:



Collaboration diagram for Concentration:



Static Private Attributes

- string `_symbol` = 'mol/m***3'

Additional Inherited Members

7.34.1 Detailed Description

concentration

Definition at line 438 of file physical_quantity.py.

7.34.2 Member Data Documentation

7.34.2.1 string `_symbol` = 'mol/m***3' [static], [private]

Definition at line 439 of file physical_quantity.py.

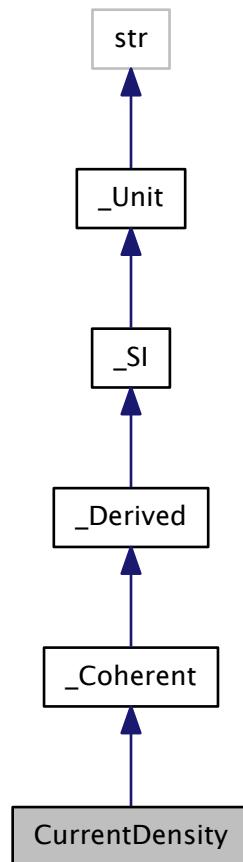
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

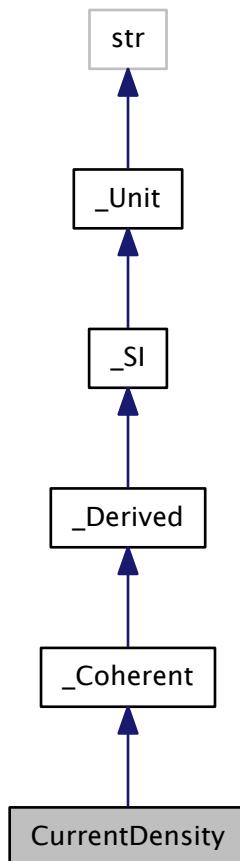
7.35 CurrentDensity Class Reference

Amps per square meter.

Inheritance diagram for CurrentDensity:



Collaboration diagram for CurrentDensity:



Static Private Attributes

- string `_symbol` = 'A/m**2'

Additional Inherited Members

7.35.1 Detailed Description

Amps per square meter.

Definition at line 420 of file `physical_quantity.py`.

7.35.2 Member Data Documentation

7.35.2.1 string `_symbol` = 'A/m**2' [static], [private]

Definition at line 421 of file `physical_quantity.py`.

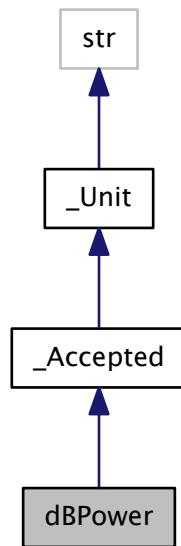
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

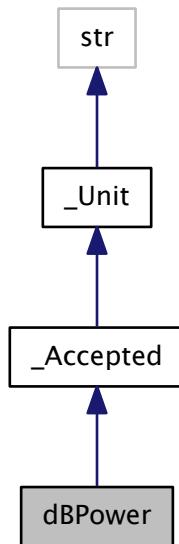
7.36 dBPower Class Reference

decibel [Power](#) -is not power- it's just a number: (Alexander Graham Bell (March 3, 1847 – August 2, 1922))

Inheritance diagram for dBPower:



Collaboration diagram for dBPower:



Static Private Attributes

- string `_symbol` = 'dbW'

Additional Inherited Members

7.36.1 Detailed Description

decibel [Power](#) -is not power- it's just a number: (Alexander Graham Bell (March 3, 1847 – August 2, 1922))

Definition at line 390 of file `physical_quantity.py`.

7.36.2 Member Data Documentation

7.36.2.1 string `_symbol` = 'dbW' [static], [private]

Definition at line 391 of file `physical_quantity.py`.

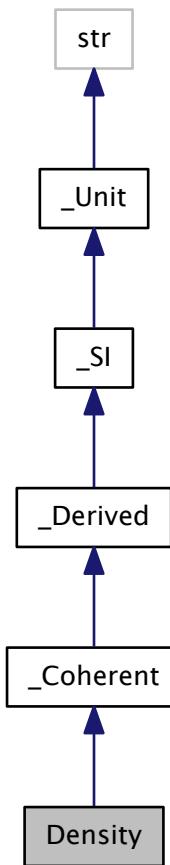
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

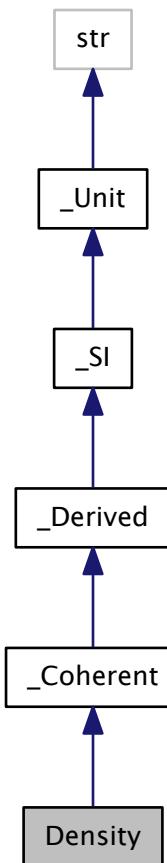
7.37 Density Class Reference

ρ

Inheritance diagram for Density:



Collaboration diagram for Density:



Static Private Attributes

- string `_symbol` = 'kg/m**3'

Additional Inherited Members

7.37.1 Detailed Description

ρ

Definition at line 408 of file physical_quantity.py.

7.37.2 Member Data Documentation

7.37.2.1 string `_symbol` = 'kg/m**3' [static], [private]

Definition at line 409 of file physical_quantity.py.

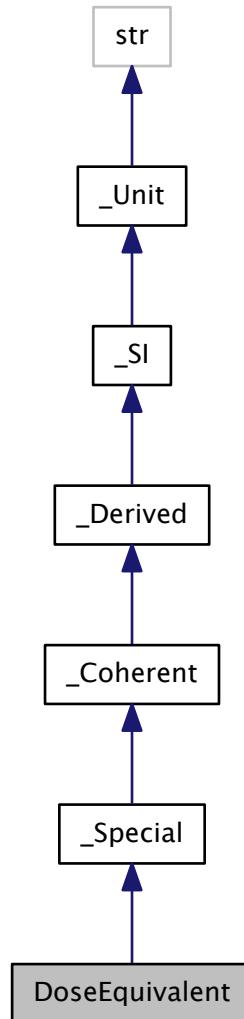
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

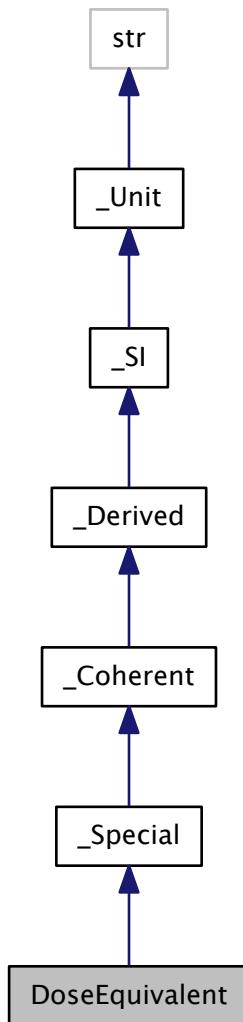
7.38 DoseEquivalent Class Reference

Professor Rolf Maximilian Sievert (6 May 1896 – 3 October 1966)

Inheritance diagram for DoseEquivalent:



Collaboration diagram for DoseEquivalent:



Static Private Attributes

- string `_symbol` = 'Sv'

Additional Inherited Members

7.38.1 Detailed Description

Professor Rolf Maximilian Sievert (6 May 1896 – 3 October 1966)

Definition at line 353 of file physical_quantity.py.

7.38.2 Member Data Documentation

7.38.2.1 `string _symbol = 'Sv' [static], [private]`

Definition at line 354 of file `physical_quantity.py`.

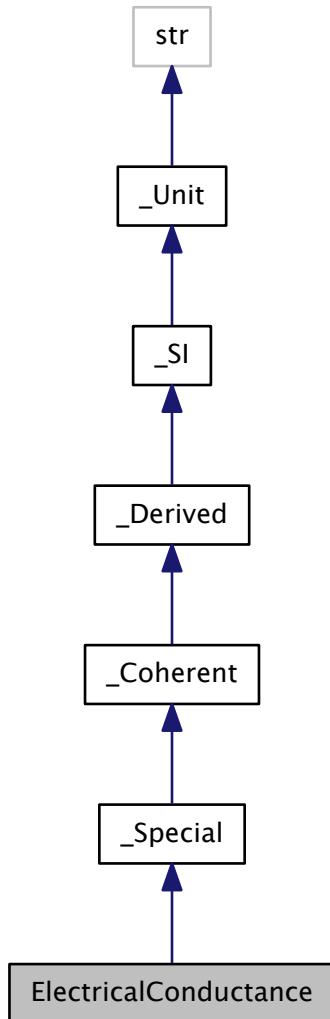
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

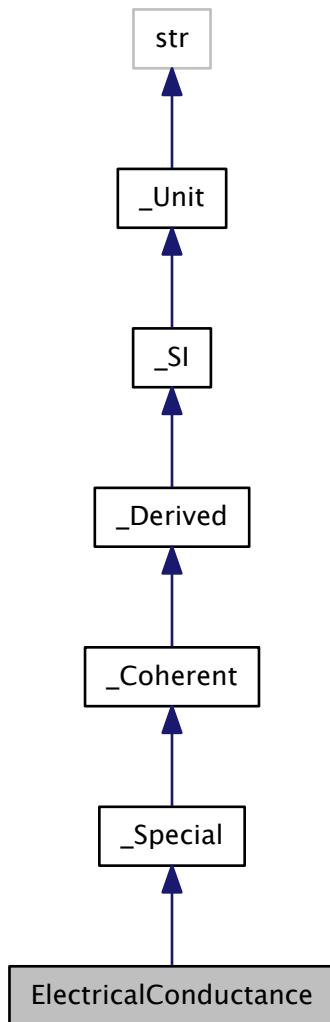
7.39 ElectricalConductance Class Reference

Ernst Werner Siemens, von Siemens since 1888, (13 December 1816 – 6 December 1892)

Inheritance diagram for ElectricalConductance:



Collaboration diagram for ElectricalConductance:



Static Private Attributes

- string `_symbol` = 'S'

Additional Inherited Members

7.39.1 Detailed Description

Ernst Werner Siemens, von Siemens since 1888, (13 December 1816 – 6 December 1892)

Definition at line 299 of file `physical_quantity.py`.

7.39.2 Member Data Documentation

7.39.2.1 `string _symbol = 'S'` [static], [private]

Definition at line 300 of file `physical_quantity.py`.

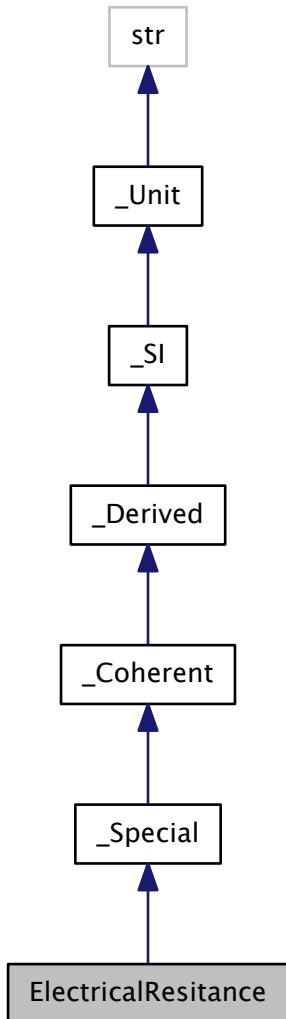
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

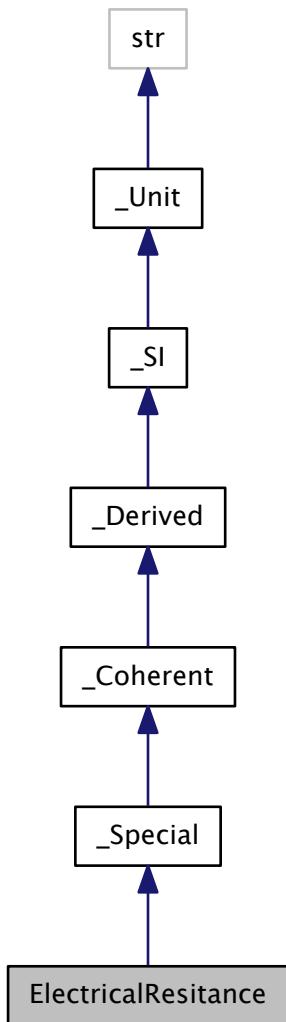
7.40 ElectricalResistance Class Reference

Georg Simon Ohm (16 March 1789 – 6 July 1854)

Inheritance diagram for ElectricalResistance:



Collaboration diagram for ElectricalResistance:



Static Private Attributes

- string `_symbol` = 'ohm'

Additional Inherited Members

7.40.1 Detailed Description

Georg Simon Ohm (16 March 1789 – 6 July 1854)

Definition at line 292 of file `physical_quantity.py`.

7.40.2 Member Data Documentation

7.40.2.1 string _symbol = 'ohm' [static], [private]

Definition at line 293 of file physical_quantity.py.

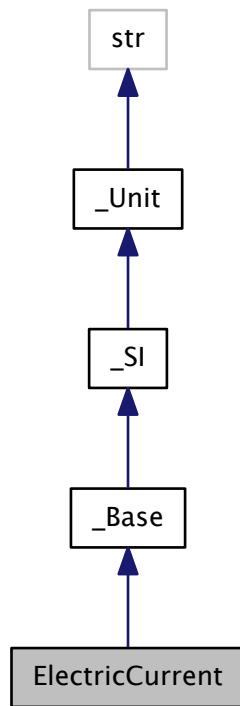
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

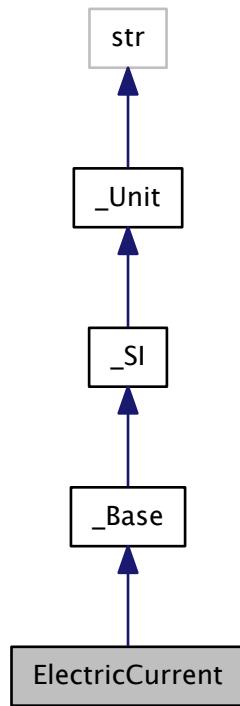
7.41 ElectricCurrent Class Reference

Andre-Marie Ampère (22 January 1775 – 10 June 1836)

Inheritance diagram for ElectricCurrent:



Collaboration diagram for ElectricCurrent:



Static Private Attributes

- string `_symbol` = 'amp'

Additional Inherited Members

7.41.1 Detailed Description

Andre-Marie Ampère (22 January 1775 – 10 June 1836)

Ampere

Definition at line 204 of file `physical_quantity.py`.

7.41.2 Member Data Documentation

7.41.2.1 string `_symbol` = 'amp' [static], [private]

Definition at line 206 of file `physical_quantity.py`.

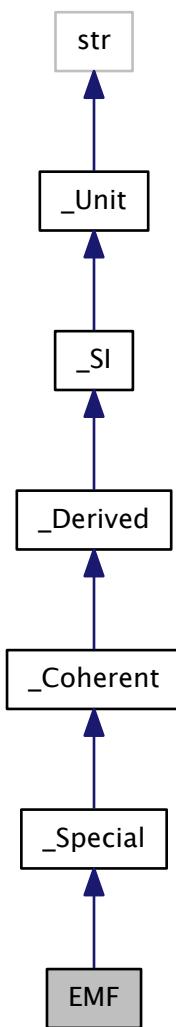
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

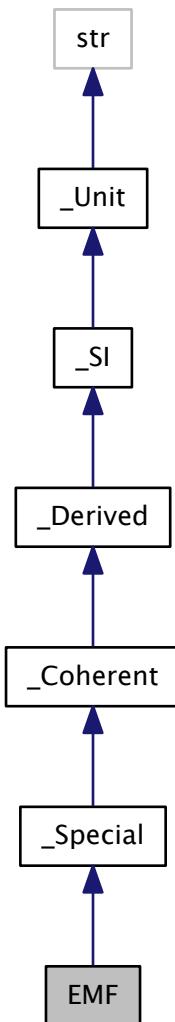
7.42 EMF Class Reference

Alessandro Giuseppe Antonio Anastasio Volta (18 February 1745 – 5 March 1827)

Inheritance diagram for EMF:



Collaboration diagram for EMF:



Static Private Attributes

- string `_symbol` = 'V'

Additional Inherited Members

7.42.1 Detailed Description

Alessandro Giuseppe Antonio Anastasio Volta (18 February 1745 – 5 March 1827)

Definition at line 280 of file `physical_quantity.py`.

7.42.2 Member Data Documentation

7.42.2.1 string _symbol = 'V' [static], [private]

Definition at line 281 of file physical_quantity.py.

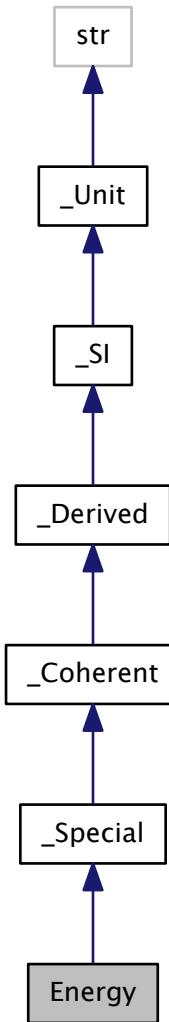
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

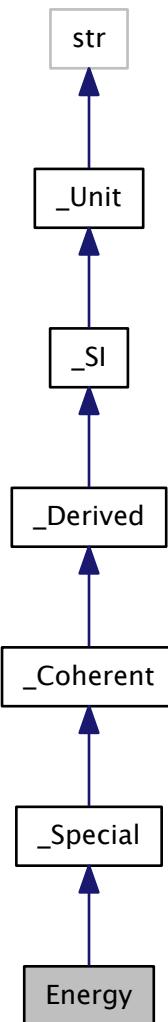
7.43 Energy Class Reference

James Prescott Joule FRS (24 December 1818 – 11 October 1889)

Inheritance diagram for Energy:



Collaboration diagram for Energy:



Static Private Attributes

- string `_symbol` = 'J'

Additional Inherited Members

7.43.1 Detailed Description

James Prescott Joule FRS (24 December 1818 – 11 October 1889)

Definition at line 261 of file `physical_quantity.py`.

7.43.2 Member Data Documentation

7.43.2.1 `string _symbol = 'J' [static], [private]`

Definition at line 262 of file `physical_quantity.py`.

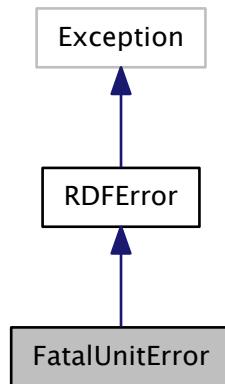
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

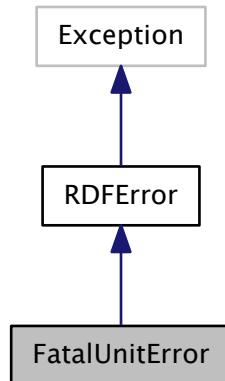
7.44 FatalUnitError Class Reference

Fatal error for unknown units (not really an acceptable idea)

Inheritance diagram for FatalUnitError:



Collaboration diagram for FatalUnitError:



7.44.1 Detailed Description

Fatal error for unknown units (not really an acceptable idea)

```
raise for unrecognized units (fatally)
```

Definition at line 7 of file errors.py.

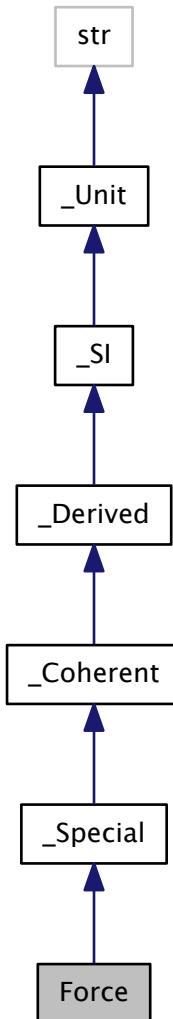
The documentation for this class was generated from the following file:

- [rdf/units/errors.py](#)

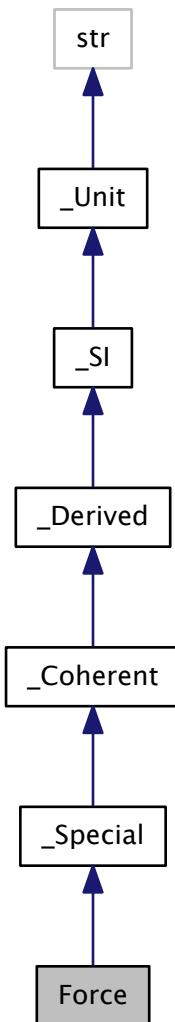
7.45 Force Class Reference

Sir Isaac Newton PRS MP (25 December 1642 – 20 March 1727)

Inheritance diagram for Force:



Collaboration diagram for Force:



Static Private Attributes

- string `_symbol` = 'N'

Additional Inherited Members

7.45.1 Detailed Description

Sir Isaac Newton PRS MP (25 December 1642 – 20 March 1727)

Definition at line 255 of file physical_quantity.py.

7.45.2 Member Data Documentation

7.45.2.1 string _symbol = 'N' [static], [private]

Definition at line 256 of file physical_quantity.py.

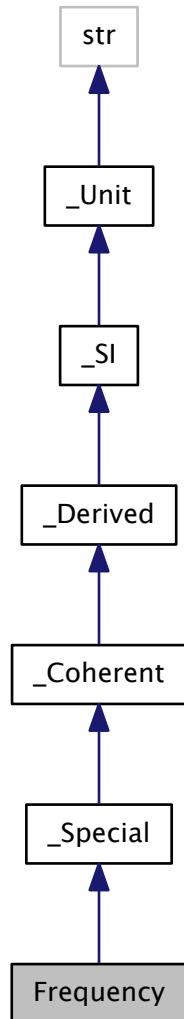
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

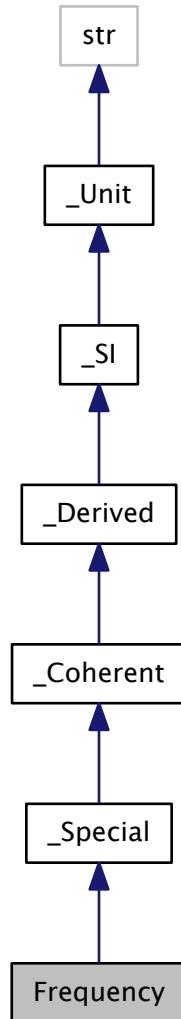
7.46 Frequency Class Reference

Heinrich Rudolf Hertz (22 February 1857 – 1 January 1894)

Inheritance diagram for Frequency:



Collaboration diagram for Frequency:



Static Private Attributes

- string `_symbol` = 'Hz'

Additional Inherited Members

7.46.1 Detailed Description

Heinrich Rudolf Hertz (22 February 1857 – 1 January 1894)

Definition at line 249 of file `physical_quantity.py`.

7.46.2 Member Data Documentation

7.46.2.1 string _symbol = 'Hz' [static], [private]

Definition at line 250 of file physical_quantity.py.

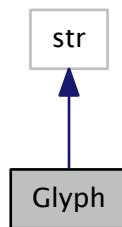
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

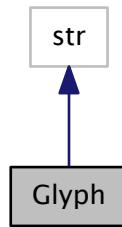
7.47 Glyph Class Reference

A character that knows how to find itself in strings.

Inheritance diagram for Glyph:



Collaboration diagram for Glyph:



Public Member Functions

- def [__call__](#)
 split line on self
- def [__pos__](#)
 $(+glyph)(line) \rightarrow glyph.right(line)$
- def [__neg__](#)
 $(-glyph)(line) \rightarrow glyph.left(line)$

Private Member Functions

- def `_left`
Get line left of self.
- def `_right`
Get line right of self.

7.47.1 Detailed Description

A character that knows how to find itself in strings.

A Glyph is a str sub-class that can be called to figure itseld out.
For example:

```
>>> line = "unit = equal"
>>> g      = Glyph("=")
```

You get:

OP	VALUE	Comments
g(line)	['unit', 'equal']	split string on glyph
(+g)(line)	'equal'	right side string
(-g)(line)	'unit'	left side string

Definition at line 25 of file punctuation.py.

7.47.2 Member Function Documentation

7.47.2.1 def __call__(self, line)

split line on self

Parameters

<code>line</code>	A line
-------------------	--------

Returns

(left, right) side of line (with possible null str on right)

Definition at line 43 of file punctuation.py.

```
43
44     def __call__(self, line):
45         try:
46             index = line.index(self)
47         except ValueError:
48             left, right = line, ""
49         else:
50             left = line[:index]
51             right = line[index+1:]
52         return map(str.strip, (left, right))
```

7.47.2.2 def __neg__(self)

(-glyph)(line) -> glyph.left(line)

-glyph --> glyph.left

Definition at line 73 of file punctuation.py.

