<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>What’s New?</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Compatibility</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>IMS Global Learning Consortium Specifications</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>The Open Data Protocol (OData)</td>
<td>119</td>
</tr>
<tr>
<td>5</td>
<td>Hypertext Transfer Protocol (RFC2616)</td>
<td>231</td>
</tr>
<tr>
<td>6</td>
<td>Other Supporting Standards</td>
<td>279</td>
</tr>
<tr>
<td>7</td>
<td>Welcome to Pyslet</td>
<td>461</td>
</tr>
<tr>
<td></td>
<td>Python Module Index</td>
<td>465</td>
</tr>
</tbody>
</table>
To improve PEP-8 compliance a number of name changes are being made to methods and class attributes with each release. There is a module, pyslet.pep8, which contains a compatibility class for remapping missing class attribute names to their new forms and generating deprecation warnings, run your code with “python -Wd” to force these warnings to appear. As Pyslet makes the transition to Python 3 some of the old names may go away completely. The warning messages explain any changes you need to make. Although backwards compatible, using the new names is slightly faster as they don’t go through the extra deprecation wrapper.

It is still possible that some previously documented names could now fail (module level functions, function arguments, etc.) but I’ve tried to include wrappers or aliases so please raise an issue on Github if you discover a bug caused by the renaming. I’ll restore any missing old-style names to improve backwards compatibility on request.

### 1.1 Version Numbering

Pyslet version numbers use the check-in date as their last component so you can always tell if one build is newer than another. At the moment there is only one actively maintained branch of the code: version ‘0’. Changes are reported against the versions released to PyPi. The main version number increases with each PyPi release. Starting with version 0.7 changes are documented in order of build date to make it easier to see what is changing in the master branch on GitHub.

Not sure which version you are using? Try:

```python
from pyslet.info import version
print version
```

### 1.2 Version 0.8

TBC
1.3 Version 0.7.20170805

1.3.1 Summary of new features

Pyslet now supports Python 3, all tests are passing in Python 3.

Travis now builds and tests Python 2.7 and Python 3.5, I’ve dropped 2.6 from the continuous integration testing because the latest Ubuntu images have dropped Python2.6 but you can still run tox on your own environments as it includes 2.6 in tox.ini.

Various bug fixes in OData and HTTP code.

Warning: for future compatibility with Python 3 you should ensure that you use the bytes type (and the ‘b’ prefix on any string constants) when initialising OData entity properties of type Edm.Binary. Failure to do so will raise an error in Python 3.

1.3.2 Tracked issues

The following issues are resolved (or substantially resolved) in this release.

#3 PEP-8 Compliance

The pep8-regression.py script now checks all source files using flake8; all reported errors have been resolved

Added a new metaclass-based solution to enable method renaming while maintaining support for derived classes that override using the old names. Crazy I know, but it works.

#12 Bug in odata2/sqlds.py

Bug when using numeric or named parameters in DB API. Added support for pyformat in DB APIs as part of enabling support for PyMySQL.

#23 Framework for WSGI-based LTI Applications (beta quality)

Re-engineered Session support in the wsgi module to reduce database load, replacing the Session table completely with signed cookies. If you have used the wsgi.SessionApp class directly this will be a breaking change but these classes will remain experimental until this item is closed out. The database schema required to support LTI has changed slightly as a result.

Changed from Django templates to use Jinja2 (this requires almost no changes to the actual sample code templates and makes the intention of the samples much clearer). Thanks to Christopher Lee for recommending this change.

Possible breaking change to wsgi module to refactor authority setting to “canonical_root”, modified WSGIContext object to accept an optional canonical_root argument and removed the optional authority argument from get_app_root and get_url. The authority setting was previously a misnomer and the wsgi samples were not working properly with localhost.

Changed wsgi module to use the OSFilePath wrapper for file paths for better compatibility with Posix file systems that use binary strings for file paths. This module was causing test failures due to some use of os.path module with mixed string types.

#29 https connections fail on POST after remote server hangup

The currently implemented solution is to allow an open ssl socket to be idle in the ‘blocked’ state for a maximum of 2s before sending a new request. After that time we tear down the socket and build a new one. This may now be a bit aggressive given the newer SSL behaviour (which differentiates issues in the underlying socket with different SSL exceptions).

#30 Provide http connection clean-up thread
The implementation is not as intelligent as I’d like it to be. The protocol version that a server last used is stored on the connection object and is lost when we clean up idle connections. Although it is likely that a new connection will speak the same protocol as the previous one there is little harm in going in to protocol detection mode again (we declare ourselves HTTP/1.1) other than the problem of using transfer encodings on an initial POST. In particular, we only drop out of keep-alive mode when the server has actually responded with an HTTP/1.0 response.

#38 Make Pyslet run under Python 3
See above for details.

#43 Fixes for Python running on Windows
This issue came back again, both unicode file name problems and further problems due to timing in unittests. Fixed this time by mocking and monkey-patching the time.time function in the QTI tests.

#47 Improve CharClass-derived doc strings
Fixed - no functional changes.

#49 Typo in pyslet/odata2/csd.py
Fixed OData serialisation of LongDescription element - thanks to @thomaseitler

#51 Fix processing of dates in JSON format OData communication by the #server
We now accept ISO string formatted dates for both DateTime and DateTimeOffset. Note that providing a timezone other than Z (+00:00) when setting a DateTime will cause the time to be zone-shifted to UTC before the value is set.
Thanks to @ianwj5int.

#53 Use datetime.date to create datetime object
You can now set DateTimeValue using a standard python datetime.date, the value is extended to be 00:00:00 on that date. Thanks to @nmichaud

#54 Fix use of super to remove self
Fixed Atom Date handling bug, thanks to @nmichaud

#55 Replace print_exception with logging (this includes the traceback)
Thanks to @ianwj5int for reporting.

#56 Garbage received when server delays response
This was caused by a bug when handling 401 responses in HTTP client
The issue affected any response that was received as a result of a resend (after a redirect or 401 response). The stream used to receive the data in the follow-up request was not being reset correctly and this resulted in a chunk of 0x00 bytes being written before the actual content.
This bug was discovered following changes in the 20160209 build when StringIO was replaced with BytesIO for Python 3 compatibility. StringIO.truncate moves the stream pointer, BytesIO.truncate does not. As a result all resends where the 3xx or 401 response had a non-zero length body were being affected. Previously the bug only affected the rarer use case of resends of streamed downloads to real files, i.e., requests created by passing an open file in the res_body argument of ClientRequest.
With thanks to @karurosu for reporting.

#58 OData default values (PUT/PATCH/MERGE)
Warning: if you use Pyslet for an OData server please check that PUTs are still working as required.
Changed the SQL data stores to use DEFAULT values from the metadata file as part of the CREATE TABLE queries. Modified update_entity in memds, and SQL storage layers to use MERGE semantics by default, added option to enable replace (PUT) semantics using column defaults. This differs from the previous (incorrect behaviour) where unselected properties were set to NULL.

1.3. Version 0.7.20170805
Updated OData server to support MERGE and ensured that PUT now uses the correct semantics (set to default instead of NULL) for values missing from the incoming request.

Improved error handling to reduce log noise in SQL layer.

#60 authentication example in docs

Added a first cut at a documentation page for HTTP auth.

#61 Add support for NTLM

Experimental support for NTLM authentication now available using the python-ntlm3 module from pip/GitHub which must be installed before you can use NTLM. The module is in pyslet.ntlmauth and it can be used in a similar way to Basic auth (see set_ntlm_credentials for details.)

Improved handling of error responses in all HTTP requests (includes a Python 3 bug fix) to enable the connection to be kept open more easily during pipelined requests that are terminated early by a final response from the server. This allows a large POST that generates a 401 response to abort sending of chunked bodies and retry without opening a new connection - vital for NTLM which is connection based.

Added automated resend after 417 Expectation failed responses as per latest HTTP guidance. (Even for POST requests!)

#64 Add a LICENSE file

Added to distribution

#65 syntax error in sqlds.SQLCollectionBase.sql_expression_substring

Also added an override for SQLite given the lack of support for the standard substring syntax.

#70 Fix for grouped unary expressions

The bug is best illustrated by attempting to parse OData expressions containing “(not false)”. Thanks to @torokokill for spotting the issue.

#71 $filter fails when querying fieldnames matching OData literal types

The names that introduce typed literals such as time, datetime, guid, binary, X, etc. can now be used in URL expressions without raising parser errors. The reserved names null, true and false continue to be interpreted as literals so properties with any of those names cannot be referred to in expressions. Thanks to @soundstripe for reporting this.

#72 Travis CI tests failing in Python 3.5

Resolved but Travis no longer builds Python 2.6, see above for details.

#74 New release with bugfixes?

Resolved with the release of 0.7

1.3.3 Untracked Fixes

HTTP related:

Fixed an issue with HTTP resends (e.g., when following redirects) that meant that the retry algorithm was causing the client to back off when more than 1 resend was required.

Added compatibility in HTTP client for parsing dates from headers where the server uses the zone designator “UTC” instead of the required “GMT”.

Fixed a bug where the HTTP client would fail if it received multiple WWW-Authenticate headers in the same response (parser bug).
Better handling of non-blocking io in HTTP client fixing issues when a message body is being received to a local stream that is itself blocked. Includes a new wrapper for RawIOBase in Python 2.6 (with a fix for blocking stream bug)

Fixed bug in HTTP client when following relative path redirects

XML/HTML Parser:

Deprecated XML Element construction with name override to improve handling of super.

Fixed a bug in the parsing of HTML content where unexpected elements that belong in the <head> were causing any preceding <body> content to be ignored. Added the get_or_add_child method to XML Elements to deal with cases where add_child’s ‘reset’ of the element’s children is undesired.

Fixed a bug in the XML parser where the parsed DTD was not being set in the Document instance.

CDATA sections were not being generated properly by the (old) function pyslet.xml_structures.EscapeCDSect(), causing the HTML style and script tags to have their content rendered incorrectly. These tags are not part of the QTI content model so this bug is unlikely to have had an impact on real data.

XMLEntity class is now a context manager to help ensure that files are closed before garbage collection. Unittests were triggering resource leak warnings in Python 3.

Fixed a bug in the XML tests that shows up on Windows if the xml test files are checked out with auto-translation of line ends.

Misc:

Fixed a bug in the detect_encoding function in unicode5 module (most likely benign).

Added support for expanded dates to iso8601 module (merged from OData v4 branch).

Refactoring of second truncation in iso8601 to use Python decimals.

Fix for comparison of midnight TimePoints not in canonical form

vfs: VirtualFilePath objects are now sortable.

Use of nested generators was triggering future warnings in Python 3, refactored to catch StopIteration as per: https://www.python.org/dev/peps/pep-0479/

Added SortableMixin to emulate Python 3 TypeError in comparisons and to simplify implementation of comparison/hash operators in custom classes. As a result, some Time/TimePoint comparisons which used to raise ValueError (e.g., due to incompatible precision) now return False for == and != operators and raise TypeError for inequalities (<, >, etc). OData is unaffected as OData time values of the same EDM type are always comparable.

Re-factored previously undocumented stream classes into their own module, in particular the Pipe implementation used for inter-thread communication. Adding documentation for them.

Re-factored the WSGI InputWrapper from rfc5023 into the http modules.

Sample code:

The sample code has also been updated to work in Python 3, including the weather OData service using MySQL but this now connects through PyMySQL as MySQLdb is not supported in Python 3.

scihub.esa.int has been renamed to scihub.copernicus.eu and the sample code has been updated accordingly with the latest metadata-fixes and tested using Python 3.

1.4 Version 0.6.20160201

Summary of New Features: LTI module rewritten, now suitable for real applications! WSGI-based web-app framework built using Pyslet’s DAL MySQL Database connector for Pyslet’s DAL SSL, Certificates and HTTP Basic
Authentication HTTP Cookies URNs

#3 PEP-8 driven refactoring (ongoing)

Added new method decorators to make supporting renamed and redirected methods easier. Added checks for ambiguous names in classes likely to have been sub-classed by third-party code.

#8 Support for SSL Certificates in HTTP Clients

Fixed certificate support in OData and Atom clients. See blog post for further information on how to use certificates: http://swl10.blogspot.co.uk/2014/11/basic-authentication-ssl-and-pyslets.html

#9 HTTP client retry strategy

Improved HTTP retries with simple Fibonacci-based back-off. Also fixed a bug where, if the first request after a server timed out an idle connection is a POST, the request would fail.

#12 bug when using numeric or named parameters in DB API

The basic bug is fixed and I’ve also added support for paramstyle ‘format’.

#14 content element missing in media-link entries

Fixed. Affected atom xml formatted entities only.

#15 MySQL implementation of Pyslet’s DAL (ongoing)

Changes to the core DAL to deal to better support other DB modules. These included added support for LIMIT clauses to speed up paged access to large entity sets. Implementation of a retry strategy when database commands return OperationalError (e.g., MySQL idle timeouts). An updated connection pool manager and an optional pool cleaner method to clean up idle database connections.

#18 Possible bug in parsing AssociationSet names

Added a compatibility mode to odata2.csd1 to enable the metadata model to optionally accept hyphen or dash characters in simple identifiers using:

```
import pyslet.odata2.csd1 as edm
edm.set_simple_identifier_re(edm.SIMPLE_IDENTIFIER_COMPATIBILITY_RE)
```

#19 OData Function parameter handling

Enabled function parameter passing in OData service operations. Only primitive types are supported but they are now parsed correctly from the query string and coerced to the declared parameter type. Bound functions now receive them as a dictionary of SimpleValue instances.

#20 HTTP Basic Authentication

Fixed an issue with the OData basic authentication support, in some cases the HTTP client was waiting for a 401 when it could have offered the credentials preemptively. See also the following blog article: http://swl10.blogspot.co.uk/2014/11/basic-authentication-ssl-and-pyslets.html

#22 Support for navigation properties in OData expressions

Although the code always contained support in general, the mapping to SQL did not previously support the use of table joins in SQL expressions. This release adds support for joins (but not for nested joins).

#23 A Framework for WSGI-based LTI Applications

Added a new module to make it easier to write WSGI-based applications. Re-factored the existing Basic LTI module to use the new oauthlib and Pyslet’s own OData-inspired data access layer.

#24 ESA Sentinel mission compatibility

Added the capability to override the metadata used by an OData server to deal with validation issues in some services. Clients can now also be created from an offline copy of the service root document.
#26 HTTP client eats memory when downloading large unchunked files
Fixed the download buffer which was failing to write out data until an entire chunk (or the entire download) was complete.

#29 https connections fail on POST after remote server hangup
Partial mitigation with an aggressive 2s window in which to start sending a follow-up request when pipelining through https. This is a crude solution and the bug remains open for a more robust solution based around use of the Expect header in HTTP/1.1.

#30 HTTP client cleanup thread
Added an optional parameter to the HTTP client constructor that creates a cleanup thread to close down idle connections periodically.

#31 Removed reliance on Host header in wsgi app class
There are a number of ways an application can be attacked using a forged Host header, wsgi now ignores the Host header and uses a new setting for the preferred scheme//host:port.

#32 get_certificate_chain
Implemented a function to create a complete certificate chain. Implemented using pyOpenSSL with a lot of help from this article

#33 Fixed exception: ‘NoneType’ object has no attribute ‘current_thread’ on exit
Caused by an overly ambitious __del__ method in SQLEntityContainer.

#34 Fixed missing Edm prefix in OData sample code
#35 Fixed missing import in rfc5023 (atom protocol) module
#36 Fixed incorrect error messages in OData $filter queries
#37 Extended comparison operators in OData to include DateTimeOffset values
All thanks to @ianwj5int for spotting

#38 Python 3 compatibility work
I have started revising modules to support Python 3. This is not yet production ready but it is a small impact on existing modules. I have done my best to maintain compatibility, in practice code should continue to work with no changes required.

The most likely failure mode is that you may find a unicode string in Python 2 where you expected a plain str. This can have a knock-on effect of promoting data to unicode, e.g., through formatting operations. In general the returned types of methods are just being clarified and unicode values are returned only where they may have been returned previously anyway. However, in the case of the URI attributes in the rfc2396 module the types have changed from str to unicode in this release.

This is work in progress but the impact is likely to be minimal at this stage.

#40 & #41 Composite keys and Slug headers
Key hints were not working properly between the OData client and server implementations, and were not working at all when the key was composite. It is now possible to pass the formatted entity key predicate (including the brackets) as a Slug to the OData server and it will attempt to parse it and use that key where allowed by the underlying data layer.

#43 Fixes for Python running on Windows
The only substantive changes required were to the way we check for io failures when IOError is raised and the way we handle URI containing non-ASCII characters. Some of the unit tests were also affected due to issues with timing, including the reduced precision of time.time() on Windows-based systems.

Untracked enhancements:

1.4. Version 0.6.20160201
Added a new module to support HTTP cookies. The HTTP/OData client can now be configured to accept cookies. The default behaviour is to ignore them so this won’t affect existing applications.

Added a new module to support URN syntax to provide a better implementation of the IMS LTI vocabularies.

Added an optional params dictionary to the OData expression parser to make it much easier to parse parameterized OData queries.

Added new methods for creating and executing drop table statements in the DAL.

Reworked sample code for the weather data server, included example driver files for mod_wsgi

Other fixes:

Fixed an issue in the OData client that caused basic key lookup in filtered entity collections to use both a key predicate and a $filter query option. This was causing the filter to be ignored, now the key predicate will be added to the filter rather than the path segment.

Fixed the OData DateTime parser to accept (and discard) any time zone specifier given in the literal form as it is now allowed in the ABNF and may therefore be generated by OData servers.

Fixed a bug in the OData server which meant that requests for JSON format responses were not being limited by the builtin topmax and would therefore attempt to return all matching entities in a single response.

Fixed a bug in the OData server which meant that use of $count was causing the $filter to be ignored!

Fixed a bug in the OData URI parser that prevent compound keys from working properly when zealous escaping was used.

Fixed a bug in the OData server which meant that error messages that contained non-ASCII characters were causing a 500 error due to character encoding issues when outputting the expected OData error format.

Fixed a bug in the OData expression evaluator when evaluating expressions that traversed navigation properties over optional relations. If there was no associated entity an error was being raised.

Fixed a bug in the SQL DAL implementation which means that navigation properties that require joining across a composite key were generating syntax errors, e.g., in SQLite the message ‘near “=”: syntax error’ would be seen.

Fixed a bug in the SQLite DAL implementation which means that in-memory databases were not working correctly in multi-threaded environments.

Fixed XML parser bug, ID elements in namespaced documents were not being handled properly.

Fixed bug in the OData server when handling non-URI characters in entity keys

Fixed a bug with composite key handling in media streams when using the SQL layer

1.5 Version 0.5.20140801

Summary of New Features:

- OData Media Resources
- HTTP Package refactoring and retry handling
- Python 2.6 Support

Tracked issues addressed in this release:

#1 added a Makefile to make it easier for others to build and develop the code

Added a tox.ini file to enable support for tox (a tool for running the unittests in multiple Python environments).

#3 PEP-8 driven refactoring (ongoing)
#2 Migrated the code from SVN to git: https://github.com/swl10/pyslet
#4 Added support for read-only properties and tests for auto generated primary and foreign key values
#6 added integration between git and travis ci (thanks @sassman for your help with this)
#10 restored support for Python 2.6

1.5.1 Other Fixes

OData URLs with reserved values in their keys were failing. For example Entity(‘why%3F’) was not being correctly percent-decoded by the URI parsing class ODataURI. Furthermore, the server implementation was fixed to deal with the fact that PATH_INFO in the WSGI environ dictionary follows the CGI convention of being URL-decoded.

1.6 Version 0.4 and earlier

These are obsolete, version 0.4 was developed on Google Code as an integral part of the QTI Migration tool.

1.7 PyAssess

A precursor to Pyslet. For more information see: https://code.google.com/p/qtimigration/wiki/PyAssess
2.1 Python 2.6 Compatibility

When imported, this module modifies a number of standard modules. This patching is done at run time by the pyslet.py26 module and will affect any script that uses Pyslet. It does not modify your Python installation!

- **io** Benign addition of the SEEK_* constants as defined in Python 2.7.

- **wsgiref.simple_server** Modifies the behaviour of the WSGI server when processing HEAD requests so that Content-Length headers are not stripped. There is an issue in Python 2.6 that causes HEAD requests to return a Content-Length of 0 if the WSGI application does not return any data. The behaviour changed in Python 2.7 to be more as expected.

- **zipfile** Patches is_zipfile to add support for passing open files which is allowed under Python 2.7 but not under 2.6.

2.1.1 Module Reference

```python
pyslet.py26.py26 = False
```

If you must know whether or not you are running under Python 2.6 then you can check using this flag, which is True in that case.

```python
class pyslet.py26.RawIOBase
```

There is a bug in the implementation of RawIOBase.read in Python 2.6 that means it never returns None, even if the stream is non-blocking and the underlying readinto method returns None. By importing RawIOBase from this module instead you will get a patched version of the class with correct read behaviour in Python 2.6.

2.2 Python 2 Compatibility

The goal of Pyslet is to work using the same code in both Python 3 and Python 2. Pyslet was originally developed in very early versions of Python 2, it then became briefly dependent on Python 2.7 before settling down to target Python 2.6 and Python 2.7.
One approach to getting the code working with Python 3 would be to implement a compatibility module like six which helps code targeted at Python 2 to run more easily in Python 3. Unfortunately, the changes required are still extensive and so more significant transformation is required.

The purpose of this module is to group together the compatibility issues that specifically affect Pyslet. It provides definitions that make the intent of the Pyslet code clearer.

```python
pyslet.py2.py2 = True
```

Unfortunately, sometimes you just need to know if you are running under Python 2, this flag provides a common way for version specific code to check. (There are multiple ways of checking, this flag just makes it easier to find places in Pyslet where we care.)

```python
pyslet.py2.suffix
```

In some cases you may want to use a suffix to differentiate something that relates specifically to Python 3 versus Python 2. This string takes the value ‘3’ when Python 3 is in use and is an empty string otherwise.

One example where Pyslet uses this is in the stem of a pickled file name as such objects tend to be version specific.

### 2.2.1 Text, Characters, Strings and Bytes

This is the main area where Pyslet has had to change. In most cases, Pyslet explicitly wants either Text or Binary data so the Python 3 handling of these concepts makes a lot of sense.

```python
pyslet.py2.u8(arg)
```

A wrapper for string literals, obviating the need to use the ‘u’ character that is not allowed in Python 3 prior to 3.3. The return result is a unicode string in Python 2 and a str object in Python 3. The argument should be a binary string in UTF-8 format, it is not a simple replacement for ‘u’. There are other approaches to this problem such as the `u` function defined by compatibility libraries such as six. Use whichever strategy best suits your application.

`u8` is forgiving if you accidentally pass a unicode string provided that string contains only ASCII characters. Recommended usage:

```python
my_string = u8(b'hello')
my_string = u8('hello') # works for ASCII text
my_string = u8(u'hello') # wrong, but will work for ASCII text
my_string = u8(b'\xe8\x8b\xb1\xe5\x9b\xbd')
my_string = u8('\xe8\x8b\xb1\xe5\x9b\xbd') # raises ValueError
my_string = u8(u'\u82f1\u56fd') # raises ValueError in Python 3 only
```

The latter examples above resolve to the following two characters: ““.

In cases where you only want to encode characters from the ISO-8859-1 aka Latin-1 character set you may prefer to use the `ul` function instead.

```python
pyslet.py2.ul(arg)
```

An alternative wrapper for string literals, similar to `u8()` but using the latin-1 codec. `ul` is a little more forgiving than `u8`:

```python
my_string = ul(b'Caf\xe9')
my_string = ul('Caf\xe9') # works for Latin text
my_string = ul(u'Caf\xe9') # wrong, but will work for Latin text
```

Notice that unicode escapes for characters outside the first 256 are not allowed in either wrapper. If you want to use a wrapper that interprets strings like ‘\u82f1\u56fd’ in both major Python versions you should use a module like six.
which will pass strings to the unicode_literal codec. The approach taken by Pyslet is deliberately different, but has the advantage of dealing with some awkward cases:

```python
ul(b'\\user')
```

The `u` wrapper in six will throw an error for strings like this:

```python
six.u('\\user')
```

```
Traceback (most recent call last):
... 
UnicodeDecodeError: 'unicodeescape' codec can't decode bytes in escape sequence
```

Finally, given the increased overhead in calling a function when interpreting literals consider moving literal definitions to module level where they appear in performance critical functions:

```python
CAFE = ul(b"Caf\xe9")

def at_cafe_1(location):
    return location == u"Caf\xe9"
def at_cafe_2(location):
    return location == CAFE
def at_cafe_3(location):
    return location == ul(b"Caf\xe9")
```

In a quick test with Python 2, using the execution time of version 1 as a bench mark version 2 was approximately 1.1 times slower but version 3 was 19 times slower (the results from six.u are about 16 times slower). The same tests with Python 3 yield about 9 and 3 times slower for ul and six.u respectively.

Compatibility comes with a cost, if you only need to support Python 3.3 and higher (while retaining compatibility with Python 2) then you should use the first form and ignore these literal functions in performance critical code. If you want more compatibility then define all string literals ahead of time, e.g., at module level.

### Character Constants

These constants are provided to define common character strings (forcing the unicode type in Python 2).

- `pyslet.py2.uempty`  
The empty string.
- `pyslet.py2.uspace`  
Single space character, character(0x20).

### Text Functions

- `pyslet.py2.is_string(arg)`  
Returns True if `arg` is either a character or binary string.
- `pyslet.py2.is_text(arg)`  
Returns True if `arg` is text and False otherwise. In Python 3 this is simply a test of whether `arg` is of type `str` but in Python 2 both `str` and `unicode` types return True. An example usage of this function is when checking arguments that may be either text or some other type of object.
pyslet.py2.force_text(arg)
Returns arg as text or raises TypeError. In Python 3 this simply checks that arg is of type str, in Python 2 this allows either string type but always returns a unicode string. No codec is used so this has the side effect of ensuring that only ASCII compatible str instances will be acceptable in Python 2.

pyslet.py2.to_text(arg)
Returns arg as text, converting it if necessary. In Python 2 this always returns a unicode string. In Python 3, this function is almost identical to the built-in str except that it takes binary data that can be interpreted as ascii and converts it to text. In other words:

```
to_text(b"hello") == "hello"
```

In both Python 2 and Python 3. Whereas the following is only true in Python 2:

```
str(b"hello") == "hello"
```

arg need not be a string, this function will cause an arbitrary object’s __str__ (or __unicode__ in Python 2) method to be evaluated.

pyslet.py2.is_ascii(arg)
Returns True if arg is of type str in both Python 2 and Python 3. The only difference is that in Python 3 unicode errors will be raised if arg contains non-ascii characters. If arg is not of str type then False is returned.

This function is used to check a value in situations where unicode is not expected in Python 2.

pyslet.py2.force_ascii(arg)
Returns arg as ascii text, converting it if necessary. The result is an object of type str, in both python 2 and python 3. The difference is that in Python 2 unicode strings are accepted and forced to type str by encoding with the ‘ascii’ codec whereas in Python 3 bytes instances are accepted and forced to type str by decoding with the ‘ascii’ codec.

This function is not needed very often but in some cases Python interfaces required type str in Python 2 when the intention was to accept ASCII text rather than arbitrary bytes. When migrated to Python 3 these interfaces can be problematic as inputs may be generated as ASCII bytes rather than strings in Python 3, e.g., the output of base64 encoding.

class pyslet.py2.UnicodeMixin
    Mixin class to handle string formatting
    For classes that need to define a __unicode__ method of their own this class is used to ensure that the correct behaviour exists in Python versions 2 and 3.

    The mixin class implements __str__ based on your existing (required) __unicode__ or (optional) __bytes__ implementation. In python 2, the output of __unicode__ is encoded using the default system encoding if no __bytes__ implementation is provided. This may well generate errors but that seems more appropriate as it will catch cases where the str function has been used instead of to_text().

pyslet.py2.is_unicode(arg)
Returns True if arg is unicode text and False otherwise. In Python 3 this is simply a test of whether arg is of type str but in Python 2 arg must be a unicode string. This is used in contexts where we want to discriminate between bytes and text in all Python versions.

pyslet.py2.character(codepoint)
Given an integer codepoint returns a single unicode character. You can also pass a single byte value (defined as the type returned by indexing a binary string). Bear in mind that in Python 2 this is a single-character string, not an integer. See byte() for how to create byte values dynamically.

pyslet.py2.join_characters(iterable)
Convenience function for concatenating an iterable of characters (or character strings). In Python 3 this is just:
In Python 2 it ensures the result is a unicode string.

**Bytes**

**pyslet.py2.to_bytes**(arg)

Returns `arg` as bytes, converting it if necessary. In Python 2 this always returns a plain string and is in fact just an alias for the builtin `str`. In Python 3, this function is more complex. If `arg` is an object with a `__bytes__` attribute then this is called, otherwise the object is converted to a string (using `str`) and then encoded using the `ascii` codec.

The behaviour of `to_bytes` in Python 3 may appear similar to the built in `bytes` function but there is an important exception:

```
x = 2
str(x) == '2'  # in python 2 and 3
bytes(x) == b'2'  # in python 2
bytes(x) == b'\x00\x00'  # in python 3
to_bytes(x) == b'2'  # in python 2 and 3
```

**pyslet.py2.force_bytes**(arg)

Given either a binary string or a character string, returns a binary string of bytes. If `arg` is a character string then it is encoded with the `ascii` codec.

**pyslet.py2.byte**(value)

Given either an integer value in the range 0..255, a single-character binary string or a single-character with Unicode codepoint in the range 0..255: returns a single byte representing that value. This is one of the main differences between Python 2 and 3. In Python 2 bytes are characters and in Python 3 they’re integers.

**pyslet.py2.byte_value**(b)

Given a value such as would be returned by `byte()` or by indexing a binary string, returns the corresponding integer value. In Python 3 this a no-op but in Python 2 it maps to the builtin function `ord`.

**pyslet.py2.join_bytes**(arg)

Given an arg that iterates to yield bytes, returns a bytes object containing those bytes. It is important not to confuse this operation with the more common joining of binary strings. No function is provided for that as the following construct works as expected in both Python 2 and Python 3:

```
b''.join(bstr_list)
```

The usage of `join_bytes` can best be illustrated by the following two interpreter sessions.

Python 2.7.10:

```
>>> from pyslet.py2 import join_bytes
>>> join_bytes(list(b'abc'))
'abc'
```

Python 3.5.1:

```
>>> from pyslet.py2 import join_bytes
>>> join_bytes(list(b'abc'))
b'abc'
>>> b''.join(list(b'abc'))
```
pyslet.py2.byte_to_bstr(arg)
Given a single byte, returns a bytes object of length 1 containing that byte. This is a more efficient way of writing:

join_bytes([arg])

In Python 2 this is a no-operation but in Python 3 it is effectively the same as the above.

### 2.2.2 Printing to stdout

pyslet.py2.output(txt)
Simple function for writing to stdout

Not as sophisticated as Python 3’s print function but designed to be more of a companion to the built in input.

### 2.2.3 Numeric Definitions

pyslet.py2.long2()
Missing from Python 3, equivalent to the builtin int.

### 2.2.4 Iterable Fixes

Python 3 made a number of changes to the way objects are iterated.

pyslet.py2.range3(*args)
Uses Python 3 range semantics, maps to xrange in Python 2.

pyslet.py2.dict_keys(d)
Returns an iterable object representing the keys in the dictionary d.

pyslet.py2.dict_values(d)
Returns an iterable object representing the values in the dictionary d.

### 2.2.5 Comparisons

```python
class pyslet.py2.SortableMixin
    Bases: object
    
    Mixin class for handling comparisons
    
    Utility class for helping provide comparisons that are compatible with Python 2 and Python 3. Classes must define a method sortkey() which returns a sortable key value representing the instance.
    
    Derived classes may optionally override the classmethod otherkey() to provide an ordering against other object types.
    
    This mixin then adds implementations for all of the comparison methods: __eq__, __ne__, __lt__, __le__, __gt__, __ge__.
```
sortkey()

Returns a value to use as a key for sorting.

By default returns NotImplemented. This value causes the comparison functions to also return NotImplemented.

otherkey(other)

Returns a value to use as a key for sorting

The difference between this method and sortkey() is that this method takes an arbitrary object and either returns the key to use when comparing with this instance or NotImplemented if the sorting is not supported.

You don’t have to override this implementation, by default it returns other.sortkey() if other is an instance of the same class as self, otherwise it returns NotImplemented.

class pyslet.py2.CmpMixin

Bases: object

Mixin class for handling comparisons

For compatibility with Python 2’s __cmp__ method this class defines an implementation of __eq__, __lt__, __le__, __gt__, __ge__ that are redirected to __cmp__. These are the minimum methods required for Python’s rich comparisons.

In Python 2 it also provides an implementation of __ne__ that simply inverts the result of __eq__. (This is not required in Python 3.)

2.2.6 Misc Fixes

Imports the builtins module enabling you to import it from py2 instead of having to guess between __builtin__ (Python 2) and builtins (Python 3).

pyslet.py2.urlopen(*args, **kwargs)

Imported from urllib.request in Python 3, from urllib in Python 2.

pyslet.py2.urlencode(*args, **kwargs)

Imported from urllib.parse in Python 3, from urlib in Python 2.

pyslet.py2.urlquote(*args, **kwargs)

Imported from urllib.parse.quote in Python 3, from urlib.quote in Python 2.

pyslet.py2.parse_qs(*args, **kwargs)

Imported from urllib.parse in Python 3, from urlparse in Python 2.

2.3 PEP-8 Compatibility

class pyslet.pep8.MigratedClass

Base class to assist with method renaming. This base class defines a metaclass for use in conjunction with the @old_method decorator. It automatically provides old method definitions that generate warnings when first called.

Any derived classes will also be defined with this metaclass (providing they don’t themselves use an overriding metaclass of course). Therefore, the associated metaclass also checks each derived class to see if it has overridden any old methods, renaming those definitions accordingly in order to preserve the purpose of the original decorator. An example will help:
With these definitions, the author of Base has renamed a method previously called ‘OldName’ to ‘new_name’. Authors of older code are unaware and continue to use the old name. The metaclass provides the magic to ensure their code does not break:

```python
>>> b = Base()
>>> b.OldName()
__main__:1: DeprecationWarning: Base.OldName is deprecated, use, new_name instead
 'Found it!'
```

The warning is only shown when python is run with the -Wd option.

The metadata also handles the slightly harder problem of dealing with derived classes that must work with new code:

```python
class Derived(Base):
    def OldName(self):
        return 'My old code works!'

>>> d = Derived()
>>> d.new_name()
'My old code works!'
```

Although more complex, the same is true for class methods. When using the @classmethod decorator you must put it before the old_method decorator, like this:

```python
@classmethod
@pep8.old_method('OldClassMethod')
def new_class_method(cls):
    return "Found it!"
```

And similarly for staticmethod:

```python
@staticmethod
@pep8.old_method('OldStaticMethod')
def new_static_method():
    return "Found it!"
```

Again, older code that uses old names will have their calls automatically redirected to the new methods and derived classes that provide an implementation using the old names will find that their implementation is also callable with the new names.

The power of metaclasses means that there is no significant performance hit as the extra work is largely done during the definition of the class so typically affects module load times rather than instance creation and method calling. Calling the old names is slower as calls are directed through a wrapper which generates the deprecation warnings. This provides an incentive to migrate older code to use the new names of course.

pyslet.pep8.make_attr_name(name)
Converts name to pep8_style

Upper case letters are replaced with their lower-case equivalent optionally preceded by ‘_’ if one of the following conditions is met:
• it was preceded by a lower case letter
• it is preceded by an upper case letter and followed by a lower-case one

As a result:

```python
make_attr_name('aName') == 'a_name'
make_attr_name('ABCName') == 'abc_name'
make_attr_name('Name') == 'name'
```

Pyslet requires Python 2.6, Python 2.7 or Python 3.3+

### 2.4 Python 3

Pyslet support in Python 3 is at a beta stage. All unittests are now running under Python 3 and the setup script can be used to install Pyslet in Python 3 without errors. Try running your own code that uses Pyslet with python options `-3Wd` to expose any issues that you are likely to need to fix on any future transition.

Support is currently limited to Python 3.3 and higher as some modules continue to require use of the ‘u’ prefix on unicode strings. There are only a handful of instances where this is a problem and these could be resolved if desired - please open an issue on GitHub if you need earlier Python 3 support.

Pyslet includes it’s own module containing compatibility definitions that target the particular idioms I’ve used in the package. You are obviously free to use these definitions yourself to help you create code that also targets both Python 2 and 3 from the same source.

#### Python 2 Compatibility

The tox configuration has been modified to enable Python3 compatibility to be checked with:

```bash
tox -e py35
```

which shoud now succeed if you have Python 3.5 and tox installed on your system.

**Warning:** Due to the dictionary-like approach taken by Pyslet in the OData modules the standard 2to3 script will suggest changing calls like `itervalues()` to `values()` on collections of entities. If you are using OData in Pyslet you are likely to need to use `-x dict` to prevent these automatic transformations.

### 2.5 Python 2.6

When run under Python 2.6 Pyslet will patch some modules to make them more compatible with Python 2.7 code. For details see:

#### Python 2.6 Compatibility

Earlier versions of Python 2.6 have typically been built with a version of sqlite3 that does not support validation of foreign key constraints, the unittests have been designed to skip these tests when such a version is encountered.

**Note:** When run under Python 2.6, Pyslet may not support certificate validation of HTTP connections properly, this seems to depend on the version of OpenSSL that Python is linked to. If you have successfully used pip to install Pyslet then your Python is probably unaffected though.
Please be aware of the following bug in Python 2.6: [http://bugs.python.org/issue2531](http://bugs.python.org/issue2531) this problem caused a number of Pyslet’s tests to fail initially and remains a potential source of problems if you are using Decimal types in OData models.

Python 2.6 support will be withdrawn in a future version, the Travis continuous integration service no longer supports Python 2.6.

### 2.6 PEP-8

The code has been widely refactored for PEP-8 compliant. Where critical, methods are renamed from CamelCase to PEP-8 compliant lower_case_form and the old names are defined as wrappers which raise deprecation warnings.

You can test your code with the `-Wd` option to python to check the warning messages in case you are relying on the old-style names.

Pyslet uses a special module that defines decorators and other code to help with method renaming. The purpose of the module is to ensure that the old names can be used with minimal impact on existing code.

*PEP-8 Compatibility*
The section contains modules that implement specifications published by the IMS Global Learning Consortium. For more information see http://www.imsglobal.org/

Contents:

3.1 IMS Content Packaging (version 1.2)

The IMS Content Packaging specification defines methods for packaging and organizing resources and their associated metadata for transmission between systems. There is a small amount of information on Wikipedia about content packaging in general, see http://en.wikipedia.org/wiki/Content_package. The main use of IMS Content Packaging in the market place is through the SCORM profile. Content Packaging is also used as the basis for the new IMS Common Cartridge, and a method of packaging assessment materials using the specifcation is also described by IMS QTI version 2.1.

Official information about the specification is available from the IMS GLC: http://www.imsglobal.org/content/packaging/index.html

3.1.1 Example

The following example script illustrates the use of this module. The script takes two arguments, a resource file to be packaged (such as an index.html file) and the path to save the zipped package to. The script creates a new package containing a single resource with the entry point set to point to the resource file. It also adds any other files in the same directory as the resource file, using the python os.walk function to include files in sub-directories too. The ContentPackage.IgnoreFilePath() method is used to ensure that hidden files are not added:

```python
#!/usr/bin/env python

import sys, os, os.path, shutil
from pyslet.imscpv1p2 import ContentPackage, PathInPath
from pyslet.rfc2396 import URIFactory
```
def main():
    if len(sys.argv) != 3:
        print "Usage: makecp <resource file> <package file>"
        return
    resFile = sys.argv[1]
    pkgFile = sys.argv[2]
    pkg = ContentPackage()
    try:
        if os.path.isdir(resFile):
            print "Resource entry point must be a file, not a directory."
            return
        resHREF = URI.from_path(resFile)
        srcDir, srcFile = os.path.split(resFile)
        r = pkg.manifest.root.Resources.add_child(pkg.manifest.root.Resources.ResourceClass)
        r.href = str(resHREF.relative(URI.from_path(os.path.join(srcDir, 'imsmanifest.xml'))))
        r.type = 'webcontent'
        for dirpath, dirnames, filenames in os.walk(srcDir):
            for f in filenames:
                srcPath = os.path.join(dirpath, f)
                if pkg.IgnoreFilePath(srcPath):
                    print "Skipping: %s" % srcPath
                    continue
                dstPath = os.path.join(pkg.dPath, PathInPath(srcPath, srcDir))
                # copy the file
                dname, fName = os.path.split(dstPath)
                if not os.path.isdir(dname):
                    os.makedirs(dname)
                print "Copying: %s" % srcPath
                shutil.copy(srcPath, dstPath)
                pkg.File(r, URI.from_path(dstPath))
        if os.path.exists(pkgFile):
            if raw_input("Are you sure you want to overwrite %s? (y/n) " % pkgFile).lower() != 'y':
                return
            pkg.manifest.update()
            pkg.ExportToPIF(pkgFile)
    finally:
        pkg.Close()

if __name__ == "__main__":
    main()
3.1.2 Reference

```python
class pyslet.imscpv1p2.ContentPackage(dpath=None)
    Bases: pyslet.pep8.MigratedClass

    Represents a content package.

    When constructed with no arguments a new package is created. A temporary folder to hold the contents of the
    package is created and will not be cleaned up until the Close() method is called.

    Alternatively, you can pass an operating system or virtual file path to a content package directory, to an ims-
    manifest.xml file or to a Package Interchange Format file. In the latter case, the file is unzipped into a temporary
    folder to facilitate manipulation of the package contents.

    A new manifest file is created and written to the file system when creating a new package, or if it is missing
    from an existing package or directory.

    ManifestDocumentClass
        the default class for representing the Manifest file
        alias of ManifestDocument

dPath = None
    the VirtualFilePath to the package's

manifest = None
    The ManifestDocument object representing the imsmanifest.xml file.

    The file is read (or created) on construction.

fileTable = None
    The fileTable is a dictionary that maps package relative file paths to the File objects that represent them
    in the manifest.

    It is possible for a file to be referenced multiple times (although dependencies were designed to take care
    of most cases it is still possible for two resources to share a physical file, or even for a resource to contain
    multiple references to the same file.) Therefore, the dictionary values are lists of File objects.

    If a file path maps to an empty list then a file exists in the package which is not referenced by any resource.
    In some packages it is common for auxiliary files such as supporting schemas to be included in packages
    without a corresponding File object so an empty list does not indicate that the file can be removed safely.
    These files are still included when packaging the content package for interchange.

    Finally, if a file referred to by a File object in the manifest is missing an entry is still created in the
    fileTable. You can walk the keys of the fileTable testing if each file exists to determine if some expected
    files are missing from the package.

    The keys in fileTable are VirtualFilePath instances. To convert a string to an appropriate instance use the
    FilePath() method.

make_file_path(*path)
    Converts a string into a pyslet.vfs.VirtualFilePath instance suitable for using as a key into the
    fileTable. The conversion is done using the file system of the content package's directory, dPath.

set_ignore_files(ignore_files)
    Sets the regular expression used to determine if a file should be ignored.

    Some operating systems and utilities create hidden files or other spurious data inside the content package
    directory. For example, 's OS X creates .DS_Store files and the svn source control utility creates .svn di-
    rectories. The files shouldn't generally be included in exported packages as they may confuse the recipient
    (who may be using a system on which these files and directories are not hidden) and be deemed to violate
```

3.1. IMS Content Packaging (version 1.2) 23
the specification, not to mention adding unnecessarily to the size of the package and perhaps even leaking information unintentionally.

To help avoid this type of problem the class uses a regular expression to determine if a file should be considered part of the package. When listing directories, the names of the files found are compared against this regular expression and are ignored if they match.

By default, the pattern is set to match all directories and files with names beginning ‘.’ so you will not normally need to call this method.

**ignore_file** *(f)*

Compares a file or directory name against the pattern set by **set_ignore_files()**.

*f* is a unicode string.

**ignore_file_path** *(path)*

Compares a file path against the pattern set by **set_ignore_files()**.

The path is normalised before comparison and any segments consisting of the string ‘.’ are skipped. The method returns True if any of the remaining path components matches the ignore pattern. In other words, if the path describes a file that is in a directory that should be ignored it will also be ignored.

The path can be relative or absolute. Relative paths are not made absolute prior to comparison so this method is not affected by the current directory, even if the current directory would itself be ignored.

**rebuild_file_table** *

Rescans the file system and manifest and rebuilds the fileTable.

**AddToZip** *(args, **kwargs)*

Deprecated equivalent to **add_to_zip()**

**Close** *(args, **kwargs)*

Deprecated equivalent to **close()**

**DeleteFile** *(args, **kwargs)*

Deprecated equivalent to **delete_file()**

**ExpandZip** *(args, **kwargs)*

Deprecated equivalent to **expand_zip()**

**ExportToPIF** *(args, **kwargs)*

Deprecated equivalent to **export_to_pif()**

**File** *(args, **kwargs)*

Deprecated equivalent to **new_file()**

**FileCopy** *(args, **kwargs)*

Deprecated equivalent to **file_copy()**

**FilePath** *(args, **kwargs)*

Deprecated equivalent to **make_file_path()**

**FileScanner** *(args, **kwargs)*

Deprecated equivalent to **file_scanner()**

**GetPackageName** *(args, **kwargs)*

Deprecated equivalent to **get_package_name()**

**GetUniqueFile** *(args, **kwargs)*

Deprecated equivalent to **get_unique_path()**

**IgnoreFile** *(args, **kwargs)*

Deprecated equivalent to **ignore_file()**
IgnoreFilePath (*args, **kwargs)
  Deprecated equivalent to ignore_file_path()

PackagePath (*args, **kwargs)
  Deprecated equivalent to package_path()

RebuildFileTable (*args, **kwargs)
  Deprecated equivalent to rebuild_file_table()

SetIgnoreFiles (*args, **kwargs)
  Deprecated equivalent to set_ignore_files()

package_path (fpath)
  Converts an absolute file path into a canonical package-relative path
  Returns None if fpath is not inside the package.

export_to_pif (zpath)
  Exports the content package, saving the zipped package in zpath
  zpath is overwritten by this operation.
  In order to make content packages more interoperable this method goes beyond the basic zip specification and ensures that pathnames are always UTF-8 encoded when added to the archive. When creating instances of ContentPackage from an existing archive the reverse transformation is performed. When exchanging PIF files between systems with different native file path encodings, encoding errors may occur.

get_unique_path (suggested_path)
  Returns a unique file path suitable for creating a new file in the package.
  suggested_path is used to provide a suggested path for the file. This may be relative (to the root and manifest) or absolute but it must resolve to a file (potentially) in the package. The suggested_path should either be a VirtualFilePath (of the same type as the content package’s dPath) or a string suitable for conversion to a VirtualFilePath.
  When suggested_path is relative, it is forced to lower-case. This is consistent with the behaviour of norm-case on systems that are case insensitive. The trouble with case insensitive file systems is that it may be impossible to unpack a content package created on a case sensitive system and store it on a case insensitive one. By channelling all file storage through this method (and constructing any URIs after the file has been stored) the resulting packages will be more portable.
  If suggested_path already corresponds to a file already in the package, or to a file already referred to in the manifest, then a random string is added to it while preserving the suggested extension in order to make it unique.
  The return result is always normalized and returned relative to the package root.

new_file (resource, href)
  Returns a new File object attached to resource
  href is the URI of the file expressed relative to the resource element in the manifest. Although this is normally the same as the URI expressed relative to the package, a resource may have an xml:base attribute that alters the base for resolving relative URIs.
  href may of course be an absolute URI to an external resource. If an absolute URI is given to a local file it must be located inside the package.
  Attempting to add a File object representing the manifest file itself will raise CPFilePathError.
  The fileTable is updated automatically by this method.

file_copy (resource, src_url)
  Returns a new File object copied into the package from src_url, attached to resource.
The file is copied to the same directory as the resource’s entry point or to the main package directory if the resource has no entry point.

The File object is actually created with the File() method.

Note that if src_url points to a missing file then no file is copied to the package but the associated File is still created. It will point to a missing file.

delete_file (href)
Removes the file at href from the file system

This method also removes any file references to it from resources in the manifest. href may be given relative to the package root directory. The entry in fileTable is also removed.

CPFileTypeError is raised if the file is not a regular file

CPFilePathError is raised if the file is an ignore_file(), the manifest itself or outside of the content package.

CPProtocolError is raised if the indicated file is not in the local file system.

get_package_name()
Returns a human readable name for the package

The name is determined by the method used to create the object. The purpose is to return a name that would be intuitive to the user if it were to be used as the name of the package directory or the stem of a file name when exporting to a PIF file.

Note that the name is returned as a unicode string suitable for showing to the user and may need to be encoded before being used in file path operations.

close()
Closes the content package, removing any temporary files.

This method must be called to clean up any temporary files created when processing the content package. Temporary files are created inside a special temporary directory created using the builtin python tempdir.mkdtemp function. They are not automatically cleaned up when the process exits or when the garbage collector disposes of the object. Use of try:... finally: to clean up the package is recommended. For example:

```python
pkg=ContentPackage("MyPackage.zip")
try:
    # do stuff with the content package here
finally:
    pkg.Close()
```

class pyslet.imscpvl2.ManifestDocument(**args)
Bases: pyslet.xml.namespace.NSDocument

Represents the imsmanifest.xml file itself.

Building on pyslet.xml.namespace.XMLNSDocument this class is used for parsing and writing manifest files.

The constructor defines three additional prefixes using make_prefix(), mapping xsi onto XML schema, imsmd onto the IMS LRM namespace and imsqti onto the IMS QTI 2.1 namespace. It also adds a schemaLocation attribute. The elements defined by the pyslet.imsmdvlp2p1 and pyslet.IMSQTLV2P1 modules are added to the classMap to ensure that metadata from those schemas are bound to the special classes defined there.

defaultNS = None
the default namespace is set to IMSCP_NAMESPACE
get_element_class(name)
    Overrides pyslet.xml.namespace.XMLNSDocument.get_element_class() to look up name.

    The class contains a mapping from (namespace,element name) pairs to class objects representing the elements. Any element not in the class map returns XMLNSElement() instead.

Constants

The following constants are used for setting and interpreting XML documents that conform to the Content Packaging specification

```python
pyslet.imscpv1p2.IMSCP_NAMESPACE = 'http://www.imsglobal.org/xsd/imscp_v1p1'
str(object='') -> string

    Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyslet.imscpv1p2.IMSCP_SCHEMALOCATION = 'http://www.imsglobal.org/xsd/imscp_v1p1.xsd'
str(object='') -> string

    Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pyslet.imscpv1p2.IMSCPX_NAMESPACE = 'http://www.imsglobal.org/xsd/imscp_extensionv1p2'
str(object='') -> string

    Return a nice string representation of the object. If the argument is a string, the return value is the same object.
```

Elements

```python
class pyslet.imscpv1p2.CPElement (parent, name=None)
    Bases: pyslet.xml.namespace.NSElement

    Base class for all elements defined by the Content Packaging specification.

class pyslet.imscpv1p2.Manifest (parent)
    Bases: pyslet.imscpv1p2.CPElement

    Represents the manifest element, the root element of the imsmanifest file.

    MetadataClass
        the default class to represent the metadata element
        alias of Metadata

    OrganizationsClass
        the default class to represent the organizations element
        alias of Organizations

    ResourcesClass
        the default class to represent the resources element
        alias of Resources

    ManifestClass
        the default class to represent child manifest elements
        alias of Manifest

    Metadata = None
        the manifest’s metadata element
```

3.1. IMS Content Packaging (version 1.2)
Organizations = None
the organizations element

Resources = None
the resources element

Manifest = None
a list of child manifest elements

class pyslet.imscp1p2.Metadata (parent)
    Bases: pyslet.imscp1p2.CPElement
    Represents the Metadata element.

    SchemaClass
    the default class to represent the schema element
    alias of Schema

    SchemaVersionClass
    alias of SchemaVersion

    Schema = None
    the optional schema element

    SchemaVersion = None
    the optional schemaversion element

class pyslet.imscp1p2.Schema (parent, name=None)
    Bases: pyslet.imscp1p2.CPElement
    Represents the schema element.

class pyslet.imscp1p2.SchemaVersion (parent, name=None)
    Bases: pyslet.imscp1p2.CPElement
    Represents the schemaversion element.

class pyslet.imscp1p2.Organizations (parent)
    Bases: pyslet.imscp1p2.CPElement
    Represents the organizations element.

    OrganizationClass
    the default class to represent the organization element
    alias of Organization

    Organization = None
    a list of organization elements

class pyslet.imscp1p2.Organization (parent, name=None)
    Bases: pyslet.imscp1p2.CPElement
    Represents the organization element.

class pyslet.imscp1p2.Resources (parent)
    Bases: pyslet.imscp1p2.CPElement
    Represents the resources element.

    ResourceClass
    the default class to represent the resource element
    alias of Resource
**Resource** = None
the list of resources in the manifest

class pyslet.imscp1p2.Resource(
    parent
) Bases: pyslet.imscp1p2.CPElement

Represents the resource element.

**MetadataClass**
the default class to represent the metadata element
alias of Metadata

**FileClass**
the default class to represent the file element
alias of File

**DependencyClass**
the default class to represent the dependency element
alias of Dependency

type = None
the type of the resource

href = None
the href pointing at the resource’s entry point

**Metadata** = None
the resource’s optional metadata element

**File** = None
a list of file elements associated with the resource

**Dependency** = None
a list of dependencies of this resource

**get_entry_point**()
Returns the File object that is identified as the entry point.
If there is no entry point, or no File object with a matching href, then None is returned.

**set_entry_point**(
    f
) Set’s the File object that is identified as the resource’s entry point.
The File must already exist and be associated with the resource.

**DeleteDependency**(*args, **kwargs)
Deprecated equivalent to delete_dependency()

**DeleteFile**(*args, **kwargs)
Deprecated equivalent to delete_file()

**GetEntryPoint**(*args, **kwargs)
Deprecated equivalent to get_entry_point()

**SetEntryPoint**(*args, **kwargs)
Deprecated equivalent to set_entry_point()

class pyslet.imscp1p2.File(
    parent
) Bases: pyslet.imscp1p2.CPElement

Represents the file element.
href = None
the href used to locate the file object

package_path(cp)
Returns the normalized file path relative to the root of the content package, cp.
If the href does not point to a local file then None is returned. Otherwise, this function calculates an
absolute path to the file and then calls the content package's ContentPackage.package_path() method.

PackagePath(*args, **kwargs)
Deprecated equivalent to package_path()

class pyslet.imscp1p2.Dependency (parent)
Bases: pyslet.imscp1p2.CPElement
Represents the dependency element.

identifierref = None
the identifier of the resource in this dependency

Utilities

pyslet.imscp1p2.PathInPath(*args, **kwargs)
Deprecated equivalent to path_in_path()

3.2 IMS Question and Test Interoperability (version 1.2)

The IMS Question and Test Interoperability (QTI) specification version 1.2 was finalized in 2002. After a gap of
1-2 years work started on a major revision, culminating in version 2 of the specification, published first in 2005. For
information about the history of the specification see http://en.wikipedia.org/wiki/QTI - official information about the
specification is available from the IMS GLC: http://www.imsglobal.org/question/index.html

The purpose of this module is to allow documents in QTI v1 format to be parsed and then transformed into objects
representing the QTI v2 data model where more sophisticated processing can be performed. Effectively, the native
model of assessment items in Pyslet (and in the PyAssess package it supersedes) is QTI v2 and this module simply
provides an import capability for legacy data marked up as QTI v1 items.