References Glyph._left().

```

73
74     def __neg__(self):
75         """-glyph --> glyph.left"""
76         return self._left
77

```

Here is the call graph for this function:



7.47.2.3 def __pos__(self)

(+glyph)(line) -> glyph.right(line)

+glyph --> glyph.right

Definition at line 68 of file punctuation.py.

References Glyph._right().

```

68
69     def __pos__(self):
70         """+glyph --> glyph.right"""
71         return self._right

```

Here is the call graph for this function:



7.47.2.4 def _left(self, line) [private]

Get line left of self.

Parameters

<i>line</i>	A line with or without self
-------------	-----------------------------

Return values

<i>left</i>	line left of self left symbol
-------------	----------------------------------

Definition at line 56 of file punctuation.py.

Referenced by Glyph.__neg__().

```
56
57     def _left(self, line):
58         """left symbol"""
59         return self(line)[0]
```

Here is the caller graph for this function:



7.47.2.5 def _right(self, line) [private]

Get line right of self.

Parameters

<i>line</i>	A line with or without self
-------------	-----------------------------

Return values

<i>right</i>	line right of self right symbol
--------------	------------------------------------

Definition at line 63 of file punctuation.py.

Referenced by Glyph.__pos__().

```
63
64     def _right(self, line):
65         """right symbol"""
66         return self(line)[-1]
```

Here is the caller graph for this function:



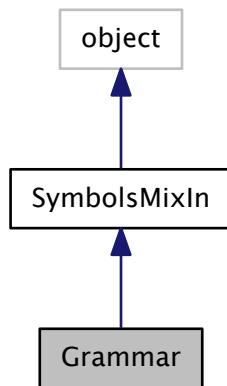
The documentation for this class was generated from the following file:

- [rdf/language/grammar/punctuation.py](#)

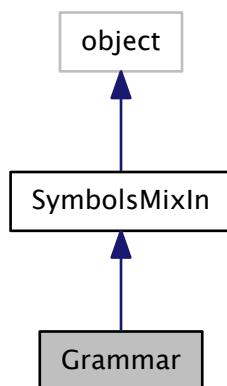
7.48 Grammar Class Reference

[Grammar](#) tracks the state of grammar, and uses it to dispatch work when it is called.

Inheritance diagram for Grammar:



Collaboration diagram for Grammar:



Public Member Functions

- def [__init__](#)

Grammar always boots into the default state, and can only be changed by processing RDF lines.

- def `operator`

Getter.
- def `operator`

Setter has mutators to ensure it is an `rdf.language.grammar.punctuation.Glyph` object.
- def `comment`

Getter.
- def `comment`

Setter has mutators to ensure it is a `rdf.language.grammar.punctuation.Glyph` object.
- def `prefix`

Getter.
- def `prefix`

Ensure `Grammar._prefix` is an `rdf.language.grammar.morpheme.Prefix`.
- def `suffix`

Getter.
- def `suffix`

Ensure `Grammar._suffix` is an `rdf.language.grammar.morpheme.Suffix`.
- def `__str__`

str reflects the current grammar state
- def `__int__`

`int()` → `Grammar.depth`
- def `__iadd__`

`+= -> change depth and append affixes w/ morpheme.Affix.descend (which knows how to do it)`
- def `__isub__`

`+= -> change depth and truncate affixes w/ morpheme.Affix.ascend (b/c grammar just implements it)`
- def `__call__`

*`Grammar(line) -> rdf.data.entries.RDFRecord`
It's the money method— not it's not a pure function- it can change the state of grammar.*
- def `__add__`

`Grammar + <str> = rdf.language.Prosodic`.
- def `set_affix`

Setitem in an AFFIX in `Grammar._appendables`.
- def `change_symbol`

Change a symbol in `Grammar._mutables`.

Public Attributes

- `depth`

The recursion depth from which the rdf lines are coming.
- `operator`

copy `rdf.reserved.SymbolsMixIn.Operator` via `Grammar.operator`
- `comment`

copy `rdf.reserved.SymbolsMixIn.Comment` via `Grammar.operator`
- `prefix`

Dynamic prefix is a copy of a list -and depends on depth.
- `suffix`

Dynamic suffix is a copy of a list -and depends on depth.

Static Public Attributes

- **wrap** = reserved.WRAP
wrap tell read how to unwrap lines— it's just a str
- **sep** = reserved.SEPARATOR
sep is not used -yet, it would appear in RDF_eval at some point.
- list **Prefix** = [""]
Static default prefix.
- list **Suffix** = [""]
Static default suffix.

Private Member Functions

- def **_process**
Process the line a Verb or a Line.

Private Attributes

- **_operator**
- **_comment**
- **_prefix**
- **_suffix**

Static Private Attributes

- **__metaclass__** = lexicon
Add lexicon at class creation.
- dictionary **_appendables** = {"prefix": "prefix", "suffix": "suffix"}
dict to prevent ill use of set_affix
- dictionary **_mutables** = {'operator': 'operator', 'comment': 'comment'}
dict to prevent ill use of change_symbol

7.48.1 Detailed Description

Grammar tracks the state of grammar, and uses it to dispatch work when it is called.

Grammar() is the state of the grammar. See __init__ for why it supports only nullary instantiation.

ALL_CAP class attributes a Pragamatic (i.e. meta) words.
_lower_case private instance attributes are punctuation Glyphs
lower_case mutator methods ensure glyphs setting is kosher.
Capitalized class attributes are default values for the lower_case version.

Overloads:

Function Emulation

```
line --> __call__(line) ---> RDFRecord #That is, grammar is a (semipure)
                                              function that makes lines into
                                              RDFRecords.

(meta)line-> __call__(line)---> None      # Pragamatic (KeyWord) lines
                                              return None (they aren't rdf
                                              records) but they change the
                                              internal state of 'self'. Hence
                                              grammar is an impure function.
```

```

other -->     __call__(line)---> None      # Comments do nothing, Errors are
                                         identified, reported to stderr
                                         and forgotten.

Integer:
int(grammar) returns the depth-- which is a non-negative integer telling
how deep the processor is in the include file tree
(IFT) Should not pass sys.getrecursionlimit().

grammar += 1  There are called when the depth_processor goes up or
grammar -= 1  down the IFT. The change int(grammar) and manage the
affixes.

```

Definition at line 53 of file syntax.py.

7.48.2 Constructor & Destructor Documentation

7.48.2.1 def __init__(self)

Grammar always boots into the default state, and can only be changed by processing RDF lines.

Nullary instantiation: you cannot inject dependencies (DI) in the constructor. You allways start with the default grammar- which is defined in static class attributes.

Only rdf Pragamatics (i.e commands or key words) can change the grammar -- infact, the attributes encapsulated in mutators.

Definition at line 114 of file syntax.py.

```

114
115     def __init__(self):
116         """Nullary instantiation: you cannot inject dependcies (DI)
117         in the constructor. You allways start with the default grammar-
118         which is defined in static class attributes.
119
120         Only rdf Pragamatics (i.e commands or key words) can change the
121         grammar -- infact, the attributes encapsulated in mutators.
122         """
123         ## The recursion depth from which the rdf lines are coming.
124         self.depth = 0
125         ## copy rdf.reserved.SymbolsMixIn.Operator via Grammar.operator
126         self.operator = self.Operator
127         ## copy rdf.reserved.SymbolsMixIn.Comment via Grammar.operator
128         self.comment = self.Comment
129         ## Dynamic prefix is a copy of a list -and depends on depth
130         self.prefix = self.Prefix[:]
131         ## Dynamic suffixx is a copy of a list -and depends on depth
132         self.suffix = self.Suffix[:]

```

7.48.3 Member Function Documentation

7.48.3.1 def __add__(self, line)

Grammar + <str> = rdf.language.Prosodic.

Definition at line 230 of file syntax.py.

```

230
231     def __add__(self, line):
232         from rdf.language import Prosodic
233         return Prosodic(line, self)

```

7.48.3.2 def __call__(self, line)

Grammar(line) -> [rdf.data.entries.RDFRecord](#)

It's the money method– not it's not a pure function- it can change the state of grammar.

```
grammar(line) --> grammar.process(line) (with error catching)
```

Definition at line 216 of file [syntax.py](#).

References [Grammar._process\(\)](#).

```
216     def __call__(self, line):
217         """grammar(line) --> grammar.process(line) (with error catching)"""
218         from rdf.units.errors import UnrecognizedUnitWarning
219         try:
220             result = self._process(line)
221         except UnrecognizedUnitWarning as err:
222             print >> sys.stderr, "WARNING:" + repr(err) + ":" + line
223             result = []
224         except errors.RDFWarning as err:
225             print >> sys.stderr, "WARNING:" + repr(err) + ":" + line
226             result = []
227         return result
228
```

Here is the call graph for this function:



7.48.3.3 def __iadd__(self, n)

`+=>` change depth and append affixes w/ [morpheme.Affix.descend](#)

(which knows how to do it)

Parameters

<code>n</code>	+1 or ValueError
----------------	------------------

Side Effects:

[Affix.desend\(\)](#)

Return values

<code>self</code>	self, changed
-------------------	---------------

Definition at line 194 of file [syntax.py](#).

References [Grammar.depth](#).

```
194     def __iadd__(self, n):
195         self.depth += int(n)
196         self.prefix.descend()
197         self.suffix.descend()
198         return self
199
```

7.48.3.4 def __int__(self)

int() -> Grammar.depth

Definition at line 184 of file syntax.py.

References Grammar.depth.

```
184     def __int__(self):
185         return self.depth
186
```

7.48.3.5 def __isub__(self, n)

+=> change depth and truncate affixes w/ morpheme.Affix.ascend (b/c grammar just implements it)

Parameters

<i>n</i>	+1 or ValueError
----------	------------------

Side Effects:

Affix.ascend()

Return values

<i>self</i>	self, changed
-------------	---------------

Definition at line 207 of file syntax.py.

References Grammar.depth.

```
207
208     def __isub__(self, n):
209         self.depth -= int(n)
210         self.prefix.ascend()
211         self.suffix.ascend()
212         return self
```

7.48.3.6 def __str__(self)

str reflects the current grammar state

Definition at line 178 of file syntax.py.

References Grammar.comment, Grammar.depth, Grammar.operator, Grammar.prefix, and Grammar.suffix.

```
178
179     def __str__(self):
180         return (str(self.depth) + " " +
181                 self.operator + " " +
182                 self.comment + " " + str(self.prefix) + str(self.
suffix))
```

7.48.3.7 def _process(self, line) [private]

Process the line a Verb or a Line.

Parameters

<i>line</i>	rdf sentence (a clitic)
-------------	-------------------------

Side Effects:

word might change self

Return values

<i>word(prosodic)</i>	The processed line
-----------------------	--------------------

Definition at line 239 of file syntax.py.

Referenced by Grammar.__call__().

```

239
240     def _process(self, line):
241         import operator
242         # combine line and grammar into 1 thing
243         prosodic = self + line
244         # Short Circuit Loop: Iterator ends when word is found,
245         # then break out of loop with an return result, this is "IFless":
246         for word in itertools.dropwhile(operator.methodcaller("is_not",
247                                         prosodic),
248                                         itertools.chain(self._VERBS,
249                                         self._NOUNS)):
250             break
251         return word(prosodic)

```

Here is the caller graph for this function:

**7.48.3.8 def change_symbol(self, attr, value)**

Change a symbol in Grammar._mutables.

```

change_symbol(self, attr, value)
attr in (operator, comment)
value = , # ! ... whatevs.

```

Definition at line 265 of file syntax.py.

References Grammar._mutables.

```

265
266     def change_symbol(self, attr, value):
267         """change_symbol(self, attr, value)
268         attr in (operator, comment)
269         value = , # ! ... whatevs."""
270         try:
271             setattr(self, self._mutables[attr], value)
272         except KeyError:
273             raise ValueError("Can't change %s with change_symbol" %
274                           repr(attr))

```

7.48.3.9 def comment(self)

Getter.

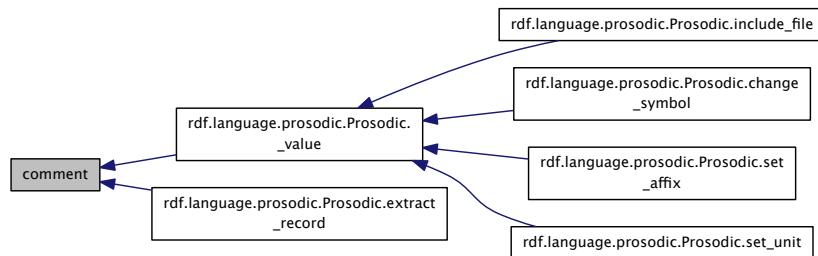
Definition at line 147 of file syntax.py.

References Grammar._comment, and Grammar.comment.

Referenced by Prosodic._value(), and Prosodic.extract_record().

```
147      def comment(self):
148          return self._comment
149
```

Here is the caller graph for this function:



7.48.3.10 def comment(self, value)

Setter has mutators to ensure it is a `rdf.language.grammar.punctuation.Glyph` object.

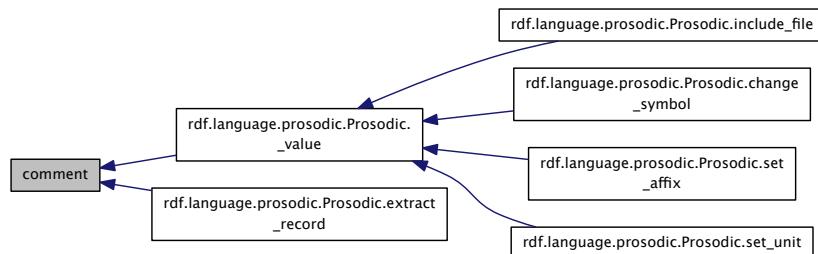
Definition at line 154 of file syntax.py.

References Grammar.comment.

Referenced by Prosodic._value(), and Prosodic.extract_record().

```
154      def comment(self, value):
155          self._comment = punctuation.Glyph(value)
156
```

Here is the caller graph for this function:



7.48.3.11 def operator(self)

Getter.

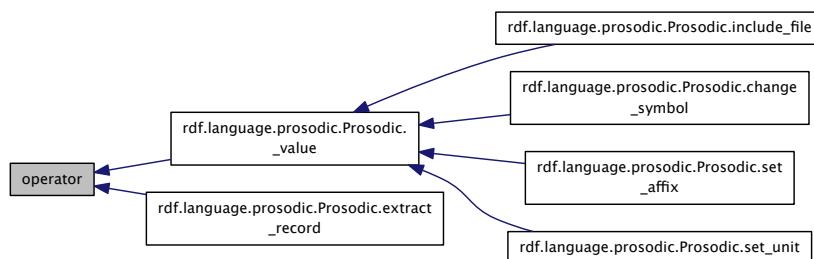
Definition at line 135 of file syntax.py.

References Grammar._operator, and Grammar.operator.

Referenced by Prosodic._value(), and Prosodic.extract_record().

```
135     def operator(self):
136         return self._operator
137
```

Here is the caller graph for this function:



7.48.3.12 def operator(self, value)

Setter has mutators to ensure it is an [rdf.language.grammar.punctuation.Glyph](#) object.

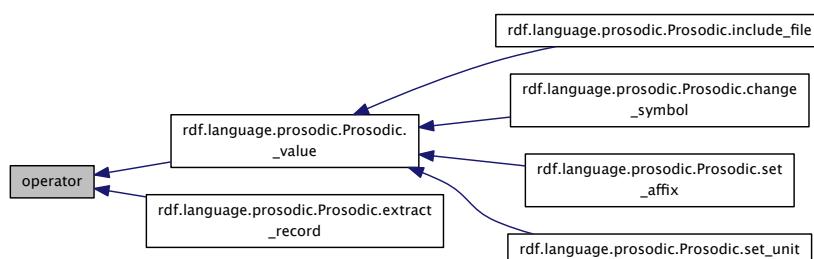
Definition at line 142 of file syntax.py.

References Grammar.operator.

Referenced by Prosodic._value(), and Prosodic.extract_record().

```
142
143     def operator(self, value):
144         self._operator = punctuation.Glyph(value)
```

Here is the caller graph for this function:



7.48.3.13 def prefix (self)

Getter.

Definition at line 159 of file syntax.py.

References Grammar._prefix, and Grammar.prefix.

```
159
160     def prefix(self):
161         return self._prefix
```

7.48.3.14 def prefix (self, value)

Ensure Grammar._prefix is an [rdf.language.grammar.morpheme.Prefix](#).

Definition at line 164 of file syntax.py.

References Grammar.prefix.

```
164
165     def prefix(self, value):
166         self._prefix = morpheme.Prefix(value)
```

7.48.3.15 def set_affix (self, classname, value)

Setitem in an AFFIX in Grammar._appendables.

```
set_affix(self, classname, value)
classname in (prefix, suffix)
value = affix at depth int(self)
```

Definition at line 253 of file syntax.py.

References Grammar._appendables.

```
253
254     def set_affix(self, classname, value):
255         """set_affix(self, classname, value)
256         classname in (prefix, suffix)
257         value = affix at depth int(self)
258         """
259         try:
260             setattr(self, self._appendables[classname])[int(self)] = value
261         except KeyError:
262             raise ValueError("Can't set %s with affix protocol" %
263                             repr(classname))
```

7.48.3.16 def suffix (self)

Getter.

Definition at line 169 of file syntax.py.

References Grammar._suffix, and Grammar.suffix.

```
169
170     def suffix(self):
171         return self._suffix
```

7.48.3.17 def suffix (self, value)

Ensure Grammar._suffix is an [rdf.language.grammar.morpheme.Suffix](#).

Definition at line 174 of file syntax.py.

References Grammar.suffix.

```
174
175     def suffix(self, value):
176         self._suffix = morpheme.Suffix(value)
```

7.48.4 Member Data Documentation

7.48.4.1 __metaclass__ = lexicon [static], [private]

Add lexicon at class creation.

Definition at line 92 of file syntax.py.

7.48.4.2 dictionary _appendables = {"prefix": "prefix", "suffix": "suffix"} [static], [private]

dict to prevent ill use of set_affix

Definition at line 107 of file syntax.py.

Referenced by Grammar.set_affix().

7.48.4.3 _comment [private]

Definition at line 155 of file syntax.py.

Referenced by Grammar.comment().

7.48.4.4 dictionary _mutables = {'operator': 'operator', 'comment': 'comment'} [static], [private]

dict to prevent ill use of change_symbol

Definition at line 110 of file syntax.py.

Referenced by Grammar.change_symbol().

7.48.4.5 _operator [private]

Definition at line 143 of file syntax.py.

Referenced by Grammar.operator().

7.48.4.6 _prefix [private]

Definition at line 165 of file syntax.py.

Referenced by Prosodic.make_affix(), and Grammar.prefix().

7.48.4.7 _suffix [private]

Definition at line 175 of file syntax.py.

Referenced by Prosodic.make_affix(), and Grammar.suffix().

7.48.4.8 comment

copy rdf.reserved.SymbolsMixIn.Comment via [Grammar.operator](#)

Definition at line 127 of file syntax.py.

Referenced by Grammar.__str__(), Prosodic.__value(), Grammar.comment(), and Prosodic.extract_record().

7.48.4.9 depth

The recursion depth from which the rdf lines are coming.

Definition at line 123 of file syntax.py.

Referenced by Grammar.__iadd__(), Grammar.__int__(), Grammar.__isub__(), and Grammar.__str__().

7.48.4.10 operator

copy [rdf.reserved.SymbolsMixIn.Operator](#) via Grammar.operator

Definition at line 125 of file syntax.py.

Referenced by Grammar.__str__(), Prosodic._value(), Prosodic.extract_record(), and Grammar.operator().

7.48.4.11 list Prefix = [""] [static]

Static default prefix.

Definition at line 101 of file syntax.py.

7.48.4.12 prefix

Dynamic prefix is a copy of a list -and depends on depth.

Definition at line 129 of file syntax.py.

Referenced by Grammar.__str__(), and Grammar.prefix().

7.48.4.13 sep = reserved.SEPARATOR [static]

sep is not used -yet, it would appear in RDF._eval at some point.

Definition at line 98 of file syntax.py.

7.48.4.14 list Suffix = [""] [static]

Static default suffix.

Definition at line 104 of file syntax.py.

7.48.4.15 suffix

Dynamic suffixx is a copy of a list -and depends on depth.

Definition at line 131 of file syntax.py.

Referenced by Grammar.__str__(), and Grammar.suffix().

7.48.4.16 wrap = reserved.WRAP [static]

wrap tell read how to unwrap lines– it's just a str

Definition at line 95 of file syntax.py.

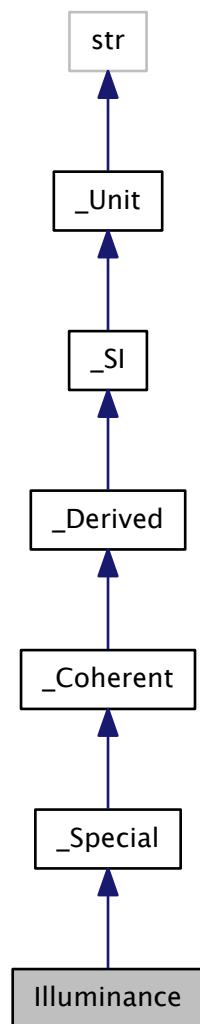
The documentation for this class was generated from the following file:

- [rdf/language/grammar/syntax.py](#)

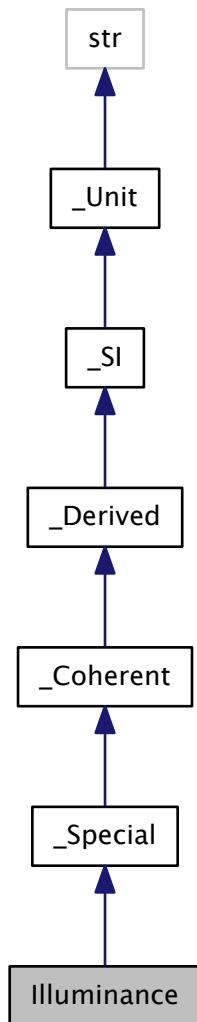
7.49 Illuminance Class Reference

lux

Inheritance diagram for Illuminance:



Collaboration diagram for Illuminance:



Static Private Attributes

- string `_symbol` = 'lx'

Additional Inherited Members

7.49.1 Detailed Description

lux

Definition at line 335 of file `physical_quantity.py`.

7.49.2 Member Data Documentation

7.49.2.1 `string _symbol = 'lx' [static], [private]`

Definition at line 336 of file `physical_quantity.py`.

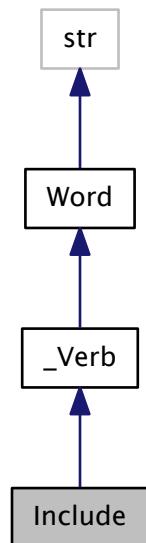
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

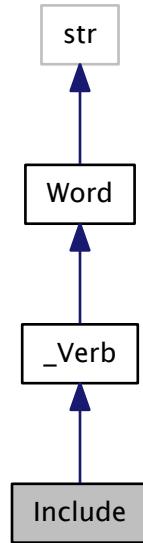
7.50 Include Class Reference

Open an [Include](#) File.

Inheritance diagram for Include:



Collaboration diagram for Include:



Public Member Functions

- def `act`
`Include` via `rdf.language.prosodic.include_file`.

7.50.1 Detailed Description

Open an `Include` File.

Verb can identify the INCLUDE lines

Definition at line 47 of file pragmatics.py.

7.50.2 Member Function Documentation

7.50.2.1 def `act(self, prosodic)`

`Include` via `rdf.language.prosodic.include_file`.

return contents of file as an interable full of RDFRecords

Definition at line 50 of file pragmatics.py.

```

50
51     def act(self, prosodic):
52         """return contents of file as an interable full of RDFRecords"""
53         return prosodic.include_file()
54
  
```

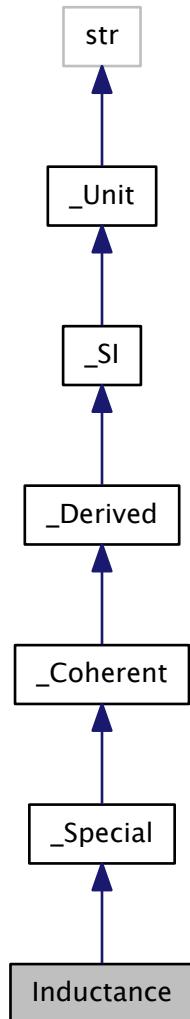
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

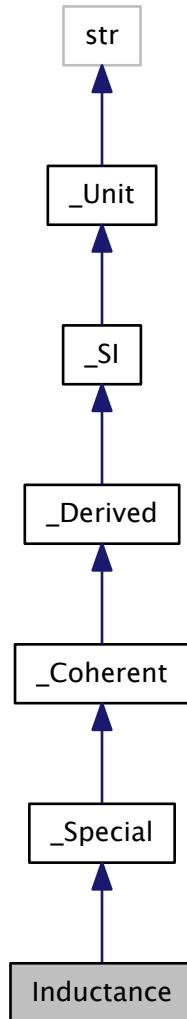
7.51 Inductance Class Reference

Joseph Henry (December 17, 1797 – May 13, 1878)

Inheritance diagram for Inductance:



Collaboration diagram for Inductance:



Static Private Attributes

- string `_symbol` = 'H'

Additional Inherited Members

7.51.1 Detailed Description

Joseph Henry (December 17, 1797 – May 13, 1878)

Definition at line 317 of file `physical_quantity.py`.