Class methods or functions with names beginning migrate_... use a common pattern for performing the conversion.
Errors and warnings are logged during conversion to a list passed in as the log parameter.

3.2.1 Core Types and Utilities

This module contains a number core classes used to support the standard.

Enumerations

Where the DTD defines enumerated attribute values we define special enumeration classes. These follow a common
pattern in which the values are represented by constant members of the class. The classes are not designed to be
instantiated but they do define class methods for decoding and encoding from and to text strings.

class pyslet.qtiv1.core.Action
    Bases: pyslet.xml.xsdatatypesEnumeration
    Action enumeration (for pyslet.qtiv1.common.SetVar:
Defines constants for the above action types. Usage example:

```
Action.Add
```

Note that:

```
Action.DEFAULT == Action.Set
```

For more methods see `Enumeration`

```python
class pyslet.qtiv1.core.Area
    Bases: pyslet.xml.xsdatatypes.Enumeration

Area enumeration:

- (Ellipse | Rectangle | Bounded) = 'Ellipse'

Defines constants for the above area types. Usage example:

```
Area.Rectangle
```

Note that:

```
Area.DEFAULT == Area.Ellipse
```

For more methods see `Enumeration`

```python
pyslet.qtiv1.core.migrate_area_to_v2(area, value, log)
```

Returns a tuple of (shape,coords object) representing the area.

(Also callable as MigrateV2AreaCoords for backwards compatibility.)

- `area` is one of the `Area` constants.
- `value` is the string containing the content of the element to which the area applies.

This conversion is generous because the separators have never been well defined and in some cases content uses a mixture of space and ','.

Note also that the definition of rarea was updated in the 1.2.1 errata and that affects this algorithm. The clarification on the definition of ellipse from radii to diameters might mean that some content ends up with hotspots that are too small but this is safer than hotspots that are too large.

Example:

```python
import pyslet.qtiv1.core as qticore1
import pyslet.qtiv2.core as qticore2
import pyslet.html40_1991224 as html
log=[]
shape,coords=qticore1.MigrateV2AreaCoords(qticore1.Area.Ellipse,"10,10,2,2",log)
# returns (qticore2.Shape.circle, html.Coords([10, 10, 1]) )
```

Note that Ellipse was deprecated in QTI version 2:

```python
import pyslet.qtiv1.core as qticore1
import pyslet.qtiv2.core as qticore2
import pyslet.html40_1991224 as html
log=[]
shape,coords=qticore1.MigrateV2AreaCoords(qticore1.Area.Ellipse,"10,10,2,4",log)
```
class pyslet.Qtivi.core.FeedbackStyle
Bases: pyslet.xml.xsd.datatypes.Enumeration

feedbackstyle enumeration:

(Complete | Incremental | Multilevel | Proprietary ) 'Complete'

Defines constants for the above feedback style. Usage example:

FeedbackStyle.Decimal

Note that:

FeedbackStyle.DEFAULT == FeedbackStyle.Complete

For more methods see Enumeration

class pyslet.Qtivi.core.FeedbackType
Bases: pyslet.xml.xsd.datatypes.Enumeration

feedbacktype enumeration:

(Response | Solution | Hint ) 'Response'

Defines constants for the above types of feedback. Usage example:

FeedbackType.Decimal

Note that:

FeedbackType.DEFAULT == FeedbackType.Response

For more methods see Enumeration

class pyslet.Qtivi.core.FIBType
Bases: pyslet.xml.xsd.datatypes.Enumeration

Fill-in-the-blank type enumeration:

(String | Integer | Decimal | Scientific ) 'String'

Defines constants for the above fill-in-the-blank types. Usage example:

FIBType.Decimal

Note that:

FIBType.DEFAULT == FIBType.String

For more methods see Enumeration

class pyslet.Qtivi.core.MDOperator
Bases: pyslet.xml.xsd.datatypes.EnumerationNoCase

Metadata operator enumeration for pyslet.Qtivi.sao.SelectionMetadata:
(EQ | NEQ | LT | LTE | GT | GTE )

Defines constants for the above operators. Usage example:

MDOperator.EQ

Lower-case aliases of the constants are provided for compatibility.

For more methods see Enumeration

**class** pyslet.qti.v1.core.NumType
**Bases:** pyslet.xml.xsd.datatypes.Enumeration

numtype enumeration:

(Integer | Decimal | Scientific ) 'Integer'

Defines constants for the above numeric types. Usage example:

NumType.Scientific

Note that:

NumType.DEFAULT == NumType.Integer

For more methods see Enumeration

**class** pyslet.qti.v1.core.Orientation
**Bases:** pyslet.xml.xsd.datatypes.Enumeration

Orientation enumeration:

(Horizontal | Vertical ) 'Horizontal'

Defines constants for the above orientation types. Usage example:

Orientation.Horizontal

Note that:

Orientation.DEFAULT == Orientation.Horizontal

For more methods see Enumeration

pyslet.qti.v1.core.migrate_orientation_to_v2(orientation)
Maps a v1 orientation onto the corresponding v2 constant.

(Also callable as MigrateV2Orientation for backwards compatibility.)

 Raises KeyError if orientation is not one of the Orientation constants.

**class** pyslet.qti.v1.core.PromptType
**Bases:** pyslet.xml.xsd.datatypes.Enumeration

Prompt type enumeration:

(Box | Dashline | Asterisk | Underline )

Defines constants for the above prompt types. Usage example:
For more methods see `Enumeration`.

```python
class pyslet.qti.v1.core.RCardinality
    Bases: pyslet.xml.xsd.datatypes.Enumeration

rcardinality enumeration:

   (Single | Multiple | Ordered ) 'Single'

Defines constants for the above cardinality types. Usage example:

   RCardinality.Multiple

Note that:

   RCardinality.DEFAULT == RCardinality.Single
```

For more methods see `Enumeration`.

```python
pyslet.qti.v1.core.migrate_cardinality_to_v2(rcardinality)
    Maps a v1 cardinality onto the corresponding v2 constant.

    (Also callable as MigrateV2Cardinality for backwards compatibility.)

    Raises KeyError if `rcardinality` is not one of the `RCardinality` constants.
```

```python
class pyslet.qti.v1.core.TestOperator = <class 'pyslet.qti.v1.core.MDOperator'>
    A simple alias of `MDOperator` defined for `pyslet.qti.v1.outcomes.VariableTest`.
```

```python
class pyslet.qti.v1.core.VarType
    Bases: pyslet.xml.xsd.datatypes.Enumeration

vartype enumeration:

   (Integer | String | Decimal | Scientific | Boolean | Enumerated | Set ) 'Integer'

Defines constants for the above view types. Usage example:

   VarType.String

Note that:

   VarType.DEFAULT == VarType.Integer
```

For more methods see `Enumeration`.

```python
pyslet.qti.v1.core.migrate_vartype_to_v2(vartype, log)
    Returns the v2 BaseType representing the v1 vartype.

    (Also callable as MigrateV2VarType for backwards compatibility.)

    Note that we reduce both Decimal and Scientific to the float types. In version 2 the BaseType values were chosen to map onto the typical types available in most programming languages. The representation of the number in decimal or exponent form is considered to be part of the interaction or the presentation rather than part of the underlying processing model. Although there clearly are use cases where retaining this distinction would have been an advantage the quality of implementation was likely to be poor and use cases that require a distinction are now implemented in more cumbersome, but probably more interoperable ways.
```
Note also that the poorly defined Set type in version 1 maps to an identifier in version 2 on the assumption that the cardinality will be upgraded as necessary.

Raises KeyError if vartype is not one of the VarType constants.

```python
class pyslet.qtiv1.core.View
    Bases: pyslet.xml.xsdatatypesEnumerationNoCase

    View enumeration:

    (All | Administrator | AdminAuthority | Assessor | Author | Candidate | InvigilatorProctor | Psychometrician | Scorer | Tutor) 'All'

    Defines constants for the above view types. Usage example:

    View.Candidate

    Note that:

    View.DEFAULT == View.All

    In addition to the constants defined in the specification we add two aliases which are in common use:

    (Invigilator | Proctor)

    For more methods see Enumeration
```

```python
pyslet.qtiv1.core.migrate_view_to_v2(view, log)
    Returns a list of v2 view values representing the v1 view.

    (Also callable as MigrateV2View for backwards compatibility.)

    The use of a list as the return type enables mapping of the special value ‘All’, which has no direct equivalent in version 2 other than providing all the defined views.

    Raises KeyError if view is not one of the View constants.

    This function will log warnings when migrating the following v1 values: Administrator, AdminAuthority, Assessor and Psychometrician
```

**Utility Functions**

```python
pyslet.qtiv1.core.make_valid_name(name)
    This function takes a string that is supposed to match the production for Name in XML and forces it to comply by replacing illegal characters with ‘_’. If name starts with a valid name character but not a valid name start character, it is prefixed with ‘_’ too.

    (Also callable as MakeValidName for backwards compatibility.)
```

```python
pyslet.qtiv1.core.yn_from_str(src)
    Returns a True/False parsed from a “Yes” / “No” string.

    This function is generous in what it accepts, it will accept mixed case and strips surrounding space. It returns True if the resulting string matches “yes” and False otherwise.

    Reverses the transformation defined by yn_to_str().
```

```python
pyslet.qtiv1.core.yn_to_str(value)
    Returns “Yes” if value is True, “No” otherwise.

    Reverses the transformation defined by yn_from_str().
```
Constants

```python
pyslet.qtiv1.core.QTI_SOURCE = 'QTIv1'
str(object='') -> string
```

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

Exceptions

```python
class pyslet.qtiv1.core.QTIError
    Bases: exceptions.Exception
    All errors raised by this module are derived from QTIError.
class pyslet.qtiv1.core.QTIUnimplementedError
    Bases: pyslet.qtiv1.core.QTIError
    A feature of QTI v1 that is not yet implemented by this module.
```

Abstract Elements

```python
class pyslet.qtiv1.core.QTIElement (parent, name=None)
    Bases: pyslet.xml.structures.Element
    Base class for all elements defined by the QTI specification
    declare_metadata (label, entry, definition=None)
        Declares a piece of metadata to be associated with the element.
        Most QTIElements will be contained by some type of metadata container that collects metadata in a format
        suitable for easy lookup and export to other metadata formats. The default implementation simply passes
        the call to the parent element or, if there is no parent, the declaration is ignored.
        For more information see MetadataContainer.
class pyslet.qtiv1.core.ObjectMixin
    Mix-in class for elements that can be inside ObjectBank:
    (section | item)+
class pyslet.qtiv1.core.SectionItemMixin
    Mix-in class for objects that can be in section objects:
    (itemref | item | sectionref | section)*
class pyslet.qtiv1.core.SectionMixin
    Bases: pyslet.qtiv1.core.SectionItemMixin
    Mix-in class for objects that can be in assessment objects:
    (sectionref | section)+
```

3.2.2 Common Classes

This module contains the common data elements defined in section 3.6 of the binding document. The doc string of
each element defined by IMS is introduced with a quote from that document to provide context. For more information
see: http://www.imsglobal.org/question/qtiv1p2/imsqtiasi_bindv1p2.html
Content Model

Perhaps the biggest change between version 1 and version 2 of the specification was the content model. There were attempts to improve the original model through the introduction of the flow concept in version 1.2 but it wasn’t until the externally defined HTML content model was formally adopted in version 2 that some degree of predictability in rendering became possible.

class pyslet.qtiv1.common.ContentMixin
    Bases: object

    Mixin class for handling all content-containing elements.

    This class is used by all elements that behave as content, the default implementation provides an additional contentChildren member that should be used to collect any content-like children.

    contentChildren = None
        the list of content children

    content_child(child_class)
        Returns True if child_class is an allowed subclass of ContentMixin in this context.

    add_child(child_class, name=None)
        Creates a new child of this element.

        Overrides the underlying Element class to implement special handling for ContentMixin and its subclasses.
        By default we accept any type of content but derived classes override this behaviour by providing an implementation of content_child to limit the range of elements to match their own content models.

    get_content_children()
        Returns an iterable of the content children.

    is_inline()
        True if this element can be inlined, False if it is block level

        The default implementation returns True if all contentChildren can be inlined.

    inline_children()
        True if all of this element’s contentChildren can all be inlined.

    extract_text()
        Returns a tuple of (<text string>, <lang>).

        Sometimes it is desirable to have a plain text representation of a content object. For example, an element may permit arbitrary content but a synopsis is required to set a metadata value.

        Our algorithm for determining the language of the text is to first check if the language has been specified for the context. If it has then that language is used. Otherwise the first language attribute encountered in the content is used as the language. If no language is found then None is returned as the second value.

    migrate_content_to_v2(parent, child_type, log, children=None)
        Migrates this content element to QTIv2.

        The resulting QTIv2 content is added to parent.

        child_type indicates whether the context allows block, inline or a mixture of element content types (flow). It is set to one of the following HTML classes: pyslet.html401.BlockMixin, pyslet.html401.InlineMixin or pyslet.html401.FlowMixin.

        The default implementation adds each of children or, if children is None, each of the local contentChildren. The algorithm handles flow elements by creating <p> elements where the context permits. Nested flows are handled by the addition of <br/>.
class pyslet.qtiv1.common.Material(\texttt{parent})
Bases: \texttt{pyslet.qtiv1.common.ContentMixin}, \texttt{pyslet.qtiv1.common.QTICommentContainer}

This is the container for any content that is to be displayed by the question-engine. The supported content types are text (emphasized or not), images, audio, video, application and applet. The content can be internally referenced to avoid the need for duplicate copies. Alternative information can be defined - this is used if the primary content cannot be displayed:

```xml
<!ELEMENT material (qticomment?, (mattext | matemtext | matimage | mataudio | matvideo | matapplet | matapplication | matref | matbreak | mat_extension)+ , altmaterial*)>
<!ATTLIST material
  label CDATA #IMPLIED
  xml:lang CDATA #IMPLIED >
```

class pyslet.qtiv1.common.AltMaterial(\texttt{parent})
Bases: \texttt{pyslet.qtiv1.common.ContentMixin}, \texttt{pyslet.qtiv1.common.QTICommentContainer}

This is the container for alternative content. This content is to be displayed if, for whatever reason, the primary content cannot be rendered. Alternative language implementations of the host \texttt{<material>} element are also supported using this structure:

```xml
<!ELEMENT altmaterial (qticomment?,
  (mattext | matemtext | matimage | mataudio | matvideo | matapplet | matapplication | matref | matbreak | mat_extension)+)>  
<!ATTLIST altmaterial xml:lang CDATA #IMPLIED >
```

class pyslet.qtiv1.common.MatThingMixin
Bases: \texttt{pyslet.qtiv1.common.ContentMixin}

An abstract class used to help identify the mat* elements.

class pyslet.qtiv1.common.PositionMixin
Mixin to define the positional attributes

```
width CDATA #IMPLIED
height CDATA #IMPLIED
y0 CDATA #IMPLIED
x0 CDATA #IMPLIED
```

class pyslet.qtiv1.common.MatText(\texttt{parent})
Bases: \texttt{pyslet.qtiv1.core.QTIElement}, \texttt{pyslet.qtiv1.common.PositionMixin}, \texttt{pyslet.qtiv1.common.MatThingMixin}

The \texttt{<mattext>} element contains any text that is to be displayed to the users

```xml
<!ELEMENT mattext (#PCDATA)>
<!ATTLIST mattext
texttype CDATA 'text/plain'
label CDATA #IMPLIED
charset CDATA 'ascii-us'
uri CDATA #IMPLIED
xml:space (preserve | default) 'default'
xml:lang CDATA #IMPLIED
entityref ENTITY #IMPLIED
width CDATA #IMPLIED
```
**inlineWrapper** = None

an inline html object used to wrap inline elements

class **pyslet.qtiv1.common.MatEmText** (parent)

Bases: **pyslet.qtiv1.common.MatText**

The `<matemtext>` element contains any emphasized text that is to be displayed to the users. The type of emphasis is dependent on the question-engine rendering the text:

```xml
<!ELEMENT matemtext (#PCDATA)>  
<!ATTLIST matemtext
texttype CDATA 'text/plain'
label CDATA #IMPLIED
charset CDATA 'ascii-us'
uri CDATA #IMPLIED
xml:space (preserve | default ) 'default'
xml:lang CDATA #IMPLIED
entityref ENTITY #IMPLIED
width CDATA #IMPLIED
height CDATA #IMPLIED
y0 CDATA #IMPLIED
x0 CDATA #IMPLIED >
```

class **pyslet.qtiv1.common.MatBreak** (parent)

Bases: **pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.MatThingMixin**

The element that is used to insert a break in the flow of the associated material. The nature of the 'break' is dependent on the display-rendering engine:

```xml
<!ELEMENT matbreak EMPTY>
```

**extract_text()**

Returns a simple line break

class **pyslet.qtiv1.common.MatImage** (parent)

Bases: **pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.PositionMixin, pyslet.qtiv1.common.MatThingMixin**

The `<matimage>` element is used to contain image content that is to be displayed to the users:

```xml
<!ELEMENT matimage (#PCDATA)>  
<!ATTLIST matimage
imagtype CDATA 'image/jpeg'
label CDATA #IMPLIED
height CDATA #IMPLIED
uri CDATA #IMPLIED
embedded CDATA 'base64'
width CDATA #IMPLIED
y0 CDATA #IMPLIED
x0 CDATA #IMPLIED
entityref ENTITY #IMPLIED >
```

**extract_text()**

We cannot extract text from matimage so we return a simple string.
class pyslet.qtiv1.common.MatAudio(parent)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.MatThingMixin

The `<mataudio>` element is used to contain audio content that is to be displayed to the users:

```xml
<!ELEMENT mataudio (#PCDATA)>
<!ATTLIST mataudio
    audiotype CDATA 'audio/base'
    label CDATA #IMPLIED
    uri CDATA #IMPLIED
    embedded CDATA 'base64'
    entityref ENTITY #IMPLIED>
```

`extract_text()`
We cannot extract text from mataudio so we return a simple string.

class pyslet.qtiv1.common.MatVideo(parent)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.PositionMixin, pyslet.qtiv1.common.MatThingMixin

The `<matvideo>` element is used to contain video content that is to be displayed to the users:

```xml
<!ELEMENT matvideo (#PCDATA)>
<!ATTLIST matvideo
    videotype CDATA 'video/avi'
    label CDATA #IMPLIED
    uri CDATA #IMPLIED
    width CDATA #IMPLIED
    height CDATA #IMPLIED
    y0 CDATA #IMPLIED
    x0 CDATA #IMPLIED
    embedded CDATA 'base64'
    entityref ENTITY #IMPLIED>
```

`extract_text()`
We cannot extract text from matvideo so we return a simple string.

class pyslet.qtiv1.common.MatApplet(parent)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.PositionMixin, pyslet.qtiv1.common.MatThingMixin

The `<matapplet>` element is used to contain applet content that is to be displayed to the users. Parameters that are to be passed to the applet being launched should be enclosed in a CDATA block within the content of the `<matapplet>` element:

```xml
<!ELEMENT matapplet (#PCDATA)>
<!ATTLIST matapplet
    label CDATA #IMPLIED
    uri CDATA #IMPLIED
    y0 CDATA #IMPLIED
    height CDATA #IMPLIED
    width CDATA #IMPLIED
    x0 CDATA #IMPLIED
    embedded CDATA 'base64'
    entityref ENTITY #IMPLIED>
```

`extract_text()`
We cannot extract text from matapplet so we return a simple string.
class pyslet.qtiv1.common.MatApplication(parent)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.MatThingMixin

The `<matapplication>` element is used to contain application content that is to be displayed to the users. Parameters that are to be passed to the application being launched should be enclosed in a CDATA block within the content of the `<matapplication>` element:

```xml
<!ELEMENT matapplication (#PCDATA)>
<!ATTLIST matapplication
   apptype CDATA #IMPLIED
   label CDATA #IMPLIED
   uri CDATA #IMPLIED
   embedded CDATA 'base64'
   entityref ENTITY #IMPLIED >
```

extract_text()

We cannot extract text from matapplication so we return a simple string.

class pyslet.qtiv1.common.MatRef(parent)
Bases: pyslet.qtiv1.common.MatThingMixin, pyslet.qtiv1.core.QTIElement

The `<matref>` element is used to contain a reference to the required material. This material will have had an identifier assigned to enable such a reference to be reconciled when the instance is parsed into the system. `<matref>` should only be used to reference a material component and not a `<material>` element (the element `<material_ref>` should be used for the latter):

```xml
<!ELEMENT matref EMPTY>
<!ATTLIST matref linkrefid CDATA #REQUIRED >
```

class pyslet.qtiv1.common.MatExtension(parent, name=None)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.MatThingMixin

The extension facility to enable proprietary types of material to be included with the corresponding data object:

```xml
<!ELEMENT mat_extension ANY>
```

class pyslet.qtiv1.common.FlowMixin
Bases: object

Mix-in class to identify all flow elements:

```python
( flow | flow_mat | flow_label)
```

class pyslet.qtiv1.common.FlowMatContainer(parent)
Bases: pyslet.qtiv1.common.ContentMixin, pyslet.qtiv1.common.QTICommentContainer

Abstract class used to represent objects that contain flow_mat:

```xml
<!ELEMENT XXXXXXXXXX (qticomment? , (material+ | flow_mat+))>
```

class pyslet.qtiv1.common.FlowMat(parent)

This element allows the materials to be displayed to the users to be grouped together using flows. The manner in which these flows are handled is dependent upon the display-engine:

```xml
<!ELEMENT flow_mat (qticomment? , (flow_mat | material | material_ref|))+>
<!ATTLIST flow_mat class CDATA 'Block' >
```
is_inline()

flowmat is always treated as a block if flow_class is specified, otherwise it is treated as a block unless it is an only child.

migrate_content_to_v2(parent, child_type, log)

flow typically maps to a div element.

A flow with a specified class always becomes a div.

class pyslet.qtiv1.common.PresentationMaterial(parent)

Bases: pyslet.qtiv1.common.FlowMatContainer

This is material that must be presented to set the context of the parent evaluation. This could be at the Section level to contain common question material that is relevant to all of the contained Sections/Items. All the contained material must be presented:

```xml
<!ELEMENT presentation_material (qticomment? , flow_mat+)>
```

Our interpretation is generous here, we also accept <material> by default from FlowMatContainer. This element is one of the newer definitions in QTI v1, after the introduction of <flow>. It excludes <material> because it was assumed there would no legacy content. Adoption of flow was poor and it was replaced with direct inclusion of the html model in version 2 (where content is either inline or block level and flow is a general term to describe both for contexts where either is allowed).

class pyslet.qtiv1.common.Reference

Bases: pyslet.qtiv1.common.ContentMixin, pyslet.qtiv1.common.QTICommentContainer

The container for all of the materials that can be referenced by other structures e.g. feedback material, presentation material etc. The presentation of this material is under the control of the structure that is referencing the material. There is no implied relationship between any of the contained material components:

```xml
<!ELEMENT reference (qticomment? , (material | mattext | matemtext | matimage | mataudio | matvideo | matapplet | matapplication | matbreak | mat_extension)+)>
```

class pyslet.qtiv1.common.MaterialRef(parent)

Bases: pyslet.qtiv1.core.QTIElement

The <material_ref> element is used to contain a reference to the required full material block. This material will have had an identifier assigned to enable such a reference to be reconciled when the instance is parsed into the system:

```xml
<!ELEMENT material_ref EMPTY>
<!ATTLIST material_ref linkrefid CDATA #REQUIRED >
```

Metadata Model

class pyslet.qtiv1.common.MetadataContainerMixin

Bases: object

A mix-in class used to hold dictionaries of metadata.

There is a single dictionary maintained to hold all metadata values, each value is a list of tuples of the form (value string, defining element). Values are keyed on the field label or tag name with any leading qmd_ prefix removed.

class pyslet.qtiv1.common.QTIMetadata(parent)

Bases: pyslet.qtiv1.core.QTIElement
The container for all of the vocabulary-based QTI-specific meta-data. This structure is available to each of the four core ASI data structures:

```xml
<!ELEMENT qtimetadata (vocabulary? , qtimetadatafield+)>
```

class `pyslet.qtiv1.common.Vocabulary` *(parent)*

Bases: `pyslet.qtiv1.core.QTIElement`

The vocabulary to be applied to the associated meta-data fields. The vocabulary is defined either using an external file or it is included as a comma separated list:

```xml
<!ELEMENT vocabulary (#PCDATA)>
<!ATTLIST vocabulary
  uri CDATA #IMPLIED
  entityref ENTITY #IMPLIED
  vocab_type CDATA #IMPLIED>
```

class `pyslet.qtiv1.common.QTIMetadataField` *(parent)*

Bases: `pyslet.qtiv1.core.QTIElement`

The structure responsible for containing each of the QTI-specific meta-data fields:

```xml
<!ELEMENT qtimetadatafield (fieldlabel , fieldentry)>
<!ATTLIST qtimetadatafield xml:lang CDATA #IMPLIED>
```

class `pyslet.qtiv1.common.FieldLabel` *(parent, name=None)*

Bases: `pyslet.qtiv1.core.QTIElement`

Used to contain the name of the QTI-specific meta-data field:

```xml
<!ELEMENT fieldlabel (#PCDATA)>
```

class `pyslet.qtiv1.common.FieldEntry` *(parent, name=None)*

Bases: `pyslet.qtiv1.core.QTIElement`

Used to contain the actual data entry of the QTI-specific meta-data field named using the associated ‘fieldlabel’ element:

```xml
<!ELEMENT fieldentry (#PCDATA)>
```

**Objectives & Rubric**

**class** `pyslet.qtiv1.common.Objectives` *(parent)*

Bases: `pyslet.qtiv1.common.FlowMatContainer`

The objectives element is used to store the information that describes the educational aims of the Item. These objectives can be defined for each of the different ‘view’ perspectives. This element should not be used to contain information specific to an Item because the question-engine may not make this information available to the Item during the actual test:

```xml
<!ELEMENT objectives (qticomment? , (material+ | flow_mat+))>
<!ATTLIST objectives view (All | Administrator | AdminAuthority | Assessor | Author | Candidate | InvigilatorProctor | Psychometrician | Scorer | Tutor ) 'All' >
```

**migrate_to_v2** *(v2_item, log)*

Adds rubric representing these objectives to the given item’s body
migrate_objectives_to_lrm(lom, log)
    Adds educational description from these objectives.

class pyslet.qtiv1.common.Rubric(parent)
    Bases: pyslet.qtiv1.common.FlowMatContainer
    The rubric element is used to contain contextual information that is important to the element e.g. it could contain
    standard data values that might or might not be useful for answering the question. Different sets of rubric can
    be defined for each of the possible ‘views’. The material contained within the rubric must be displayed to the
    participant:

    ```xml
    <!ELEMENT rubric (qticomment?, (material+ | flow_mat+))>
    <!ATTLIST rubric view (All | Administrator | AdminAuthority |
    Assessor | Author | Candidate |
    InvigilatorProctor | Psychometrician |
    Scorer | Tutor ) 'All' >
    ```

Response Processing Model

class pyslet.qtiv1.common.DecVar(parent)
    Bases: pyslet.qtiv1.core.QTIElement
    The <decvar> element permits the declaration of the scoring variables

    ```xml
    <!ELEMENT decvar (#PCDATA)>
    <!ATTLIST decvar varname CDATA 'SCORE' ::
    vartype (Integer | String | Decimal | Scientific |
    Boolean | Enumerated | Set ) 'Integer'
    defaultval CDATA #IMPLIED
    minvalue CDATA #IMPLIED
    maxvalue CDATA #IMPLIED
    members CDATA #IMPLIED
    cutvalue CDATA #IMPLIED >
    ```

    content_changed()
    The decvar element is supposed to be empty but QTI v1 content is all over the place.

class pyslet.qtiv1.common.InterpretVar(parent)
    Bases: pyslet.qtiv1.common.ContentMixin, pyslet.qtiv1.core.QTIElement
    The <interpretvar> element is used to provide statistical interpretation information about the associated vari-
    ables:

    ```xml
    <!ELEMENT interpretvar (material | material_ref)>
    <!ATTLIST interpretvar view (All | Administrator | AdminAuthority |
    Author | Candidate | InvigilatorProctor |
    Psychometrician | Scorer | Tutor ) 'All'
    varname CDATA 'SCORE' >
    ```

class pyslet.qtiv1.common.SetVar(parent)
    Bases: pyslet.qtiv1.core.QTIElement
    The <setvar> element is responsible for changing the value of the scoring variable as a result of the associated
    response processing test:

    ```xml
    <!ELEMENT setvar (#PCDATA)>
    <!ATTLIST setvar varname CDATA 'SCORE'
```
class pyslet.qtiv1.common.DisplayFeedback (parent)
Bases: pyslet.qtiv1.core.QTIElement