7.51.2 Member Data Documentation

7.51.2.1 `string _symbol = 'H' [static], [private]`

Definition at line 318 of file `physical_quantity.py`.

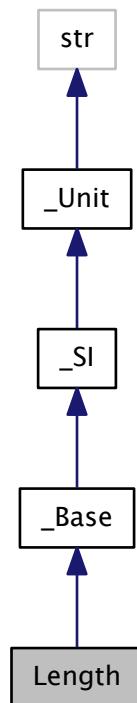
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

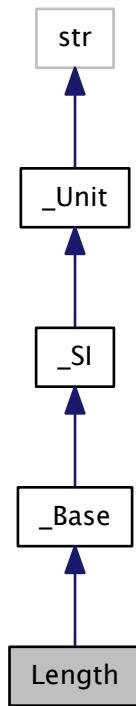
7.52 Length Class Reference

[Length](#) conversion to meters.

Inheritance diagram for Length:



Collaboration diagram for Length:



Static Private Attributes

- string [_symbol](#) = 'm'

Additional Inherited Members

7.52.1 Detailed Description

[Length](#) conversion to meters.

Meter

Definition at line 183 of file [physical_quantity.py](#).

7.52.2 Member Data Documentation

7.52.2.1 string [_symbol](#) = 'm' [static], [private]

Definition at line 185 of file [physical_quantity.py](#).

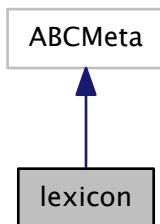
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

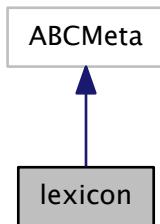
7.53 lexicon Class Reference

Metaclass for [Grammar](#) gets defines the pragamatics and semantics at load-time" pragmatics.Verbs instances are assigned according to the rdf.reserved.words.KEYWORDS, and the symantics.Noun instances are created- these are needed by Grammar.process.

Inheritance diagram for lexicon:



Collaboration diagram for lexicon:



Public Member Functions

- def [__new__](#)

Create class and add pragmatics and semantics.

7.53.1 Detailed Description

Metaclass for [Grammar](#) gets defines the pragamatics and semantics at load-time" pragmatics.Verbs instances are assigned according to the rdf.reserved.words.KEYWORDS, and the symantics.Noun instances are created- these are needed by Grammar.process.

lexicon meta class deal with the keywords defined in verbs.

Definition at line 37 of file [syntax.py](#).

7.53.2 Member Function Documentation

7.53.2.1 def __new__(mcs, args, kwargs)

Create class and add pragmatics and semantics.

Definition at line 42 of file [syntax.py](#).

```
42
43     def __new__(mcs, *args, **kwargs):
44         cls = type.__new__(mcs, *args, **kwargs)
45         ## Set up verbs
46         cls._VERBS = tuple(map(apply, pragmatics.VERBS))
47         ## Set up Noun instances by instantiating NOUNS's classes
48         cls._NOUNS = tuple(map(apply, semantics.NOUNS))
49         return cls
50
```

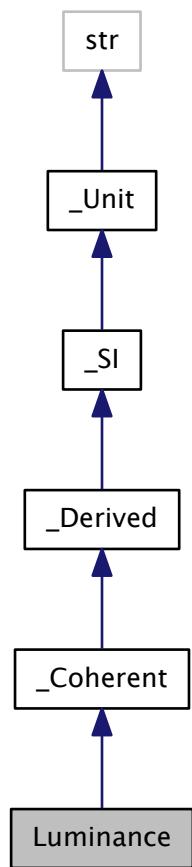
The documentation for this class was generated from the following file:

- [rdf/language/grammar/syntax.py](#)

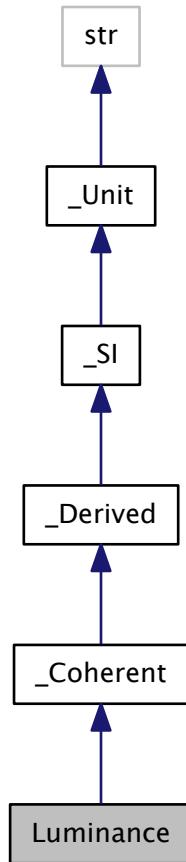
7.54 Luminance Class Reference

cd/m**2

Inheritance diagram for Luminance:



Collaboration diagram for Luminance:



Static Private Attributes

- string `_symbol` = 'cd/m**2'

Additional Inherited Members

7.54.1 Detailed Description

`cd/m**2`

Definition at line 432 of file `physical_quantity.py`.

7.54.2 Member Data Documentation

7.54.2.1 string `_symbol` = 'cd/m**2' [static], [private]

Definition at line 433 of file `physical_quantity.py`.

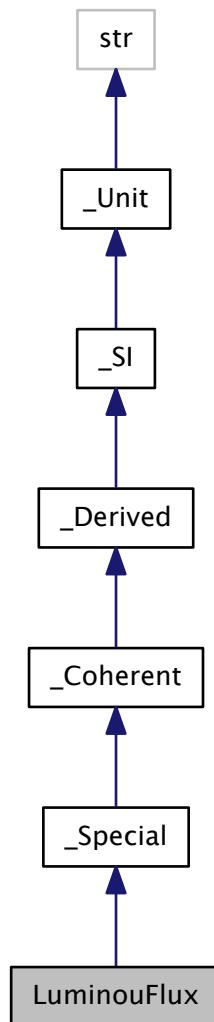
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

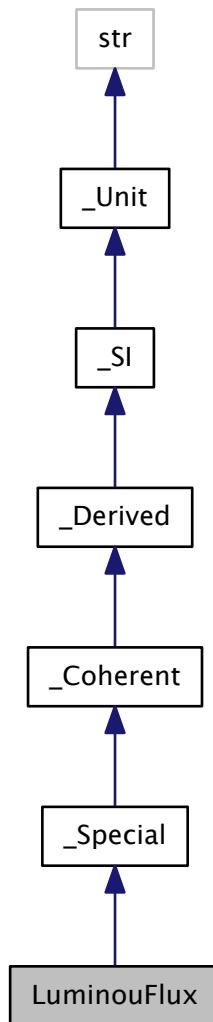
7.55 LuminouFlux Class Reference

lumen

Inheritance diagram for LuminouFlux:



Collaboration diagram for LuminouFlux:



Static Private Attributes

- string `_symbol` = 'lm'

Additional Inherited Members

7.55.1 Detailed Description

lumen

Definition at line 329 of file `physical_quantity.py`.

7.55.2 Member Data Documentation

7.55.2.1 `string _symbol = 'lm' [static], [private]`

Definition at line 330 of file `physical_quantity.py`.

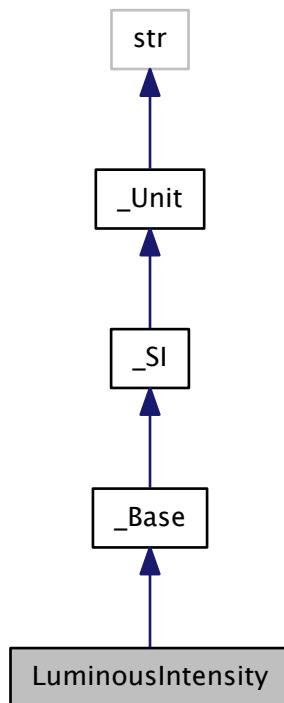
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

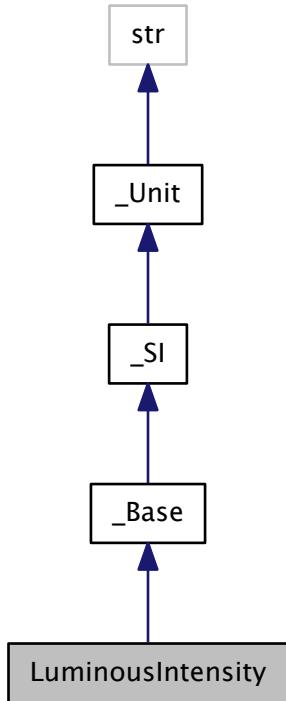
7.56 LuminousIntensity Class Reference

Brightness.

Inheritance diagram for LuminousIntensity:



Collaboration diagram for LuminousIntensity:



Static Private Attributes

- string `_symbol` = "cd"

Additional Inherited Members

7.56.1 Detailed Description

Brightness.

Definition at line 225 of file `physical_quantity.py`.

7.56.2 Member Data Documentation

7.56.2.1 string `_symbol` = "cd" [static], [private]

Definition at line 226 of file `physical_quantity.py`.

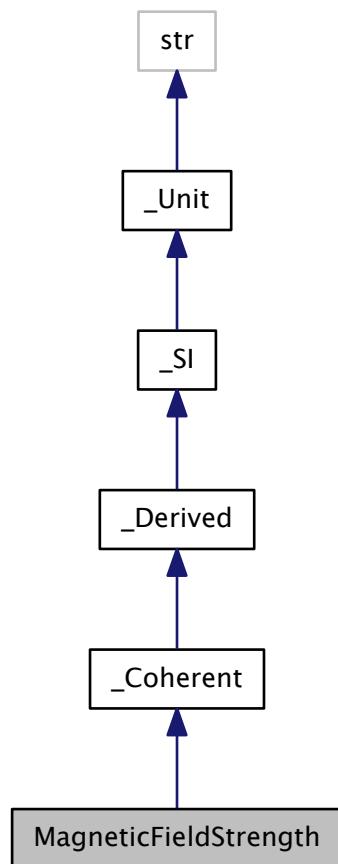
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

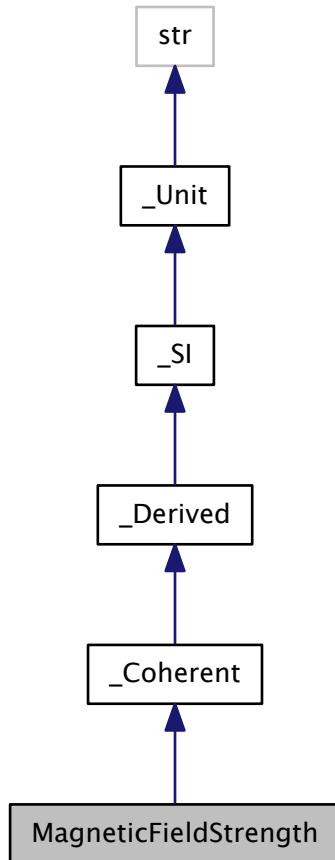
7.57 MagneticFieldStrength Class Reference

A/m.

Inheritance diagram for MagneticFieldStrength:



Collaboration diagram for MagneticFieldStrength:



Static Private Attributes

- string `_symbol` = 'A/m'

Additional Inherited Members

7.57.1 Detailed Description

A/m.

Definition at line 426 of file `physical_quantity.py`.

7.57.2 Member Data Documentation

7.57.2.1 string `_symbol` = 'A/m' [static], [private]

Definition at line 427 of file `physical_quantity.py`.

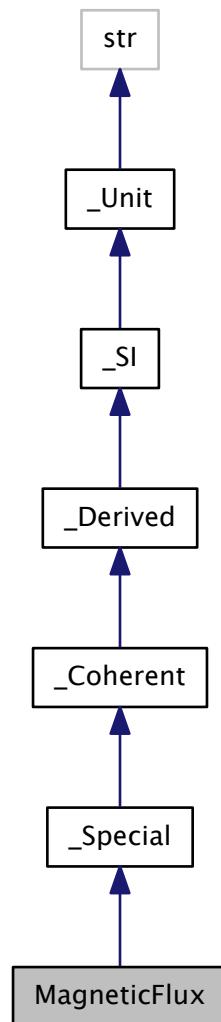
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

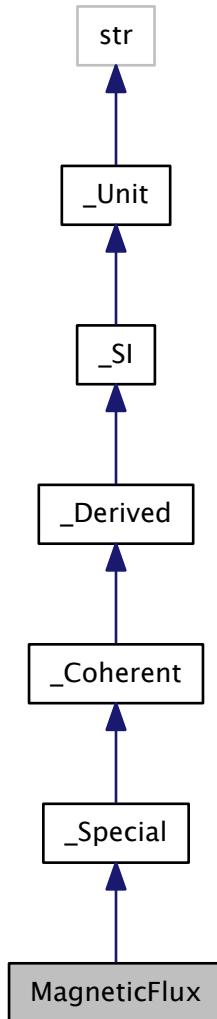
7.58 MagneticFlux Class Reference

Wilhelm Eduard Weber (24 October 1804 – 23 June 1891)

Inheritance diagram for MagneticFlux:



Collaboration diagram for MagneticFlux:



Static Private Attributes

- string `_symbol` = 'Wb'

Additional Inherited Members

7.58.1 Detailed Description

Wilhelm Eduard Weber (24 October 1804 – 23 June 1891)

Definition at line 305 of file `physical_quantity.py`.

7.58.2 Member Data Documentation

7.58.2.1 string _symbol = 'Wb' [static], [private]

Definition at line 306 of file physical_quantity.py.

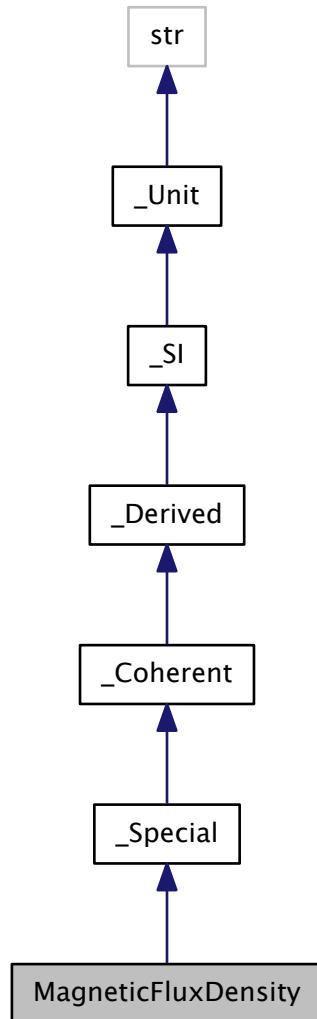
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

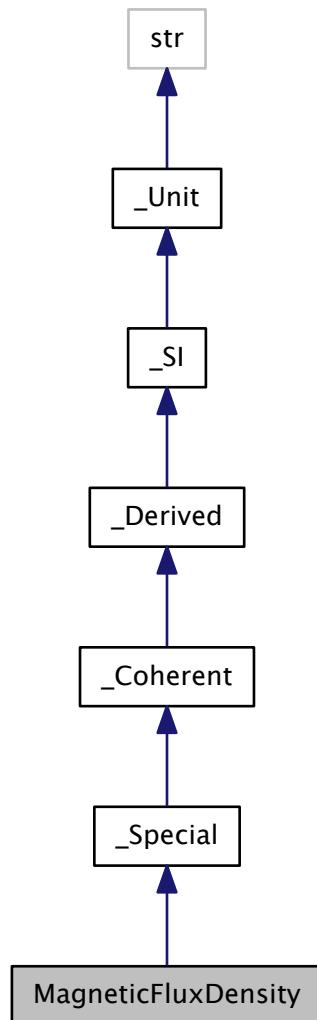
7.59 MagneticFluxDensity Class Reference

Nikola Tesla (10 July 1856 – 7 January 1943)

Inheritance diagram for MagneticFluxDensity:



Collaboration diagram for MagneticFluxDensity:



Static Private Attributes

- string `_symbol` = 'T'

Additional Inherited Members

7.59.1 Detailed Description

Nikola Tesla (10 July 1856 – 7 January 1943)

Definition at line 311 of file `physical_quantity.py`.

7.59.2 Member Data Documentation

7.59.2.1 `string _symbol = 'T' [static], [private]`

Definition at line 312 of file `physical_quantity.py`.

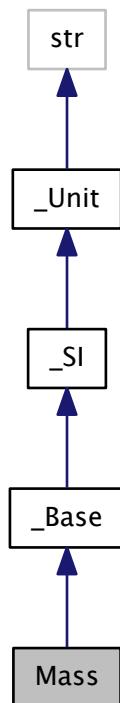
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

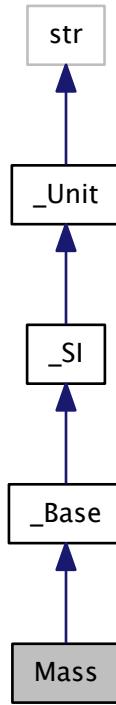
7.60 Mass Class Reference

Conversion to kilograms.

Inheritance diagram for Mass:



Collaboration diagram for Mass:



Static Private Attributes

- string [_symbol](#) = 'kg'

Additional Inherited Members

7.60.1 Detailed Description

Conversion to kilograms.

Kilogram

Definition at line 190 of file `physical_quantity.py`.

7.60.2 Member Data Documentation

7.60.2.1 string [_symbol](#) = 'kg' [static], [private]

Definition at line 192 of file `physical_quantity.py`.

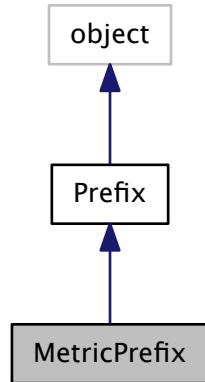
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

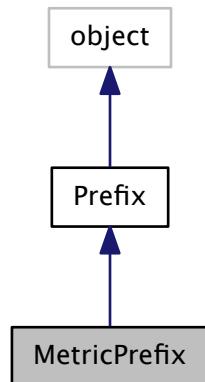
7.61 MetricPrefix Class Reference

[Metric Prefix](#).

Inheritance diagram for MetricPrefix:



Collaboration diagram for MetricPrefix:



Public Member Functions

- def [__float__](#)
cast() returns a function that calls this
- def [cast](#)
float

Static Public Attributes

- int `base` = 10

Metric is a Perfect 10.

Additional Inherited Members

7.61.1 Detailed Description

`Metric` Prefix.

Prefix based on 10

Definition at line 78 of file `prefix.py`.

7.61.2 Member Function Documentation

7.61.2.1 def __float__(self)

`cast()` returns a function that calls this

Definition at line 85 of file `prefix.py`.

References `Prefix.factor`.

```
85
86     def __float__(self):
87         return float(self.factor)
```

7.61.2.2 def cast(self)

float

Definition at line 89 of file `prefix.py`.

```
89
90     def cast(self):
91         return float
92
```

7.61.3 Member Data Documentation

7.61.3.1 int base = 10 [static]

Metric is a Perfect 10.

Definition at line 82 of file `prefix.py`.

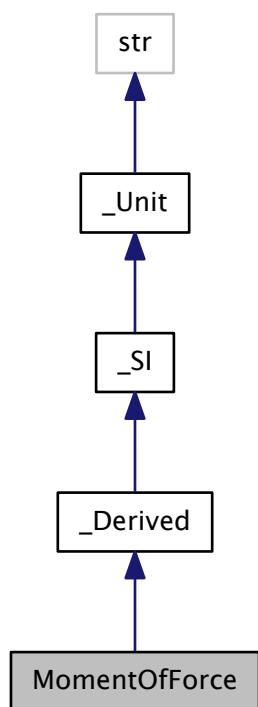
The documentation for this class was generated from the following file:

- [rdf/units/`prefix.py`](#)

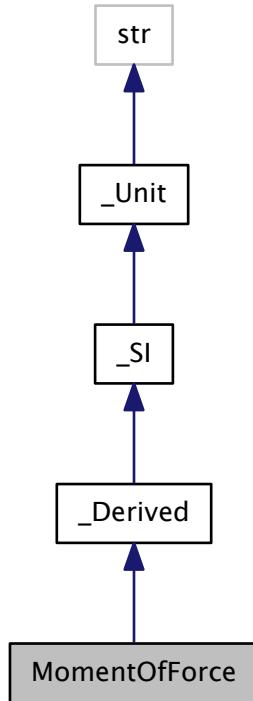
7.62 MomentOfForce Class Reference

Torque.

Inheritance diagram for MomentOfForce:



Collaboration diagram for MomentOfForce:



Static Private Attributes

- string `_symbol` = 'N*m'

Additional Inherited Members

7.62.1 Detailed Description

Torque.

Definition at line 383 of file `physical_quantity.py`.

7.62.2 Member Data Documentation

7.62.2.1 string `_symbol` = 'N*m' [static], [private]

Definition at line 384 of file `physical_quantity.py`.

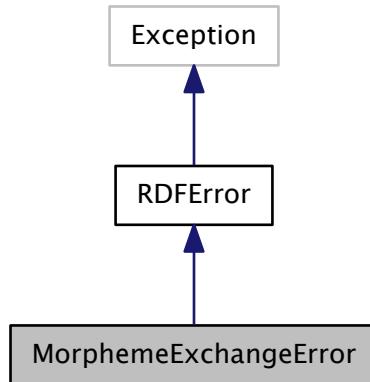
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

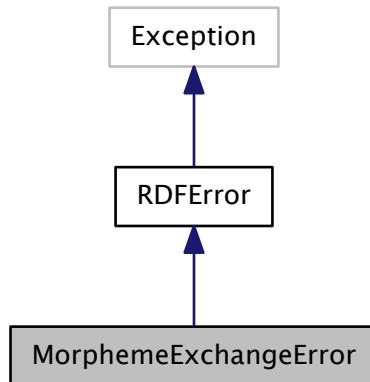
7.63 MorphemeExchangeError Class Reference

Morphere Exchange Currents?

Inheritance diagram for MorphemeExchangeError:



Collaboration diagram for MorphemeExchangeError:



7.63.1 Detailed Description

Morphere Exchange Currents?

fix-pre and/or sufix would cast list v. str type errors on "+"
anyway, so this is a TypeError

Definition at line 17 of file errors.py.

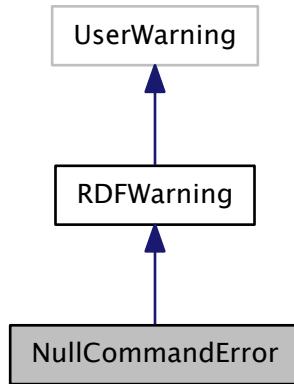
The documentation for this class was generated from the following file:

- rdf/language/[errors.py](#)

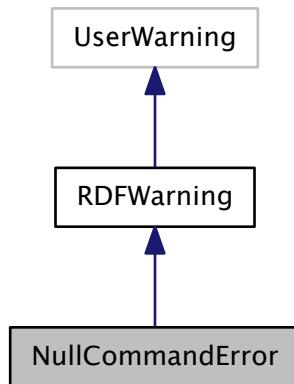
7.64 NullCommandError Class Reference

Should be thrown in constructor?

Inheritance diagram for NullCommandError:



Collaboration diagram for NullCommandError:



7.64.1 Detailed Description

Should be thrown in constructor?

Setting a required command to nothing

Definition at line 43 of file errors.py.

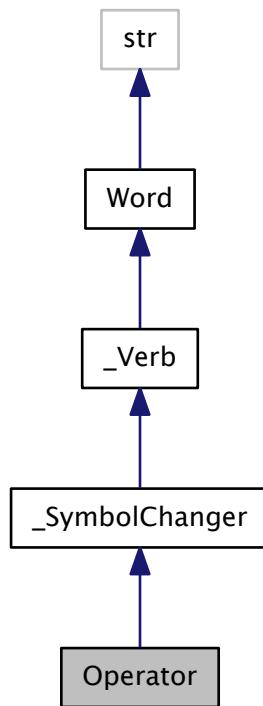
The documentation for this class was generated from the following file:

- rdf/language/errors.py

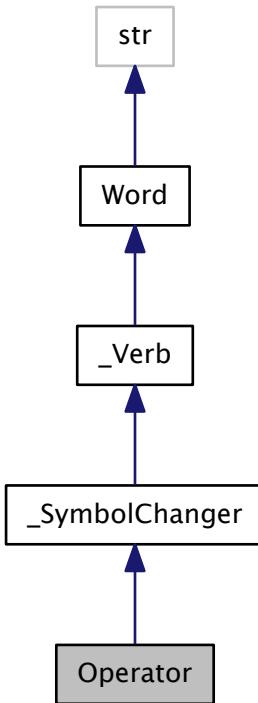
7.65 Operator Class Reference

Change [rdf.language.grammar.syntax.Grammar.operator](#) via `_SymbolChanger.act`.