The `<displayfeedback>` element is responsible for assigning an associated feedback to the response processing if the `True` state is created through the associated response processing condition test:

```xml
<!ELEMENT displayfeedback (#PCDATA)>
<!ATTLIST displayfeedback
  feedbacktype (Response | Solution | Hint ) 'Response'
  linkrefid CDATA #REQUIRED >
```

class pyslet.qtiv1.common.ConditionVar (parent)
Bases: pyslet.qtiv1.core.QTIElement

The conditional test that is to be applied to the user’s response. A wide range of separate and combinatorial test can be applied:

```xml
<!ELEMENT conditionvar (not | and | or | unanswered | other |
  varequal | varlt | varlte | vargt | vargte | varsubset |
  varinside | varsubstring | durequal | durlt | durlte |
  durgt | durgte | var_extension)+>
```

class pyslet.qtiv1.common.ExtendableExpressionMixin
Abstract mixin class to indicate an expression, including var_extension

class pyslet.qtiv1.common.ExpressionMixin
Bases: pyslet.qtiv1.common.ExtendableExpressionMixin

Abstract mixin class to indicate an expression excluding var_extension

class pyslet.qtiv1.common.VarThing (parent)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

Abstract class for var* elements

```xml
<!ATTLIST *
  respident CDATA #REQUIRED
  index CDATA #IMPLIED >
```

class pyslet.qtiv1.common.VarEqual (parent)
Bases: pyslet.qtiv1.common.VarThing

The `<varequal>` element is the test of equivalence. The data for the test is contained within the element’s PCDATA string and must be the same as one of the `<response_label>` values (this were assigned using the ident attribute):

```xml
<!ELEMENT varequal (#PCDATA)>
<!ATTLIST varequal
  case (Yes | No ) 'No'
  respident CDATA #REQUIRED
  index CDATA #IMPLIED >
```

class pyslet.qtiv1.common.VarInequality (parent)
Bases: pyslet.qtiv1.common.VarThing

Abstract class for varlt, varlte, vargt and vargte.
**migrate_inequality_to_v2()**

Returns the class to use in qtiV2

```python
class pyslet.qtiv1.common.VarLT(parent)
    Bases: pyslet.qtiv1.common.VarInequality

The <varlt> element is the ‘less than’ test. The data for the test is contained within the element’s PCDATA string
and is assumed to be numerical in nature:
```
<!ELEMENT varlt (#PCDATA)>
<!ATTLIST varlt
    respident CDATA #REQUIRED*
    index CDATA #IMPLIED >
```

```python
class pyslet.qtiv1.common.VarLTE(parent)
    Bases: pyslet.qtiv1.common.VarInequality

The <varlte> element is the ‘less than or equal’ test. The data for the test is contained within the element’s
PCDATA string and is assumed to be numerical in nature:
```
<!ELEMENT varlte (#PCDATA)>
<!ATTLIST varlte
    respident CDATA #REQUIRED*
    index CDATA #IMPLIED >
```

```python
class pyslet.qtiv1.common.VarGT(parent)
    Bases: pyslet.qtiv1.common.VarInequality

The <vargt> element is the ‘greater than’ test. The data for the test is contained within the element’s PCDATA
string and is assumed to be numerical in nature:
```
<!ELEMENT vargt (#PCDATA)>
<!ATTLIST vargt
    respident CDATA #REQUIRED*
    index CDATA #IMPLIED >
```

```python
class pyslet.qtiv1.common.VarGTE(parent)
    Bases: pyslet.qtiv1.common.VarInequality

The <vargte> element is the ‘greater than or equal to’ test. The data for the test is contained within the element’s
PCDATA string and is assumed to be numerical in nature:
```
<!ELEMENT vargte (#PCDATA)>
<!ATTLIST vargte
    respident CDATA #REQUIRED*
    index CDATA #IMPLIED >
```

```python
class pyslet.qtiv1.common.VarSubset(parent, name=None)
    Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The <varsubset> element is the ‘member of a list/set’ test. The data for the test is contained within the element’s
PCDATA string. The set is a comma separated list with no enclosing parentheses:
```
<!ELEMENT varsubset (#PCDATA)>
<!ATTLIST varsubset
    respident CDATA #REQUIRED*
    setmatch (Exact | Partial ) 'Exact'
    index CDATA #IMPLIED >
```
class pyslet.qtiv1.common.VarSubString(parent, name=None)

Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The `<varsubstring>` element is used to determine if a given string is a substring of some other string:

```xml
<!ELEMENT varsubstring (#PCDATA)>
<!ATTLIST varsubstring
    index CDATA #IMPLIED
    respident CDATA #REQUIRED
    case (Yes | No ) 'No' >
```

class pyslet.qtiv1.common.VarInside(parent)

Bases: pyslet.qtiv1.common.VarThing

The `<varinside>` element is the ‘xy-co-ordinate inside an area’ test. The data for the test is contained within the element’s PCDATA string and is a set of co-ordinates that define the area:

```xml
<!ELEMENT varinside (#PCDATA)>
<!ATTLIST varinside
    areatype (Ellipse | Rectangle | Bounded ) #REQUIRED
    respident CDATA #REQUIRED
    index CDATA #IMPLIED >
```

class pyslet.qtiv1.common.DurEqual(parent, name=None)

Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The `<durequal>` element is the ‘duration equal to’ test i.e. a test on the time taken to make the response:

```xml
<!ELEMENT durequal (#PCDATA)>
<!ATTLIST durequal
    index CDATA #IMPLIED
    respident CDATA #REQUIRED >
```

class pyslet.qtiv1.common.DurLT(parent, name=None)

Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The `<durlt>` element is the ‘duration less than’ test i.e. a test on the time taken to make the response:

```xml
<!ELEMENT durlt (#PCDATA)>
<!ATTLIST durlt
    index CDATA #IMPLIED
    respident CDATA #REQUIRED >
```

class pyslet.qtiv1.common.DurLTE(parent, name=None)

Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The `<durlte>` element is the ‘duration less than or equal to’ test i.e. a test on the time taken to make the response:

```xml
<!ELEMENT durlte (#PCDATA)>
<!ATTLIST durlte
    index CDATA #IMPLIED
    respident CDATA #REQUIRED >
```

class pyslet.qtiv1.common.DurGT(parent, name=None)

Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The `<durgt>` element is the ‘duration greater than’ test i.e. a test on the time taken to make the response:

```xml
<!ELEMENT durgt (#PCDATA)>
<!ATTLIST durgt
    index CDATA #IMPLIED
    respident CDATA #REQUIRED >
```
class pyslet.qtiv1.common.DurGTE(parent, name=None)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The <durgte> element is the ‘duration greater than or equal to’ test i.e. a test on the time taken to make the response:

```xml
<!ELEMENT durgte (#PCDATA)>
<!ATTLIST durgte
    index CDATA #IMPLIED
    respident CDATA #REQUIRED/>
```

class pyslet.qtiv1.common.Not(parent)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The <not> element inverts the logical test outcome that is required. In the case of the <varequal> element produces a ‘not equals’ test:

```xml
<!ELEMENT not (and | or | not | unanswered | other | varequal | varlt | varlte | vargt | vargte | varsubset | varinside | varsubstring | durequal | durlt | durlte | durgt | durgte)>```

class pyslet.qtiv1.common.And(parent)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The <and> element is used to create the Boolean ‘AND’ operation between the two or more enclosed tests. The result ‘True’ is returned if all of the tests return a ‘True’ value:

```xml
<!ELEMENT and (not | and | or | unanswered | other | varequal | varlt | varlte | vargt | vargte | varsubset | varinside | varsubstring | durequal | durlt | durlte | durgt | durgte)+>```

class pyslet.qtiv1.common.Or(parent)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The <or> element is used to create the Boolean ‘OR’ operation between the two or more enclosed tests. The result ‘True’ is returned if one or more of the tests return a ‘True’ value:

```xml
<!ELEMENT or (not | and | or | unanswered | other | varequal | varlt | varlte | vargt | vargte | varsubset | varinside | varsubstring | durequal | durlt | durlte | durgt | durgte)+>```

class pyslet.qtiv1.common.Unanswered(parent, name=None)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin

The <unanswered> element is the condition to be applied if a response is not received for the Item i.e. it is unanswered:

```xml
<!ELEMENT unanswered (#PCDATA)>
<!ATTLIST unanswered
    respident CDATA #REQUIRED */>
```

class pyslet.qtiv1.common.Other(parent, name=None)
Bases: pyslet.qtiv1.core.QTIElement, pyslet.qtiv1.common.ExpressionMixin
The `<other>` element is used to trigger the condition when all of the other tests have not returned a ‘True’ state:

```xml
<!ELEMENT other (#PCDATA)>
```

class pyslet.qtiiv1.common.VarExtension (parent, name=None)
Bases: pyslet.qtiiv1.core.QTIElement, pyslet.qtiiv1.common.ExtendableExpressionMixin

This element contains proprietary extensions to be applied to condition tests. This enables vendors to create their own conditional tests to be used on the participant responses:

```xml
<!ELEMENT var_extension ANY>
```

### Miscellaneous Classes

class pyslet.qtiiv1.common.QTICCommentContainer (parent)
Bases: pyslet.qtiiv1.core.QTIElement

Basic element to represent all elements that can contain a comment as their first child:

```xml
<!ELEMENT XXXXXXXXXXXX (qticomment? , ....... )>
```

class pyslet.qtiiv1.common.QTICComment (parent, name=None)
Bases: pyslet.qtiiv1.core.QTIElement

This element contains the comments that are relevant to the host element. The comment is contained as a string:

```xml
<!ELEMENT qticomment (#PCDATA)>
<!ATTLIST qticomment xml:lang CDATA #IMPLIED >
```

class pyslet.qtiiv1.common.Duration (parent, name=None)
Bases: pyslet.qtiiv1.core.QTIElement

The duration permitted for the completion of a particular activity. The duration is defined as per the ISO8601 standard. The information is entered as a string:

```xml
<!ELEMENT duration (#PCDATA)>
```

The starting point for parsing and managing QTI v1 content.

class pyslet.qtiiv1.xml.QTIDocument (**args)
Bases: pyslet.xml.structures.Document

Class for working with QTI documents.

We turn off the parsing of external general entities to prevent a missing DTD causing the parse to fail. This is a significant limitation as it is possible that some sophisticated users have used general entities to augment the specification or to define boiler-plate code. If this causes problems then you can turn the setting back on again for specific instances of the parser that will be used with that type of data.

XMLParser (entity)

Adds some options to the basic XMLParser to improve QTI compatibility.

get_element_class (name)

Returns the class to use to represent an element with the given name.

This method is used by the XML parser. The class object is looked up in the classMap, if no specialized class is found then the general `pyslet.xml.structures.Element` class is returned.
register_mat_thing(mat_thing)
Registers a MatThing instance in the dictionary of matThings.

find_mat_thing(link_ref_id)
Returns the mat<thing> element with label matching the link_ref_id.  

The specification says that material_ref should be used if you want to refer a material object, not matref,  
however this rule is not universally observed so if we don’t find a basic mat<thing> we will search the  
material objects too and return a Material instance instead.

register_material(material)
Registers a Material instance in the dictionary of labelled material objects.

find_material(link_ref_id)
Returns the material element with label matching link_ref_id.

Like find_mat_thing() this method will search for instances of MatThingMixin if it can’t find a Material element to match. The specification is supposed to be strict about matching the two types of  
reference but errors are common, even in the official example set.

migrate_to_v2(cp)
Converts the contents of this document to QTI v2

The output is stored into the content package passed in cp. Errors and warnings generated by the migration  
process are added as annotations to the resulting resource objects in the content package.

The function returns a list of 4-tuples, one for each object migrated.

Each tuple comprises ( <QTI v2 Document>, <LOM Metadata>, <log>, <Resource> )

3.2.3 QuesTestInterop Elements

class pyslet.qtiv1.xml.QuesTestInterop(parent)
Bases: pyslet.qtiv1.common.QTICommentContainer

Outermost container for QTI content

The <questestinterop> element is the outermost container for the QTI contents i.e. the container of the Assessment(s), Section(s) and Item(s):

<!ELEMENT questestinterop (qticomment? , (objectbank | assessment | 
 (section | item)+))>

migrate_to_v2()
Converts this element to QTI v2

Returns a list of tuples of the form: ( <QTIv2 Document>, <Metadata>, <List of Log Messages> ).

One tuple is returned for each of the objects found. In QTiv2 there is no equivalent of QuesTestInterop.  
The baseURI of each document is set from the baseURI of the QuesTestInterop element using the object identifier to derive a file name.

3.2.4 Object Bank Elements

class pyslet.qtiv1.objectbank.ObjectBank(parent)
QTICommentContainer

This is the container for the Section(s) and/or Item(s) that are to be grouped as an object-bank. The object-bank  
is assigned its own unique identifier and can have the full set of QTI-specific meta-data:
3.2.5 Assessment Elements

class pyslet.qtiv1.assessment.Assessment(parent)
Bases: pyslet.qtiv1.common.QTICommentContainer

The Assessment data structure is used to contain the exchange of test data structures. It will always contain at least one Section and may contain meta-data, objectives, rubric control switches, assessment-level processing, feedback and selection and sequencing information for sections:

```xml
<!ATTLIST assessment ident CDATA #REQUIRED %I_Title; xml:lang CDATA #IMPLIED >
```

migrate_to_v2 (output)

Converts this assessment to QTI v2

For details, see QuesTestInterop.migrate_to_v2.

class pyslet.qtiv1.assessment.AssessmentControl(parent)
Bases: pyslet.qtiv1.common.QTICommentContainer

The control switches that are used to enable or disable the display of hints, solutions and feedback within the Assessment:

```xml
<!ELEMENT assessmentcontrol (qticomment?)>
<!ATTLIST assessmentcontrol
   hintswitch (Yes | No ) 'Yes'
solutionswitch (Yes | No ) 'Yes'
view (All | Administrator | AdminAuthority | Assessor | Author | Candidate | InvigilatorProctor | Psychometrician | Scorer | Tutor ) 'All'
feedbackswitch (Yes | No ) 'Yes' >
```

class pyslet.qtiv1.assessment.AssessProcExtension(parent, name=None)
Bases: pyslet.qtiv1.core.QTIElement

This is used to contain proprietary alternative Assessment-level processing functionality:
3.3 IMS Question and Test Interoperability (version 2.1)

The IMS Question and Test Interoperability specification version 2.1 has yet to be finalized and is currently only available as a “Public Draft Specification” from the IMS GLC website: http://www.imsglobal.org/question/index.html

Version 2.1 is an extension of the pre-existing version 2.0 which was finalized in 2005. For more information on the history of the specification see http://en.wikipedia.org/wiki/QTI

This module implements version 2.1 of the specification in anticipation of the finalization of the specification by the consortium.

3.3.1 Items

class pyslet.qtiv2.items.AssessmentItem (parent)
Bases: pyslet.qtiv2.core.QTIElement, pyslet.qtiv2.core.DeclarationContainer

An assessment item encompasses the information that is presented to a candidate and information about how to score the item:

```xml
<xsd:attributeGroup name="assessmentItem.AttrGroup">
    <xsd:attribute name="identifier" type="string.Type" use="required"/>
    <xsd:attribute name="title" type="string.Type" use="required"/>
    <xsd:attribute name="label" type="string256.Type" use="optional"/>
    <xsd:attribute ref="xml:lang"/>
    <xsd:attribute name="adaptive" type="boolean.Type" use="required"/>
    <xsd:attribute name="timeDependent" type="boolean.Type" use="required"/>
    <xsd:attribute name="toolName" type="string256.Type" use="optional"/>
    <xsd:attribute name="toolVersion" type="string256.Type" use="optional"/>
</xsd:attributeGroup>
<xsd:group name="assessmentItem.ContentGroup">
```
<xsd:sequence>
  <xsd:element ref="responseDeclaration"
    minOccurs="0"
    maxOccurs="unbounded"/>
  <xsd:element ref="outcomeDeclaration"
    minOccurs="0"
    maxOccurs="unbounded"/>
  <xsd:element ref="templateDeclaration"
    minOccurs="0"
    maxOccurs="unbounded"/>
  <xsd:element ref="templateProcessing"
    minOccurs="0"
    maxOccurs="1"/>
  <xsd:element ref="stylesheet"
    minOccurs="0"
    maxOccurs="unbounded"/>
  <xsd:element ref="itemBody"
    minOccurs="0"
    maxOccurs="1"/>
  <xsd:element ref="responseProcessing"
    minOccurs="0"
    maxOccurs="1"/>
  <xsd:element ref="modalFeedback"
    minOccurs="0"
    maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:group>

sort_declarations()

Sort each of the variable declaration lists so that they are in identifier order. This is not essential but it does help ensure that output is predictable. This method is called automatically when reading items from XML files.

render_html(item_state, html_parent=None)

Renders this item in html, adding nodes to html_parent. The state of the item (e.g., the values of any controls and template variables), is taken from item_state, a variables.ItemSessionState instance.

The result is the top-level div containing the item added to the html_parent. If html_parent is None then a parentless div is created. If the item has no itemBody then an empty Div is returned.

add_to_content_package(cp, lom, dname=None)

Adds a resource and associated files to the content package.

AddToContentPackage(*args, **kwargs)

Deprecated equivalent to add_to_content_package()

3.3.2 Tests

class pyslet.qtiv2.tests.AssessmentTest(parent)
Bases: pyslet.qtiv2.core.QTIElement, pyslet.qtiv2.core.DeclarationContainer

A test is a group of assessmentItems with an associated set of rules that determine which of the items the candidate sees, in what order, and in what way the candidate interacts with them. The rules describe the valid paths through the test, when responses are submitted for response processing and when (if at all) feedback is to be given:
<xsd:attributeGroup name="assessmentTest.AttrGroup">
    <xsd:attribute name="identifier" type="string.Type" use="required"/>
    <xsd:attribute name="title" type="string.Type" use="required"/>
    <xsd:attribute name="toolName" type="string256.Type" use="optional"/>
    <xsd:attribute name="toolVersion" type="string256.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="assessmentTest.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="outcomeDeclaration" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="timeLimits" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="testPart" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element ref="outcomeProcessing" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="testFeedback" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>

sort_declarations()
Sort the outcome declarations so that they are in identifier order. This is not essential but it does help ensure that output is predictable. This method is called automatically when reading items from XML files.

register_part (part)
Registers a testPart, assessmentSection or assessmentItemRef in parts.

get_part (identifier)
Returns the testPart, assessmentSection or assessmentItemRef with the given identifier.

GetPart (*args, **kwargs)
Deprecated equivalent to get_part()

RegisterPart (*args, **kwargs)
Deprecated equivalent to register_part()

Navigation and Submission

class pyslet.qtiv2.tests.NavigationMode
    Bases: pyslet.xml.xsdatypes Enumeration

    The navigation mode determines the general paths that the candidate may take. A testPart in linear mode restricts the candidate to attempt each item in turn. Once the candidate moves on they are not permitted to return. A testPart in nonlinear mode removes this restriction - the candidate is free to navigate to any item in the test at any time:

    <xsd:simpleType name="navigationMode.Type">
        <xsd:restriction base="xsd:NMTOKEN">
            <xsd:enumeration value="linear"/>
            <xsd:enumeration value="nonlinear"/>
        </xsd:restriction>
    </xsd:simpleType>
Defines constants for the above modes. Usage example:

```python
NavigationMode.linear
```

Note that:

```python
NavigationMode.DEFAULT == None
```

For more methods see `Enumeration`

```python
class pyslet.qti.v2.tests.SubmissionMode
Bases: pyslet.xml.xsd.datatypes Enumeration
```

The submission mode determines when the candidate’s responses are submitted for response processing. A testPart in individual mode requires the candidate to submit their responses on an item-by-item basis. In simultaneous mode the candidate’s responses are all submitted together at the end of the testPart:

```xml
<xsd:simpleType name="submissionMode.Type">
    <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="individual"/>
        <xsd:enumeration value="simultaneous"/>
    </xsd:restriction>
</xsd:simpleType>
```

Defines constants for the above modes. Usage example:

```python
SubmissionMode.individual
```

Note that:

```python
SubmissionMode.DEFAULT == None
```

For more methods see `Enumeration`

```python
class pyslet.qti.v2.tests.TestPart (parent)
Bases: pyslet.qti.v2.core.QTIElement
```

Each test is divided into one or more parts which may in turn be divided into sections, sub-sections, and so on:

```xml
<xsd:attributeGroup name="testPart.AttrGroup">
    <xsd:attribute name="identifier" type="identifier.Type" use="required"/>
    <xsd:attribute name="navigationMode" type="navigationMode.Type" use="required"/>
    <xsd:attribute name="submissionMode" type="submissionMode.Type" use="required"/>
</xsd:attributeGroup>

<xsd:group name="testPart.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="preCondition" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="branchRule" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="itemSessionControl" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="timeLimits" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:group>
```
check_pre_conditions(state)
Returns True if this testPart's pre-conditions are satisfied or if there are no pre-conditions in effect.

get_branch_target(state)
Returns the identifier of the testPart to branch to, or the pre-defined EXIT_TEST identifier. If there is no branch rule in effect then None is returned. state is a variables.TestSessionState instance used to evaluate the branch rule expressions.

Test Structure

class pyslet.qtiv2.tests.Selection (parent)
Bases: pyslet.qtiv2.core.QTIElement
The selection class specifies the rules used to select the child elements of a section for each test session:

```xml
<xsd:element ref="assessmentSection" minOccurs="1" maxOccurs="unbounded"/>
<xsd:element ref="testFeedback" minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.tests.Ordering (parent)
Bases: pyslet.qtiv2.core.QTIElement
The ordering class specifies the rule used to arrange the child elements of a section following selection. If no ordering rule is given we assume that the elements are to be ordered in the order in which they are defined:

```xml
<xsd:attributeGroup name="ordering.AttrGroup">
  <xsd:attribute name="shuffle" type="boolean.Type" use="required"/>
  <xsd:anyAttribute namespace="##other"/>
</xsd:attributeGroup>
<xsd:group name="ordering.ContentGroup">
  <xsd:sequence>
    <xsd:element name="any" minOccurs="0" maxOccurs="unbounded" processContents="skip"/>
  </xsd:sequence>
</xsd:group>
```
class pyslet.qtiv2.tests.SectionPart (parent)
    Bases: pyslet.qtiv2.core.QTIElement

Sections group together individual item references and/or sub-sections. A number of common parameters are
shared by both types of child element:

```xml
<xsd:attributeGroup name="sectionPart.AttrGroup">
    <xsd:attribute name="identifier" type="identifier.Type" use="required"/>
    <xsd:attribute name="required" type="boolean.Type" use="optional"/>
    <xsd:attribute name="fixed" type="boolean.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="sectionPart.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="preCondition" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="branchRule" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="itemSessionControl" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="timeLimits" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:group>
```

**check_pre_conditions** (*state*)

Returns True if this item or section’s pre-conditions are satisfied or if there are no pre-conditions in effect.

**get_branch_target** (*state*)

Returns the identifier of the next item or section to branch to, or one of the pre-defined EXIT_* identifiers. If there is no branch rule in effect then None is returned. *state* is a variables.TestSessionState instance used to evaluate the branch rule expressions.

class pyslet.qtiv2.tests.AssessmentSection (parent)
    Bases: pyslet.qtiv2.tests.SectionPart

Represents assessmentSection element

```xml
<xsd:attributeGroup name="assessmentSection.AttrGroup">
    <xsd:attributeGroup ref="sectionPart.AttrGroup"/>
    <xsd:attribute name="title" type="string.Type" use="required"/>
    <xsd:attribute name="visible" type="boolean.Type" use="required"/>
    <xsd:attribute name="keepTogether" type="boolean.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="assessmentSection.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="sectionPart.ContentGroup"/>
        <xsd:element ref="selection" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="ordering" minOccurs="0" maxOccurs="1"/>
        <xsd:element ref="rubricBlock" minOccurs="0"/>
    </xsd:sequence>
</xsd:group>
```
class pyslet.qtiv2.tests.AssessmentItemRef (parent)

Bases: pyslet.qtiv2.tests.SectionPart

Items are incorporated into the test by reference and not by direct aggregation:

get_item()

Returns the AssessmentItem referred to by this reference.

GetItem (*args, **kwargs)

Deprecated equivalent to get_item()

SetTemplateDefaults (*args, **kwargs)

Deprecated equivalent to set_template_defaults()

3.3.3 Content Model

class pyslet.qtiv2.content.ItemBody (parent)

Bases: pyslet.qtiv2.content.BodyElement

The item body contains the text, graphics, media objects, and interactions that describe the item’s content and information about how it is structured:
render_html (parent, profile, item_state)

Overrides BodyElement.render_html(), the result is always a Div with class set to “itemBody”. Unlike other such method parent may by None, in which case a new parentless Div is created.

class pyslet.qti2.content.BodyElement (parent)

Bases: pyslet.qti2.core.QTIElement

The root class of all content objects in the item content model is the bodyElement. It defines a number of attributes that are common to all elements of the content model:

render_html (parent, profile, item_state)

Renders this element in html form, adding nodes to parent. This method effectively overrides html401.XPATHElement.render_html enabling QTI and XHTML elements to be mixed freely.

The state of the item (e.g., the values of any controls), is taken from item_state, a ItemSessionState instance.

render_html_children (parent, profile, item_state)

Renders this element’s children to an external document represented by the parent node

3.3. IMS Question and Test Interoperability (version 2.1)
XHTML Elements

Again, these classes are defined in the accompanying html401 module, however we do define some profiles here to make it easier to constraint general HTML content to the profile defined here.

```python
pyslet.qtiv2.content.TextElements = {'em': ('id', 'class', 'label'), 'pre': ('id', 'class', ... 'class', 'label'), 'p': ('id', 'class', 'label'), 'div': ('id', 'class', 'label'), 'abbr': ('id', 'class', 'label')}
```

Basic text formatting elements

```python
pyslet.qtiv2.content.ListElements = {'dl': ('id', 'class', 'label'), 'ol': ('id', 'class', ... 'class', 'label'), 'dt': ('id', 'class', 'label')}
```

Elements required for lists

```python
pyslet.qtiv2.content.ObjectElements = {'object': ('id', 'class', 'label', 'data', 'type', ... 'class', 'label')}
```

The object element

```python
pyslet.qtiv2.content.PresentationElements = {'caption': ('id', 'class', 'label'), 'tfoot': ('id', 'class', 'label')}
```

Tables

```python
pyslet.qtiv2.content.ImageElement = {'img': ('id', 'class', 'label', 'src', 'alt', 'longdesc', 'height', 'width')}
```

Images

```python
pyslet.qtiv2.content.HypertextElement = {'a': ('id', 'class', 'label', 'href', 'type')}
```

Hyperlinks

```python
pyslet.qtiv2.content.HTMLProfile = {'em': ('id', 'class', 'label'), 'pre': ('id', 'class', ... 'label')}
```

The full HTML profile defined by QTI

### 3.3.4 Interactions

```python
class pyslet.qtiv2.interactions.Interaction(parent)
   Bases: pyslet.qtiv2.content.BodyElement
```

Interactions allow the candidate to interact with the item. Through an interaction, the candidate selects or constructs a response:

```xml
<xsd:attributeGroup name="interaction.AttrGroup">
    <xsd:attributeGroup ref="bodyElement.AttrGroup"/>
    <xsd:attribute name="responseIdentifier" type="identifier.Type" use="required"/>
</xsd:attributeGroup>
```

```python
class pyslet.qtiv2.interactions.InlineInteraction(parent)
   Bases: pyslet.html401.InlineMixin, pyslet.qtiv2.interactions.Interaction
```

Abstract class for interactions that appear inline.