Inheritance diagram for Operator:



Collaboration diagram for Operator:



Additional Inherited Members

7.65.1 Detailed Description

Change [rdf.language.grammar.syntax.Grammar.operator](#) via `_SymbolChangrr.act`.

Change grammar's operator

Definition at line 70 of file `pragmatics.py`.

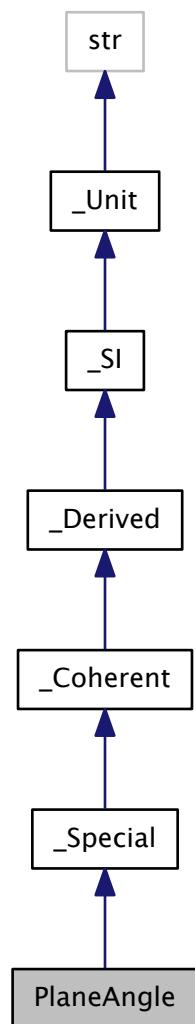
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

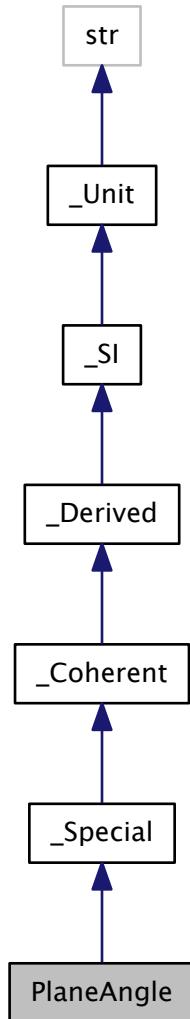
7.66 PlaneAngle Class Reference

Angle conversion to degrees.

Inheritance diagram for PlaneAngle:



Collaboration diagram for PlaneAngle:



Static Private Attributes

- string `_symbol` = 'rad'

Additional Inherited Members

7.66.1 Detailed Description

Angle conversion to degrees.

Definition at line 231 of file `physical_quantity.py`.

7.66.2 Member Data Documentation

7.66.2.1 `string _symbol = 'rad' [static], [private]`

Definition at line 232 of file `physical_quantity.py`.

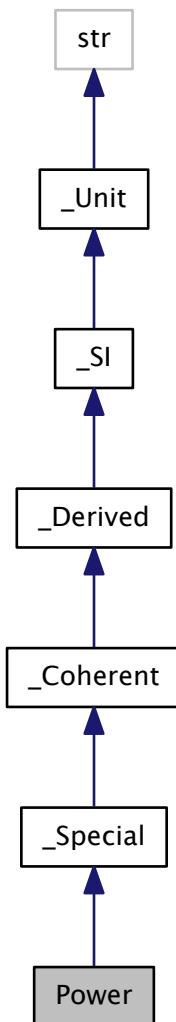
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

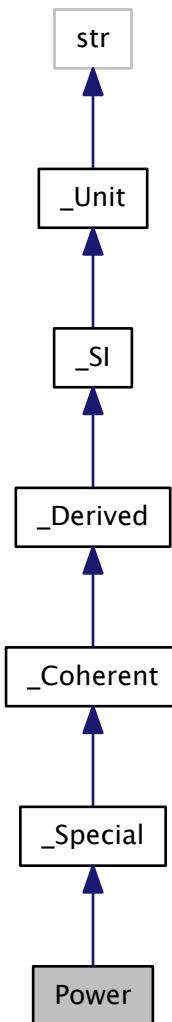
7.67 Power Class Reference

James Watts, FRS, FRSE (19 January 1736 – 25 August 1819)

Inheritance diagram for Power:



Collaboration diagram for Power:



Static Private Attributes

- string `_symbol` = 'W'

Additional Inherited Members

7.67.1 Detailed Description

James Watts, FRS, FRSE (19 January 1736 – 25 August 1819)

Definition at line 267 of file `physical_quantity.py`.

7.67.2 Member Data Documentation

7.67.2.1 `string _symbol = 'W' [static], [private]`

Definition at line 268 of file `physical_quantity.py`.

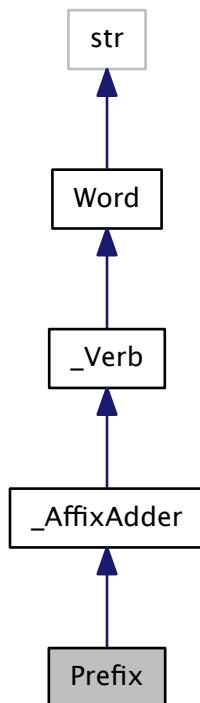
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

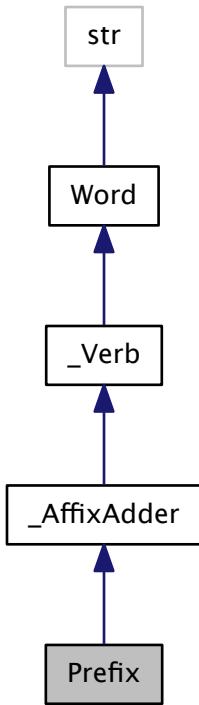
7.68 Prefix Class Reference

Add to [rdf.language.grammar.syntax.Grammar.prefix](#) via `_AffixAdder.act`.

Inheritance diagram for Prefix:



Collaboration diagram for Prefix:



Additional Inherited Members

7.68.1 Detailed Description

Add to [rdf.language.grammar.syntax.Grammar.prefix](#) via `_AffixAdder.act`.

`Prefix` is an `_AffixAdder` that cooperates with `Gramar.prefix`

Definition at line 93 of file `pragmatics.py`.

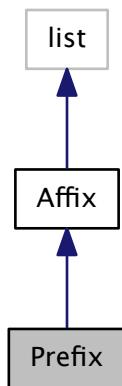
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

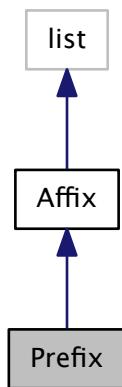
7.69 Prefix Class Reference

Appears Before the stem:

Inheritance diagram for Prefix:



Collaboration diagram for Prefix:



Public Member Functions

- def [__add__](#)
prefix + stem (overrides list concatenation)

7.69.1 Detailed Description

Appears Before the stem:

prefix + stem

is the only allowed operator overload- it, by definition, must be prepended

Definition at line 69 of file morpheme.py.

7.69.2 Member Function Documentation

7.69.2.1 def __add__(self, stem)

prefix + stem (overrides list concatenation)

Definition at line 75 of file morpheme.py.

```
75
76     def __add__(self, stem):
77         return self() + stem
78
```

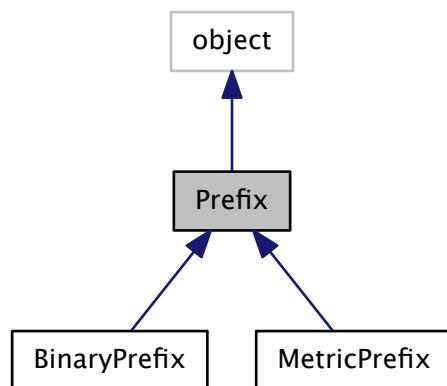
The documentation for this class was generated from the following file:

- rdf/language/grammar/morpheme.py

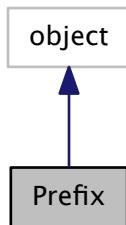
7.70 Prefix Class Reference

Abstract Base class for class decorator factory that makes an instance of the decorated _Unit subclass according to the prefix and the power.

Inheritance diagram for Prefix:



Collaboration diagram for Prefix:



Public Member Functions

- def `base`
Sub's need a base to define what their prefixing means.
- def `cast`
- def `__init__`
Construct with a symbol and an exponent.
- def `__str__`
str(prefix) is the prefix's symbol.
- def `__call__`
Class decorator:
- def `__pow__`
get factor and raise it to n

Public Attributes

- `symbol`
The prefix's official symbol.
- `factor`
 $f = B^x$

Static Private Attributes

- `__metaclass__` = abc.ABCMeta
Without a self.base, this class is no good.

7.70.1 Detailed Description

Abstract Base class for class decorator factory that makes an instance of the decorated `_Unit` subclass according to the prefix and the power.

```

prefix = Prefix("symbol", exponent)

INPUT:
    symbol    The prefix string symbol, e.g: "M" for mega
    exponent  The exponent for the factor... 6    for 10**6
  
```

OUTPUT:

```
prefix A class decorator that creates a new instance of
       the decorated class (that must be a sub-class of _Unit)
       See _Unit.__doc__ for why that works.
```

```
@prefix
class Dimension(_Unit)
    symbol = <what ever measure dimension>
```

Note: Of course, you can stack them up.

Definition at line 8 of file prefix.py.

7.70.2 Constructor & Destructor Documentation

7.70.2.1 def __init__(self, symbol, exponent)

Construct with a symbol and an exponent.

Parameters

<i>symbol</i>	A string symbol that IS the abbreviation
<i>exponent</i>	sets the scale factor = base ** exponent

Definition at line 43 of file prefix.py.

```
43
44     def __init__(self, symbol, exponent):
45         ## The prefix's official symbol
46         self.symbol = str(symbol)
47         ## \f$ f = B^x \f$
48         self.factor = self.base ** exponent
49     return None
```

7.70.3 Member Function Documentation

7.70.3.1 def __call__(self, n=1)

Class decorator:

Parameters

<i>cls</i>	A _Unit sub-class
------------	-------------------

Side Effects:

instaniate deocrated intance and loh into Glossary

Return values

<i>cls</i>	Class decorators return classes. prefix(cls)-->cls' with SIDE EFFECTS
------------	---

Definition at line 59 of file prefix.py.

```
59
60     def __call__(self, n=1):
61         """prefix(cls)-->cls'
62         with SIDE EFFECTS"""
63
64         ## Here we can look up base class to see if it
```

```

65      ## a log on not?
66      def decorator(cls):
67          ## Instantiate cls with prefixed name, and scale factor
68          ## (with exponent)
69          cls(str(self) + cls._symbol, self**n)
70          return cls
71
72      return decorator

```

7.70.3.2 def __pow__(self, n)

get factor and raise it to n

Definition at line 73 of file prefix.py.

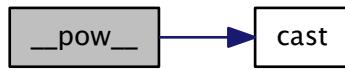
References Prefix.cast().

```

73      def __pow__(self, n):
74          return (self.cast()(self)) ** n
75
76

```

Here is the call graph for this function:



7.70.3.3 def __str__(self)

str(prefix) is the prefix's symbol.

Definition at line 51 of file prefix.py.

References Prefix.symbol.

```

51      def __str__(self):
52          return self.symbol
53

```

7.70.3.4 def base(self)

Sub's need a base to define what their prefixing means.

Definition at line 33 of file prefix.py.

```

33      def base(self):
34          pass
35

```

7.70.3.5 def cast(self)

Definition at line 37 of file prefix.py.

References Prefix.factor.

Referenced by Prefix.__pow__().

```

37
38     def cast(self):
39         return self.factor

```

Here is the caller graph for this function:



7.70.4 Member Data Documentation

7.70.4.1 __metaclass__ = abc.ABCMeta [static], [private]

Without a `self.base`, this class is no good.

Definition at line 29 of file `prefix.py`.

7.70.4.2 factor

$$f = B^x$$

Definition at line 47 of file `prefix.py`.

Referenced by `MetricPrefix.__float__()`, `BinaryPrefix.__long__()`, and `Prefix.cast()`.

7.70.4.3 symbol

The prefix's official symbol.

Definition at line 45 of file `prefix.py`.

Referenced by `Prefix.__str__()`.

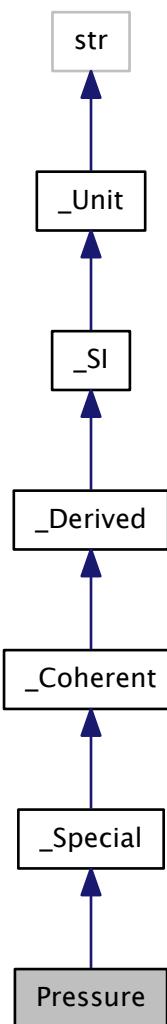
The documentation for this class was generated from the following file:

- `rdf/units/prefix.py`

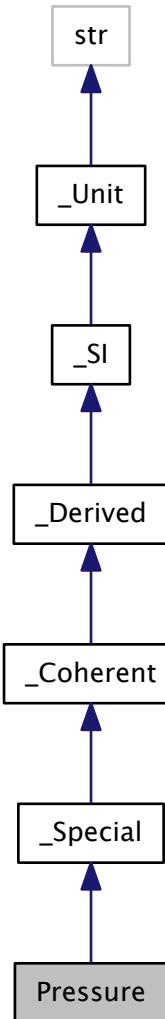
7.71 Pressure Class Reference

Blaise Pascal (19 June 1623 – 19 August 1662)

Inheritance diagram for Pressure:



Collaboration diagram for Pressure:



Static Private Attributes

- string symbol = 'Pa'

Additional Inherited Members

7.71.1 Detailed Description

Blaise Pascal (19 June 1623 – 19 August 1662)

Definition at line 243 of file physical_quantity.py.

7.71.2 Member Data Documentation

7.71.2.1 `string _symbol = 'Pa' [static], [private]`

Definition at line 244 of file `physical_quantity.py`.

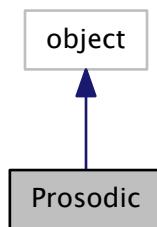
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

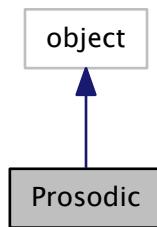
7.72 Prosodic Class Reference

In Dependency Grammar, the clitic (an RDF line) depends on its host (the current grammar)—together they are a **Prosodic**—this is important because an RDF line on its own has no definite meaning—it must be interpreted in terms of its host Grammar, this tuple exists because the pair is passed all over the place.

Inheritance diagram for Prosodic:



Collaboration diagram for Prosodic:



Public Member Functions

- def `__init__`
- def `comment`
Return host's operator glyph.
- def `operator`
Return host's comment glyph.

- def `__iter__`
- def `__int__`
- def `strip`

Stripped sentence (as in no white spaces)
- def `include_file`

INCLUDE.
- def `change_symbol`

Change: OPERATOR / COMMENT.
- def `set_affix`

Change: PREFIX / SUFFIX.
- def `set_unit`

Add to an rdf.units.physical_quantity.Unit to the rdf.units.GLOSSARY.
- def `make_affix`

Add affixes—note: Grammar just adds, the overloaded `add` and `radd` invoke the affix protocol.
- def `extract_record`

RDF Record Reader.
- def `extract_comment`

RDF Comment Reader.

Public Attributes

- `clitic`

The line.
- `host`

The rdf.language.grammar.syntax.Grammar hosting the clitic.

Private Member Functions

- def `_prefix`
- def `_suffix`
- def `_value`

*Extract the "value" field by
getting the line left of Prosodic.comment
getting the remains right of Prosodic.operator.*

Static Private Attributes

- tuple `__slots__` = ('`clitic`', '`host`)

7.72.1 Detailed Description

In Dependency Grammar, the clitic (an RDF line) depends on its host (the current grammar)—together they are a `Prosodic`—this is important because an RDF line on its own has no definite meaning—it must be interpreted in terms of its host Grammar, this tuple exists because the pair is passed all over the place.

```
Prosodic(clitic, host)

clitic      is a string reppin' an RDF line
host       is a Grammar() instance
```

The actions of the lexis are carried out in methods.

Definition at line 28 of file prosodic.py.

7.72.2 Constructor & Destructor Documentation

7.72.2.1 def __init__(self, clitic, host)

Definition at line 39 of file prosodic.py.

```
39      def __init__(self, clitic, host):
40          ## The line
41          self.critic = clitic
42          ## The rdf.language.grammar.syntax.Grammar hosting the clitic
43          self.host = host
```

7.72.3 Member Function Documentation

7.72.3.1 def __int__(self)

Definition at line 66 of file prosodic.py.

References Prosodic.host.

```
66      def __int__(self):
67          return int(self.host)
```

7.72.3.2 def __iter__(self)

Definition at line 63 of file prosodic.py.

References Prosodic.__slots__, and _RDFRecord.__slots__.

```
63      def __iter__(self):
64          return iter((getattr(self, attr) for attr in self.__slots__))
```

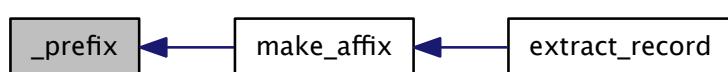
7.72.3.3 def __prefix(self) [private]

Definition at line 46 of file prosodic.py.

Referenced by Prosodic.make_affix().

```
46      def __prefix(self):
47          return self.host.prefix
48
```

Here is the caller graph for this function:



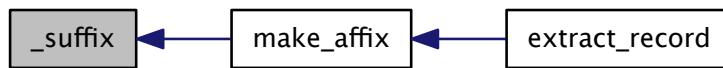
7.72.3.4 def _suffix(self) [private]

Definition at line 50 of file prosodic.py.

Referenced by Prosodic.make_affix().

```
50
51     def _suffix(self):
52         return self.host.suffix
```

Here is the caller graph for this function:



7.72.3.5 def _value(self) [private]

Extract the "value" field by

getting the line left of [Prosodic.comment](#)

getting the remains right of [Prosodic.operator](#).

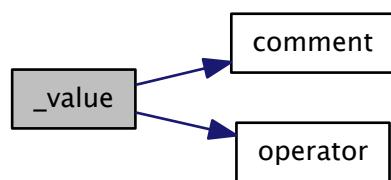
Definition at line 72 of file prosodic.py.

References Prosodic.clitic, Prosodic.comment(), Grammar.comment, Prosodic.operator(), and Grammar.operator.

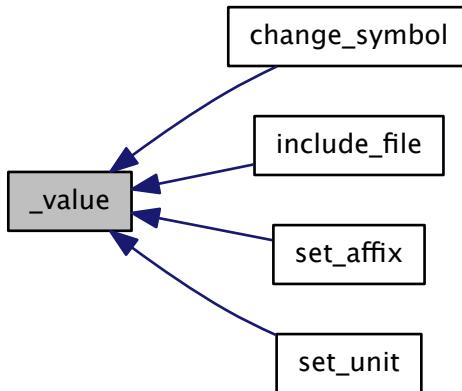
Referenced by Prosodic.change_symbol(), Prosodic.include_file(), Prosodic.set_affix(), and Prosodic.set_unit().

```
72
73     def _value(self):
74         return (+self.operator)((-self.comment)(self.clitic))
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.72.3.6 def change_symbol(self, attr)

Change: OPERATOR / COMMENT.

Send request to host to change attr to `_value()`

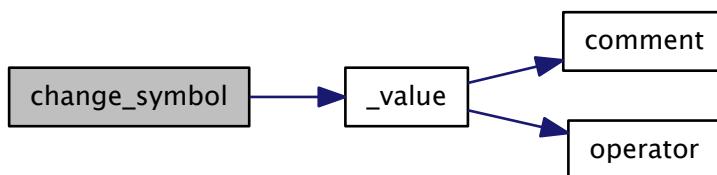
Definition at line 85 of file prosodic.py.

References `Prosodic._value()`.

```

85
86     def change_symbol(self, attr):
87         """Send request to host to change attr to _value()"""
88         self.host.change_symbol(attr, self._value())
  
```

Here is the call graph for this function:



7.72.3.7 def comment(self)

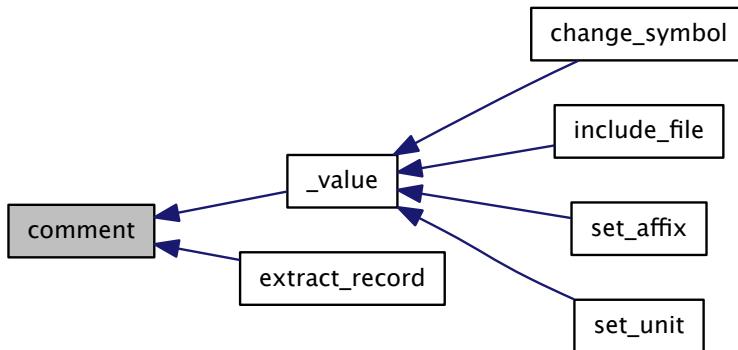
Return host's operator glyph.

Definition at line 55 of file prosodic.py.

Referenced by Prosodic._value(), and Prosodic.extract_record().

```
55
56     def comment(self):
57         return self.host.comment
```

Here is the caller graph for this function:



7.72.3.8 def extract_comment(self)

RDF Comment Reader.

convert in a comment

Definition at line 131 of file prosodic.py.

References Prosodic.strip().

```
131
132     def extract_comment(self):
133         """convert in a comment"""
134         from rdf.data.entries import RDFComment
135         line = self.strip()
136         return RDFComment(line) if line else None # semi-Guard
```

Here is the call graph for this function:



7.72.3.9 def extract_record(self)

RDF Record Reader.

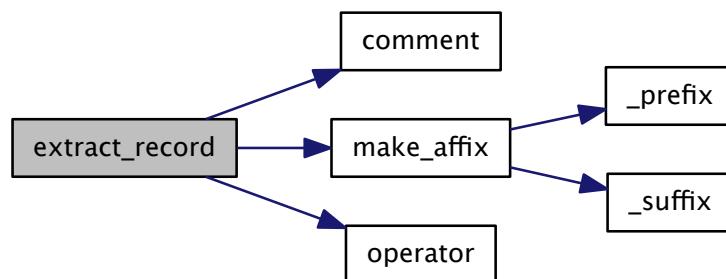
```
if self is a record, you'll get it-- if not who knows
```

Definition at line 117 of file prosodic.py.

References Prosodic.critic, Prosodic.comment(), Grammar.comment, Prosodic.make_affix(), Prosodic.operator(), and Grammar.operator.

```
117
118     def extract_record(self):
119         """if self is a record, you'll get it-- if not who knows"""
120         from rdf.language.grammar import punctuation
121         from rdf.data.entries import RDFField
122         left, comments = self.comment(self.critic)
123         left, value = self.operator(left)
124         base_key, units, dimensions, element = punctuation.parse_left(left)
125         return self.make_affix(base_key) + RDFField(value.strip(),
126                                         units=units,
127                                         dimensions=dimensions,
128                                         element=element,
129                                         comments=comments)
```

Here is the call graph for this function:



7.72.3.10 def include_file(self)

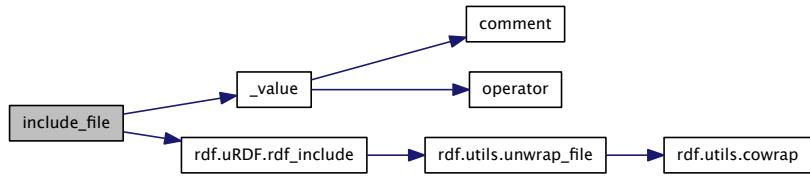
INCLUDE.

Definition at line 80 of file prosodic.py.

References Prosodic._value(), Prosodic.host, and rdf.uRDF.rdf_include().

```
80
81     def include_file(self):
82         from rdf.uRDF import rdf_include
83         return rdf_include(self._value(), _grammar=self.host)
```

Here is the call graph for this function:



7.72.3.11 def make_affix(self, stem)

Add affixes—note: Grammar just adds, the overloaded `add` and `radd` invoke the affix protocol.

Definition at line 113 of file prosodic.py.

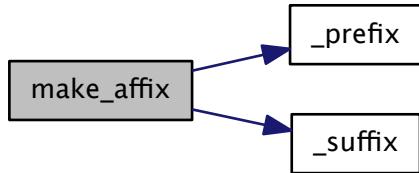
References `Prosodic._prefix()`, `Grammar._prefix`, `Prosodic._suffix()`, and `Grammar._suffix`.

Referenced by `Prosodic.extract_record()`.

```

113
114     def make_affix(self, stem):
115         return self._prefix + stem + self._suffix
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.72.3.12 def operator(self)

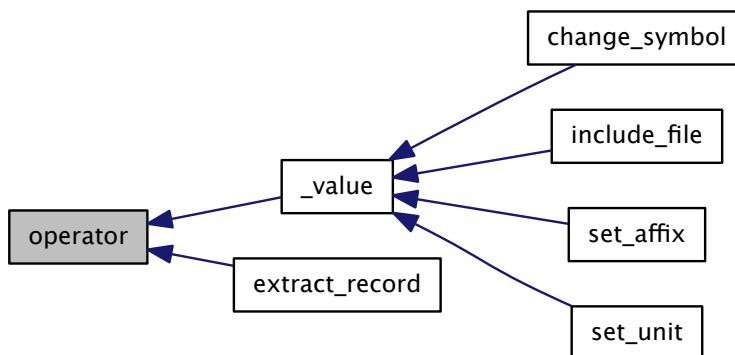
Return host's comment glyph.

Definition at line 60 of file prosodic.py.

Referenced by Prosodic._value(), and Prosodic.extract_record().

```
60
61     def operator(self):
62         return self.host.operator
```

Here is the caller graph for this function:



7.72.3.13 def set_affix (self, classname)

Change: PREFIX / SUFFIX.

```
Send request to host to set
host.classname[int(host)] = _value()
```

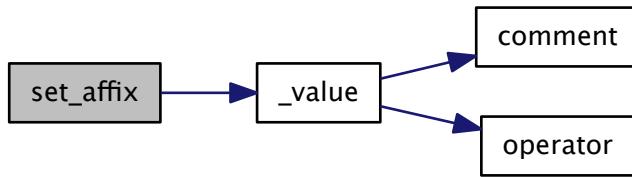
(that is, set PREFIX or SUFFIX's host.dpeth item to the value given by the clitic-- that is, in the AFFFIX = <value> line

Definition at line 90 of file prosodic.py.

References Prosodic._value().

```
90
91     def set_affix(self, classname):
92         """Send request to host to set
93         host.classname[int(host)] = _value()
94
95         (that is, set PREFIX or SUFFIX's host.dpeth item to the value
96         given by the clitic-- that is, in the AFFFIX = <value>
97         line"""
98         self.host.set_affix(classname, self._value())
```

Here is the call graph for this function:



7.72.3.14 def set_unit(self)

Add to an rdf.units.physical_quantity.Unit to the [rdf.units.GLOSSARY](#).

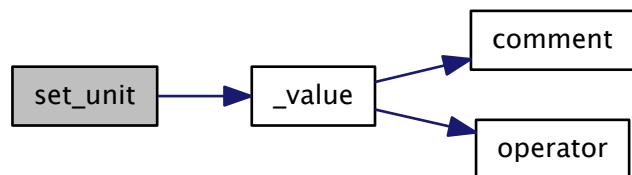
Definition at line 100 of file prosodic.py.

References Prosodic._value().

```

100
101     def set_unit(self):
102         from rdf import utils
103         from rdf.units import physical_quantity
104         from rdf.units.errors import RedefiningUnitWarning
105         ## Convert line into arguments and/or keywords
106         args, kwargs = utils.parse_tuple_input(
107             self._value().lstrip("(").rstrip(")")
108         )
109         ## Insatiation is memoized.
110         physical_quantity._UnAccepted(*args, **kwargs)
  
```

Here is the call graph for this function:



7.72.3.15 def strip(self)

Stripped sentence (as in no white spaces)

Definition at line 76 of file prosodic.py.

Referenced by Prosodic.extract_comment().

```

76
77     def strip(self):
78         return self.clitic.strip()
  
```

Here is the caller graph for this function:



7.72.4 Member Data Documentation

7.72.4.1 tuple __slots__ = ('clitic', 'host') [static], [private]

Definition at line 37 of file prosodic.py.

Referenced by Prosodic.__iter__().

7.72.4.2 clitic

The line.

Definition at line 41 of file prosodic.py.

Referenced by Prosodic._value(), and Prosodic.extract_record().

7.72.4.3 host

The [rdf.language.grammar.syntax.Grammar](#) hosting the clitic.

Definition at line 43 of file prosodic.py.

Referenced by Prosodic.__int__(), and Prosodic.include_file().

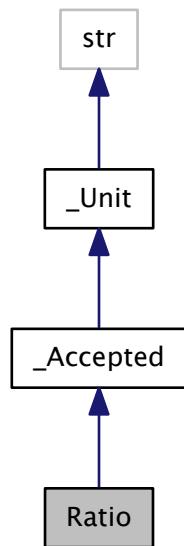
The documentation for this class was generated from the following file:

- [rdf/language/prosodic.py](#)

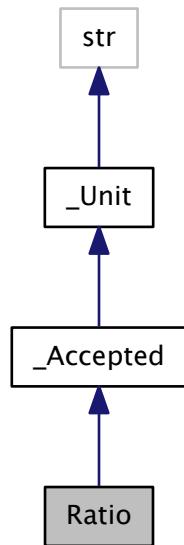
7.73 Ratio Class Reference

TBD.

Inheritance diagram for Ratio:



Collaboration diagram for Ratio:



Additional Inherited Members

7.73.1 Detailed Description

TBD.

Definition at line 475 of file `physical_quantity.py`.

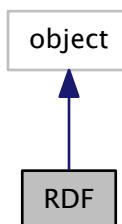
The documentation for this class was generated from the following file:

- `rdf/units/physical_quantity.py`

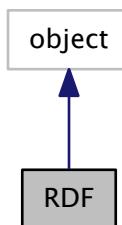
7.74 RDF Class Reference

An [RDF](#) Mothership: A fully interpresed [RDF](#) File.

Inheritance diagram for `RDF`:



Collaboration diagram for `RDF`:



Public Member Functions

- def `fromdict`
Make an instance from DICT argument.
- def `fromstar`
Make it from keyword arguments.
- def `__init__`
Don't use init– it's internal.

- def `__getattr__`
Get attr from DICT If needed.
- def `rdf`
rdf is rdf
- def `__getitem__`
Return value, vs "get" returns RDFField.
- def `__setitem__`
Set the item to an RDFField.
- def `__delitem__`
Delete key, field.
- def `__iter__`
`iteritems()`
- def `__call__`
Access as method or property (this is really for clients)
- def `record`
key → rdf.data.entries.RDFRecord
- def `records`
get record() for all keys.
- def `__str__`
A nice formatted string.
- def `__repr__`
rep the data attribute
- def `__len__`
`len()`
- def `__rshift__`
rdf >> dst \n see tofile().
- def `__lshift__`
{ rdf << src } Sucks up an new rdf file (see fromfile())
- def `__ilshift__`
rdf << rdf' # lets rdf suck up an new rdf file.
- def `__add__`
Add is concatenation, and is not commutative.
- def `__iadd__`
See __add__.
- def `tofile`
Write to file, with some formatting.
- def `todict`
Convert to a standard (key, value) DICT.

Static Public Member Functions

- def `fromfile`
Instantiate from a file.

Public Attributes

- `data`
The data are an associative array-or the rdf spec is worthless.

Private Member Functions

- `def __max_index`
Get maximum index (column) of OPERATOR in file's strings.

7.74.1 Detailed Description

An [RDF](#) Mothership: A fully interpresed [RDF](#) File.

RDF object is made from the `read_rdf` helper function:

```
>>>data = rdf_reader('rdf.txt')
```

It is an associate array, so like a dict:

```
>>>data[key]
```

returns a value- as a float or string-or whatever "eval" returns.

All the standard OrderDict methods can be used, and will return the full RDFField object that represent the value, units, dimension.... comments.

You may `__setitem__`:

```
>>>rdf[key] = value #equivalent to  
>>>rdf[key] = RDFField(value)
```

That is, it transforms assignee into an RDFField for you.

Definition at line 27 of file files.py.

7.74.2 Constructor & Destructor Documentation

7.74.2.1 def __init__(self, args)

Don't use init- it's internal.

Parameters

<code>*args</code>	Internal constructor takes *args RDF(*args) matched dict built-in
--------------------	--

Definition at line 85 of file files.py.

```
85  
86     def __init__(self, *args):  
87         """RDF(*args) matched dict built-in"""  
88         ## The data are an associative array-or the rdf spec is worthless.  
89         self.data = DICT(args)
```

7.74.3 Member Function Documentation

7.74.3.1 def __add__(self, other)

Add is concatenation, and is not communative.

Parameters

<i>other</i>	An RDF instance
--------------	---------------------------------

Return values

<i>result</i>	Is another RDF instance
---------------	---

Definition at line 192 of file files.py.

```
192     def __add__(self, other):
193         result = self
194         for key, field in other.items():
195             result[key] = field
196         return result
```

7.74.3.2 def __call__(self)

Access as method or property (this is really for clients)

```
self() -> self so that x.rdf() () -->x.rdf() ->x.rdf
```

Definition at line 136 of file files.py.

```
136     def __call__(self):
137         """self() -> self so that x.rdf() () -->x.rdf() ->x.rdf"""
138         return self
139
```

7.74.3.3 def __delitem__(self, key)

Delete key, field.

Parameters

<i>key</i>	an RDF key
------------	----------------------------

Side Effects:

deletes key from [RDF.data](#)

Returns

None

Definition at line 128 of file files.py.

```
128
129     def __delitem__(self, key):
130         self.data.__delitem__(key)
```

7.74.3.4 def __getattr__(self, attr)

Get attr from [DICT](#) If needed.

Parameters

<i>attr</i>	Attribubute
-------------	-------------

Return values

<i>self.attr</i>	OR
<i>self.data.attr</i>	If needed

Definition at line 94 of file files.py.

References RDF.data.

```
94
95     def __getattr__(self, attr):
96         try:
97             result = object.__getattribute__(self, attr)
98         except AttributeError:
99             result = getattr(self.data, attr)
100        return result
```

7.74.3.5 def __getitem__(self, key)

Return value, vs "get" returns RDFField.

Definition at line 108 of file files.py.

References RDF.data.

```
108
109     def __getitem__(self, key):
110         return self.data[key]() # .value ?
```

7.74.3.6 def __iadd__(self, other)

See `_add_`.

Definition at line 199 of file files.py.

```
199
200     def __iadd__(self, other):
201         self = self+other
202         return self
```

7.74.3.7 def __ilshift__(self, src)

`rdf << rdf' # lets rdf suck up an new rdf file.`

Definition at line 186 of file files.py.

```
186
187     def __ilshift__(self, src):
188         return self << src
```

7.74.3.8 def __iter__(self)

`iteritems()`

Definition at line 132 of file files.py.

```
132
133     def __iter__(self):
134         return iter(self.data.iteritems())
```

7.74.3.9 def __len__(self)

len()

Definition at line 171 of file files.py.

References RDF.data.

```
171
172     def __len__(self):
173         return len(self.data)
```

7.74.3.10 def __lshift__(self, src)

{ rdf << src } Sucks up an new rdf file (see [fromfile\(\)](#))

Parameters

<i>src</i>	RDF file name
------------	---------------

Returns

src + RDF

Definition at line 182 of file files.py.

References RDF.fromfile().

```
182
183     def __lshift__(self, src):
184         return self + type(self).fromfile(src)
```

Here is the call graph for this function:



7.74.3.11 def __repr__(self)

rep the data attribute

Definition at line 167 of file files.py.

References RDF.data.

```
167
168     def __repr__(self):
169         return repr(self.data)
```

7.74.3.12 def __rshift__(self, dst)

rdf >> dst \n see [tofile\(\)](#).

Definition at line 176 of file files.py.

References RDF.tofile().

```

176
177     def __rshift__(self, dst):
178         return self.tofile(dst)

```

Here is the call graph for this function:



7.74.3.13 def __setitem__(self, key, value)

Set the item to an RDFField.

Parameters

<i>key</i>	an RDF key
<i>value</i>	any kind of value

Side Effects:

assigns key in [RDF.data](#) with RDFField value

Returns

None

Definition at line 118 of file files.py.

References [RDF.data](#).

```

118
119     def __setitem__(self, key, value):
120         from rdf.data.entries import RDFField
121         self.data[key] = RDFField(value)

```

7.74.3.14 def __str__(self)

A nice formatted string.

Definition at line 156 of file files.py.

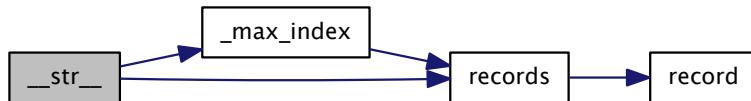
References [RDF._max_index\(\)](#), and [RDF.records\(\)](#).

```

156
157     def __str__(self):
158         from rdf.data.entries import RDFRecord
159         max_index = self._max_index()
160         ## now insert space...
161         final_result = []
162         for record in self.records():
163             line = record.__str__(index=max_index)
164             final_result.append(line + '\n')
165         return "".join(final_result)

```

Here is the call graph for this function:



7.74.3.15 def _max_index(self) [private]

Get maximum index (column) of OPERATOR in file's strings.

Definition at line 152 of file files.py.

References RDF.records().

Referenced by RDF.__str__().

```

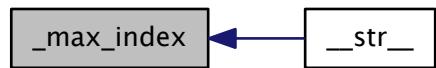
152
153     def _max_index(self):
154         return max(map(int, self.records()))

```

Here is the call graph for this function:



Here is the caller graph for this function:



7.74.3.16 def fromdict(cls, dict_)

Make an instance from DICT argument.

Parameters

<i>dict_</i>	is an rdf-enough dictionary
--------------	-----------------------------

Returns**RDF instance**

instantiate from a DICT

Definition at line 53 of file files.py.

```

53     def fromdict(cls, dict_):
54         """instantiate from a DICT"""
55         result = cls()
56         for key, value in dict_.items():
57             result[key] = value
58
59     return result

```

7.74.3.17 def fromfile(*src*) [static]

Instantiate from a file.

Parameters

<i>src</i>	RDF file name
------------	---------------

Return values

RDF	an rdf instance
------------	-----------------

<i>src</i>	\rightarrow RDF
------------	-------------------

Definition at line 78 of file files.py.

References `rdf.rdfparse`.Referenced by `RDF.__lshift__()`.

```

78
79     def fromfile(src):
80         """src -> RDF"""
81         from rdf import rdfparse
82         return rdfparse(src)

```

Here is the caller graph for this function:

**7.74.3.18 def fromstar(*cls*, *args*, *dict_*)**

Make it from keyword arguments.

Parameters

*args	is an rdf-enough arguments (like dict)
**dict_	is an rdf-enough dictionary

Returns**RDF** instance

instantiate from *args **dict_()

Definition at line 65 of file files.py.

```

65      def fromstar(cls, *args, **dict_):
66          """instantiate from *args **dict_"""
67          # todo: (dict(*args) + dict_)
68          rdf_ = cls()
69          for pair in args:
70              key, value = pair
71              rdf_[str(key)] = value
72          return rdf_ + cls.fromdict(dict_)
```

7.74.3.19 def rdf(self)

rdf is rdf

Definition at line 103 of file files.py.

```

103      def rdf(self):
104          return self
105
```

7.74.3.20 def record(self, key)key -> **rdf.data.entries.RDFRecord**

convery self[key] to an RDFRecord

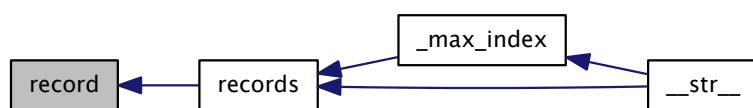
Definition at line 141 of file files.py.

Referenced by **RDF.records()**.

```

141      def record(self, key):
142          """convery self[key] to an RDFRecord"""
143          from rdf.data.entries import RDFRecord
144          return RDFRecord(key, self.get(key))
```

Here is the caller graph for this function:



7.74.3.21 def records (self)

get record() for all keys.

Get all records from record()

Definition at line 147 of file files.py.

References RDF.record().

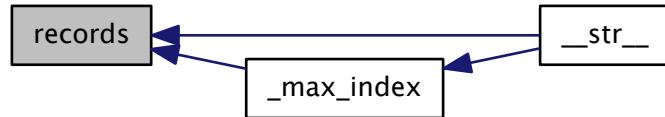
Referenced by RDF.__str__(), and RDF._max_index().

```
147
148     def records(self):
149         """Get all records from record()"""
150         return map(self.record, self.keys())
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.74.3.22 def todict (self, caster=lambda __: __)

Convert to a standard (key, value) DICT.

Convert to a normal dict()

caster key words casts the values

Definition at line 216 of file files.py.

```
216
217     def todict(self, caster=lambda __: __):
218         """Convert to a normal dict()
219
220         caster key words casts the values
221
222         """
223         result = {}
224         for key, field in self.iteritems():
225             result.update({key: caster(field.value)})
226
227         return result
```

7.74.3.23 def tofile(self, dst)

Write to file, with some formatting.

Parameters

<i>dst</i>	file name (writable)
------------	----------------------

Side Effects:

writes dst

Returns

write data to a file

Definition at line 208 of file files.py.

Referenced by RDF.__rshift__().

```
208
209     def tofile(self, dst):
210         """write data to a file"""
211         with open(dst, 'w') as fdst:
212             fdst.write(str(self))
213         ## return dst to make idempotent
214         return dst
```

Here is the caller graph for this function:



7.74.4 Member Data Documentation

7.74.4.1 data

The data are an associative array-or the rdf spec is worthless.

Definition at line 88 of file files.py.

Referenced by RDF.__getattr__(), RDF.__getitem__(), RDF.__len__(), RDF.__repr__(), and RDF.__setitem__().

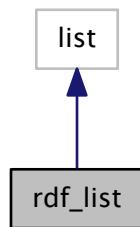
The documentation for this class was generated from the following file:

- [rdf/data/files.py](#)

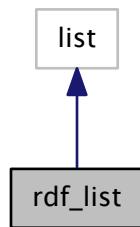
7.75 rdf_list Class Reference

A list of rdf records that can filter itself and make an RDF.

Inheritance diagram for rdf_list:



Collaboration diagram for rdf_list:



Public Member Functions

- def `rdf`

Convert list to an RDF instance: filter out comments and pass to RDF constructor.

7.75.1 Detailed Description

A list of rdf records that can filter itself and make an RDF.

see the list constructor.

Adds method `rdf()` to make it into an `rdf`

Definition at line 21 of file iRDF.py.

7.75.2 Member Function Documentation

7.75.2.1 def `rdf(self)`

Convert list to an RDF instance: filter out comments and pass to RDF constructor.

Returns

RDF instance from filter(bool, self)

Convert list's contents into an RDF instance

Definition at line 30 of file iRDF.py.

```
30
31     def rdf(self):
32         """Convert list's contents into an RDF instance"""
33         from rdf.data.files import RDF
34         # filter out comments and send over to RDF.
35         return RDF(*filter(bool, self))
36
```

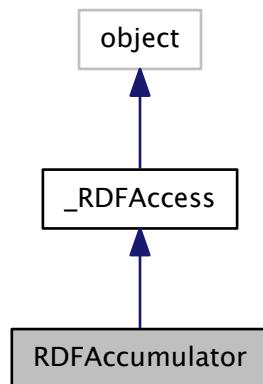
The documentation for this class was generated from the following file:

- [rdf/iRDF.py](#)

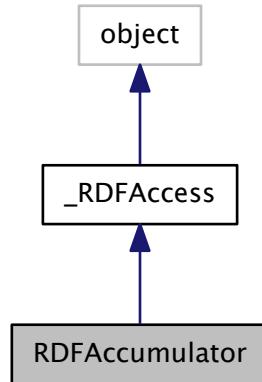
7.76 RDFAccumulator Class Reference

A TBD rdf accumulator - prolly slower than [RDFAnalyzer](#) as it rebuild dictionary all the time— see [RDF.__iadd__](#).

Inheritance diagram for RDFAccumulator:



Collaboration diagram for RDFAccumulator:



Public Member Functions

- def [fromfile](#)

The following are equivalent:

```
>>>RDFAccumulator.fromfile(src).rdf()
>>>rdf.rdfparse(src)
```

- def [__init__](#)

There are no inputs, but some private static attributes are created.

- def [__call__](#)

Call rdf.language.grammar.syntax.Grammar() and remember.

- def [rdf](#)

Unpack RDFRecord elements into RDF.

Public Attributes

- [record_list](#)

Remember the list (as an rdf_list) -starts empty.

7.76.1 Detailed Description

A TBD rdf accumulator - prolly slower than [RDFAnalyzer](#) as it rebuild dictionary all the time— see [RDF.__iadd__](#).

```
a = RDFAccumulator()
```

creates and accumulator, who's __call__ eats rdf lines and appends their results to 'record_list', which is an rdf_list.

```
rdf() method calls rdf_list.rdf().
```

Definition at line 94 of file iRDF.py.

7.76.2 Constructor & Destructor Documentation

7.76.2.1 def __init__(self)

There are no inputs, but some private static attributes are created.

Parameters

<code>None</code>	There are no inputs allowed.
-------------------	------------------------------

Definition at line 117 of file iRDF.py.

```
117
118     def __init__(self):
119         # call super
120         super(RDFAccumulator, self).__init__()
121         ## Remeber the list (as an rdf_list) -starts empty.
122         self.record_list = rdf_list()
123
124     return None
```

7.76.3 Member Function Documentation**7.76.3.1 def __call__(self, line)**

Call [rdf.language.grammar.syntax.Grammar\(\)](#) and remember.

Parameters

<code>line</code>	Any type of rdf string, including continued.
-------------------	--

Returns

`None`

rdf sentence goes in

Definition at line 129 of file iRDF.py.

```
129
130     def __call__(self, line):
131         """rdf sentence goes in"""
132         self.record_list.extend(super(RDFAccumulator, self).__call__(line))
```

7.76.3.2 def fromfile(cls, src)

The following are equivalent:

```
>>>RDFAccumulator.fromfile(src).rdf()
>>>rdf.rdfparse(src)
```

Parameters

<code>src</code>	RDF source file
------------------	-----------------

Return values

<code>accumulator</code>	And RDFAccumulator instance (full of src).
	instaniate from src

Definition at line 109 of file iRDF.py.

```
109
110     def fromfile(cls, src):
111         """instantiate from src"""
112         accumulator = RDFAccumulator()
113         accumulator("INCLUDE = %s" % src)
114
115     return accumulator
```

7.76.3.3 def rdf(self)

Unpack RDFRecord elements into RDF.

see `rdf_list.rdf()`

Definition at line 135 of file iRDF.py.

```
135
136     def rdf(self):
137         """see rdf_list.rdf()"""
138         return self.record_list.rdf()
139
```

7.76.4 Member Data Documentation

7.76.4.1 record_list

Remeber the list (as an `rdf_list`) -starts empty.

Definition at line 121 of file iRDF.py.

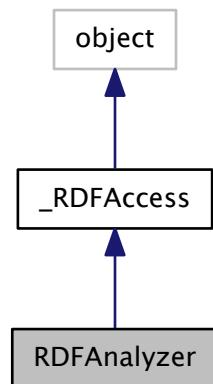
The documentation for this class was generated from the following file:

- `rdf/iRDF.py`

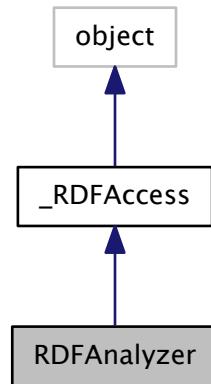
7.77 RDFAnalyzer Class Reference

`RDFAnalyzer` is created with an `rdf.language.syntax.Grammar` object and then emulates a function that converts single line inputs into a pre-RDF output -note: it is overly complicated, and its sole purpose is to provide an interactive RDF reader.

Inheritance diagram for `RDFAnalyzer`:



Collaboration diagram for RDFAnalyzer:



Public Member Functions

- def [__call__](#)
FUNCTION EMULATION: The instance processes RDF lines sequentially.

7.77.1 Detailed Description

[RDFAnalyzer](#) is created with an `rdf.language.syntax.Grammar` object and then emulates a function that converts single line inputs into a pre-RDF output -note: it is overly complicated, and its sole purpose is to provide an interactive RDF reader.

```
a = RDFAnalyzer()
creates an RDFAnalyzer with 'fresh' Grammar. __call__ then runs it:
>>>a(line) --> RDFRecord.
```

Definition at line 69 of file iRDF.py.

7.77.2 Member Function Documentation

7.77.2.1 def [__call__](#)(*self*, *line*)

FUNCTION EMULATION: The instance processes RDF lines sequentially.

Parameters

<i>line</i>	A complete (or incomplete) rdf sentence.
-------------	--

Return values

<i>rdf_list</i>	<p>An <code>rdf_list</code> of <code>rdf.data.entries</code> objects...</p> <p><code>self(line)</code> --> processes the line and updates the grammar wrapped lines are OK.</p>
-----------------	--

Definition at line 79 of file iRDF.py.