```python
class pyslet.qtiv2.interactions.BlockInteraction(parent)
   Bases: pyslet.html401.BlockMixin, pyslet.qtiv2.interactions.Interaction
```

An interaction that behaves like a block in the content model. Most interactions are of this type:

```xml
<xsd:group name="blockInteraction.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="prompt" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:group>
```

```python
class pyslet.qtiv2.interactions.Prompt(parent)
   Bases: pyslet.qtiv2.content.BodyElement
```
The prompt used in block interactions

```
<xsd:group name="prompt.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="inlineStatic.ElementGroup"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

class `pyslet.qtiv2.interactions.Choice` *(parent)*

Bases: `pyslet.qtiv2.content.BodyElement`

Many of the interactions involve choosing one or more predefined choices

```
<xsd:attributeGroup name="choice.AttrGroup">
  <xsd:attributeGroup ref="bodyElement.AttrGroup"/>
  <xsd:attribute name="identifier" type="identifier.Type"
    use="required"/>
  <xsd:attribute name="fixed" type="boolean.Type"
    use="optional"/>
  <xsd:attribute name="templateIdentifier"
    type="identifier.Type" use="optional"/>
  <xsd:attribute name="showHide" type="showHide.Type"
    use="optional"/>
</xsd:attributeGroup>
```

class `pyslet.qtiv2.interactions.AssociableChoice` *(parent)*

Bases: `pyslet.qtiv2.interactions.Choice`

Other interactions involve associating pairs of predefined choices

```
<xsd:attributeGroup name="associableChoice.AttrGroup">
  <xsd:attributeGroup ref="choice.AttrGroup"/>
  <xsd:attribute name="matchGroup" use="optional">
    <xsd:simpleType>
      <xsd:list itemType="identifier.Type"/>
    </xsd:simpleType>
  </xsd:attribute>
</xsd:attributeGroup>
```

### Simple Interactions

class `pyslet.qtiv2.interactions.ChoiceInteraction` *(parent)*

Bases: `pyslet.qtiv2.interactions.BlockInteraction`

The choice interaction presents a set of choices to the candidate. The candidate’s task is to select one or more of the choices, up to a maximum of maxChoices:

```
<xsd:attributeGroup name="choiceInteraction.AttrGroup">
  <xsd:attributeGroup ref="blockInteraction.AttrGroup"/>
  <xsd:attribute name="shuffle" type="boolean.Type"
    use="required"/>
  <xsd:attribute name="maxChoices" type="integer.Type"
    use="required"/>
  <xsd:attribute name="minChoices" type="integer.Type"
    use="optional"/>
</xsd:attributeGroup>
```
In an order interaction the candidate’s task is to reorder the choices, the order in which the choices are displayed initially is significant:

A SimpleChoice is a choice that contains flow objects; it must not contain any nested interactions:

An associate interaction is a blockInteraction that presents candidates with a number of choices and allows them to create associations between them:
class `pyslet.QtIV2.interactions.MatchInteraction` *(parent)*

Bases: `pyslet.QtIV2.interactions.BlockInteraction`

A match interaction is a blockInteraction that presents candidates with two sets of choices and allows them to create associates between pairs of choices in the two sets, but not between pairs of choices in the same set:

```xml
<xsd:attributeGroup name="matchInteraction.AttrGroup">
    <xsd:attributeGroup ref="blockInteraction.AttrGroup"/>
    <xsd:attribute name="shuffle" type="boolean.Type" use="required"/>
    <xsd:attribute name="maxAssociations" type="integer.Type" use="required"/>
    <xsd:attribute name="minAssociations" type="integer.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="matchInteraction.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="blockInteraction.ContentGroup"/>
        <xsd:element ref="simpleMatchSet" minOccurs="2" maxOccurs="2"/>
    </xsd:sequence>
</xsd:group>
```

class `pyslet.QtIV2.interactions.SimpleAssociableChoice` *(parent)*


associableChoice is a choice that contains flowStatic objects, it must not contain nested interactions:

```xml
<xsd:attributeGroup name="simpleAssociableChoice.AttrGroup">
    <xsd:attributeGroup ref="associableChoice.AttrGroup"/>
    <xsd:attribute name="matchMax" type="integer.Type" use="required"/>
    <xsd:attribute name="matchMin" type="integer.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="simpleAssociableChoice.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="flowStatic.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```
class pyslet.qtiv2.interactions.SimpleMatchSet (parent)
Bases: pyslet.qtiv2.core.QTIElement
Contains an ordered set of choices for the set

```xml
<xsd:group name="simpleMatchSet.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="simpleAssociableChoice"
            minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.interactions.GapMatchInteraction (parent)
Bases: pyslet.qtiv2.interactions.BlockInteraction
A gap match interaction is a blockInteraction that contains a number gaps that the candidate can fill from an associated set of choices:

```xml
<xsd:attributeGroup name="gapMatchInteraction.AttrGroup">
    <xsd:attributeGroup ref="blockInteraction.AttrGroup"/>
    <xsd:attribute name="shuffle" type="boolean.Type" use="required"/>
</xsd:attributeGroup>
<xsd:group name="gapMatchInteraction.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="blockInteraction.ContentGroup"/>
        <xsd:group ref="gapChoice.ElementGroup" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:group ref="blockStatic.ElementGroup" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.interactions.Gap (parent)
Bases: pyslet.html401.InlineMixin, pyslet.qtiv2.interactions.AssociableChoice
A gap is an inline element that must only appear within a gapMatchInteraction

```xml
<xsd:attributeGroup name="gap.AttrGroup">
    <xsd:attributeGroup ref="associableChoice.AttrGroup"/>
    <xsd:attribute name="required" type="boolean.Type" use="optional"/>
</xsd:attributeGroup>
```

class pyslet.qtiv2.interactions.GapChoice (parent)
Bases: pyslet.qtiv2.interactions.AssociableChoice
The choices that are used to fill the gaps in a gapMatchInteraction are either simple runs of text or single image objects, both derived from gapChoice:

```xml
<xsd:attributeGroup name="gapChoice.AttrGroup">
    <xsd:attributeGroup ref="associableChoice.AttrGroup"/>
    <xsd:attribute name="matchMax" type="integer.Type" use="required"/>
    <xsd:attribute name="matchMin" type="integer.Type" use="optional"/>
</xsd:attributeGroup>
```
class `pyslet.qtiv2.interactions.GapText` *(parent)*
Bases: `pyslet.qtiv2.interactions.GapChoice`

A simple run of text to be inserted into a gap by the user, may be subject to variable value substitution with `printedVariable`:

```xml
<xsd:group name="gapText.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="printedVariable" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

class `pyslet.qtiv2.interactions.GapImg` *(parent)*
Bases: `pyslet.qtiv2.interactions.GapChoice`

A gap image contains a single image object to be inserted into a gap by the candidate:

```xml
<xsd:attributeGroup name="gapImg.AttrGroup">
    <xsd:attributeGroup ref="gapChoice.AttrGroup"/>
    <xsd:attribute name="objectLabel" type="string.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="gapImg.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="object" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:group>
```

### Text-based Interactions

class `pyslet.qtiv2.interactions.InlineChoiceInteraction` *(parent)*
Bases: `pyslet.qtiv2.interactions.InlineInteraction`

An inline choice is an inlineInteraction that presents the user with a set of choices, each of which is a simple piece of text:

```xml
<xsd:attributeGroup name="inlineChoiceInteraction.AttrGroup">
    <xsd:attributeGroup ref="inlineInteraction.AttrGroup"/>
    <xsd:attribute name="shuffle" type="boolean.Type" use="required"/>
    <xsd:attribute name="required" type="boolean.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="inlineChoiceInteraction.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="inlineChoice" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

class `pyslet.qtiv2.interactions.InlineChoice` *(parent)*
Bases: `pyslet.qtiv2.interactions.Choice`
A simple run of text to be displayed to the user, may be subject to variable value substitution with printedVariable:

```xml
<xsd:group name="inlineChoice.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="printedVariable" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

class `pyslet.qtiv2.interactions.StringInteractionMixin`

Abstract mix-in class for interactions based on free-text input. String interactions can be bound to numeric response variables, instead of strings, if desired:

```xml
<xsd:attributeGroup name="stringInteraction.AttrGroup">
    <xsd:attribute name="base" type="integer.Type" use="optional"/>
    <xsd:attribute name="stringIdentifier" type="identifier.Type" use="optional"/>
    <xsd:attribute name="expectedLength" type="integer.Type" use="optional"/>
    <xsd:attribute name="patternMask" type="string.Type" use="optional"/>
    <xsd:attribute name="placeholderText" type="string.Type" use="optional"/>
</xsd:attributeGroup>
```

class `pyslet.qtiv2.interactions.TextEntryInteraction` *(parent)*


A textEntry interaction is an inlineInteraction that obtains a simple piece of text from the candidate.

class `pyslet.qtiv2.interactions.TextFormat`

Bases: `pyslet.xml.xsdatatypes.EnumerationNoCase`

Used to control the format of the text entered by the candidate:

```xml
<xsd:simpleType name="textFormat.Type">
    <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="plain"/>
        <xsd:enumeration value="preFormatted"/>
        <xsd:enumeration value="xhtml"/>
    </xsd:restriction>
</xsd:simpleType>
```

Defines constants for the above formats. Usage example:

TextFormat.plain

Note that:

TextFormat.DEFAULT == TextFormat.plain

For more methods see `Enumeration`

class `pyslet.qtiv2.interactions.ExtendedTextInteraction` *(parent)*

An extended text interaction is a blockInteraction that allows the candidate to enter an extended amount of text:

```xml
<xsd:attributeGroup name="extendedTextInteraction.AttrGroup">
  <xsd:attributeGroup ref="blockInteraction.AttrGroup"/>
  <xsd:attributeGroup ref="stringInteraction.AttrGroup"/>
  <xsd:attribute name="maxStrings" type="integer.Type" use="optional"/>
  <xsd:attribute name="minStrings" type="integer.Type" use="optional"/>
  <xsd:attribute name="expectedLines" type="integer.Type" use="optional"/>
  <xsd:attribute name="format" type="textFormat.Type" use="optional"/>
</xsd:attributeGroup>
```

class `pyslet.qtiv2.interactions.HottextInteraction` *(parent)*  
Bases: `pyslet.qtiv2.interactions.BlockInteraction`

The hottest interaction presents a set of choices to the candidate represented as selectable runs of text embedded within a surrounding context, such as a simple passage of text:

```xml
<xsd:attributeGroup name="hottextInteraction.AttrGroup">
  <xsd:attributeGroup ref="blockInteraction.AttrGroup"/>
  <xsd:attribute name="maxChoices" type="integer.Type" use="required"/>
  <xsd:attribute name="minChoices" type="integer.Type" use="optional"/>
</xsd:attributeGroup>
<xsd:group name="hottextInteraction.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="blockInteraction.ContentGroup"/>
    <xsd:group ref="blockStatic.ElementGroup" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

class `pyslet.qtiv2.interactions.Hottext` *(parent)*  

A hottext area is used within the content of an hottextInteraction to provide the individual choices:

```xml
<xsd:group name="hottext.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="inlineStatic.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

### Graphical Interactions

**class** `pyslet.qtiv2.interactions.GapImg` *(parent)*  
**Bases:** `pyslet.qtiv2.interactions.GapChoice`

A gap image contains a single image object to be inserted into a gap by the candidate:

```xml
<xsd:attributeGroup name="gapImg.AttrGroup">
  <xsd:attributeGroup ref="gapChoice.AttrGroup"/>
</xsd:attributeGroup>
```
A gap image contains a single image object to be inserted into a gap by the candidate:

```
<xsd:attributeGroup name="gapImg.AttrGroup">
  <xsd:attributeGroup ref="gapChoice.AttrGroup"/>
  <xsd:attribute name="objectLabel" type="string.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="gapImg.ContentGroup">
  <xsd:sequence>
    <xsd:element ref="object" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.interactions.GapImg (parent)
Bases: pyslet.qtiv2.interactions.GapChoice

A gap image contains a single image object to be inserted into a gap by the candidate:

```
<xsd:attributeGroup name="gapImg.AttrGroup">
  <xsd:attributeGroup ref="gapChoice.AttrGroup"/>
  <xsd:attribute name="objectLabel" type="string.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="gapImg.ContentGroup">
  <xsd:sequence>
    <xsd:element ref="object" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

### 3.3.5 Item Variables

This module contains the basic run-time data model. Although the specification does contain elements to represent the values of variables set at runtime the XML schema sometimes relies too much on context for an efficient implementation. For example, a `<value>` element is always a value of a specific base type but the base type is rarely specified on the value element itself as it is normally implicit in the context, such as a variable declaration.

Although the expression model does contain an element that provides a more complete representation of single values (namely `<baseValue>`) we decide to make the distinction in this module with `ValueElement` representing the element and the abstract `Value` being used as the root of the runtime object model.
For example, to get the default value of a variable from a variable declaration you’ll use the `get_default_value()` method and it will return a `Value` instance which could be of any cardinality or base type.

```python
class pyslet.qtiv2.variables.VariableDeclaration(parent)
    Bases: pyslet.qtiv2.core.QTIElement, pyslet.py2.SortableMixin

    Item variables are declared by variable declarations... The purpose of the declaration is to associate an identifier with the variable and to identify the runtime type of the variable’s value:

    get_default_value()
        Returns a `Value` instance representing either the default value or an appropriately typed NULL value if there is no default defined.

    GetDefaultValue(*args, **kwargs)
        Deprecated equivalent to `get_default_value()`
```

```xml
<xs:attributeGroup name="variableDeclaration.AttrGroup">
    <xs:attribute name="identifier" type="identifier.Type" use="required"/>
    <xs:attribute name="cardinality" type="cardinality.Type" use="required"/>
    <xs:attribute name="baseType" type="baseType.Type" use="optional"/>
</xs:attributeGroup>

<xs:group name="variableDeclaration.ContentGroup">
    <xs:sequence>
        <xs:element ref="defaultValue" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
</xs:group>
```

```python
get_default_value()
    Returns a `Value` instance representing either the default value or an appropriately typed NULL value if there is no default defined.

GetDefaultValue(*args, **kwargs)
    Deprecated equivalent to `get_default_value()`
```
Defines constants for the above carinalities. Usage example:

Cardinality.multiple

There is no default:

Cardinality.DEFAULT == None

For more methods see Enumeration

class pyslet.qtiv2.variables.BaseType
Bases: pyslet.xml.xsdatatypes EnumerationNoCase

A base-type is simply a description of a set of atomic values (atomic to this specification). Note that several of
the baseTypes used to define the runtime data model have identical definitions to those of the basic data types
used to define the values for attributes in the specification itself. The use of an enumeration to define the set of
baseTypes used in the runtime model, as opposed to the use of classes with similar names, is designed to help
distinguish between these two distinct levels of modelling:

Defines constants for the above base types. Usage example:

BaseType.float

There is no default:

BaseType.DEFAULT == None

For more methods see Enumeration

class pyslet.qtiv2.variables.Mapping (parent)
Bases: pyslet.qtiv2.core.QTIElement

A special class used to create a mapping from a source set of any baseType (except file and duration) to a single
float:
content_changed()
Builds an internal dictionary of the values being mapped.

In order to fully specify the mapping we need to know the baseType of the source values. (The targets are always floats.) We do this based on our parent, orphan Mapping elements are treated as mappings from source strings.

map_value(value)
Maps an instance of Value with the same base type as the mapping to an instance of Value with base type float.

MapValue(*args, **kwargs)
Deprecated equivalent to map_value()

class pyslet.qtiv2.variables.MapEntry(parent)
Bases: pyslet.qtiv2.core.QTIElement
An entry in a Mapping

mapKey = None
The source value

mappedValue = None
The mapped value

Response Variables

class pyslet.qtiv2.variables.ResponseDeclaration(parent)
Bases: pyslet.qtiv2.variables.VariableDeclaration
Response variables are declared by response declarations and bound to interactions in the itemBody:
**get_correct_value()**

Returns a *Value* instance representing either the correct response value or an appropriately typed NULL value if there is no correct value.

**get_stage_dimensions()**

For response variables with point type, returns a pair of integer values: width, height

In HTML, shapes (including those used in the AreaMapping) can use relative coordinates. To interpret relative coordinates we need to know the size of the stage used to interpret the point values. For a response variable that is typically the size of the image or object used in the interaction.

This method searches for the interaction associated with the response and obtains the width and height of the corresponding object.

[TODO: currently returns 100,100]

```python
class pyslet.qtiv2.variables.CorrectResponse(parent):
    Bases: pyslet.qtiv2.variables.DefinedValue

    A response declaration may assign an optional correctResponse. This value may indicate the only possible value of the response variable to be considered correct or merely just a correct value.
```

```python
class pyslet.qtiv2.variables.AreaMapping(parent):
    Bases: pyslet.qtiv2.core.QTIElement

    A special class used to create a mapping from a source set of point values to a target set of float values:
```

```xml
<xsd:attributeGroup name="areaMapping.AttrGroup">
    <xsd:attribute name="lowerBound" type="float.Type" use="optional"/>
    <xsd:attribute name="upperBound" type="float.Type" use="optional"/>
    <xsd:attribute name="defaultValue" type="float.Type" use="required"/>
</xsd:attributeGroup>

<xsd:group name="areaMapping.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="areaMapEntry" minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

**map_value(value, width, height)**

Maps a point onto a float.

Returns an instance of *Value* with base type float.

- value is a *Value* of base type point
- width is the integer width of the object on which the area is defined
• height is the integer height of the object on which the area is defined

The width and height of the object are required because HTML allows relative values to be used when defining areas.

MapValue(*args, **kwargs)

Deprecated equivalent to map_value()

class pyslet.qtiv2.variables.AreaMapEntry(parent)

Bases: pyslet.qtiv2.core.QTIElement, pyslet.qtiv2.core.ShapeElementMixin

An AreaMapping is defined by a set of areaMapEntries, each of which maps an area of the coordinate space onto a single float:

```xml
<xsd:attributeGroup name="areaMapEntry.AttrGroup">
  <xsd:attribute name="shape" type="shape.Type" use="required"/>
  <xsd:attribute name="coords" type="coords.Type" use="required"/>
  <xsd:attribute name="mappedValue" type="float.Type" use="required"/>
</xsd:attributeGroup>
```

mappedValue = None

The mapped value

Outcome Variables

class pyslet.qtiv2.variables.OutcomeDeclaration(parent)

Bases: pyslet.qtiv2.variables.VariableDeclaration

Outcome variables are declared by outcome declarations

```xml
<xsd:attributeGroup name="outcomeDeclaration.AttrGroup">
  <xsd:attributeGroup ref="variableDeclaration.AttrGroup"/>
  <xsd:attribute name="view" use="optional">
    <xsd:simpleType>
      <xsd:list itemType="view.Type"/>
    </xsd:simpleType>
  </xsd:attribute>
  <xsd:attribute name="interpretation" type="string.Type" use="optional"/>
  <xsd:attribute name="longInterpretation" type="uri.Type" use="optional"/>
  <xsd:attribute name="normalMaximum" type="float.Type" use="optional"/>
  <xsd:attribute name="normalMinimum" type="float.Type" use="optional"/>
  <xsd:attribute name="masteryValue" type="float.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="outcomeDeclaration.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="variableDeclaration.ContentGroup"/>
    <xsd:group ref="lookupTable.ElementGroup" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```
class pyslet.qtiv2.variables.LookupTable(parent)
Bases: pyslet.qtiv2.core.QTIElement

An abstract class associated with an outcomeDeclaration used to create a lookup table from a numeric source value to a single outcome value in the declared value set:

```xml
<xsd:attributeGroup name="lookupTable.AttrGroup">
    <xsd:attribute name="defaultValue" type="valueType.Type"
        use="optional"/>
</xsd:attributeGroup>
```

default = None

A Value instance representing the default

class pyslet.qtiv2.variables.MatchTable(parent)
Bases: pyslet.qtiv2.variables.LookupTable

A matchTable transforms a source integer by finding the first matchTableEntry with an exact match to the source:

```xml
<xsd:group name="matchTable.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="matchTableEntry" minOccurs="1"
            maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

current_changed() Builds an internal dictionary of the values being mapped.

lookup(value) Maps an instance of Value with integer base type to an instance of Value with the base type of the match table.

class pyslet.qtiv2.variables.MatchTableEntry(parent)
Bases: pyslet.qtiv2.core.QTIElement

sourceValue The source integer that must be matched exactly.

targetValue The target value that is used to set the outcome when a match is found

```xml
<xsd:attributeGroup name="matchTableEntry.AttrGroup">
    <xsd:attribute name="sourceValue" type="integer.Type"
        use="required"/>
    <xsd:attribute name="targetValue" type="valueType.Type"
        use="required"/>
</xsd:attributeGroup>
```

class pyslet.qtiv2.variables.InterpolationTable(parent)
Bases: pyslet.qtiv2.variables.LookupTable

An interpolationTable transforms a source float (or integer) by finding the first interpolationTableEntry with a sourceValue that is less than or equal to (subject to includeBoundary) the source value:

```xml
<xsd:group name="interpolationTable.ContentGroup">
    <xsd:sequence>
        <xsd:element ref="interpolationTableEntry" minOccurs="1"
            maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```
content_changed()
    Builds an internal table of the values being mapped.

lookup(value)
    Maps an instance of Value with integer or float base type to an instance of Value with the base type of
    the interpolation table.

class pyslet.qtiv2.variables.InterpolationTableEntry(parent)
    Bases: pyslet.qtiv2.core.QTIElement

    sourceValue The lower bound for the source value to match this entry.
    includeBoundary Determines if an exact match of sourceValue matches this entry. If true, the default, then an
    exact match of the value is considered a match of this entry.
    targetValue The target value that is used to set the outcome when a match is found

<xsd:attributeGroup name="interpolationTableEntry.AttrGroup">
    <xsd:attribute name="sourceValue" type="float.Type" use="required"/>
    <xsd:attribute name="includeBoundary" type="boolean.Type" use="optional"/>
    <xsd:attribute name="targetValue" type="valueType.Type" use="required"/>
</xsd:attributeGroup>

Template Variables

class pyslet.qtiv2.variables.TemplateDeclaration(parent)
    Bases: pyslet.qtiv2.variables.VariableDeclaration

    Template declarations declare item variables that are to be used specifically for the purposes of cloning items

<xsd:attributeGroup name="templateDeclaration.AttrGroup">
    <xsd:attributeGroup ref="variableDeclaration.AttrGroup"/>
    <xsd:attribute name="paramVariable" type="boolean.Type" use="optional"/>
    <xsd:attribute name="mathVariable" type="boolean.Type" use="optional"/>
</xsd:attributeGroup>

Runtime Object Model

class pyslet.qtiv2.variables.SessionState
    Bases: pyslet.pep8.MigratedClass

    Abstract class used as the base class for namespace-like objects used to track the state of an item or test session.
    Instances can be used as if they were dictionaries of Value.

    get_declaration(var_name)
    Returns the declaration associated with var_name or None if the variable is one of the built-in variables. If
    var_name is not a variable KeyError is raised. To test for the existence of a variable just use the object
    as you would a dictionary:

    # state is a SessionState instance
    if 'RESPONSE' in state:
        print("RESPONSE declared!")
is_response(var_name)
    Return True if var_name is the name of a response variable.

is_outcome(var_name)
    Return True if var_name is the name of an outcome variable.

is_template(var_name)
    Return True if var_name is the name of a template variable.

__getitem__(var_name)
    Returns the Value instance corresponding to var_name or raises KeyError if there is no variable with that name.

__setitem__(var_name, value)
    Sets the value of var_name to the Value instance value.
    The baseType and cardinality of value must match those expected for the variable.

This method does not actually update the dictionary with the value instance but instead, it copies the value of value into the Value instance already stored in the session. The side-effect of this implementation is that a previous look-up will be updated by a subsequent assignment:

```python
# state is a SessionState instance
state['RESPONSE']=IdentifierValue('Hello')
r1=state['RESPONSE']
state['RESPONSE']=IdentifierValue('Bye')
r2=state['RESPONSE']
r1==r2          # WARNING: r1 has been updated so still evaluates to True!
```

GetDeclaration(*args, **kwargs)
    Deprecated equivalent to get_declaration()

IsOutcome(*args, **kwargs)
    Deprecated equivalent to is_outcome()

IsResponse(*args, **kwargs)
    Deprecated equivalent to is_response()

IsTemplate(*args, **kwargs)
    Deprecated equivalent to is_template()

class pyslet.qti.v2.variables.ItemSessionState(item)
    Bases: pyslet.qti.v2.variables.SessionState

    Represents the state of an item session. item is the item from which the session should be created.

    On construction, all declared variables (included built-in variables) are added to the session with NULL values,
    except the template variables which are set to their defaults.

    In addition to the variables defined by the specification we add meta variables corresponding to response and
    outcome defaults, these have the same name as the variable but with “.DEFAULT” appended. Similarly, we de-
    fine names for the correct values of response variables using “.CORRECT”. The values of these meta-variables
    are all initialised from the item definition on construction.

select_clone()
    Item templates describe a range of possible items referred to as clones.

    If the item used to create the session object is an item template then you must call select_clone before
    beginning the candidate’s session with begin_session().

    The main purpose of this method is to run the template processing rules. These rules update the values of
    the template variables and may also alter correct responses and default outcome (or response) values.
begin_session()
Called at the start of an item session. According to the specification:

“The session starts when the associated item first becomes eligible for delivery to the candidate”

The main purpose of this method is to set the outcome values to their defaults.

begin_attempt (html_parent=None)
Called at the start of an attempt.

This method sets the default RESPONSE values and completionStatus if this is the first attempt and increments numAttempts accordingly.

save_session (params, html_parent=None)
Called when we wish to save unsubmitted values.

submit_session (params, html_parent=None)
Called when we wish to submit values (i.e., end an attempt).

d end_attempt()
Called at the end of an attempt. Invokes response processing if present.

is_response (var_name)
Return True if var_name is the name of a response variable.

We add handling of the built-in response variables numAttempts and duration.

is_outcome (var_name)
Return True if var_name is the name of an outcome variable.

We add handling of the built-in outcome variable completionStatus.

class pyslet.Qtiv2.variables.TestSessionState (form)
Bases: pyslet.Qtiv2.variables.SessionState

Represents the state of a test session. The keys are the names of the variables including qualified names that can be used to look up the value of variables from the associated item session states. form is the test form from which the session should be created.

On construction, all declared variables (including built-in variables) are added to the session with NULL values.

test = None
the tests.AssessmentTest that this session is an

t = None
the time of the last event

salt = None
a random string of bytes used to add entropy to the session key

key = None
A key representing this session in its current state, this key is initialised to a random value and changes as each event is received. The key must be supplied when triggering subsequent events. The key is designed to be unguessable and unique so a caller presenting the correct key when triggering an event can be securely assumed to be the owner of the existing session.

prevKey = None
The key representing the previous state. This can be used to follow session state transitions back through a chain of states back to the beginning of the session (i.e., for auditing).

keyMap = None
A mapping of keys previously used by this session. A caller presenting an expired key when triggering an event generates a SessionKeyExpired exception. This condition might indicate that a session
response was not received (e.g., due to a connection failure) and that the session should be re-started with
the previous response.

**get_current_test_part()**

Returns the current test part or None if the test is finished.

**get_current_question()**

Returns the current question or None if the test is finished.

**begin_session(key, html_parent=None)**

Called at the start of a test session. Represents a ‘Start Test’ event.

The main purpose of this method is to set the outcome values to their defaults and to select the first
question.

**get_namespace(var_name)**

Returns a tuple of namespace/var_name from variable name

The resulting namespace will be a dictionary or a dictionary-like object from which the value of the re-
turned var_name object can be looked up.

**is_response(var_name)**

Return True if var_name is the name of a response variable. The test-level duration values are treated as
built-in responses and return True.

**__len__()**

Returns the total length of all namespaces combined.

**__getitem__(var_name)**

Returns the Value instance corresponding to var_name or raises KeyError if there is no variable with that
name.

**class pyslet.qti.v2.variables.Value**


Represents a single value in the processing model.

This class is the heart of the QTI processing model. This is an abstract base class of a class hierarchy that
represents the various types of value that may be encountered when processing.

**baseType = None**

One of the BaseType constants or None if the baseType is unknown.

An unknown baseType acts like a wild-card. It means that the baseType is not determined and could
potentially be any of the BaseType values. This distinction has implications for the way evaluation is
done. A value with a baseType of None will not raise TypeErrors during evaluation if the cardinalities
match the context. This allows expressions which contain types bound only at runtime to be evaluated for
validity checking.

**value = None**

The value of the variable. The following representations are used for values of single cardinality:

**NULL value** Represented by None

**boolean** One of the built-in Python values True and False

**directedPair** A tuple of strings (<source identifier>, <destination identifier>)

**duration** real number of seconds

**file** a file like object (supporting seek)

**float** real number

**identifier** A text string
**integer**  A plain python integer (QTI does not support long integer values)

**pair**  A *sorted* tuple of strings (<identifier A>, <identifier B>). We sort the identifiers in a pair by python’s native string sorting to ensure that pair values are comparable.

**point**  A tuple of integers (<x-coordinate>, <y-coordinate>)

**string**  A python string

**uri**  An instance of **URI**

For containers, we use the following structures:

- **ordered**  A list of one of the above value types.
- **multiple**  A dictionary with keys that are one of the above value types and values that indicate the frequency of that value in the container.
- **record**  A dictionary with keys that are the field identifiers and values that Value instances.

**set_value** (*value*)

Sets the value.

All single values can be set from a single text string corresponding to their XML schema defined lexical values (*without* character level escaping). If v is a single Value instance then the following always leaves v unchanged:

```python
v.set_value(unicode(v))  # str() in Python 3
```

Value instances can also be set from values of the appropriate type as described in **value**. For base types that are represented with tuples we also accept and convert lists.

Containers values cannot be set from strings.

**value_error** (*value*)

Raises a ValueError with a debug-friendly message string.

**cardinality** ()

Returns the cardinality of this value. One of the **Cardinality** constants.

By default we return None - indicating unknown cardinality. This can only be the case if the value is a NULL.

**is_null** ()

Returns True if this value is NULL, as defined by the QTI specification.

**classmethod new_value** (*cardinality*, **base_type=None*)

Creates a new value instance with **cardinality** and **base_type**.

**classmethod copy_value** (*value*)

Creates a new value instance copying **value**.

**Cardinality** (*args, **kwargs*)

Deprecated equivalent to **cardinality()**

**classmethod CopyValue** (*args, **kwargs*)

Deprecated equivalent to **copy_value()**

**IsNull** (*args, **kwargs*)

Deprecated equivalent to **is_null()**

**classmethod NewValue** (*args, **kwargs*)

Deprecated equivalent to **new_value()**
ValueError(*args, **kwargs)
    Deprecated equivalent to value_error()

class pyslet.qti.v2.variables.SingleValue
    Bases: pyslet.qti.v2.variables.Value

    Represents all values with single cardinality.

classmethod new_value(base_type, value=None)
    Creates a new instance of a single value with base_type and value

class pyslet.qti.v2.variables[BooleanValue](value=None)
    Bases: pyslet.qti.v2.variables.SingleValue

    Represents single values of type BaseType.boolean.

    set_value(value)
        If value is a string it will be decoded according to the rules for representing boolean values. Booleans
        and integers can be used directly in the normal python way but other values will raise ValueError. To take
        advantage of a non-zero test you must explicitly force it to be a boolean. For example:

        # x is a value of unknown type with non-zero test implemented
        v=True if x else False

        class pyslet.qti.v2.variables.DirectedPairValue(value=None)
            Bases: pyslet.qti.v2.variables.SingleValue

            Represents single values of type BaseType.directedPair.

            set_value(value, name_check=False)
                See Identifier.SetValue() for usage of name_check.

                Note that if value is a string then name_check is ignored and identifier validation is always performed.

class pyslet.qti.v2.variables.DurationValue(value=None)
    Bases: pyslet.qti.v2.variables.FloatValue

    Represents single value of type BaseType.duration.

class pyslet.qti.v2.variables.FileValue
    Bases: pyslet.qti.v2.variables.SingleValue

    Represents single value of type BaseType.file.

    contentType = None
        The content type of the file, a pyslet.http.params.MediaType instance.

    file_name = None
        The file name to use for the file.

    set_value(value, type='application/octet-stream', name='data.bin')
        Sets a file value from a file like object or a string.

        There are some important and subtle distinctions in this method.

        If value is a Unicode text string then it is parsed according to the MIME-like format defined in the QTI
        specification. The values of type and name are only used as defaults if those values cannot be read from
        the value’s headers.

        If value is a plain string then it is assumed to represent the file’s data directly, type and name are used to
        interpret the data. Other file type objects are set in the same way.
class pyslet.qtiv2.variables.FloatValue(value=None)
    Bases: pyslet.qtiv2.variables.SingleValue
    Represents single value of type BaseType.float.
    
    set_value(value)
    
    This method will **not** convert integers to float values, you must do this explicitly if you want automatic
    conversion, for example
    
    ```
    # x is a numeric value that may be float or integer
    v=FloatValue()
    v.set_value(float(x))
    ```

class pyslet.qtiv2.variables.IdentifierValue(value=None)
    Bases: pyslet.qtiv2.variables.SingleValue
    Represents single value of type BaseType.identifier.
    
    set_value(value, name_check=True)
    
    In general, to speed up computation we do not check the validity of identifiers unless parsing the value
    from a string representation (such as a value read from an XML input document).
    
    As values of baseType identifier are represented natively as strings we cannot tell if this method is being
called with an existing, name-checked value or a new value being parsed from an external source. To
speed up computation you can suppress the name check in the first case by setting `name_check` to False
(the default is True).

class pyslet.qtiv2.variables.IntegerValue(value=None)
    Bases: pyslet.qtiv2.variables.SingleValue
    Represents single value of type BaseType.integer.
    
    set_value(value)
    
    Note that integers and floats are distinct types in QTI: we do not accept floats where we would expect
integers or *vice versa*. However, integers are accepted from long or plain integer values provided they are
within the ranges specified in the QTI specification: -2147483648...2147483647.

class pyslet.qtiv2.variables.PairValue(value=None)
    Bases: pyslet.qtiv2.variables.DirectedPairValue
    Represents single values of type BaseType.pair.
    
    set_value(value, name_check=True)
    
    Overrides DirectedPair's implementation to force a predictable ordering on the identifiers.

class pyslet.qtiv2.variables.PointValue(value=None)
    Bases: pyslet.qtiv2.variables.SingleValue
    Represents single value of type BaseType.point.

class pyslet.qtiv2.variables.StringValue(value=None)
    Bases: pyslet.qtiv2.variables.SingleValue
    Represents single value of type BaseType.string.

class pyslet.qtiv2.variables.URIValue(value=None)
    Bases: pyslet.qtiv2.variables.SingleValue
    Represents single value of type BaseType.uri.
    
    set_value(value)
    
    Sets a uri value from a string or another URI instance.
class pyslet.qtiv2.variables.Container(base_type=None)
    Bases: pyslet.qtiv2.variables.Value
    An abstract class for all container types.
    By default containers are empty (and are treated as NULL values). You can force the type of an empty container
    by passing a baseType constant to the constructor. This will cause the container to generate TypeError if used in
    a context where the specified baseType is not allowed.

get_values()
    Returns an iterable of the container's values.

classmethod new_value(cardinality, base_type=None)
    Creates a new container with cardinality and base_type.

class pyslet.qtiv2.variables.OrderedContainer(base_type=None)
    Bases: pyslet.qtiv2.variables.Container
    Represents containers with ordered Cardinality.

    set_value(value, base_type=None)
        Sets the value of this container from a list, tuple or other iterable. The list must contain valid representations
        of base_type, items may be None indicating a NULL value in the list. In accordance with the specification's
        multiple operator NULL values are ignored.
        If the input list of values empty, or contains only NULL values then the resulting container is empty.
        If base_type is None the base type specified when the container was constructed is assumed.

    get_values()
        Returns an iterable of values in the ordered container.

class pyslet.qtiv2.variables.MultipleContainer(base_type=None)
    Bases: pyslet.qtiv2.variables.Container
    Represents containers with multiple Cardinality.

    set_value(value, base_type=None)
        Sets the value of this container from a list, tuple or other iterable. The list must contain valid representations
        of base_type, items may be None indicating a NULL value in the list. In accordance with the specification's
        multiple operator NULL values are ignored.
        If the input list of values is empty, or contains only NULL values then the resulting container is empty.
        If base_type is None the base type specified when the container was constructed is assumed.

    get_values()
        Returns an iterable of values in the ordered container.

class pyslet.qtiv2.variables.RecordContainer
    Bases: pyslet.qtiv2.variables.Container
    Represents containers with record Cardinality.

    set_value(value)
        Sets the value of this container from an existing dictionary in which the keys are the field identifiers and
        the values are Value instances. You cannot parse containers from strings.
        Records are always treated as having a wild-card base type.
        If the input value contains any keys which map to None or to a NULL value then these fields are omitted
        from the resulting value.
`

**__getitem__**(field_identifier)

Returns the `Value` instance corresponding to `field_identifier` or raises KeyError if there is no field with that name.

**__setitem__**(field_identifier, value)

Sets the value in the named field to `value`.

We add some special behaviour here. If `value` is None or is a NULL value then we remove the field with the give name. In other words:

```python
r=RecordContainer()
r['pi']=FloatValue(3.14)
r['pi']=FloatValue()    # a NULL value
r['pi']                # raises KeyError
```

### 3.3.6 Response Processing

#### Generalized Response Processing

class `pyslet.qtiv2.processing.ResponseProcessing`(parent)

**Bases:** `pyslet.qtiv2.core.QTIElement`

Response processing is the process by which the Delivery Engine assigns outcomes based on the candidate's responses:

```xml
<xsd:attributeGroup name="responseProcessing.AttrGroup">
    <xsd:attribute name="template" type="uri.Type" use="optional"/>
    <xsd:attribute name="templateLocation" type="uri.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="responseProcessing.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="responseRule.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

**run**(state)

Runs response processing using the values in `state`.

- `state` is an `ItemSessionState` instance.

**Run**(*args, **kwargs)

Deprecated equivalent to `run()`

class `pyslet.qtiv2.processing.ResponseRule`(parent, name=None)

**Bases:** `pyslet.qtiv2.core.QTIElement`

Abstract class to represent all response rules.

**run**(state)

Abstract method to run this rule using the values in `state`.

**Run**(*args, **kwargs)

Deprecated equivalent to `run()`
**class** pyslet.qtiv2.processing.ResponseCondition *(parent)*  
**Bases:** pyslet.qtiv2.processing.ResponseRule

If the expression given in a responseIf or responseElseIf evaluates to true then the sub-rules contained within it are followed and any following responseElseIf or responseElse parts are ignored for this response condition:

```xml
<xsd:group name="responseCondition.ContentGroup">
  <xsd:sequence>
    <xsd:element ref="responseIf" minOccurs="1" maxOccurs="1"/>
    <xsd:element ref="responseElseIf" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="responseElse" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.processing.ResponseIf *(parent)*  
**Bases:** pyslet.qtiv2.core.QTIElement

A responseIf part consists of an expression which must have an effective baseType of boolean and single cardinality. If the expression is true then the sub-rules are processed, otherwise they are skipped (including if the expression is NULL):

```xml
<xsd:group name="responseIf.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
    <xsd:group ref="responseRule.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

run *(state)*  
Run this test and, if True, any resulting rules.  
Returns True if the condition evaluated to True.

Run *(*args, **kwargs)*  
Deprecated equivalent to run()

class pyslet.qtiv2.processing.ResponseElseIf *(parent)*  
**Bases:** pyslet.qtiv2.processing.ResponseIf

Represents the responseElse element, see ResponseIf

```xml
<xsd:group name="responseElseIf.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
    <xsd:group ref="responseRule.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.processing.ResponseElse *(parent)*  
**Bases:** pyslet.qtiv2.core.QTIElement

Represents the responseElse element, see ResponseCondition
run (state)
Runs the sub-rules.

Run (*args, **kwargs)
Deprecated equivalent to run()

class pyslet.qtiv2.processing.SetOutcomeValue (parent)
Bases: pyslet.qtiv2.processing.ResponseRule

The setOutcomeValue rule sets the value of an outcome variable to the value obtained from the associated expression:

class pyslet.qtiv2.processing.StopProcessing
Bases: pyslet.qtiv2.core.QTIError

Raised when a rule which stops processing is encountered.

class pyslet.qtiv2.processing.ExitResponse (parent, name=None)
Bases: pyslet.qtiv2.processing.ResponseRule

The exit response rule terminates response processing immediately (for this invocation). It does this by raising StopProcessing:

3.3.7 Template Processing

class pyslet.qtiv2.processing.TemplateProcessing (parent)
Bases: pyslet.qtiv2.core.QTIElement

Template processing consists of one or more templateRules that are followed by the cloning engine or delivery system in order to assign values to the template variables:
run \((\text{state})\)

Runs template processing rules using the values in \text{state}.

- \text{state} is an \text{ItemSessionState} instance.

Run \((*\text{args}, **\text{kwargs})\)

Deprecated equivalent to \text{run()}

class pyslet.qtiv2.processing.TemplateRule\((\text{parent}, \text{name=None})\)

Abstract class to represent all template rules.

run \((\text{state})\)

Abstract method to run this rule using the values in \text{state}.

Run \((*\text{args}, **\text{kwargs})\)

Deprecated equivalent to \text{run()}

class pyslet.qtiv2.processing.TemplateCondition\((\text{parent})\)

Bases: \text{pyslet.qtiv2.processing.TemplateRule}

If the expression given in the templateIf or templateElseIf evaluates to true then the sub-rules contained within it are followed and any following templateElseIf or templateElse parts are ignored for this template condition:

```
<xsd:group name="templateCondition.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="templateIf.ElementGroup" minOccurs="1" maxOccurs="1"/>
    <xsd:group ref="templateElseIf.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:group ref="templateElse.ElementGroup" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.processing.TemplateIf\((\text{parent})\)

Bases: \text{pyslet.qtiv2.core.QTIElement}

A templateIf part consists of an expression which must have an effective baseType of boolean and single cardinality. If the expression is true then the sub-rules are processed, otherwise they are skipped (including if the expression is NULL):

```
<xsd:group name="templateIf.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
    <xsd:group ref="templateRule.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

run \((\text{state})\)

Run this test and, if True, any resulting rules.

Returns \text{True} if the condition evaluated to \text{True}. 

86 Chapter 3. IMS Global Learning Consortium Specifications
class pyslet.qtiv2.processing.TemplateElseIf (parent)
Bases: pyslet.qtiv2.processing.TemplateIf

Represents the templateElseIf element, see templateIf

```xml
<xsd:group name="templateElseIf.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
        <xsd:group ref="templateRule.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

```python
class pyslet.qtiv2.processing.TemplateElse (parent)
Bases: pyslet.qtiv2.core.QTIElement

Represents the templateElse element, see TemplateCondition

```xml
<xsd:group name="templateElse.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="templateRule.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

```python
run (state)
Runs the sub-rules.
```

```python
Run (*args, **kwargs)
Deprecated equivalent to run()
```

class pyslet.qtiv2.processing.SetTemplateValue (parent)
Bases: pyslet.qtiv2.processing.TemplateRule

The setTemplateValue rule sets the value of a template variable to the value obtained from the associated expression:

```xml
<xsd:attributeGroup name="setTemplateValue.AttrGroup">
    <xsd:attribute name="identifier" type="identifier.Type" use="required"/>
</xsd:attributeGroup>

<xsd:group name="setTemplateValue.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
</xsd:group>
```

```python
class pyslet.qtiv2.processing.SetCorrectResponse (parent)
Bases: pyslet.qtiv2.processing.TemplateRule

The setCorrectResponse rule sets the correct value of a response variable to the value obtained from the associated expression:

```xml
```
class pyslet.qtiv2.processing.SetDefaultValue (parent)
Bases: pyslet.qtiv2.processing.TemplateRule

The setDefaultValue rule sets the default value of a response or outcome variable to the value obtained from the associated expression:

```xml
<pyslet.qtiv2.processing.SetDefaultValue>
    <xsd:attributeGroup name="setDefaultValue.AttrGroup">
        <xsd:attribute name="identifier" type="identifier.Type" use="required"/>
    </xsd:attributeGroup>

    <xsd:group name="setDefaultValue.ContentGroup">
        <xsd:sequence>
            <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:group>
</pyslet.qtiv2.processing.SetDefaultValue>
```

class pyslet.qtiv2.processing.ExitTemplate (parent, name=None)
Bases: pyslet.qtiv2.processing.TemplateRule

The exit template rule terminates template processing immediately. It does this by raising StopProcessing:

```xml
<pyslet.qtiv2.processing.ExitTemplate />
```

3.3.8 Pre-conditions and Branching

class pyslet.qtiv2.processing.TestPartCondition (parent)
Bases: pyslet.qtiv2.core.QTIElement

evaluate (state)
    Evaluates the condition using the values in state.
        • state is a TestSessionState instance.

evaluate (*args, **kwargs)
    Deprecated equivalent to evaluate()

class pyslet.qtiv2.processing.PreCondition (parent)
Bases: pyslet.qtiv2.processing.TestPartCondition

A preCondition is a simple expression attached to an assessmentSection or assessmentItemRef that must evaluate to true if the item is to be presented:

```xml
<pyslet.qtiv2.processing.PreCondition>
    <xsd:group name="preCondition.ContentGroup">
        <xsd:sequence>

```

Chapter 3. IMS Global Learning Consortium Specifications
class pyslet.qtiv2.processing.BranchRule(parent)
    Bases: pyslet.qtiv2.processing.TestPartCondition

    A branch-rule is a simple expression attached to an assessmentItemRef, assessmentSection or testPart that is
    evaluated after the item, section, or part has been presented to the candidate:

    <xsd:attributeGroup name="branchRule.AttrGroup">
        <xsd:attribute name="target" type="identifier.Type" use="required"/>
    </xsd:attributeGroup>

    <xsd:group name="branchRule.ContentGroup">
        <xsd:sequence>
            <xsd:group ref="expression.ElementGroup"
                minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:group>

class pyslet.qtiv2.processing.TemplateDefault(parent)
    Bases: pyslet.qtiv2.core.QTIElement

    Overrides the default value of a template variable based on the test context in which the template is instantiated:

    <xsd:attributeGroup name="templateDefault.AttrGroup">
        <xsd:attribute name="templateIdentifier" type="identifier.Type" use="required"/>
    </xsd:attributeGroup>

    <xsd:group name="templateDefault.ContentGroup">
        <xsd:sequence>
            <xsd:group ref="expression.ElementGroup"
                minOccurs="1" maxOccurs="1"/>
        </xsd:sequence>
    </xsd:group>

run(item_state, test_state)
    Updates the value of a template variable in item_state based on the values in test_state.

Run(*args, **kwargs)
    Deprecated equivalent to run()

3.3.9 Expressions

class pyslet.qtiv2.expressions.Expression(parent, name=None)
    Bases: pyslet.qtiv2.core.QTIElement

    Abstract class for all expression elements.

    evaluate(state)
        Evaluates this expression in the context of the session state.

    integer_or_template_ref(state, value)
        Given a value of type integerOrTemplateRef this method returns the corresponding integer by looking up
the value, if necessary, in `state`. If value is a variable reference to a variable with NULL value then None is returned.

`float_or_template_ref(state, value)`

Given a value of type floatOrTemplateRef this method returns the corresponding float by looking up the value, if necessary, in `state`. If value is a variable reference to a variable with NULL value then None is returned.

`string_or_template_ref(state, value)`

Given a value of type stringOrTemplateRef this method returns the corresponding string by looking up the value, if necessary, in `state`. If value is a variable reference to a variable with NULL value then None is returned. Note that unlike the integer and float expansions this expansion will not raise an error if `value` is a syntactically valid reference to a non-existent template variable, as per this condition in the specification.

“if a string attribute appears to be a reference to a template variable but there is no variable with the given name it should be treated simply as string value”

`Evaluate(*args, **kwargs)`

Deprecated equivalent to `evaluate()`

class `pyslet.qtiv2.expressions.NOperator(parent)`
Bases: `pyslet.qtiv2.expressions.Expression`

An abstract class to help implement operators which take multiple sub-expressions.

evaluate_children(state)`

Evaluates all child expressions, returning an iterable of Value instances.

`EvaluateChildren(*args, **kwargs)`

Deprecated equivalent to `evaluate_children()`

class `pyslet.qtiv2.expressions.UnaryOperator(parent)`
Bases: `pyslet.qtiv2.expressions.Expression`

An abstract class to help implement unary operators.

**Built-in General Expressions**

class `pyslet.qtiv2.expressions.BaseValue(parent)`
Bases: `pyslet.qtiv2.expressions.Expression`

The simplest expression returns a single value from the set defined by the given baseType

```xml
<xs:attributeGroup name="baseValue.AttrGroup">
    <xs:attribute name="baseType" type="baseType.Type" use="required"/>
</xs:attributeGroup>

<xs:complexType name="baseValue.Type">
    <xs:simpleContent>
        <xs:extension base="xsd:string">
            <xs:attributeGroup ref="baseValue.AttrGroup"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```

class `pyslet.qtiv2.expressions.Variable(parent)`
Bases: `pyslet.qtiv2.expressions.Expression`
This expression looks up the value of an itemVariable that has been declared in a corresponding variableDeclaration or is one of the built-in variables:

```xml
<xs:attributeGroup name="variable.AttrGroup">
  <xs:attribute name="identifier" type="identifier.Type" use="required"/>
  <xs:attribute name="weightIdentifier" type="identifier.Type" use="optional"/>
</xs:attributeGroup>
```

```python
class pyslet.qtiv2.expressions.Default(parent):
    Bases: pyslet.qtiv2.expressions.Expression
    
    This expression looks up the declaration of an itemVariable and returns the associated defaultValue or NULL if no default value was declared:
```

```xml
<xs:attributeGroup name="default.AttrGroup">
  <xs:attribute name="identifier" type="identifier.Type" use="required"/>
</xs:attributeGroup>
```

```python
class pyslet.qtiv2.expressions.Correct(parent):
    Bases: pyslet.qtiv2.expressions.Expression
    
    This expression looks up the declaration of a response variable and returns the associated correctResponse or NULL if no correct value was declared:
```

```xml
<xs:attributeGroup name="correct.AttrGroup">
  <xs:attribute name="identifier" type="identifier.Type" use="required"/>
</xs:attributeGroup>
```

```python
class pyslet.qtiv2.expressions.MapResponse(parent):
    Bases: pyslet.qtiv2.expressions.Expression
    
    This expression looks up the value of a response variable and then transforms it using the associated mapping, which must have been declared. The result is a single float:
```

```xml
<xs:attributeGroup name="mapResponse.AttrGroup">
  <xs:attribute name="identifier" type="identifier.Type" use="required"/>
</xs:attributeGroup>
```

```python
class pyslet.qtiv2.expressions.MapResponsePoint(parent):
    Bases: pyslet.qtiv2.expressions.Expression
    
    This expression looks up the value of a response variable that must be of base-type point, and transforms it using the associated areaMapping:
```

```xml
<xs:attributeGroup name="mapResponsePoint.AttrGroup">
  <xs:attribute name="identifier" type="identifier.Type" use="required"/>
</xs:attributeGroup>
```

```python
class pyslet.qtiv2.expressions.Null(parent, name=None):
    Bases: pyslet.qtiv2.expressions.Expression
    
    null is a simple expression that returns the NULL value - the null value is treated as if it is of any desired baseType
```
**RandomInteger** class

```
class pyslet.qtiv2.expressions.RandomInteger(parent)
Bases: pyslet.qtiv2.expressions.Expression

Selects a random integer from the specified range [min,max] satisfying min + step * n for some integer n:
```

<table>
<thead>
<tr>
<th>Attribute Group Name</th>
<th>Attribute Name</th>
<th>Type</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>randomInteger.AttrGroup</td>
<td>min</td>
<td>integerOrTemplateRef.Type</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>integerOrTemplateRef.Type</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>step</td>
<td>integerOrTemplateRef.Type</td>
<td>optional</td>
</tr>
</tbody>
</table>

**RandomFloat** class

```
class pyslet.qtiv2.expressions.RandomFloat(parent)
Bases: pyslet.qtiv2.expressions.Expression

Selects a random float from the specified range [min,max]
```

<table>
<thead>
<tr>
<th>Attribute Group Name</th>
<th>Attribute Name</th>
<th>Type</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>randomFloat.AttrGroup</td>
<td>min</td>
<td>floatOrTemplateRef.Type</td>
<td>required</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>floatOrTemplateRef.Type</td>
<td>required</td>
</tr>
</tbody>
</table>

**Expressions Used only in Outcomes Processing**

**Operators**

**Multiple** class

```
class pyslet.qtiv2.expressions.Multiple(parent)
Bases: pyslet.qtiv2.expressions.NOperator

The multiple operator takes 0 or more sub-expressions all of which must have either single or multiple cardinality:
```

<table>
<thead>
<tr>
<th>Group Name</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple.ContentGroup</td>
<td></td>
</tr>
</tbody>
</table>

**Ordered** class

```
class pyslet.qtiv2.expressions.Ordered(parent)
Bases: pyslet.qtiv2.expressions.NOperator

The multiple operator takes 0 or more sub-expressions all of which must have either single or multiple cardinality:
```

<table>
<thead>
<tr>
<th>Group Name</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>ordered.ContentGroup</td>
<td></td>
</tr>
</tbody>
</table>
class pyslet.qtiv2.expressions.ContainerSize(parent)
Bases: pyslet.qtiv2.expressions.UnaryOperator

The containerSize operator takes a sub-expression with any base-type and either multiple or ordered cardinality:

```xml
<xsd:group name="containerSize.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.IsNull(parent)
Bases: pyslet.qtiv2.expressions.UnaryOperator

The isNull operator takes a sub-expression with any base-type and cardinality:

```xml
<xsd:group name="isNull.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.Index(parent)
Bases: pyslet.qtiv2.expressions.UnaryOperator

The index operator takes a sub-expression with an ordered container value and any base-type:

```xml
<xsd:attributeGroup name="index.AttrGroup">
  <xsd:attribute name="n" type="integer.Type" use="required"/>
</xsd:attributeGroup>

<xsd:group name="index.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.FieldValue(parent)
Bases: pyslet.qtiv2.expressions.UnaryOperator

The field-value operator takes a sub-expression with a record container value. The result is the value of the field with the specified fieldIdentifier:

```xml
<xsd:attributeGroup name="fieldValue.AttrGroup">
  <xsd:attribute name="fieldIdentifier" type="identifier.Type" use="required"/>
</xsd:attributeGroup>

<xsd:group name="fieldValue.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```
class pyslet.qtiv2.expressions.Random

Bases: pyslet.qtiv2.expressions.UnaryOperator

The random operator takes a sub-expression with a multiple or ordered container value and any base-type. The result is a single value randomly selected from the container:

```xml
<xsd:group name="random.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
      minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.Member