```

79
80     def __call__(self, line):
81         """self(line) --> processes the line and updates the grammar
82
83         wrapped lines are OK.
84         """
85         ## Deal with wraps:
86         while line.strip().endswith(self.grammar.wrap):
87             line = line.strip().rstrip(self.grammar.wrap) + raw_input('... ')
88         # Process the line and unpack all the results.
89         result = super(RDFAnalyzer, self).__call__(line)
90         return rdf_list([item for item in result] if result else [result])
91

```

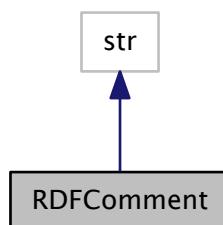
The documentation for this class was generated from the following file:

- [rdf/iRDF.py](#)

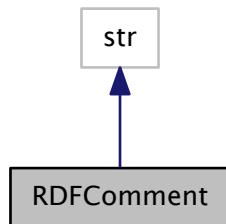
7.78 RDFComment Class Reference

The RDF Comment is a comment string, endowed with False RDF-ness and null response to iteration.

Inheritance diagram for RDFComment:



Collaboration diagram for RDFComment:



Public Member Functions

- def `__nonzero__`
RDF comments are False in an RDF sense—regardless of their content.
- def `__iter__`
Iter iterates over nothing-NOT the contents of the string.

7.78.1 Detailed Description

The RDF Comment is a comment string, endowed with False RDF-ness and null response to iteration.

This is string that always evaluates to False.

Why?

False gets thrown out before being sent to the RDF constructor

But!

It is not None, so you can keep it in your RDFAccumulator

Definition at line 207 of file entries.py.

7.78.2 Member Function Documentation

7.78.2.1 def __iter__(self)

Iter iterates over nothing-NOT the contents of the string.

Return values

<code>iter()</code>	An empty iterator that passes a for-loop silently
---------------------	---

Definition at line 227 of file entries.py.

```

227
228     def __iter__(self):
229         return iter(())

```

7.78.2.2 def __nonzero__(self)

RDF comments are False in an RDF sense—regardless of their content.

Return values

<	False	Always returns False— ALWAYS
---	-------	------------------------------

Definition at line 222 of file entries.py.

```
222
223     def __nonzero__(self):
224         return False
```

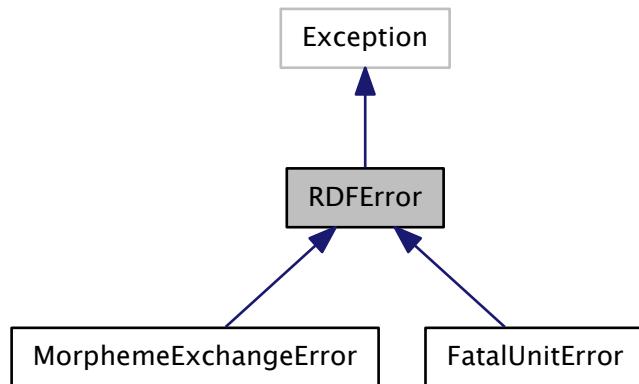
The documentation for this class was generated from the following file:

- [rdf/data/entries.py](#)

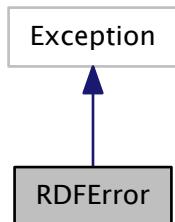
7.79 RDSError Class Reference

Fatal attempt to CODE badly.

Inheritance diagram for RDSError:



Collaboration diagram for RDSError:



7.79.1 Detailed Description

Fatal attempt to CODE badly.

Base RDF Error for BAD RDF coding (Fatal)

Definition at line 6 of file errors.py.

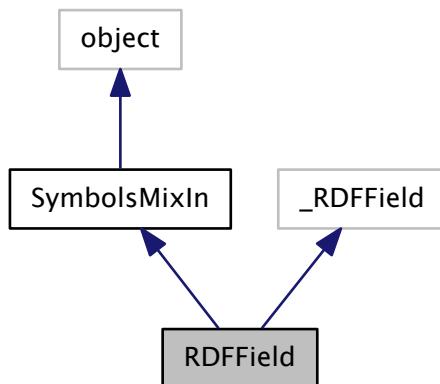
The documentation for this class was generated from the following file:

- [rdf/language/errors.py](#)

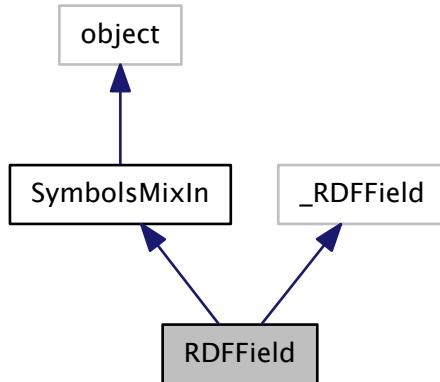
7.80 RDFField Class Reference

Add methods and constants to `_RDFField` so that it lives up to its name.

Inheritance diagram for RDFField:



Collaboration diagram for RDFField:



Public Member Functions

- def `__new__`
Do a namedtuple with defaults as follows...
- def `eval`
eval(self.value) -with some protection/massage safe for list, tuples, nd.arrays, set, dict, anything that can survive repr - this is really a work in progress, since there is a lot of python subtly involved.
- def `right_field`
Construct string on the right side of OPERATOR (w/o an IF)
- def `left_field`
Construct string on the left side of OPERATOR.
- def `__str__`
FORMAT CONTROL TBD.
- def `__call__`
Call returns value.
- def `__radd__`
key + field -> _RDFPreRecord, the whole thing is private.

Static Public Attributes

- `default_units = _default_units`
non-private version: it is used in units.py

Private Member Functions

- def `_handle_units`
Do the unit conversion.

Static Private Attributes

- string `_default_units` = "-"
(-) appears as default
- string `_default_comments` = ""
does not appear b/c it's False
- string `_default_dimensions` = ""
ditto
- string `_default_element` = ""
ditto
- tuple `_index_` = `_cast(bin)`
- tuple `_hex_` = `_cast(hex)`
- tuple `_oct_` = `_cast(oct)`
- tuple `_int_` = `_cast(int)`
- tuple `_long_` = `_cast(long)`
- tuple `_float_` = `_cast(float)`
- tuple `_complex_` = `_cast(complex)`

7.80.1 Detailed Description

Add methods and constants to `_RDFField` so that it lives up to its name.

```
RDFField(value, units=None, dimensions=None, element=None, comments=None)
represents a fully interpreted logical entry in an RDF file (sans key)
```

Definition at line 34 of file entries.py.

7.80.2 Member Function Documentation

7.80.2.1 def __call__(self, caster=lambda __: __)

Call returns value.

Parameters

<code>[func]</code>	= $f(x) : x \rightarrow x$ A callable (like float).
---------------------	---

Returns

$f(x)$ with x from `eval()` method.

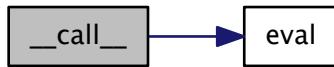
```
You can cast with call via, say:
field(float)
```

Definition at line 132 of file entries.py.

References `RDFField.eval()`.

```
132
133     def __call__(self, caster=lambda __: __):
134         """You can cast with call via, say:
135             field(float)"""
136         return caster(self.eval())
```

Here is the call graph for this function:



7.80.2.2 def __new__(cls, value, units=None, dimensions=None, element=None, comments=None)

Do a namedtuple with defaults as follows...

Parameters

<i>[cls]</i>	class is implicitly passed...
<i>value</i>	Is the value of the rdf field
<i>[units]</i>	defaults to RDFField._default_units
<i>[dimensions]</i>	defaults to RDFField._default_dimensions
<i>[element]</i>	defaults to RDFField._default_element
<i>[comments]</i>	defaults to RDFField._default_comments

Returns

`__new__` returns new instances, or value if value already is an RDFField instance (which makes `RDF.___setitem__` easy to use)

Definition at line 63 of file entries.py.

```

63
64             comments=None):
65     ## Bail on redundant instantiation
66     if isinstance(value, cls): # guard
67         return value
68     ## Unit conversion
69     value, units = cls._handle_units(value, units)
70     return _RDFField.__new__(cls,
71                             value,
72                             str(units or cls._default_units),
73                             str(dimensions or cls._default_dimensions),
74                             str(element or cls._default_element),
75                             str(comments or cls._default_comments))

```

7.80.2.3 def __radd__(self, key)

key + field -> _RDFPreRecord, the whole thing is private.

Definition at line 146 of file entries.py.

```

146
147     def __radd__(self, key):
148         return RDFPreRecord(key, self)
149

```

7.80.2.4 def __str__(self, index=0)

FORMAT CONTROL TBD.

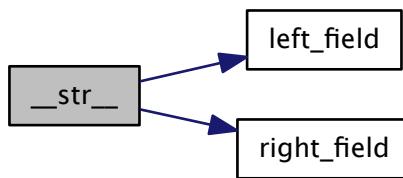
place OPERATOR at index or don't

Definition at line 123 of file entries.py.

References RDFField.left_field(), SymbolsMixIn.Operator, and RDFField.right_field().

```
123
124     def __str__(self, index=0):
125         """place OPERATOR at index or don't"""
126         return (self.left_field(index=index) +
127                 self.Operator + SPACE +
128                 self.right_field())
```

Here is the call graph for this function:



7.80.2.5 def _handle_units(cls, value, units) [private]

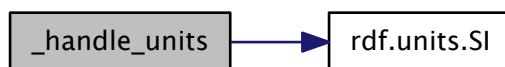
Do the unit conversion.

Definition at line 78 of file entries.py.

References rdf.units_SI().

```
78
79     def _handle_units(cls, value, units):
80         from rdf.units import SI, errors
81         # convert units, If they're neither None nor "-".
82         if units and units != cls._default_units:
83             try:
84                 value, units = SI(value, units)
85             except errors.UnrecognizedUnitWarning:
86                 print >> sys.stderr, ("UnrecognizedUnitWarning: %s" %
87                                     str(units))
88         return value, units
```

Here is the call graph for this function:



7.80.2.6 def eval(self)

eval(self.value) -with some protection/massage safe for list, tuples, nd.arrays, set, dict, anything that can survive repr - this is really a work in progress, since there is a lot of python subtly involved.

Returns

evaluated version of RDFField.value
 eval() uses eval built-in to interpret value

Definition at line 94 of file entries.py.

Referenced by RDFField.__call__().

```
94
95     def eval(self):
96         """eval() uses eval built-in to interpret value"""
97         try:
98             result = eval(str(self.value))
99         except (TypeError, NameError, AttributeError, SyntaxError):
100            try:
101                result = eval(repr(self.value))
102            except (TypeError, NameError, AttributeError, SyntaxError):
103                result = self.value
104
105        return result
```

Here is the caller graph for this function:

**7.80.2.7 def left_field(self, index = 0)**

Construct string on the left side of OPERATOR.

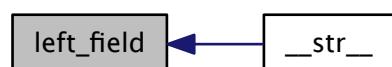
```
Parse left of OPERATOR
place OPERATOR at index or don't
```

Definition at line 113 of file entries.py.

Referenced by RDFField.__str__().

```
113
114     def left_field(self, index=0):
115         """Parse left of OPERATOR
116         place OPERATOR at index or don't
117         """
118         result = punctuation.write_brackets(self.units,
119                                              self.dimensions,
120                                              self.element)
121
122         return result + (SPACE * max(0, index-len(result)))
```

Here is the caller graph for this function:



7.80.2.8 def right_field(self)

Construct string on the right side of OPERATOR (w/o an IF)

Parse right of operator

Definition at line 106 of file entries.py.

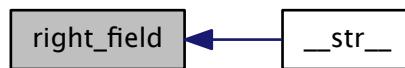
References SymbolsMixIn.Comment.

Referenced by RDFField.__str__().

```
106
107     def right_field(self):
108         """Parse right of operator"""
109         return (str(self.value) +
110                 (SPACE + self.Comment) * bool(self.comments) +
111                 (self.comments or ""))

```

Here is the caller graph for this function:



7.80.3 Member Data Documentation

7.80.3.1 tuple __complex__ = __cast(complex) [static], [private]

Definition at line 143 of file entries.py.

7.80.3.2 tuple __float__ = __cast(float) [static], [private]

Definition at line 142 of file entries.py.

7.80.3.3 tuple __hex__ = __cast(hex) [static], [private]

Definition at line 138 of file entries.py.

7.80.3.4 tuple __index__ = __cast(bin) [static], [private]

Definition at line 137 of file entries.py.

7.80.3.5 tuple __int__ = __cast(int) [static], [private]

Definition at line 140 of file entries.py.

7.80.3.6 tuple __long__ = __cast(long) [static], [private]

Definition at line 141 of file entries.py.

7.80.3.7 tuple __oct__ = __cast(oct) [static], [private]

Definition at line 139 of file entries.py.

7.80.3.8 string _default_comments = "" [static], [private]

does not appear b/c it's False

Definition at line 46 of file entries.py.

7.80.3.9 string _default_dimensions = "" [static], [private]

ditto

Definition at line 48 of file entries.py.

7.80.3.10 string _default_element = "" [static], [private]

ditto

Definition at line 50 of file entries.py.

7.80.3.11 string _default_units = "-" [static], [private]

(-) appears as default

Definition at line 42 of file entries.py.

7.80.3.12 default_units = _default_units [static]

non-private version: it is used in units.py

Definition at line 44 of file entries.py.

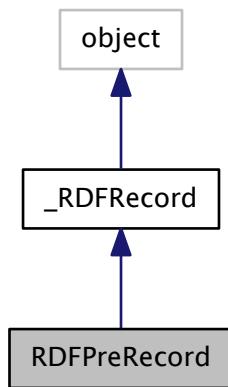
The documentation for this class was generated from the following file:

- rdf/data/entries.py

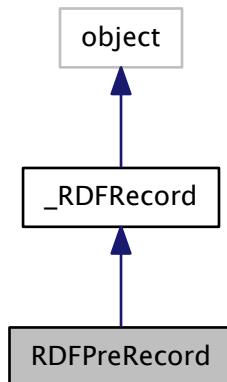
7.81 RDFPreRecord Class Reference

The pre Record is built from data and is a len=1 iterator: iterating builds the final product: [RDFRecord](#)— thus line reads or include file reads yield the same (polymorphic) result: iterators that yield Records.

Inheritance diagram for RDFPreRecord:



Collaboration diagram for RDFPreRecord:



Public Member Functions

- def [__iter__](#)
Iteration does not traverse (key, field), it:

Additional Inherited Members

7.81.1 Detailed Description

The pre Record is built from data and is a len=1 iterator: iterating builds the final product: [RDFRecord](#)— thus line reads or include file reads yield the same (polymorphic) result: iterators that yield Records.

Users should not see this class

Definition at line 172 of file entries.py.

7.81.2 Member Function Documentation

7.81.2.1 def __iter__(self)

Iteration does not traverse (key, field), it:

Returns

A generator that yields [RDFRecord](#) iter(RDFPreRecord)

Definition at line 177 of file entries.py.

References [_RDFRecord.__slots__](#).

```

177
178     def __iter__(self):
179         yield RDFRecord(*getattr(self, attr) for attr in self.
180                         __slots__)
  
```

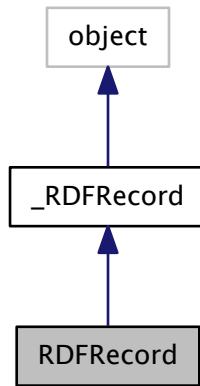
The documentation for this class was generated from the following file:

- rdf/data/[entries.py](#)

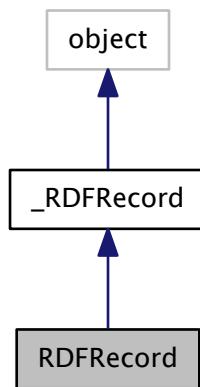
7.82 RDFRecord Class Reference

This is a fully parsed RDF record, and is an [_RDFRecord](#) with a formattable string.

Inheritance diagram for RDFRecord:



Collaboration diagram for RDFRecord:



Public Member Functions

- def [__iter__](#)

```
(key, field)
• def __int__
    int is the position of the rdf.reserved.OPERATOR (for formatting)
• def __str__
    FORMAT CONTROL TBD.
```

Additional Inherited Members

7.82.1 Detailed Description

This is a fully parsed RDF record, and is an [_RDFRecord](#) with a formatable string.

```
RDFRecord(key, field)

is the parsed RDF file line. Key is a string (or else), and
field is an RDFField.
```

Definition at line 183 of file entries.py.

7.82.2 Member Function Documentation

7.82.2.1 def __int__(self)

int is the position of the [rdf.reserved.OPERATOR](#) (for formatting)

Definition at line 194 of file entries.py.

```
194
195     def __int__(self):
196         return str(self).index(reserved.OPERATOR)
```

7.82.2.2 def __iter__(self)

(key, field)

Definition at line 190 of file entries.py.

References [_RDFRecord.__slots__](#).

```
190
191     def __iter__(self):
192         return iter((getattr(self, attr) for attr in self.__slots__))
```

7.82.2.3 def __str__(self, index = 0)

FORMAT CONTROL TBD.

```
place OPERATOR at index or don't
```

Definition at line 198 of file entries.py.

References [_RDFRecord.key](#).

```
198
199     def __str__(self, index=0):
200         """place OPERATOR at index or don't"""
201         key = str(self.key)
202         field = self.field.__str__(max(0, index-len(key)))
203         return key + field
204
```

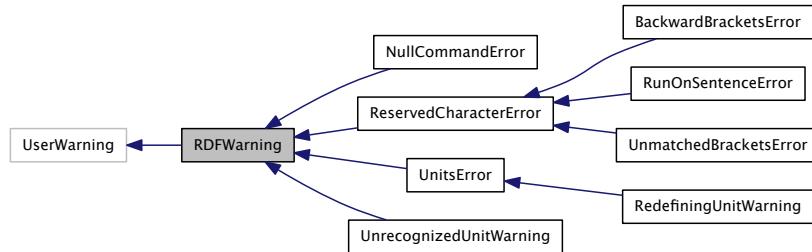
The documentation for this class was generated from the following file:

- [rdf/data/entries.py](#)

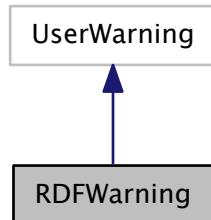
7.83 RDFWarning Class Reference

RDF Warning of INPUT problems.

Inheritance diagram for RDFWarning:



Collaboration diagram for RDFWarning:



7.83.1 Detailed Description

RDF Warning of INPUT problems.

`Base RDF Warning for bad RDF input grammar`

Definition at line 11 of file `errors.py`.

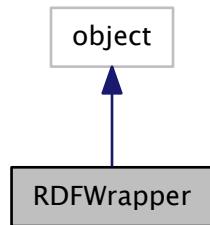
The documentation for this class was generated from the following file:

- [rdf/language/errors.py](#)

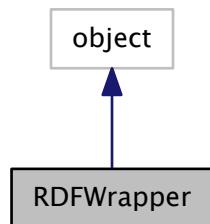
7.84 RDFWrapper Class Reference

A generic base class for RDF wrapped data structures -clients should use this when they have an object with RDF dependency injection and then further behavior as defined by the sub-classes methods.

Inheritance diagram for RDFWrapper:



Collaboration diagram for RDFWrapper:



Public Member Functions

- def [__init__](#)
Initialized with an RDF instance.
- def [rdf](#)
self.rdf == self.rdf() == self._rdf
- def [__getitem__](#)
Access rdf dictionary.
- def [__setitem__](#)
Access rdf dictionary.
- def [__delitem__](#)
Access rdf dictionary.
- def [__len__](#)
Access rdf dictionary.

Private Attributes

- [_rdf](#)
The wrapped rdf.

7.84.1 Detailed Description

A generic base class for RDF wrapped data structures -clients should use this when they have an object with RDF dependency injection and then further behavior as defined by the sub-classes methods.

```
RDFWrapper(rdf instance):
    is a base class for classes that wrap rdf instances.
```

Definition at line 8 of file eRDF.py.

7.84.2 Constructor & Destructor Documentation

7.84.2.1 def __init__(self, rdf_)

Initialized with an RDF instance.

Parameters

<code>rdf_</code>	a bonafide <code>rdf.data.files.RDF</code> object
-------------------	---

Definition at line 15 of file eRDF.py.

```
15
16     def __init__(self, rdf_):
17         ## The wrapped rdf
18         self._rdf = rdf_
19         return None
```

7.84.3 Member Function Documentation

7.84.3.1 def __delitem__(self, key)

Access rdf dictionary.

Definition at line 34 of file eRDF.py.

```
34
35     def __delitem__(self, key):
36         return self._rdf.__delitem__(self, key)
```

7.84.3.2 def __getitem__(self, key)

Access rdf dictionary.

Definition at line 26 of file eRDF.py.

```
26
27     def __getitem__(self, key):
28         return self._rdf.__getitem__(self, key)
```

7.84.3.3 def __len__(self, key)

Access rdf dictionary.

Definition at line 38 of file eRDF.py.

References `RDFWrapper._rdf`.

```
38
39     def __len__(self, key):
40         return len(self._rdf)
41
```

7.84.3.4 def __setitem__(self, key, field)

Access rdf dictionary.

Definition at line 30 of file eRDF.py.

```
30      def __setitem__(self, key, field):
31          return self._rdf.__setitem__(self, key, field)
32
```

7.84.3.5 def rdf(self)

```
self.rdf == self.rdf() == self._rdf
```

Definition at line 22 of file eRDF.py.

References RDFWrapper._rdf.

```
22      def rdf(self):
23          return self._rdf
24
```

7.84.4 Member Data Documentation**7.84.4.1 _rdf [private]**

The wrapped rdf.

Definition at line 17 of file eRDF.py.

Referenced by RDFWrapper.__len__(), and RDFWrapper.rdf().

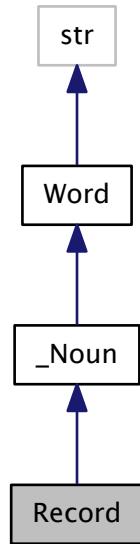
The documentation for this class was generated from the following file:

- rdf/eRDF.py

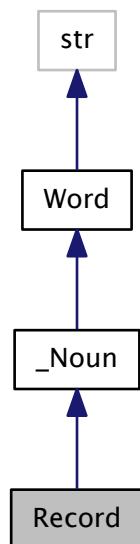
7.85 Record Class Reference

The [Record](#) Noun processes the basic input: An RDF line.

Inheritance diagram for Record:



Collaboration diagram for Record:



Static Public Member Functions

- def `concrete`

act uses `RDFField.__radd__` to build some form of an `_RDFRecord` (we don't know what that is here, no should we).

Static Private Attributes

- `_operator_in_line = True`

Additional Inherited Members

7.85.1 Detailed Description

The `Record` Noun processes the basic input: An RDF line.

Definition at line 50 of file `semantics.py`.

7.85.2 Member Function Documentation

7.85.2.1 def `concrete(prosodic) [static]`

act uses `RDFField.__radd__` to build some form of an `_RDFRecord` (we don't know what that is here, no should we).

return a iterable with 1 `RDFRecords`

Definition at line 57 of file `semantics.py`.

```
57
58     def concrete(prosodic):
59         """return a iterable with 1 RDFRecords"""
60         return prosodic.extract_record()
61
```

7.85.3 Member Data Documentation

7.85.3.1 `_operator_in_line = True [static], [private]`

Definition at line 52 of file `semantics.py`.

Referenced by `_Noun.is_()`.

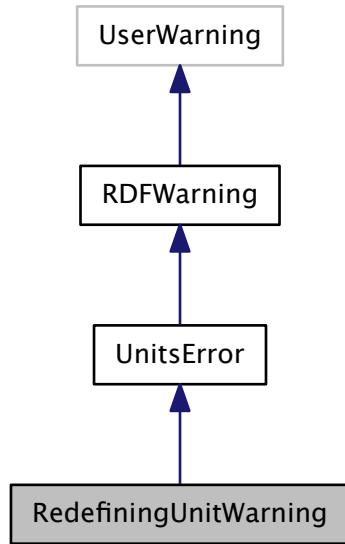
The documentation for this class was generated from the following file:

- `rdf/language/lexis/semantics.py`

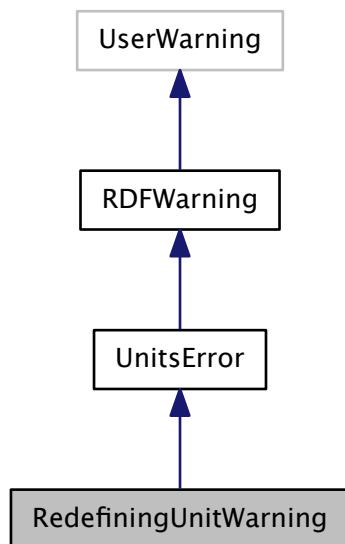
7.86 RedefiningUnitWarning Class Reference

Warning when you overwrite a unit.

Inheritance diagram for RedefiningUnitWarning:



Collaboration diagram for RedefiningUnitWarning:



7.86.1 Detailed Description

Warning when you overwrite a unit.

Redefining Units

Definition at line 22 of file errors.py.

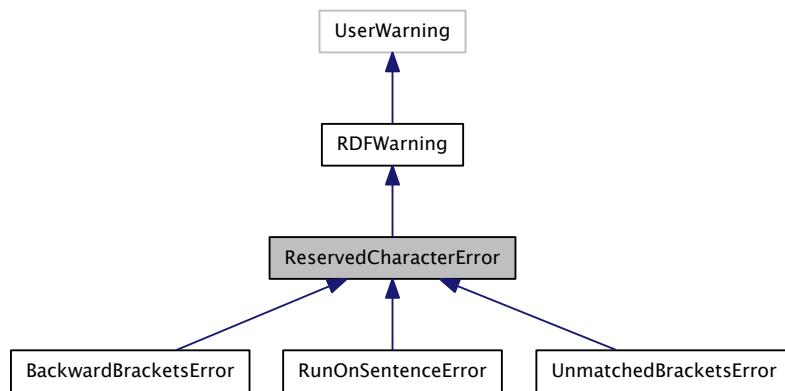
The documentation for this class was generated from the following file:

- [rdf/units/errors.py](#)

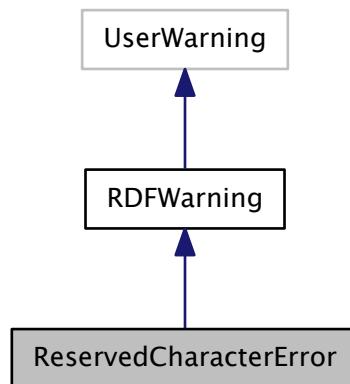
7.87 ReservedCharacterError Class Reference

Error for using a character in RESERVED.

Inheritance diagram for ReservedCharacterError:



Collaboration diagram for ReservedCharacterError:



7.87.1 Detailed Description

Error for using a character in RESERVED.

Error for using a RESERVED character badly

Definition at line 23 of file errors.py.

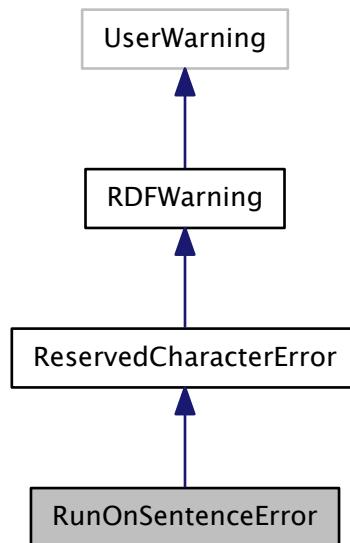
The documentation for this class was generated from the following file:

- [rdf/language/errors.py](#)

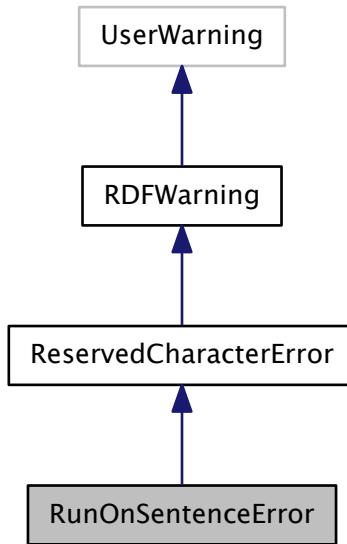
7.88 RunOnSentenceError Class Reference

Unmatched or un parsable pairs.

Inheritance diagram for RunOnSentenceError:



Collaboration diagram for RunOnSentenceError:



7.88.1 Detailed Description

Unmatched or un parsable pairs.

Too many punctuation marks

Definition at line 33 of file errors.py.

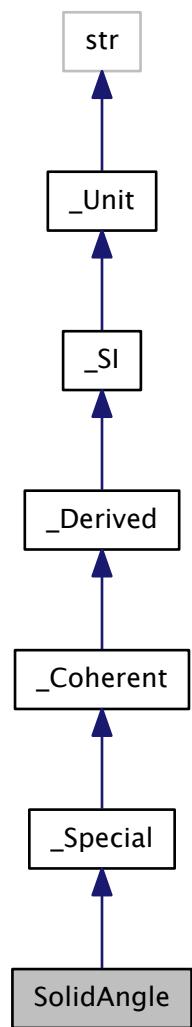
The documentation for this class was generated from the following file:

- rdf/language/[errors.py](#)

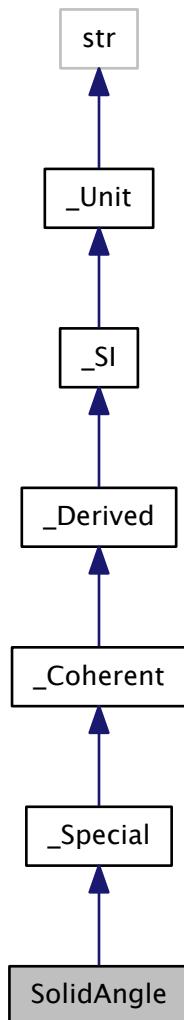
7.89 SolidAngle Class Reference

Steradians.

Inheritance diagram for SolidAngle:



Collaboration diagram for SolidAngle:



Static Private Attributes

- string symbol = 'sr'

Additional Inherited Members

7.89.1 Detailed Description

Steradians.

Definition at line 237 of file physical_quantity.py.

7.89.2 Member Data Documentation

7.89.2.1 `string _symbol = 'sr'` [static], [private]

Definition at line 238 of file `physical_quantity.py`.

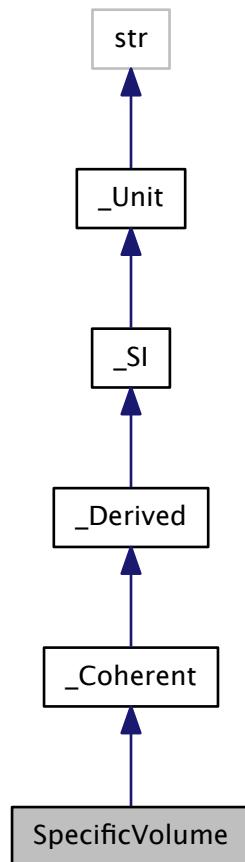
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

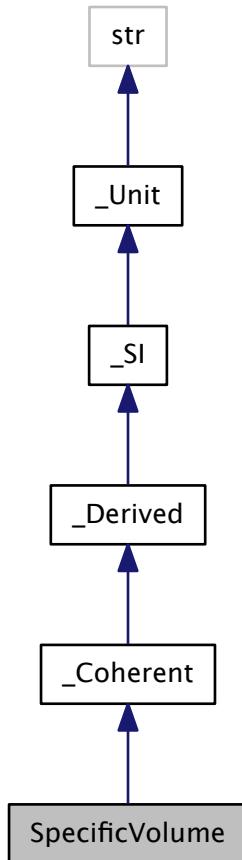
7.90 SpecificVolume Class Reference

$$\rho^{-1}$$

Inheritance diagram for `SpecificVolume`:



Collaboration diagram for SpecificVolume:



Static Private Attributes

- string `_symbol` = 'm**3/kg'

Additional Inherited Members

7.90.1 Detailed Description

$$\rho^{-1}$$

Definition at line 414 of file `physical_quantity.py`.

7.90.2 Member Data Documentation

7.90.2.1 string `_symbol` = 'm**3/kg' [static], [private]

Definition at line 415 of file `physical_quantity.py`.

The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

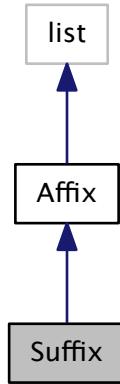
7.91 Suffix Class Reference

Appears After the stem.

Inheritance diagram for Suffix:



Collaboration diagram for Suffix:



Public Member Functions

- def [__radd__](#)
stem + prefix (overrides list concatenation)

7.91.1 Detailed Description

Appears After the stem.

```
stem + suffix  
is the only allowed operator overload- it, by definition, must  
be appended
```

Definition at line 80 of file morpheme.py.

7.91.2 Member Function Documentation

7.91.2.1 def __radd__(self, stem)

stem + prefix (overrides list concatenation)

Definition at line 86 of file morpheme.py.

```
86  
87     def __radd__(self, stem):  
88         return stem + self()
```

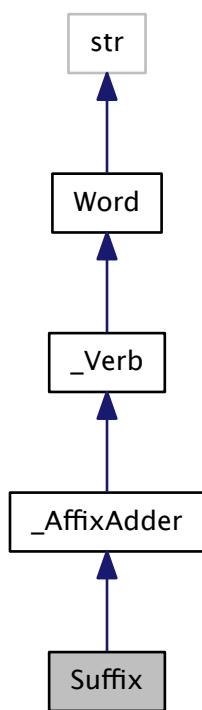
The documentation for this class was generated from the following file:

- rdf/language/grammar/morpheme.py

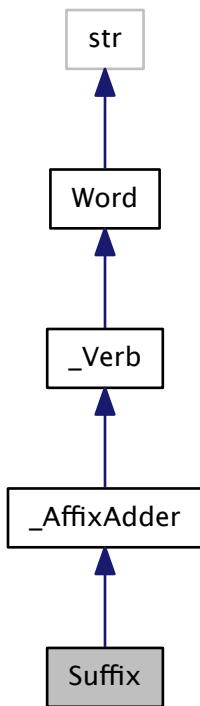
7.92 Suffix Class Reference

Add to [rdf.language.grammar.syntax.Grammar.suffix](#) via [_AffixAdder.act](#).

Inheritance diagram for Suffix:



Collaboration diagram for Suffix:



Additional Inherited Members

7.92.1 Detailed Description

Add to [rdf.language.grammar.syntax.Grammar.suffix](#) via `_AffixAdder.act`.

Suffix is an `_AffixAdder` that cooperates with `Gramar.suffix`

Definition at line 98 of file `pragmatics.py`.

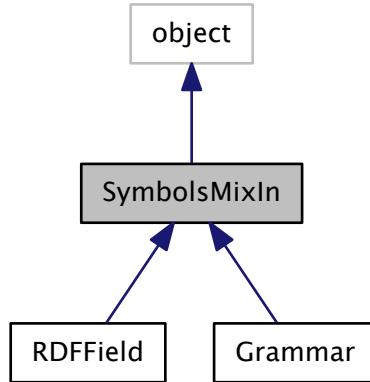
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

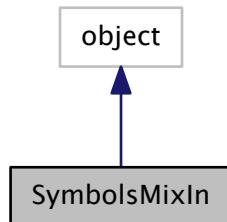
7.93 SymbolsMixin Class Reference

For classes that need to know about OPERATOR and COMMENT (statically).

Inheritance diagram for SymbolsMixIn:



Collaboration diagram for SymbolsMixIn:



Static Public Attributes

- **Operator** = OPERATOR
The operator symbol (default) -capitalized to avoid class with property.
- **Comment** = COMMENT
The comment symbol (default) -capitalized to avoid class with property.

7.93.1 Detailed Description

For classes that need to know about OPERATOR and COMMENT (statically).

Definition at line 29 of file reserved.py.

7.93.2 Member Data Documentation

7.93.2.1 Comment = COMMENT [static]

The comment symbol (default) -capitalized to avoid class with property.

Definition at line 33 of file reserved.py.

Referenced by RDFField.right_field().

7.93.2.2 Operator = OPERATOR [static]

The operator symbol (default) -capitalized to avoid class with property.

Definition at line 31 of file reserved.py.

Referenced by RDFField.__str__().

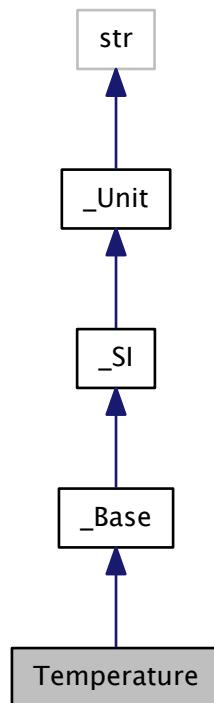
The documentation for this class was generated from the following file:

- rdf/reserved.py

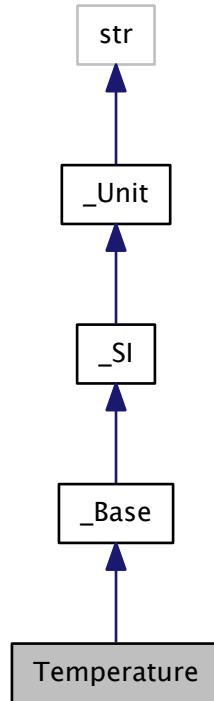
7.94 Temperature Class Reference

William Thomson, 1st Baron Kelvin OM, GCVO, PC, PRS, PRSE, (26 June 1824 – 17 December 1907)

Inheritance diagram for Temperature:



Collaboration diagram for Temperature:



Static Private Attributes

- string `_symbol` = 'K'

Additional Inherited Members

7.94.1 Detailed Description

William Thomson, 1st Baron Kelvin OM, GCVO, PC, PRS, PRSE, (26 June 1824 – 17 December 1907)

Definition at line 212 of file `physical_quantity.py`.

7.94.2 Member Data Documentation

7.94.2.1 string `_symbol` = 'K' [static], [private]

Definition at line 213 of file `physical_quantity.py`.

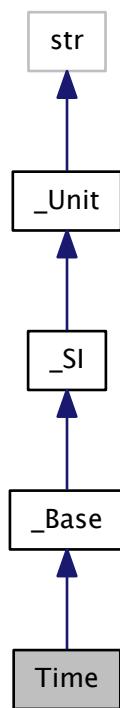
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

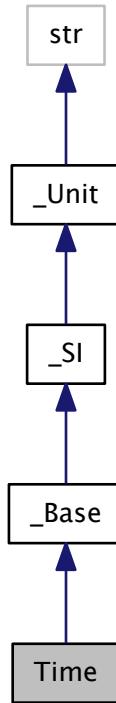
7.95 Time Class Reference

[Time](#) conversion to seconds.

Inheritance diagram for Time:



Collaboration diagram for Time:



Static Private Attributes

- string [_symbol](#) = 's'

Additional Inherited Members

7.95.1 Detailed Description

[Time](#) conversion to seconds.

Second

Definition at line 197 of file [physical_quantity.py](#).

7.95.2 Member Data Documentation

7.95.2.1 string [_symbol](#) = 's' [static], [private]

Definition at line 199 of file [physical_quantity.py](#).

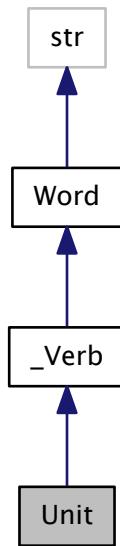
The documentation for this class was generated from the following file:

- [rdf/units/physical_quantity.py](#)

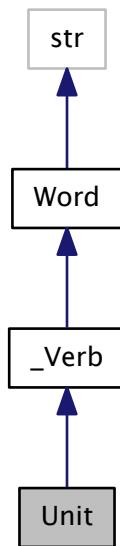
7.96 Unit Class Reference

Add to an rdf.units.physical_quantity.Unit to the [rdf.units.GLOSSARY](#) via `rdf.prosodic.Prosodic.set_unit`.

Inheritance diagram for Unit:



Collaboration diagram for Unit:



Public Member Functions

- def `act`

7.96.1 Detailed Description

Add to an `rdf.units.physical_quantity.Unit` to the [rdf.units.GLOSSARY](#) via `rdf.prosodic.Prosodic.set_unit`.

The idea:

```
UNIT = ('km', 1000, symbol='m')
will load another unit in the GLOSSARY
```

Definition at line 104 of file `pragmatics.py`.

7.96.2 Member Function Documentation

7.96.2.1 def `act(self, prosodic)`

create a unit

Definition at line 111 of file `pragmatics.py`.

```
111
112     def act(self, prosodic):
113         """create a unit"""
114         return prosodic.set_unit()
115
```

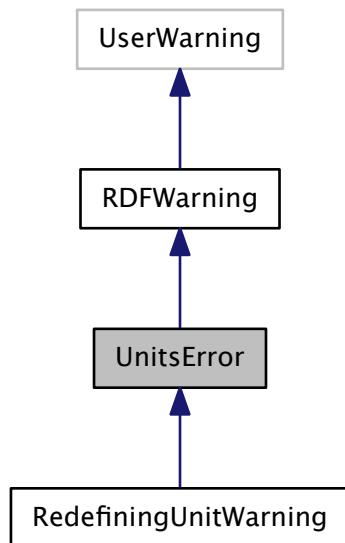
The documentation for this class was generated from the following file:

- [rdf/language/lexis/pragmatics.py](#)

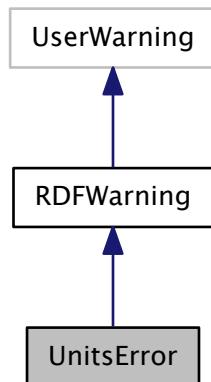
7.97 UnitsError Class Reference

RDF Error for a unit problem (not sure what kind of error this is)

Inheritance diagram for UnitsError:



Collaboration diagram for UnitsError:



7.97.1 Detailed Description

RDF Error for a unit problem (not sure what kind of error this is)

Raised for a non-existent unit

Definition at line 12 of file errors.py.

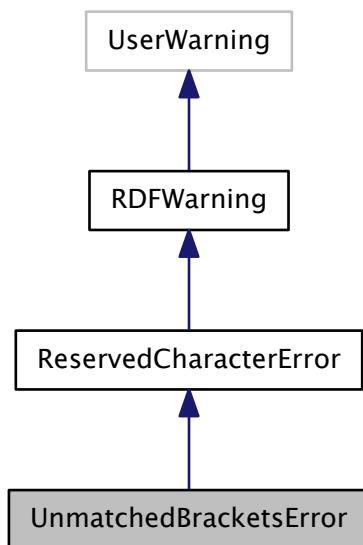
The documentation for this class was generated from the following file:

- [rdf/units/errors.py](#)

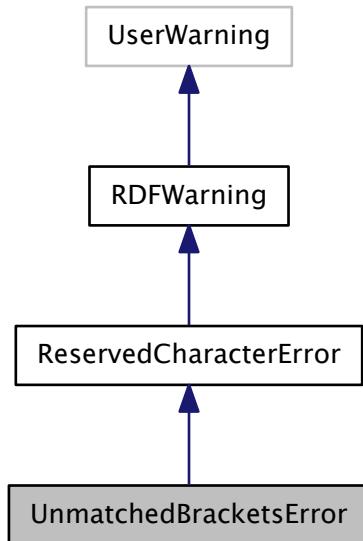
7.98 UnmatchedBracketsError Class Reference

Unmatched or un parseable pairs.

Inheritance diagram for UnmatchedBracketsError:



Collaboration diagram for UnmatchedBracketsError:



7.98.1 Detailed Description

Unmatched or un parsable pairs.

1/2 a delimiter was used

Definition at line 28 of file `errors.py`.

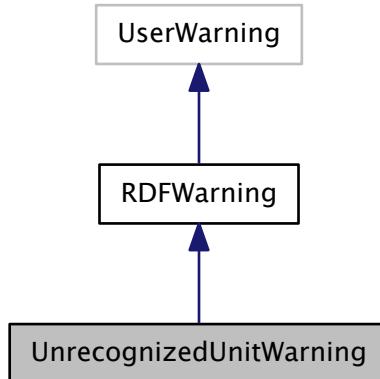
The documentation for this class was generated from the following file:

- `rdf/language/errors.py`

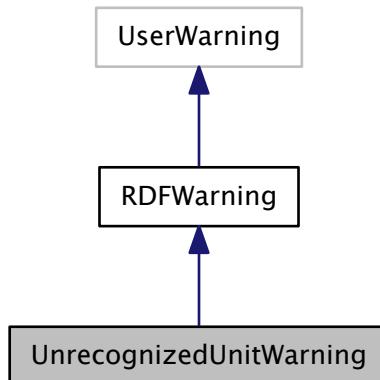
7.99 UnrecognizedUnitWarning Class Reference

Error for input w/ unknown units.

Inheritance diagram for UnrecognizedUnitWarning:



Collaboration diagram for UnrecognizedUnitWarning:



7.99.1 Detailed Description

Error for input w/ unknown units.

Unrecognized unit (ignored)

Definition at line 17 of file `errors.py`.

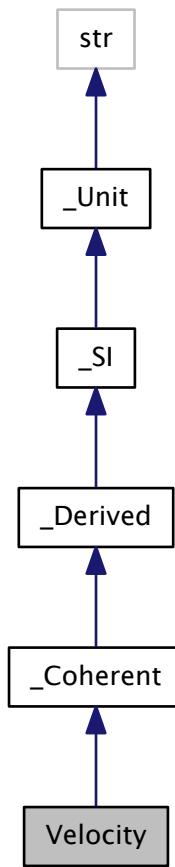
The documentation for this class was generated from the following file:

- [rdf/units/errors.py](#)

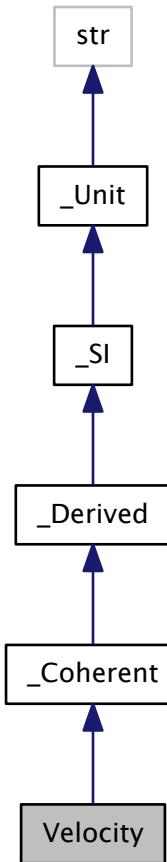
7.100 Velocity Class Reference

meters per second

Inheritance diagram for Velocity:



Collaboration diagram for Velocity:



Static Private Attributes

- string `_symbol` = 'm/s'

Additional Inherited Members

7.100.1 Detailed Description

meters per second

Definition at line 377 of file physical_quantity.py.

7.100.2 Member Data Documentation

7.100.2.1 string `_symbol` = 'm/s' [static], [private]

Definition at line 378 of file physical_quantity.py.

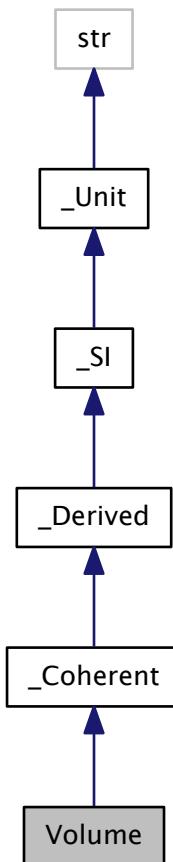
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

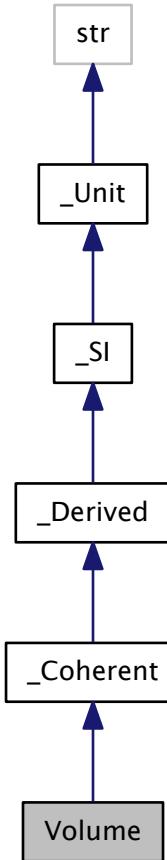
7.101 Volume Class Reference

meters cubed

Inheritance diagram for Volume:



Collaboration diagram for Volume:



Static Private Attributes

- string `_symbol` = 'm***3'

Additional Inherited Members

7.101.1 Detailed Description

meters cubed

Definition at line 371 of file physical_quantity.py.

7.101.2 Member Data Documentation

7.101.2.1 string `_symbol` = 'm***3' [static], [private]

Definition at line 372 of file physical_quantity.py.

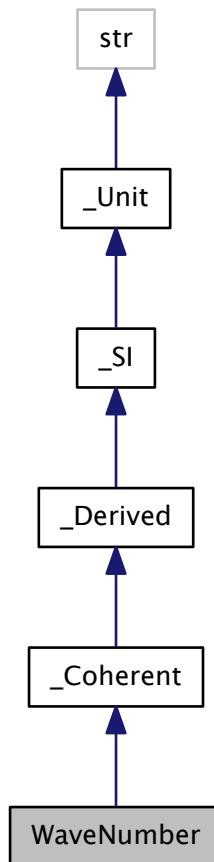
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

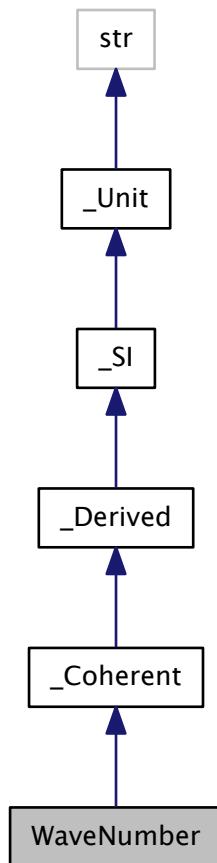
7.102 WaveNumber Class Reference

Inverse meter.

Inheritance diagram for WaveNumber:



Collaboration diagram for WaveNumber:



Static Private Attributes

- string _symbol = 'm**-1'

Additional Inherited Members

7.102.1 Detailed Description

Inverse meter.

Definition at line 396 of file physical_quantity.py.

7.102.2 Member Data Documentation

7.102.2.1 string _symbol = 'm**-1' [static], [private]

Definition at line 397 of file physical_quantity.py.

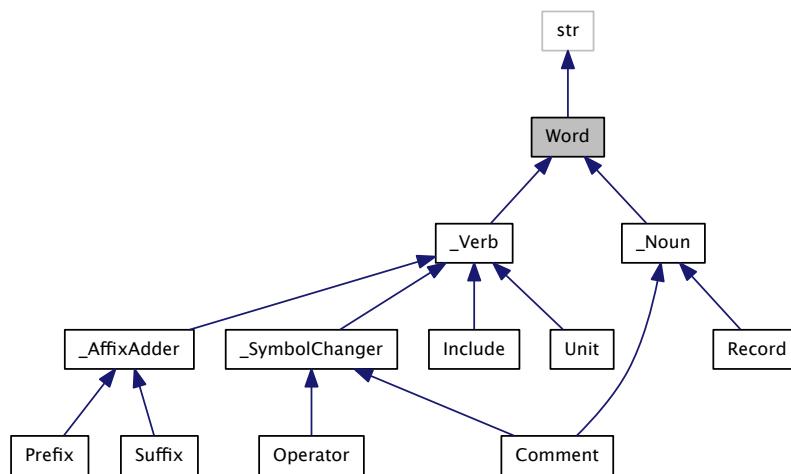
The documentation for this class was generated from the following file:

- rdf/units/physical_quantity.py

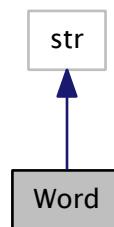
7.103 Word Class Reference

The Pragamtic's are RDF lines meaning.

Inheritance diagram for Word:



Collaboration diagram for Word:



Public Member Functions

- def `is_not`
Negative form for running `itertools.dropwhile` TODO: understand neg v.
- def `__call__`
- def `sin_qua_non`

Static Private Attributes

- `__metaclass__` = abc.ABCMeta

7.103.1 Detailed Description

The Pragamtic's are RDF lines meaning.

Word is an ABC that subclasses str. It has a call that dynamically dispatches args = (line, grammar) to the sub classes' sin qua non method-- which is the method that allows them to do their business.

Definition at line 6 of file __init__.py.

7.103.2 Member Function Documentation

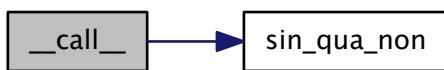
7.103.2.1 def __call__(self, prosodic)

Definition at line 22 of file __init__.py.

References Word.sin_qua_non(), and _Noun.sin_qua_non.