Bases: pyslet.qtiv2.expressions.NOperator

The member operator takes two sub-expressions which must both have the same base-type. The first sub-expression must have single cardinality and the second must be a multiple or ordered container. The result is a single boolean with a value of true if the value given by the first sub-expression is in the container defined by the second sub-expression:

```xml
<xsd:group name="member.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
      minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.Delete

Bases: pyslet.qtiv2.expressions.NOperator

The delete operator takes two sub-expressions which must both have the same base-type. The first sub-expression must have single cardinality and the second must be a multiple or ordered container. The result is a new container derived from the second sub-expression with all instances of the first sub-expression removed:

```xml
<xsd:group name="delete.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
      minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.Contains

Bases: pyslet.qtiv2.expressions.NOperator

The contains operator takes two sub-expressions which must both have the same base-type and cardinality – either multiple or ordered. The result is a single boolean with a value of true if the container given by the first sub-expression contains the value given by the second sub-expression and false if it doesn’t:

```xml
<xsd:group name="contains.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
      minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.SubString

Bases: pyslet.qtiv2.expressions.NOperator
The substring operator takes two sub-expressions which must both have an effective base-type of string and single cardinality. The result is a single boolean with a value of true if the first expression is a substring of the second expression and false if it isn’t:

```xml
<xsd:attributeGroup name="substring.AttrGroup">
  <xsd:attribute name="caseSensitive" type="boolean.Type"
    use="required"/>
</xsd:attributeGroup>

<xsd:group name="substring.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
      minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.Not (parent)
Bases: pyslet.qtiv2.expressions.UnaryOperator

The not operator takes a single sub-expression with a base-type of boolean and single cardinality. The result is a single boolean with a value obtained by the logical negation of the sub-expression’s value:

```xml
<xsd:group name="not.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
      minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.And (parent)
Bases: pyslet.qtiv2.expressions.NOperator

The and operator takes one or more sub-expressions each with a base-type of boolean and single cardinality. The result is a single boolean which is true if all sub-expressions are true and false if any of them are false:

```xml
<xsd:group name="and.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
      minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.Or (parent)
Bases: pyslet.qtiv2.expressions.NOperator

The or operator takes one or more sub-expressions each with a base-type of boolean and single cardinality. The result is a single boolean which is true if any of the sub-expressions are true and false if all of them are false:

```xml
<xsd:group name="or.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
      minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.AnyN (parent)
Bases: pyslet.qtiv2.expressions.NOperator

The anyN operator takes one or more sub-expressions each with a base-type of boolean and single cardinality.
The result is a single boolean which is true if at least min of the sub-expressions are true and at most max of the sub-expressions are true:

```
<xsd:attributeGroup name="anyN.AttrGroup">
  <xsd:attribute name="min" type="integerOrTemplateRef.Type" use="required"/>
  <xsd:attribute name="max" type="integerOrTemplateRef.Type" use="required"/>
</xsd:attributeGroup>

<xsd:group name="anyN.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:group>
```

```python
class pyslet.qtiv2.expressions.Match(parent)
Bases: pyslet.qtiv2.expressions.NOperator
```

The match operator takes two sub-expressions which must both have the same base-type and cardinality. The result is a single boolean with a value of true if the two expressions represent the same value and false if they do not:

```
<xsd:group name="match.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

```python
class pyslet.qtiv2.expressions.StringMatch(parent)
Bases: pyslet.qtiv2.expressions.NOperator
```

The stringMatch operator takes two sub-expressions which must have single and a base-type of string. The result is a single boolean with a value of true if the two strings match:

```
<xsd:attributeGroup name="stringMatch.AttrGroup">
  <xsd:attribute name="caseSensitive" type="boolean.Type" use="required"/>
  <xsd:attribute name="substring" type="boolean.Type" use="optional"/>
</xsd:attributeGroup>

<xsd:group name="stringMatch.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

```python
class pyslet.qtiv2.expressions.PatternMatch(parent)
Bases: pyslet.qtiv2.expressions.UnaryOperator
```

The patternMatch operator takes a sub-expression which must have single cardinality and a base-type of string. The result is a single boolean with a value of true if the sub-expression matches the regular expression given by pattern and false if it doesn’t:
class pyslet.qtiv2.expressions.Equal (parent)
Bases: pyslet.qtiv2.expressions.NOperator

The equal operator takes two sub-expressions which must both have single cardinality and have a numerical base-type. The result is a single boolean with a value of true if the two expressions are numerically equal and false if they are not:

class pyslet.qtiv2.expressions.ToleranceMode
Bases: pyslet.xml.xsdatatypesEnumeration

When comparing two floating point numbers for equality it is often desirable to have a tolerance to ensure that spurious errors in scoring are not introduced by rounding errors. The tolerance mode determines whether the comparison is done exactly, using an absolute range or a relative range:
ToleranceMode.exact

The default value is exact:

ToleranceMode.DEFAULT == ToleranceMode.exact

For more methods see Enumeration

class pyslet.qtv2.expressions.EqualRounded (parent)
Bases: pyslet.qtv2.expressions.NOperator

The equalRounded operator takes two sub-expressions which must both have single cardinality and have a numerical base-type. The result is a single boolean with a value of true if the two expressions are numerically equal after rounding and false if they are not:

```xml
<xsd:attributeGroup name="equalRounded.AttrGroup">
    <xsd:attribute name="roundingMode" type="roundingMode.Type" use="required"/>
    <xsd:attribute name="figures" type="integerOrTemplateRef.Type" use="required"/>
</xsd:attributeGroup>

<xsd:group name="equalRounded.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="expression.ElementGroup" minOccurs="2" maxOccurs="2"/>
    </xsd:sequence>
</xsd:group>
```

class pyslet.qtv2.expressions.RoundingMode
Bases: pyslet.xml.xsd.datatypes Enumeration

Numbers are rounded to a given number of significantFigures or decimalPlaces:

```xml
<xsd:simpleType name="roundingMode.Type">
    <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="decimalPlaces"/>
        <xsd:enumeration value="significantFigures"/>
    </xsd:restriction>
</xsd:simpleType>
```

Defines constants for the above modes. Usage example:

RoundingMode.decimalPlaces

The default value is significantFigures:

RoundingMode.DEFAULT == RoundingMode.significantFigures

For more methods see Enumeration

class pyslet.qtv2.expressions.Inside (parent)
Bases: pyslet.qtv2.expressions.UnaryOperator, pyslet.qtv2.core.ShapeElementMixin

The inside operator takes a single sub-expression which must have a baseType of point. The result is a single boolean with a value of true if the given point is inside the area defined by shape and coords. If the sub-expression is a container the result is true if any of the points are inside the area:
class pyslet.qtiv2.expressions.LT(parent)
Bases: pyslet.qtiv2.expressions.NOperator

The lt operator takes two sub-expressions which must both have single cardinality and have a numerical base-type. The result is a single boolean with a value of true if the first expression is numerically less than the second and false if it is greater than or equal to the second:

```
<xsd:group name="lt.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="expression.ElementGroup"
            minOccurs="2" maxOccurs="2"/>
    </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.GT(parent)
Bases: pyslet.qtiv2.expressions.NOperator

The gt operator takes two sub-expressions which must both have single cardinality and have a numerical base-type. The result is a single boolean with a value of true if the first expression is numerically greater than the second and false if it is less than or equal to the second:

```
<xsd:group name="gt.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="expression.ElementGroup"
            minOccurs="2" maxOccurs="2"/>
    </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.LTE(parent)
Bases: pyslet.qtiv2.expressions.NOperator

The lte operator takes two sub-expressions which must both have single cardinality and have a numerical base-type. The result is a single boolean with a value of true if the first expression is numerically less than or equal to the second and false if it is greater than the second:

```
<xsd:group name="lte.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="expression.ElementGroup"
            minOccurs="2" maxOccurs="2"/>
    </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.GTE(parent)
Bases: pyslet.qtiv2.expressions.NOperator
The gte operator takes two sub-expressions which must both have single cardinality and have a numerical base-type. The result is a single boolean with a value of true if the first expression is numerically less than or equal to the second and false if it is greater than the second:

```xml
<class pyslet.qtiv2.expressions.DurationLT (parent)
Bases: pyslet.qtiv2.expressions.NOperator

The durationLT operator takes two sub-expressions which must both have single cardinality and base-type duration. The result is a single boolean with a value of true if the first duration is shorter than the second and false if it is longer than (or equal) to the second:

```xml
<class pyslet.qtiv2.expressions.DurationGTE (parent)
Bases: pyslet.qtiv2.expressions.NOperator

The durationGTE operator takes two sub-expressions which must both have single cardinality and base-type duration. The result is a single boolean with a value of true if the first duration is longer (or equal, within the limits imposed by truncation) than the second and false if it is shorter than the second:

```xml
<class pyslet.qtiv2.expressions.Sum (parent)
Bases: pyslet.qtiv2.expressions.NOperator

The sum operator takes 1 or more sub-expressions which all have single cardinality and have numerical base-types. The result is a single float or, if all sub-expressions are of integer type, a single integer that corresponds to the sum of the numerical values of the sub-expressions:

```xml
<class pyslet.qtiv2.expressions.Product (parent)
Bases: pyslet.qtiv2.expressions.NOperator

The product operator takes 1 or more sub-expressions which all have single cardinality and have numerical base-types. The result is a single float or, if all sub-expressions are of integer type, a single integer that corresponds to the product of the numerical values of the sub-expressions:
**class** `pyslet.qtiv2.expressions.Subtract(parent)`  
*Bases: pyslet.qtiv2.expressions.NOperator*

The subtract operator takes 2 sub-expressions which all have single cardinality and numerical base-types. The result is a single float or, if both sub-expressions are of integer type, a single integer that corresponds to the first value minus the second:

```xml
<xsd:group name="subtract.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
              minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

**class** `pyslet.qtiv2.expressions.Divide(parent)`  
*Bases: pyslet.qtiv2.expressions.NOperator*

The divide operator takes 2 sub-expressions which both have single cardinality and numerical base-types. The result is a single float that corresponds to the first expression divided by the second expression:

```xml
<xsd:group name="divide.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
              minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

**class** `pyslet.qtiv2.expressions.Power(parent)`  
*Bases: pyslet.qtiv2.expressions.NOperator*

The power operator takes 2 sub-expression which both have single cardinality and numerical base-types. The result is a single float that corresponds to the first expression raised to the power of the second:

```xml
<xsd:group name="power.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
              minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

**class** `pyslet.qtiv2.expressions.IntegerDivide(parent)`  
*Bases: pyslet.qtiv2.expressions.NOperator*

The integer divide operator takes 2 sub-expressions which both have single cardinality and base-type integer. The result is the single integer that corresponds to the first expression \((x)\) divided by the second expression \((y)\) rounded down to the greatest integer \((i)\) such that \(i\leq(x/y)\):

```xml
<xsd:group name="integerDivide.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup"
              minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```
class pyslet.qtiv2.expressions.IntegerModulus(parent)

Bases: pyslet.qtiv2.expressions.NOperator

The integer modulus operator takes 2 sub-expressions which both have single cardinality and base-type integer. The result is the single integer that corresponds to the remainder when the first expression (x) is divided by the second expression (y). If z is the result of the corresponding integerDivide operator then the result is x-z*y:

```xml
<xsd:group name="integerModulus.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="2" maxOccurs="2"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.Truncate(parent)

Bases: pyslet.qtiv2.expressions.UnaryOperator

The truncate operator takes a single sub-expression which must have single cardinality and base-type float. The result is a value of base-type integer formed by truncating the value of the sub-expression towards zero:

```xml
<xsd:group name="truncate.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.Round(parent)

Bases: pyslet.qtiv2.expressions.UnaryOperator

The round operator takes a single sub-expression which must have single cardinality and base-type float. The result is a value of base-type integer formed by rounding the value of the sub-expression. The result is the integer n for all input values in the range [n-0.5,n+0.5):

```xml
<xsd:group name="round.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions.IntegerToFloat(parent)

Bases: pyslet.qtiv2.expressions.UnaryOperator

The integer to float conversion operator takes a single sub-expression which must have single cardinality and base-type integer. The result is a value of base type float with the same numeric value:

```xml
<xsd:group name="integerToFloat.ContentGroup">
  <xsd:sequence>
    <xsd:group ref="expression.ElementGroup" minOccurs="1" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```

class pyslet.qtiv2.expressions CustomOperator(parent)

Bases: pyslet.qtiv2.expressions.NOperator
The custom operator provides an extension mechanism for defining operations not currently supported by this specification:

```
<xsd:attributeGroup name="customOperator.AttrGroup">
    <xsd:attribute name="class" type="identifier.Type" use="optional"/>
    <xsd:attribute name="definition" type="uri.Type" use="optional"/>
    <xsd:anyAttribute namespace="##other"/>
</xsd:attributeGroup>

<xsd:group name="customOperator.ContentGroup">
    <xsd:sequence>
        <xsd:group ref="expression.ElementGroup" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:group>
```

### 3.3.10 Core Types and Utilities

This module contains a number core classes used to support the standard.

#### Constants

```python
pyslet.qtiv2.core.IMSQTI_NAMESPACE = 'http://www.imsglobal.org/xsd/imsqti_v2p1'
The namespace used to recognise elements in XML documents.
```

```python
pyslet.qtiv2.core.IMSQTI_SCHEMALOCATION = 'http://www.imsglobal.org/xsd/imsqti_v2p1.xsd'
The location of the QTI 2.1 schema file on the IMS website.
```

```python
pyslet.qtiv2.core.IMSQTI_ITEM_RESOURCETYPE = 'imsqti_item_xmlv2p1'
The resource type to use for the QTI 2.1 items when added to content packages.
```

#### XML Basics

```python
class pyslet.qtiv2.core.QTIElement (parent, name=None)
    Bases: pyslet.xml.namespace.NSElement
    Basic element to represent all QTI elements

    add_to_cpresource (cp, resource, been_there)
        We need to add any files with URL's in the local file system to the content package.
        been_there is a dictionary we use for mapping URLs to File objects so that we don't keep adding the same
        linked resource multiple times.
        This implementation is a little more horrid, we avoid circular module references by playing dumb about
        our children. HTML doesn't actually know anything about QTI even though QTI wants to define children
        for some XHTML elements so we pass the call only to "CP-Aware" elements.
```

#### Exceptions

```python
class pyslet.qtiv2.core.QTIError
    Bases: exceptions.Exception
    Abstract class used for all QTI v2 exceptions.
```
class pyslet.qtiv2.core.DeclarationError
    Bases: pyslet.qtiv2.core.QTIError
    Error raised when a variable declaration is invalid.

class pyslet.qtiv2.core.ProcessingError
    Bases: pyslet.qtiv2.core.QTIError
    Error raised when an invalid processing element is encountered.

class pyslet.qtiv2.core.SelectionError
    Bases: pyslet.qtiv2.core.QTIError
    Error raised when there is a problem with creating test forms.

Basic Data Types

Basic data types in QTI v2 are a mixture of custom types and basic types defined externally, for example, by XMLSchema.

The external types used are:

**boolean** Represented by python’s boolean values True and False. See `boolean_from_str()` and `boolean_to_str()`

**coords** Defined as part of support for HTML. See `Coords`

**date** Although QTI draws on the definitions in XML schema it restricts values to those from the nontimezoned timeline. This restriction is effectively implemented in the basic `Date` class.

**datetime**: See `DecodeDateTime()` and `EncodeDateTime()`

**duration**: Earlier versions of QTI drew on the ISO8601 representation of duration but QTI v2 simplifies this with a basic representation in seconds bound to XML Schema’s double type which we, in turn, represent with python’s `float`. See `double_from_str()` and `double_to_str()`

**float**: implemented by python’s `float`. Note that this is defined as having “machine-level double precision” and the python specification goes on to warn that “You are at the mercy of the underlying machine architecture”. See `double_from_str()` and `double_to_str()`

**identifier**: represented by python’s `(unicode)` string. The type is effectively just the NCName from the XML namespace specification. See `pyslet.xml.namespace.IsValidNCName()`.

    pyslet.qtiv2.core.ValidateIdentifier(*args, **kwargs)
    Deprecated equivalent to validate_identifier()

**integer**: XML schema’s `integer`, implemented by python’s `integer`. See `DecodeInteger()` and `integer_to_str()`

**language**: Currently implemented as a simple python string.

**length**: Defined as part of support for HTML. See `LengthType`

**mimeType**: Currently implemented as a simple python string

**string**: XML schema string becomes python’s `unicode` string

**string256**: Length restriction not yet implemented, see string above.

**styleclass**: Inherited from HTML, implemented with a simple `(unicode)` string.

**uri**: In some instances this is implemented as a simple `(unicode)` string, for example, in cases where a URI is being used as global identifier. In contexts where the URI will need to be interpreted it is implemented with instances of `pyslet.rfc2396.URI`.

104 Chapter 3. IMS Global Learning Consortium Specifications
QTI-specific types:

```python
class pyslet.qtiv2.core.Orientation
    Bases: pyslet.xml.xsdatatypes.Enumeration
```

Orientation attribute values provide a hint to rendering systems that an element has an inherent vertical or horizontal interpretation:

```xml
<xsd:simpleType name="orientation.Type">
    <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="horizontal"/>
        <xsd:enumeration value="vertical"/>
    </xsd:restriction>
</xsd:simpleType>
```

Defines constants for the above orientations. Usage example:

```python
Orientation.horizontal
```

Note that:

```python
Orientation.DEFAULT == None
```

For more methods see `Enumeration`

```python
class pyslet.qtiv2.core.Shape
    Bases: pyslet.xml.xsdatatypes.Enumeration
```

A value of a shape is always accompanied by coordinates and an associated image which provides a context for interpreting them:

```xml
<xsd:simpleType name="shape.Type">
    <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="circle"/>
        <xsd:enumeration value="default"/>
        <xsd:enumeration value="ellipse"/>
        <xsd:enumeration value="poly"/>
        <xsd:enumeration value="rect"/>
    </xsd:restriction>
</xsd:simpleType>
```

Defines constants for the above types of Shape. Usage example:

```python
Shape.circle
```

Note that:

```python
Shape.DEFAULT == Shape.default
```

For more methods see `Enumeration`

```python
class pyslet.qtiv2.core.ShowHide
    Bases: pyslet.xml.xsdatatypes.Enumeration
```

Used to control content visibility with variables:

```xml
<xsd:simpleType name="showHide.Type">
    <xsd:restriction base="xsd:NMTOKEN">
        <xsd:enumeration value="hide"/>
        <xsd:enumeration value="show"/>
    </xsd:restriction>
</xsd:simpleType>
```
Note that ShowHide.DEFAULT == ShowHide.show

class pyslet.qtiv2.core.View
Bases: pyslet.xml.xsdatatypes EnumerationNoCase

Used to represent roles when restricting view:

```xml
<xsd:simpleType name="view.Type">
  <xsd:restriction base="xsd:NMTOKEN">
    <xsd:enumeration value="author"/>
    <xsd:enumeration value="candidate"/>
    <xsd:enumeration value="proctor"/>
    <xsd:enumeration value="scorer"/>
    <xsd:enumeration value="testConstructor"/>
    <xsd:enumeration value="tutor"/>
  </xsd:restriction>
</xsd:simpleType>
```

Defines constants for the above views. Usage example:

```python
View.candidate
```

There is no default view. Views are represented in XML as space-separated lists of values. Typical usage:

```python
view=View.DecodeValueDict("tutor scorer")
# returns...
{ View.tutor:'tutor', View.scorer:'scorer' }
View.EncodeValueDict(view)
# returns...
"scorer tutor"
```

For more methods see `Enumeration`

The QTI specification lists valueType as a basic data type. In pyslet this is implemented as a core part of the processing model. See `pyslet.qtiv2.variables.Value` for details.

### 3.3.11 Meta-data and Usage Data

class pyslet.qtiv2.metadata.QTIMetadata (parent)
Bases: pyslet.qtiv2.core.QTIElement

A new category of meta-data for the recording of QTI specific information. It is designed to be treated as an additional top-level category to augment the LOM profile:

```xml
<xsd:group name="qti1Metadata.ContentGroup">
  <xsd:sequence>
    <xsd:element ref="itemTemplate" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="timeDependent" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="composite" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="interactionType" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="feedbackType" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="solutionAvailable" minOccurs="0" maxOccurs="1"/>
    <xsd:element ref="toolName" minOccurs="0" maxOccurs="1"/>
  </xsd:sequence>
</xsd:group>
```
The starting point for parsing and managing QTI content:

```python
import pyslet.qtiv2.xml as qti
```

**Warning:** The structure of this module has changed in version Pyslet version 0.7. You should now include `pyslet.qtiv2.xml` to get access to QTIDocument as there are now no names exposed directly through the older `pyslet.imsqtiv2p1`.

```python
class pyslet.qtiv2.xml.QTIDocument(/**args)
    Bases: pyslet.xml.namespace.NSDocument

    Used to represent all documents representing information from the QTI v2 specification.

    Simple recipe to get started:

    ```python
    import pyslet.qtiv2.xml as qti
    doc = qti.QTIDocument()
    with open('myqti.xml', 'rb') as f:
        doc.read(src=f)
    # do stuff with the QTI document here
    ```
```

The root (doc.root) element of a QTI document may one of a number of elements, if you are interested in items look for an instance of `qti.items.AssessmentItem`, etc.

```python
add_to_content_package(cp, md, dname=None)
```

Copies this QTI document into a content package and returns the resource ID used.

An optional directory name can be specified in which to put the resource files.

```python
AddToContentPackage(*args, **kwargs)
```

Deprecated equivalent to `add_to_content_package()`

---

### 3.4 IMS Basic Learning Tools Interoperability (version 1.0)

The IMS Basic Learning Tools Interoperability (BLTI) specification was released in 2010. The purpose of the specification is to provide a link between tool consumers (such as Learning Management Systems and portals) and Tools (such as specialist assessment management systems). Official information about the specification is available from the IMS GLC under the general name LTI:

This module implements the Basic LTI specification documented in the Best Practice Guide

This module requires the `oauthlib` module to be installed. The `oauthlib` module is available from PyPi. This module is written from the point of view of the Tool Provider. There are a number of pre-defined classes to help you implement LTI in your own Python applications.

---

3.4. IMS Basic Learning Tools Interoperability (version 1.0) 107
3.4.1 Classes

The ToolProviderApp class is a mini-web framework in itself which makes writing LTI tools much easier. The framework does not include a page templating language but it should be easy to integrate with your templating system of choice.

Instances of ToolProviderApp are callable objects that support the WSGI protocol for ease of integration into a wide variety of deployment scenarios.

The base class implementation takes care of many aspects of your LTI tool:

1. Application settings are read from a JSON-encoded settings file.
2. Data storage is configured using one of the concrete implementations of Pyslet’s own data access layer API. No SQL necessary in your code, minimising the risk of SQL injection vulnerabilities!
3. Session handling: the base class handles the setting of a session cookie and an initial set of redirects to ensure that cookies are being supported properly by the browser. If session handling is broken a fail page method is called. The session logic contains special measures to prevent common session-related attacks (such as session fixation, hijacking and cross-site request forgery) and the redirection sequence is designed to overcome limitations imposed by browser restrictions on third party cookies or P3P-related policy issues by providing a user-actionable flow, opening your tool in a new window if necessary. End user messages are customisable.
4. Launch authorisation is handled automatically, launch requests are checked using OAuth for validity and rejected automatically if invalid. Successful requests are automatically redirected to a resource-specific page.
5. Each resource is given its own path in your application of the form /<script.wsgi>/resource/<id>/ allowing you to spread your tool application across multiple pages if necessary. A special method, ToolProviderApp.load_visit(), is provided to extract the resource ID from the URL path and load the corresponding entity from the data store. This method also loads the related entities for the the context, user and visit entities from the session according to the parameters passed in the original launch.
6. An overridable tool permission model is provided with a default implementation that provides read/write/configure permissions to Instructors (and sub-roles) and read permissions to Learners (and sub-roles). This enables your tool to simply test a permission bit at runtime to determine whether or not to display certain page elements.
7. Tools can be launched multiple times in the same browser session. Authorisations remain active allowing the user to interact with your tool in separate tabs or even in multiple iframes on the same page. Authorisations are automatically expired if a conflicting launch request is received. In other words, if a browser session receives a new launch from the same consumer but for a different user then all the previous user’s activity is automatically logged out.
8. Consumer secrets can be encrypted when persisted in the data store using an application key. By default the application key is configured in the settings file. (The PyCrypto module is required for encryption.)

The ToolConsumer and ToolProvider classes are largely for internal use. You may want to use them if you are integrating the basic LTI functionality into a different web framework, they contain utility methods for reading information from the data store. You would use the ToolProvider.launch() method in your application when the user POSTs to your launch endpoint to check that the LTI launch has been authorised.

3.4.2 The Data Model

Implementing LTI requires some data storage to persist information between HTTP requests. This module is written using Pyslet’s own data access layer, based on the concepts of OData. For more information see The Open Data Protocol (OData).

A sample metadata file describing the required elements of the model is available as part of Pyslet itself. The entity sets (cf SQL Tables) it describes are as follows:
**AppKeys**  This entity set is used to store information about encryption keys used to encrypt the consumer secrets in the data store. For more information see `pyslet.wsgi.WSGIDataApp`

**Silos**  This entity set is the root of the information space for each tool. LTI tools tend to be multi-tenant, that is, the same tool application can be used by multiple consumers with complete isolation between each consumer’s data. The Silo provides this level of protection. Normally, each Silo will link to a single consumer but there may be cases where two or more consumers should share some data, in these cases a single Silo may link to multiple consumers.

**Consumers**  This entity set contains information about the tool consumers. Each consumer is identified by a consumer key and access is protected using a consumer secret (which can be stored in an encrypted form in the data store).

**Nonces**  LTI tools are launched from the consumer using OAuth. The protocol requires the use of a nonce (number used once only) to prevent the launch request being ‘replayed’ by an unauthorised person. This entity set is used to record which nonces have been used and when.

**Resources**  The primary launch concept in LTI is the resource. Every launch must have a resource_link_id which identifies the specific ‘place’ or ‘page’ in which the tool has been placed.

**Contexts**  LTI defines a context as an optional course or group-like organisation that provides context for a launch request. The context provides another potential scope for sharing data across launches.

**Users**  An LTI launch is typically identified with a specific user of the Tool Consumer (though this isn’t required). Information about the users is recorded in the data store so that they can be associated with any data generated by the tool using simple extensions to the data model.

**Visits**  Each time someone launches your tool a visit entity is created with information about the resource, the context and the user.

**Sessions**  Used to store information about the browser session, see `pyslet.wsgi.SessionApp` for details. The basic session entity is extended to link to the visits that are active (i.e., currently authorised) for this session. These entities are related using navigation properties enabling you to determine, for example, which Consumer a Resource belongs to, which Visits are active in a Session, and so on.

You can extend the core model by adding additional data properties (which should be nullable) or by adding optional navigation properties. For example, you might create an entity set to store information created by users of the tool and add a navigation property from the User entity to your new entity to indicate ownership. The sample Noticeboard application uses this technique and can be used as a guide.

---

**Hello LTI**

Writing your first LTI tool is easy:

```python
from optparse import OptionParser
import pyslet.imsbltiv1p0 as lti

if __name__ == '__main__':
    parser = OptionParser()
    lti.ToolProviderApp.add_options(parser)
    (options, args) = parser.parse_args()
    lti.ToolProviderApp.setup(options, args)
    app = lti.ToolProviderApp()
    app.run_server()
```

Save this script as `mytool.py` and run it from the command line like this:

```
$ python mytool.py --help
```
Built-in to the WSGI base classes is support for running your tool from the command line during development. The script above just uses Python’s builtin options parsing feature to set up the tool class before creating an instance (the WSGI callable object) and running a basic WSGI server using Python’s builtin wsgiref module.

Try running your application with the -m and –create_silo options to use an in-memory SQLite data store and a default consumer.

```
$ python mytool.py -m
```

The script may print a warning message to the console warning you that the in-memory database does not support multiple connections, it then just sits waiting for connections on the default port, 8080. The default consumer has key ‘12345’ and secret ‘secret’ (these can be changed using a configuration file!). The launch URL for your running tool is:

```
http://localhost:8080/launch
```

If you try it in the IMS test consumer at: http://www.imsglobal.org/developers/LTI/test/v1p1/lms.php you should get something that looks a bit like this:

For a more complete example see the NoticeBoard Sample LTI Tool.