```
22
23     def __call__(self, prosodic):
24         return self.sin_qua_non(prosodic)
```

Here is the call graph for this function:



7.103.2.2 def is_not(self, prosodic)

Negative form for running itertools.dropwhile TODO: undetstand neg v.

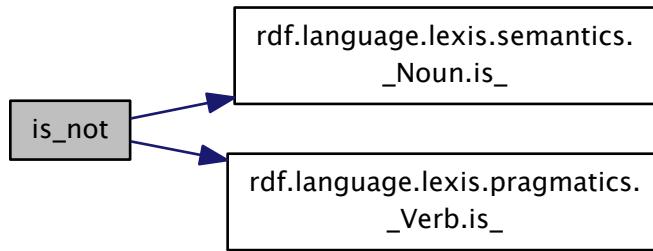
pos

Definition at line 17 of file __init__.py.

References _Noun.is_(), and _Verb.is_().

```
17
18     def is_not(self, prosodic):
19         return self.is_(prosodic)
```

Here is the call graph for this function:



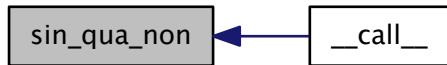
7.103.2.3 def sin_qua_non (self, prosodic)

Definition at line 26 of file `__init__.py`.

Referenced by `Word.__call__()`.

```
26
27     def sin_qua_non(self, prosodic):
28         pass
```

Here is the caller graph for this function:



7.103.3 Member Data Documentation

7.103.3.1 `__metaclass__ = abc.ABCMeta` [static], [private]

Definition at line 13 of file `__init__.py`.

The documentation for this class was generated from the following file:

- [rdf/language/lexis/`__init__.py`](#)

8 File Documentation

8.1 `rdf/__init__.py` File Reference

Namespaces

- [rdf](#)

The rdf package.

Functions

- def `test`

Variables

- `rdfparse` = `rdf_reader`
Backwards compatible rdf readers.
- `parse` = `rdf_reader`
less redundant parser

8.2 rdf/data/__init__.py File Reference

Namespaces

- `rdf.data`
RDF The final form for lines and files.

8.3 rdf/language/__init__.py File Reference

Namespaces

- `rdf.language`
The Language: grammar + lexis = language mod prosodic.

8.4 rdf/language/grammar/__init__.py File Reference

Namespaces

- `rdf.language.grammar`
Grammar (language - lexis)

8.5 rdf/language/lexis/__init__.py File Reference

Classes

- class `Word`
The Pragamtic's are RDF lines meaning.

Namespaces

- `rdf.language.lexis`
The Lexis comprises the words in the language.

8.6 rdf/test/__init__.py File Reference

Namespaces

- `rdf.test`
A brief test suite.

Functions

- def **diff**
Run diff on tested files.
- def **main**
`rdf.parse(SRC) >> DST`

Variables

- string **SRC** = "rdf.txt"
Top of the nested desting chain.
- string **DST** = "new.rdf"
File made by script.
- string **STANDARD** = "old.rdf"
Result of a sucessfile run.

8.7 rdf/units/__init__.py File Reference

Namespaces

- **rdf.units**
Units according to <http://physics.nist.gov/cuu/pdf/sp811.pdf>.

Functions

- def **SI**
Convert (value, units) to SI pair - this is the interface to RDField Search various places for units...(TBD).

Variables

- **GLOSSARY** = _Unit.Glossary
The global unit glossary dictionary:[symbol]->converter function.

8.8 rdf/data/entries.py File Reference

Classes

- class **RDFField**
Add methods and constants to _RDFField so that it lives up to its name.
- class **_RDFRecord**
Base for iterable (key, field) pair.
- class **RDFPreRecord**
*The pre Record is built from data and is a len=1 iterator: iterating builds the final product: **RDFRecord**– thus line reads or include file reads yield the same (polymorphic) result: iterators that yield Records.*
- class **RDFRecord**
*This is a fully parsed RDF record, and is an **_RDFRecord** with a formatable string.*
- class **RDFComment**
The RDF Comment is a comment string, endowed with False RDF-ness and null response to iteration.

Namespaces

- `rdf.data.entries`

Usable data objects for lines (records).

Functions

- `def __cast`

Decorator to cast values.

Variables

- string `SPACE` = " "
- tuple `_RDFField`

Base RDF Field named tuple -it's all in here -note, it's assigned (public) name differs from its variable (private) name, so that users never need to know about this private assignment.

- tuple `_RDFRecord` = `collections.namedtuple("RDFRecord", "key field")`

This assignment is a bit deeper: Just a key and a field.

8.9 rdf/data/files.py File Reference

Classes

- class `RDF`

An `RDF` Mothership: A fully interpreted `RDF` File.

Namespaces

- `rdf.data.files`

Usable data object for files.

Functions

- `def stripper`

Decorator to cast and strip arguments.

Variables

- `DICT` = `collections.OrderedDict`
- `WARN` = `False`

8.10 rdf/DOCS/mainpage.txt File Reference

8.11 rdf/eRDF.py File Reference

Classes

- class `RDFWrapper`

A generic base class for RDF wrapped data structures -clients should use this when they have an object with RDF dependency injection and then further behavior as defined by the sub-classes methods.

Namespaces

- `rdf.eRDF`
*e*xperimental RDF objects

Functions

- `def debug`
Use to find problem lines when developing.
- `def factor`
Experimental function to factor keys and rdf in least form.

8.12 rdf/iRDF.py File Reference

Classes

- `class rdf_list`
A list of rdf records that can filter itself and make an RDF.
- `class _RDFAccess`
Base class.
- `class RDFAnalyzer`
RDFAnalyzer is created with an rdf.language.syntax.Grammar object and then emulates a function that converts single line inputs into a pre-RDF output -note: it is overly complicated, and its sole purpose is to provide an interactive RDF reader.
- `class RDFAccumulator`
A TBD rdf accumulator - prolly slower than RDFAnalyzer as it rebuild dictionary all the time— see RDF._iadd_.

Namespaces

- `rdf.iRDF`
*i*nteractive RDF tools for 'perts.

Functions

- `def test`

Variables

- `tuple __all__`
The RDF Toybox.

8.13 rdf/language/errors.py File Reference

Classes

- `class RDFError`
Fatal attempt to CODE badly.
- `class RDFWarning`
RDF Warning of INPUT problems.
- `class MorphemeExchangeError`
Morphere Exchange Currents?

- class [ReservedCharacterError](#)
Error for using a character in RESERVED.
- class [UnmatchedBracketsError](#)
Unmatched or un parsable pairs.
- class [RunOnSentenceError](#)
Unmatched or un parsable pairs.
- class [BackwardBracketsError](#)
Unmatched or un parsable pairs.
- class [NullCommandError](#)
Should be thrown in constructor?

Namespaces

- [rdf.language.errors](#)
RDF Language Exceptions.

8.14 rdf/units/errors.py File Reference

Classes

- class [FatalUnitError](#)
Fatal error for unknown units (not really an acceptable idea)
- class [UnitsError](#)
RDF Error for a unit problem (not sure what kind of error this is)
- class [UnrecognizedUnitWarning](#)
Error for input w/ unknown units.
- class [RedefiningUnitWarning](#)
Warning when you overwrite a unit.

Namespaces

- [rdf.units.errors](#)
RDF Unit Errors.

8.15 rdf/language/grammar/morpheme.py File Reference

Classes

- class [Affix](#)
Abstract Base Class for Pre/Suf behavior.
- class [Prefix](#)
Appears Before the stem:
- class [Suffix](#)
Appears After the stem.

Namespaces

- [rdf.language.grammar.morpheme](#)
Key Changing Morphemes.

8.16 rdf/language/grammar/punctuation.py File Reference

Classes

- class **Glyph**
A character that knows how to find itself in strings.
- class **Brackets**
Brackets that know thyelves.

Namespaces

- **rdf.language.grammar.punctuation**
Language's Punctuation Marks.

Functions

- def **parse_left**
and RDF line (that is, left of OPERATOR)
- def **read_rdfkey**
Remove brackets from left line.
- def **read_brackets**
Read list of brackets from left line.
- def **write_brackets**
Pack bracket values into a string.

Variables

- tuple **UNITS** = Brackets(reserved.UNITS)
(units) Brackets
- tuple **DIMENSIONS** = Brackets(reserved.DIMENSIONS)
{dim} Brackets
- tuple **ELEMENT** = Brackets(reserved.ELEMENT)
[elements] Brackets

8.17 rdf/language/grammar/syntax.py File Reference

Classes

- class **lexicon**
Metaclass for Grammar gets defines the pragmatics and semantics at load-time" pragmatics. Verb instances are assigned according to the rdf.reserved.words.KEYWORDS, and the symantics.Noun instances are created- these are needed by Grammar.process.
- class **Grammar**
Grammar tracks the state of grammar, and uses it to dispatch work when it is called.

Namespaces

- **rdf.language.grammar.syntax**
Syntax glues it all together.

Functions

- def `null_command_watch`
decorate setters to prevent setting required commands to NULL
- def `unit_change`
decorate incremental ops to only allow +1 change

8.18 rdf/language/lexis/pragmatics.py File Reference

Classes

- class `_Verb`
Verbs are `rdf.language.lexis.Word` objects that "act" as their `sin_qua_non`.
- class `Include`
Open an `Include` File.
- class `_SymbolChanger`
ABC sends sub's class.`__name__.lower()` to `rdf.language.prosodic.change_symbol`.
- class `Operator`
Change `rdf.language.grammar.syntax.Grammar.operator` via `_SymbolChangrr.act`.
- class `Comment`
Change `rdf.language.grammar.syntax.Grammar.comment` via `_SymbolChangrr.act`.
- class `_AffixAdder`
ABC sends sub's class name to `rdf.language.prosodic.set_affix`.
- class `Prefix`
Add to `rdf.language.grammar.syntax.Grammar.prefix` via `_AffixAdder.act`.
- class `Suffix`
Add to `rdf.language.grammar.syntax.Grammar.suffix` via `_AffixAdder.act`.
- class `Unit`
Add to an `rdf.units.physical_quantity.Unit` to the `rdf.units.GLOSSARY` via `rdf.prosodic.Prosodic.set_unit`.

Namespaces

- `rdf.language.lexis.pragmatics`
Words with (reflexive) meaning (Verb)

Variables

- tuple `VERBS` = (`Include`, `Operator`, `Comment`, `Prefix`, `Suffix`, `Unit`)
Verb classes are auto instantiated by `rdf.language.grammar.syntax.lexicon` metaclass.

8.19 rdf/language/lexis/semantics.py File Reference

Classes

- class `_Noun`
Nouns are `rdf.language.lexis.Word` objects that "concrete" as their `sin_qua_non`.
- class `Record`
The `Record` Noun processes the basic input: An RDF line.
- class `Comment`
The `Comment` Noun remembers passive comment lines.

Namespaces

- `rdf.language.lexis.semantics`
References to Things (Noun)

Variables

- tuple `NOUNS` = (Record, Comment)
Nouns.

8.20 rdf/language/prosodic.py File Reference

Classes

- class `Prosodic`

In Dependency Grammar, the clitic (an RDF line) depends on its host (the current grammar)– together they are a `Prosodic`– this is important because and RDF line on its own has no definite meaning- it must be interpreted in terms of its host Grammar, this tuple exists because the pair is passed all over the place.

Namespaces

- `rdf.language.prosodic`
Prosodics define a `Dependency Grammar`.

Variables

- `_27` = False

8.21 rdf/parse.py File Reference

Namespaces

- `rdf.parse`
RDF Parsing script.

Variables

- `pipe` = `sys.stderr`
- tuple `message` = `getattr(sys.modules['__name__'], '__doc__')`
- `EXIT` = `errno.EINVAL`
- list `argv` = `sys.argv[1:]`
- list `src` = `argv[-1]`

8.22 rdf/reserved.py File Reference

Classes

- class `SymbolsMixIn`
For classes that need to know about OPERATOR and COMMENT (statically).

Namespaces

- `rdf.reserved`

Reserved Symbols.

Variables

- string `OPERATOR` = "="

Meta word defining operator symbol.

- string `COMMENT` = "#"

Meta word defining comment symbol (depracted !)

- string `SEPARATOR` = ","

SEPARATOR Symbol.

- string `WRAP` = '/'

Line Wrap Symbol.

- string `CR` = '\n'

Carriage Return.

- string `UNITS` = '()'

Unit Delimiter ordered glyphs.

- string `DIMENSIONS` = '{}'

Dimensions wrapper ordered glyphs.

- string `ELEMENT` = '[]'

Element wrapper ordered glyphs.

- `SINGULAR` = OPERATOR+COMMENT

Singular characters, who's repetition constitutes an rdf.language.errors.ReservedCharacterError.

8.23 rdf/units/addendum.py File Reference

Namespaces

- `rdf.units.addendum`

Non metric and user units.

Variables

- tuple `LENGTHS`

Supported _Length conversions.

- tuple `MASSES` = (Mass('g', 0.001),)

- tuple `AREAS`

Supported _Area conversions.

- tuple `TIMES`

Supported _Time conversions.

- tuple `VELOCITES`

Supported _Velocity conversions.

- tuple `POWERS` = ()

- tuple `DBPOWERS` = (dBPower('dBm', adder=-30),)

Suppoerted dB Power.

- tuple `ENERGIES` = (Energy('BTU', 1055.056),)

- tuple `FREQUENCIES` = ()

Supported Frequency conversions.

8.24 rdf/units/physical_quantity.py File Reference

Classes

- class [_Unit](#)
The [_Unit](#) class memoizes its instances.
- class [_SI](#)
SI units.
- class [_Accepted](#)
Accepted non SI units.
- class [_UnAccepted](#)
Unaccepted units (If you must)
- class [_Base](#)
Base SI units.
- class [_Derived](#)
Derived DI units.
- class [_Coherent](#)
Coherent Derived SI Units.
- class [_Special](#)
Special Coherent Derived SI Units.
- class [Length](#)
Length conversion to meters.
- class [Mass](#)
Conversion to kilograms.
- class [Time](#)
Time conversion to seconds.
- class [ElectricCurrent](#)
Andre-Marie Ampère (22 January 1775 – 10 June 1836)
- class [Temperature](#)
William Thomson, 1st Baron Kelvin OM, GCVO, PC, PRS, PRSE, (26 June 1824 – 17 December 1907)
- class [AmountOfSubstance](#)
Lorenzo Romano Amedeo Carlo Avogadro di Quaregna e di Cerreto, Count of Quaregna and Cerreto (9 August 1776, Turin, Piedmont – 9 July 1856)
- class [LuminousIntensity](#)
Brightness.
- class [PlaneAngle](#)
Angle conversion to degrees.
- class [SolidAngle](#)
Steradians.
- class [Pressure](#)
Blaise Pascal (19 June 1623 – 19 August 1662)
- class [Frequency](#)
Heinrich Rudolf Hertz (22 February 1857 – 1 January 1894)
- class [Force](#)
Sir Isaac Newton PRS MP (25 December 1642 – 20 March 1727)
- class [Energy](#)
James Prescott Joule FRS (24 December 1818 – 11 October 1889)
- class [Power](#)
James Watts, FRS, FRSE (19 January 1736 – 25 August 1819)
- class [Charge](#)
Charles-Augustin de Coulomb (14 June 1736 – 23 August 1806)

- class [EMF](#)
Alessandro Giuseppe Antonio Anastasio Volta (18 February 1745 – 5 March 1827)
- class [Capacitance](#)
Michael Faraday, FRS (22 September 1791 – 25 August 1867)
- class [ElectricalResistance](#)
Georg Simon Ohm (16 March 1789 – 6 July 1854)
- class [ElectricalConductance](#)
Ernst Werner Siemens, von Siemens since 1888, (13 December 1816 – 6 December 1892)
- class [MagneticFlux](#)
Wilhelm Eduard Weber (24 October 1804 – 23 June 1891)
- class [MagneticFluxDensity](#)
Nikola Tesla (10 July 1856 – 7 January 1943)
- class [Inductance](#)
Joseph Henry (December 17, 1797 – May 13, 1878)
- class [CelsiusTemperatue](#)
Anders Celsius (27 November 1701 – 25 April 1744)
- class [LuminouFlux](#)
lumen
- class [Illuminance](#)
lux
- class [Activity](#)
Antoine Henri Becquerel (15 December 1852 – 25 August 1908)
- class [AbsorbedDose](#)
Louis Harold Gray (10 November 1905 – 9 July 1965)
- class [DoseEquivalent](#)
Professor Rolf Maximilian Sievert (6 May 1896 – 3 October 1966)
- class [CatalyticActivity](#)
katal
- class [Area](#)
meters squared
- class [Volume](#)
meters cubed
- class [Velocity](#)
meters per second
- class [MomentOfForce](#)
Torque.
- class [dBPower](#)
decibel Power -is not power- it's just a number: (Alexander Graham Bell (March 3, 1847 – August 2, 1922))
- class [WaveNumber](#)
Inverse meter.
- class [Acceleration](#)
meters per seconds squared
- class [Density](#)
 ρ
- class [SpecificVolume](#)
 ρ^{-1}
- class [CurrentDensity](#)
Amps per square meter.
- class [MagneticFieldStrength](#)
A/m.
- class [Luminance](#)

- cd/m**2*
- class [Concentration](#)
concentration
 - class [Bit](#)
Data Volume conversion to bits.
 - class [BitPerSecond](#)
Data rate conversion to bps.
 - class [Byte](#)
Data Volume conversion to bits.
 - class [BytesPerSecond](#)
Data rate conversion to bytes per second.
 - class [Ratio](#)
TBD.

Namespaces

- [rdf.units.physical_quantity](#)
Classes for Physical Quantities.

8.25 rdf/units/prefix.py File Reference

Classes

- class [Prefix](#)
Abstract Base class for class decorator factory that makes an instance of the decorated _Unit subclass according to the prefix and the power.
- class [MetricPrefix](#)
Metric Prefix.
- class [BinaryPrefix](#)
Binary Prefix Note: limits/differences of/between JEDEC and IEC

Namespaces

- [rdf.units.prefix](#)
Auto Prefix Handlers.

Functions

- def [metric](#)
Decorate All Metric Prefixes.
- def [jedec](#)
Decorate JEDEC prefixes.
- def [iec](#)
Decorate IEC prefixes.

Variables

- tuple `yotta` = MetricPrefix('Y', 24)
 10^{24}
- tuple `zetta` = MetricPrefix('Z', 21)
 10^{21}
- tuple `exa` = MetricPrefix('E', 18)
 10^{18}
- tuple `peta` = MetricPrefix('P', 15)
 10^{15}
- tuple `tera` = MetricPrefix('T', 12)
 10^{12}
- tuple `giga` = MetricPrefix('G', 9)
 10^9
- tuple `mega` = MetricPrefix('M', 6)
 10^6
- tuple `kilo` = MetricPrefix('k', 3)
 10^3
- tuple `hecto` = MetricPrefix('h', 2)
 10^2
- tuple `deca` = MetricPrefix('da', 1)
 10^1
- tuple `base` = MetricPrefix("", 0)

Trival (but it does create an instance and put it in [Unit.Glossary](#))
- tuple `deci` = MetricPrefix('d', -1)

10^{-1}
- tuple `centi` = MetricPrefix('c', -2)

10^{-2}
- tuple `milli` = MetricPrefix('m', -3)

10^{-3}
- tuple `micro` = MetricPrefix('u', -6)

10^{-6}
(NB: "u" is used instead of "\u" for typographical reasons)
- tuple `nano` = MetricPrefix('n', -9)

10^{-9}
- tuple `pico` = MetricPrefix('p', -12)

10^{-12}
- tuple `femto` = MetricPrefix('f', -15)

10^{-15}
- tuple `atto` = MetricPrefix('a', -18)

10^{-18}
- tuple `zepto` = MetricPrefix('z', -21)

10^{-21}
- tuple `yocto` = MetricPrefix('y', -24)

10^{-24}
- tuple `base2` = BinaryPrefix("", 0)

Trival (integer measurement)
- tuple `kilo2` = BinaryPrefix('k', 1)

2^{10} , JEDEC
- tuple `mega2` = BinaryPrefix('M', 2)

$(2^{10})^2$, JEDEC

- tuple **giga2** = BinaryPrefix('G', 3)
 $(2^{10})^3$, *JEDEC*
- tuple **kibi** = BinaryPrefix('Ki', 1)
 2^{10} , *IEC*
- tuple **mebi** = BinaryPrefix('Mi', 2)
 $(2^{10})^2$, *IEC*
- tuple **gibi** = BinaryPrefix('Gi', 3)
 $(2^{10})^3$, *IEC*
- tuple **tebi** = BinaryPrefix('Ti', 4)
 $(2^{10})^4$, *IEC*
- tuple **pebi** = BinaryPrefix('Pi', 5)
 $(2^{10})^5$, *IEC*
- tuple **exbi** = BinaryPrefix('Ei', 6)
 $(2^{10})^6$, *IEC*
- tuple **zebi** = BinaryPrefix('Zi', 7)
 $(2^{10})^7$, *IEC*
- tuple **yebi** = BinaryPrefix('Yi', 8)
 $(2^{10})^8$, *IEC*
- **yobi** = yebi

TBD: pick one.

8.26 rdf/uRDF.py File Reference

Namespaces

- **rdf.uRDF**
users' interface to language.py and data.py from rdf import read

Functions

- def **rdf_include**
The rdf_include function takes a src and rdf.language.syntax.Grammar object to go process the entirety of src- it is the sole controller of Grammar.depth, unpacking of _RDFRecord and lists of them- it deals with the recursion, etc.
- def **rdf_reader**
For src it's that simple.

8.27 rdf/utils.py File Reference

Namespaces

- **rdf.utils**
Non-RDF specific utilities.

Functions

- def **coroutine**
Coroutine Decorator, returns it kick started.
- def **cowrap**
Coroutine yields None, util a full line is ingested, then it's yielded.
- def **unwrap_file**

Return unwrapped lines from a file.

- def [parse_tuple_input](#)
A basic parsing of standard python 2x args and kwargs.
- def [dialog_pickfile](#)
open a file picking GUI