3.4.3 Reference

```python
class pyslet.imsbltiv1p0.ToolProviderApp(**kwargs)
    Bases: pyslet.wsgi.SessionApp

    Represents WSGI applications that provide LTI Tools

    The key ‘ToolProviderApp’ is reserved for settings defined by this class in the settings file. The defined settings are:

    silo (‘testing’)  The name of a default silo to create when the –create_silo option is used.
    key (‘12345’)    The default consumer key created when –create_silo is used.
    secret (‘secret’) The consumer secret of the default consumer created when –create_silo is used.

    ContextClass
    We have our own context class
      alias of ToolProviderContext

    classmethod add_options(parser)
        Adds the following options:

        --create_silo    create default silo and consumer

    init_dispatcher()
        Provides ToolProviderApp specific bindings.
```

110 Chapter 3. IMS Global Learning Consortium Specifications
This method adds bindings for /launch as the launch URL for the tool and all paths within /resource as the resource pages themselves.

**set_launch_group** *(context)*  
Sets the group in the context from the launch parameters

**set_launch_resource** *(context)*  
Sets the resource in the context from the launch parameters

**set_launch_user** *(context)*  
Sets the user in the context from the launch parameters

**set_launch_permissions** *(context)*  
Sets the permissions in the context from the launch parameters

**READ_PERMISSION = 1**  
Permission bit mask representing ‘read’ permission

**WRITE_PERMISSION = 2**  
Permission bit mask representing ‘write’ permission

**CONFIGURE_PERMISSION = 4**  
Permission bit mask representing ‘configure’ permission

**classmethod get_permissions** *(role)*  
Returns the permissions that apply to a single role

- **role**  
  A single URN instance

Specific LTI tools can override this method to provide more complex permission models. Each permission type is represented by an integer bit mask, permissions can be combined with binary or ‘|’ to make an overall permissions integer. The default implementation uses the `READ_PERMISSION`, `WRITE_PERMISSION` and `CONFIGURE_PERMISSION` bit masks but you are free to use any values you wish.

In this implementation, Instructors (and all sub-roles) are granted read, write and configure whereas Learners (and all subroles) are granted read only. Any other role returns 0 (no permissions).

An LTI consumer can specify multiple roles on launch, this method is called for each role and the resulting permissions integers are combined to provide an overall permissions integer.

**get_user_display_name** *(context, user=None)*  
Given a user entity, returns a display name  
If user is None then the user from the context is used instead.

**get_resource_title** *(context)*  
Given a resource entity, returns a display title

**new_visit** *(context)*  
Called during launch to create a new visit entity  
A new visit entity is created and bound to the resource entity referred to in the launch. The visit entity stores the permissions and a link to the (optional) user entity.

If a visit to the same resource is already associated with the session it is replaced. This ensures that information about the resource, the user, roles and permissions always corresponds to the most recent launch.

Any visits from the same consumer but with a different user are also removed. This handles the case where a previous user of the browser session needs to be logged out of the tool.

**find_visit** *(context, resource_id)*  
Finds a visit that matches this resource_id
establish_session(context)
Overridden to update the Session ID in the visit

merge_session(context, merge_session)
Overridden to update the Session ID in any associated visits

load_visit(context)
Loads an existing LTI visit into the context

You'll normally call this method from each session decorated method of your tool provider that applies to
a protected resource.

This method sets the following attributes of the context...

ToolProviderContext.resource The resource record is identified from the resource id given in
the URL path.

ToolProviderContext.visit The session is searched for a visit record matching the resource.

ToolProviderContext.permissions Set from the visit record

ToolProviderContext.user The optional user is loaded from the visit.

ToolProviderContext.group The context record identified from the resource id given in the URL
path. This may be None if the resource link was not created in any context.

ToolProviderContext.consumer The consumer object is looked up from the visit entity.

If the visit can’t be set then an exception is raised, an unknown resource raises pyslet.wsgi.
PageNotFound whereas the absence of a valid visit for a known resource raises pyslet.wsgi.
PageNotAuthorized. These are caught automatically by the WSGI handlers and return 404 and 403
errors respectively.

launch_redirect(context)
Redirects to the resource identified on launch

A POST request should pretty much always redirect to a GET page and our tool launches are no different.
This allows you to reload a tool page straight away if desired without the risk of double-POST issues.

class pyslet.imsbltiv1p0.ToolProviderContext(environ, start_response, canonical_root=None)

Bases: pyslet.wsgi.SessionContext

consumer = None
a ToolConsumer instance identified from the launch

parameters = None
a dictionary of non-oauth parameters from the launch

visit = None
the effective visit entity

resource = None
the effective resource entity

user = None
the effective user entity

group = None
the effective group (context) entity

permissions = None
the effective permissions (an integer for bitwise testing)
class pyslet.imsbltiv1p0.ToolConsumer(entity, cipher)

    Bases: object

    An LTI consumer object

    entity  An Entity instance.
    cipher  An AppCipher instance.

This class is a light wrapper for the entity object that is used to persist information on the server about the consumer. The consumer is persisted in a data store using a single entity passed on construction which must have the following required properties:

ID: Int64  A database key for the consumer.
Handle: String  A convenient handle for referring to this consumer in the user interface of the silo’s owner. This handle is never exposed to users launching the tool through the LTI protocol. For example, you might use handles like “LMS Prod” and “LMS Staging” as handles to help distinguish different consumers.
Key: String  The consumer key
Secret: String  The consumer secret (encrypted using cipher).
Silo: Entity  Required navigation property to the Silo this consumer is associated with.
Contexts: Entity Collection  Navigation property to the associated contexts from which this tool has been launched.
Resources: Entity Collection  Navigation property to the associated resources from which this tool has been launched.
Users: Entity Collection  Navigation property to the associated users that have launched the tool.

entity = None
    the entity that persists this consumer
cipher = None
    the cipher used to
key = None
    the consumer key
secret = None
    the consumer secret
classmethod new_from_values(entity, cipher, handle, key=None, secret=None)

    Create an instance from an new entity
    entity  An Entity instance from a suitable entity set.
    cipher  An AppCipher instance, used to encrypt the secret before storing it.
    handle  A string
    key (optional)  A text string, defaults to a string generated with generate_key()
    secret (optional)  A text string, defaults to a string generated with generate_key()

    The fields of the entity are set from the passed in parameters (or the defaults) and then a new instance of cls is constructed from the entity and cipher and returned as a the result.

update_from_values(handle, secret)

    Updates an instance from new values
    handle  A string used to update the consumer’s handle
    secret  A string used to update the consumer’s secret
It is not possible to update the consumer key as this is used to set the ID of the consumer itself.

**nonce_key (nonce)**
- Returns a key into the nonce table
  - **nonce** A string received as a nonce during an LTI launch.
  
  This method hashes the nonce, along with the consumer entity’s Key, to return a hex digest string that can be used as a key for comparing against the nonces used in previous launches.
  
  Mixing the consumer entity’s Key into the hash reduces the chance of a collision between two nonces from separate consumers.

**get_context (context_id, title=None, label=None, ctypes=None)**
- Returns a context entity
  - **context_id** The context_id string passed on launch
  - **title (optional)** The title string passed on launch
  - **label (optional)** The label string passed on launch
  - **ctypes (optional)** An array of URI instances representing the context types of this context. See `CONTEXT_TYPE_HANDLES` for more information.

Returns the context entity.

If this context has never been seen before then a new entity is created and bound to the consumer. Otherwise, the additional information (if supplied) is compared and updated as necessary.

**get_resource (resource_link_id, title=None, description=None, context=None)**
- Returns a resource entity
  - **resource_link_id** The resource_link_id string passed on launch (required).
  - **title (optional)** The title string passed on launch, or None.
  - **description (optional)** The description string passed on launch, or None.
  - **context (optional)** The context entity referred to in the launch, or None.

If this resource has never been seen before then a new entity is created and bound to the consumer and (if specified) the context. Otherwise, the additional information (if supplied) is compared and updated as necessary, with the proviso that a resource can never change context, as per the following quote from the specification:

```
[resource_link_id] will also change if the item is exported from one system or context and imported into another system or context.
```

**get_user (user_id, name_given=None, name_family=None, name_full=None, email=None)**
- Returns a user entity
  - **user_id** The user_id string passed on launch
  - **name_given** The user’s given name (or None)
  - **name_family** The user’s family name (or None)
  - **name_full** The user’s full name (or None)
  - **email** The user’s email (or None)

If this user has never been seen before then a new entity is created and bound to the consumer, otherwise the
class pyslet.imsbltiv1p0.ToolProvider(consumers, nonces, cipher)

Bases: pyslet.imsbltiv1p0.OAuthMissing, pyslet.pep8.MigratedClass

An LTI tool provider object

consumers The EntitySet containing the tool Consumers.

nonces The EntitySet containing the used Nonces.

cipher An AppCipher instance. Used to decrypt the consumer secret from the database.

Implements the RequestValidator object required by the oauthlib package. Internally creates an instance of SignatureOnlyEndpoint

consumers = None
    The entity set containing Silos

nonces = None
    The entity set containing Nonces

cipher = None
    The cipher object used for encrypting consumer secrets

lookup_consumer(key)
    Implements the required method for consumer lookup

    Returns a ToolConsumer instance or raises a KeyError if key is not the key of any known consumer.

launch(command, url, headers, body_string)
    Checks a launch request for authorization

    command The HTTP method, as an upper-case string. Should be POST for LTI.

    url The full URL of the page requested as part of the launch. This will be the launch URL specified in the LTI protocol and configured in the consumer.

    headers A dictionary of headers, must include the Authorization header but other values are ignored.

    body_string The query string (in the LTI case, this is the content of the POST request).

    Returns a ToolConsumer instance and a dictionary of parameters on success. If the incoming request is not authorized then LTIAuthenticationError is raised.

    This method also checks the LTI message type and protocol version and will raise LTIProtocolError if this is not a recognized launch request.

Launch(*args, **kwargs)
    Deprecated equivalent to launch()

Metadata

pyslet.imsbltiv1p0.load_metadata()
    Loads the default metadata document

    Returns a pyslet.odata2.metadata.Document instance. The schema is loaded from a bundled metadata document which contains the minimum schema required for an LTI tool provider.

Constants and Data

pyslet.imsbltiv1p0.LTI_VERSION = 'LTI-1p0'
    The version of LTI we support
pyslet.imsbltiv1p0.LTI_MESSAGE_TYPE = 'basic-lti-launch-request'
The message type we support

pyslet.imsbltiv1p0.SYSROLE_HANDLES = {'Administrator': <pyslet.urn.URN object>, 'Creator': <pyslet.urn.URN object>}
A mapping from a system role handle to the full URN for the role as a URI instance.

pyslet.imsbltiv1p0.INSTROLE_HANDLES = {'None': <pyslet.urn.URN object>, 'Guest': <pyslet.urn.URN object>}
A mapping from an institution role handle to the full URN for the role as a URI instance.

pyslet.imsbltiv1p0.ROLE_HANDLES = {'Manager/CourseCoordinator': <pyslet.urn.URN object>, 'Manager/Instructor': <pyslet.urn.URN object>, 'Manager/Instructor': <pyslet.urn.URN object>}
A mapping from LTI role handles to the full URN for the role as a URI instance.

pyslet.imsbltiv1p0.split_role(role)
Splits an LTI role into vocab, type and sub-type

role A URN instance containing the full definition of the role.

Returns a triple of:

vocab One of ‘role’, ‘sysrole’, ‘instrole’ or some future vocab extension.

type The role type, e.g., ‘Learner’, ‘Instructor’

rsstype The role sub-type, e.g., ‘NonCreditLearner’, ‘Lecturer’. Will be None if there is no sub-type.

If this is not an LTI defined role, or the role descriptor does not start with the path ims/lis then ValueError is raised.

pyslet.imsbltiv1p0.is_subrole(role, parent_role)
True if role is a sub-role of parent_role

role A URN instance containing the full definition of the role to be tested.

parent_role A URN instance containing the full definition of the parent role. It must not define a subrole of

ValueError is raised.

In the special case that role does not have subrole then it is simply matched against parent_role. This ensures that:

is_subrole(role, ROLE_HANDLES['Learner'])

will return True in all cases where role is a Learner role.

pyslet.imsbltiv1p0.CONTEXT_TYPE_HANDLES = {'CourseSection': <pyslet.urn.URN object>, 'CourseOffering': <pyslet.urn.URN object>}
A mapping from LTI context type handles to the full URN for the context type as a URI instance.

Exceptions

class pyslet.imsbltiv1p0.LTIError
Bases: exceptions.Exception
Base class for all LTI errors

class pyslet.imsbltiv1p0.LTIAuthenticationError
Bases: pyslet.imsbltiv1p0.LTIError
Indicates an authentication error (on launch)

class pyslet.imsbltiv1p0.LTIProtocolError
Bases: pyslet.imsbltiv1p0.LTIError
Indicates a protocol violation
This may be raised if the message type or protocol version in a launch request do not match the expected values or if a required parameter is missing.

## Legacy Classes

Earlier Pyslet versions contained a very rudimentary memory based LTI tool provider implementation based on the older oauth module. These classes have been superceded but the main BLTIToolProvider class has been refactored as a derived class of `ToolProvider` using a SQLite ':memory:' database (instead of a Python dictionary) and the existing method signatures should continue to work as before.

The only change you’ll need to make is to install the newer `oauthlib`. Bear in mind that these classes are now deprecated and you should refactor to use the base `ToolProvider` class directly for future compatibility. Please raise an issue on GitHub if you anticipate problems.

```python
class pyslet.imsbltiv1p0.BLTIToolProvider
    Bases: pyslet.imsbltiv1p0.ToolProvider

    Legacy class for tool provider.

    Refactored to build directly on the newer `ToolProvider`. A single Silo entity is created containing all defined consumers. An in-memory SQLite database is used as the data store. Consumer keys are not encrypted (a plaintext cipher is used) as they will not be persisted.

    `generate_key(key_length=128)`
    Generates a new key
    This method is deprecated, it has been replaced by the similarly named function `pyslet.wsgi.generate_key()`.

    `key_length` The minimum key length in bits. Defaults to 128.

    The key is returned as a sequence of 16 bit hexadecimal strings separated by ‘.’ to make them easier to read and transcribe into other systems.

    `new_consumer(key=None, secret=None)`
    Creates a new BLTIConsumer instance
    This method is deprecated, it has been replaced by the similarly named function `pyslet.wsgi.generate_key()`.

    `load_from_file(f)`
    Loads the list of trusted consumers
    This method is deprecated, it has been replaced by the similarly named function `pyslet.wsgi.generate_key()`.
```

## GenerateKey

```python
GenerateKey(*args, **kwargs)
Deprecated equivalent to `generate_key()`
```

## LoadFromFile

```python
LoadFromFile(*args, **kwargs)
Deprecated equivalent to `load_from_file()`
```

## NewConsumer

```python
NewConsumer(*args, **kwargs)
Deprecated equivalent to `new_consumer()`
```
SaveToFile (*args, **kwargs)
   Deprecated equivalent to `save_to_file()`

save_to_file (f)
   Saves the list of trusted consumers
   Also available as SaveToFile
   The consumers are saved in a simple file suitable for reading with `load_from_file()`.
This sub-package defines functions and classes for working with OData, a data access protocol based on Atom and Atom Pub: http://www.odata.org/

This sub-package only deals with version 2 of the protocol at the moment.

OData is not an essential part of supporting the Standards for Learning, Education and Training (SLET) that gives pyslet its name, though I have actively promoted its use in these communities. As technical standards move towards using REST-ful web services it makes sense to converge around some common patterns for common use cases. Many of the protocols now being worked on are much more like basic data-access layers spread over the web between two co-operating systems. HTTP on its own is often good enough for these applications but when the data lends itself to tabular representations I think the OData standard is the best protocol available.

The purpose of this group of modules is to make is easy to use the conventions of the OData protocol as a general purpose data-access layer (DAL) for Python applications. To get started, look at the Data Consumers section which gives a high-level overview of the API with examples that use Microsoft’s Northwind data-service.

If you are interested in writing an OData provider, or you simply want to use these classes to implement a data access layer for your own application then look in OData Providers.

4.1 Data Consumers

Warning: the OData client doesn’t support certificate validation when accessing servers through https URLs. This feature is coming soon.

4.1.1 Introduction

Let’s start with a simple illustration of how to consume data using the DAL API by walking through the use of the OData client.

The client implementation uses Python’s logging module to provide logging, when learning about the client it may help to turn logging up to “INFO” as it makes it clearer what the client is doing. “DEBUG” would show exactly what is passing over the wire.
>>> import logging
>>> logging.basicConfig(level=logging.INFO)

To create a new client simply instantiate a Client object. You can pass the URL of the service root you wish to connect
to directly to the constructor which will then call the service to download the list of feeds and the metadata document
from which it will set the Client.model.

>>> from pyslet.odata2.client import Client
>>> c = Client("http://services.odata.org/V2/Northwind/Northwind.svc/")
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/ HTTP/1.1
INFO:root:Finished Response, status 200
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/$metadata HTTP/1.1
INFO:root:Finished Response, status 200

The Client.feeds attribute is a dictionary mapping the exposed feeds (by name) onto EntitySet instances. This makes it easy to open the feeds as EDM collections. In your code you’d typically use the with statement when opening the collection but for clarity we’ll continue on the python command line:

>>> products = c.feeds['Products'].open()
>>> for p in products: print p
1 2 3 ... [and so on]
...
20
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products?$skiptoken=20 HTTP/1.1
INFO:root:Finished Response, status 200
21 22 23 ... [and so on]
...
76 77

Note that products behaves like a dictionary, iterating through it iterates through the keys in the dictionary. In this
case these are the keys of the entities in the collection of products. Notice that the client logs several requests to the
server interspersed with the printed output. Subsequent requests use $skiptoken because the server is limiting the
maximum page size. These calls are made as you iterate through the collection allowing you to iterate through very
large collections.

The keys alone are of limited interest, let’s try a similar loop but this time we’ll print the product names as well:

>>> for k, p in products.items(): print k, p['ProductName'].value
...
Sir Rodney’s Scones sound interesting, we can grab an individual record in the usual way:

```python
>>> scones = products[21]
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(21) HTTP/1.1
INFO:root:Finished Response, status 200
>>> for k, v in scones.data_items(): print k, v.value
...
ProductID 21
ProductName Sir Rodney's Scones
SupplierID 8
CategoryID 3
QuantityPerUnit 24 pkgs. x 4 pieces
UnitPrice 10.0000
UnitsInStock 3
UnitsOnOrder 40
ReorderLevel 5
Discontinued False
```
Attempting to load a non existent entity results in a KeyError of course:

```python
>>> p = products[211]
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(211) HTTP/1.1
INFO:root:Finished Response, status 404
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File /Library/Python/2.7/site-packages/pyslet/odata2/client.py, line 165, in __getitem__
    raise
KeyError: 211
```

Finally, when we’re done, it is a good idea to close the open collection:

```python
>>> products.close()
```

### 4.1.2 The Data Access Layer in Depth

In the introduction we created an OData Client object using a URL, but in general the way you connect to a data service will vary depending on the implementation. The Client class itself isn’t actually part of the DAL API itself.

The API starts with a model of the data service. The model is typically parsed from an XML file. For the OData client the XML file is obtained from the service’s $metadata URL. Here’s an extract from the Northwind $metadata file showing the definition of the data service, I’ve removed the XML namespace definitions for brevity:

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<edmx:Edmx Version="1.0">
  <edmx:DataServices m:DataServiceVersion="1.0">
    <Schema Namespace="NorthwindModel">
      <EntityType Name="Category">
        <!-- rest of the definitions go here... -->
      </EntityType>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

Each element is represented by an object in Pyslet, the starting point for the API is the `DataServices` object. A `DataServices` object can contain multiple `Schema` elements, which in turn can contain multiple `EntityContainer` elements which in turn can contain multiple `EntitySet` elements. The following diagram illustrates these relationships and compares them with approximate equivalent concepts in a typical SQL-scenario.

In the OData client example we used a short-cut to get to the EntitySet objects we were interested in by using the feeds property of the client itself. However, we could have used the model directly as follows, continuing with the same session:

```python
>>> c.model
<pyslet.odata2.metadata.Edmx object at 0x10140a9d0>
>>> c.model.DataServices
<pyslet.odata2.metadata.DataServices object at 0x107fdb990>
```
As you can see, the same EntitySet object can be obtained by looking it up in the parent container which behaves like a dictionary, this in turn can be looked up in the parent Schema which in turn can be looked up in the DataServices enclosing object. Elements of the model also support deep references using dot-concatenation of names which makes the code easier to read:

```python
>>> print c.model.DataServices['ODataWeb.Northwind.Model']['NorthwindEntities']['Products'].get_fqname()
```

When writing an application that would normally use a single database you should pass an EntityCollection object to it as a data source rather than the DataServices ancestor. It is best not to pass an implementation-specific class like the OData Client as that will make the application dependent on a particular type of data source.

**Entity Sets**

The following attributes are useful for consumers of the API (and should be treated as read only)

- **name** The name of the entity set
- **entityTypeName** The name of the entity set’s EntityType
- **entityType** The `EntityType` object that defines the properties for entities in this set.
- **keys** A list of the names of the keys for this EntitySet. For example:

```python
>>> print products.keys
[u'ProductID']
```

For entity types with compound keys this list will contain multiple items of course.

The following methods are useful for consumers of the API.

- **get_fqname()** Returns the fully qualified name of the entity set, suitable for looking up the entity set in the enclosing DataServices object.

```python
>>> print products.get_fqname()
```

- **get_location()** Returns a `pyslet.rfc2396.URI` object that represents this entity set:

```python
>>> print products.get_location()
http://services.odata.org/V2/Northwind/Northwind.svc/Products
```

(If there is no base URL available this will be a relative URL)

- **open()** Returns a `pyslet.odata2.csdl.EntityCollection` object that can be used to access the entities in the set.

4.1. Data Consumers
get_target()  Returns the target entity set of a named navigation property.

get_multiplicity()  Returns a tuple of multiplicity constants for the named navigation property. Constants for these values are defined in `pyslet.odata2.csd1.Multiplicity`, for example:

```python
>>> from pyslet.odata2.csd1 import Multiplicity, multiplicity_to_str
>>> print Multiplicity.ZeroToOne, Multiplicity.One, Multiplicity.Many
0 1 2
>>> products.get_multiplicity('Supplier')
(2, 0)
>>> map(lambda x:multiplicity_to_str(x),products.get_multiplicity('Supplier'))
['*', '0..1']
```

is_entity_collection()  Returns True if the named navigation property points to a collection of entities or a single entity. In Pyslet, you can treat all navigation properties as collections. In the above example the collection of Supplier entities obtained by following the ‘Supplier’ navigation property of a Product entity will have at most 1 member.

Entity Collections

To continue with database analogy above, if EntitySets are like SQL Tables EntityCollections are somewhat like the database cursors that you use to actually read data - the difference is that EntityCollections can only read entities from a single EntitySet.

An `EntityCollection` may consume physical resources (like a database connection) and so must be closed with its `close()` method when you’re done. They support the context manager protocol to make this easier so you can use them in with statements to make clean-up easier:

```python
with c.feeds['Products'].open() as products:
    if 42 in products:
        print "Found it!"
```

The close method is called automatically when the with statement exits.

Entity collections also behave like a python dictionary of `Entity` instances keyed on a value representing the Entity’s key property or properties. The keys are either single values (as in the above code example) or tuples in the case of compound keys. The order of the values in the tuple is taken from the order of the PropertyRef definitions in the model.

There are two ways to obtain an EntityCollection object. You can open an entity set directly or you can open a collection by navigating from a specific entity through a named navigation property. Although dictionary-like there are some differences with true dictionaries.

When you have opened a collection from the base entity set the following rules apply:

collection[key]  Returns a new `Entity` instance by looking up the `key` in the collection. As a result, subsequent calls will return a different object, but with the same key!

collection[key] = new_entity  For an existing entity this is essentially a no-operation. This form of assignment cannot be used to create a new entity in the collection because the act of inserting the entity may alter its key (for example, when the entity set represents a database table with an auto-generated primary key). See below for information on how to create and update entities.

del collection[key]  In contrast, del will remove an entity from the collection completely.

When an EntityCollection represents a collection of entities obtained by navigation then these rules are updated as follows:
collection[key] Normally returns a new Entity instance by looking up the key in the collection but when the navigation property has been expanded it will return a cached Entity (so subsequent calls will return the same object without looking up the key in the data source again).

collection[key] = existingEntity Provided that key is the key of existingEntity this will add an existing entity to this collection, effectively creating a link from the entity you were navigating from to an existing entity.

del collection[key] Removes the entity with key from this collection. The entity is not deleted from its EntitySet, is merely unlinked from the entity you were navigating from.

The following attribute is useful for consumers of the API (and should be treated as read only)

entity_set The EntitySet of this collection. In the case of a collection opened through navigation this is the base entity set.

In addition to all the usual dictionary methods like len, itervalues and so on, the following methods are useful for consumers of the API:

get_location() Returns a pyslet.rfc2396.URI object that represents this entity collection.

get_title() Returns a user-friendly title to represent this entity collection.

new_entity() Creates a new entity suitable for inserting into this collection. The entity does not exist until it is inserted with insert_entity.

copy_entity() Creates a new entity by copying all non-key properties from another entity. The entity does not exist until it is inserted with insert_entity.

insert_entity() Inserts an entity previously created by new_entity or copy_entity. When inserting an entity any active filter is ignored.

Warning: an active filter may result in a paradoxical KeyError:

```
import pyslet.odata2.core as core
with people.open() as collection:
    collection.set_filter(core.CommonExpression.from_str("startswith(Name,'D')"))
    new_entity = collection.new_entity()
    new_entity['Key'].set_from_value(1)
    new_entity['Name'].set_from_value(u"Steve")
    collection.insert_entity(new_entity)
    # new_entity now exists in the base collection but...
    el = collection[1]
    # ...raises KeyError as new_entity did not match the filter!
```

It is recommended that collections used to insert entities are not filtered.

update_entity() Updates an existing entity following changes to the Entity’s values. You can’t update the values of key properties. To change the key you will need to create a new entity with copy_entity, insert the new entity and then remove the old one. Like insert_entity, the current filter is ignored.

set_page() Sets the top and skip values for this collection, equivalent to the $top and $skip options in OData. This value only affects calls to iterpage. See Paging for more information.

iterpage() Iterates through a subset of the entities returned by itervalues defined by the top and skip values. See Paging for more information.

set_filter() Sets the filter for this collection, equivalent to the $filter option in OData. Once set this value effects all future entities returned from the collection (with the exception of new_entity). See Filtering Collections for more information.

set_orderby() Sets the filter for this collection, equivalent to the $orderby option in OData. Once set this value effects all future iterations through the collection. See Ordering Collections for more information.
**set_expand()**  Sets expand and select options for this collection, equivalent to the $expand and $select system query options in OData. Once set these values effect all future entities returned from the collection (with the exception of new_entity). See *Expand and Select* for more information.

### Paging

*Supported from build 0.4.20140215 onwards*

The $top/$skip options in OData are a useful way to restrict the amount of data that an OData server returns. The collection dictionary always behaves as if it contains all entities so the value returned by `len` doesn’t change if you set top and skip values and nor does the set of entities returned by `itervalues` and similar methods.

In most cases, the server will impose a reasonable maximum on each request using server-enforced paging. However, you may wish to set a smaller top value or simply have more control over the automatic paging implemented by the default iterators.

To iterate through a single page of entities you’ll start by using the the `set_page()` method to specify values for top and, optionally, skip. You must then use the `iterpage()` method to iterate through the entities in just that page. The `set_next` boolean parameter indicates whether or not the next call to `iterpage` iterates over the same page or the next page of the collection.

To continue the example above, in which `products` is an open collection from the Northwind data service:

```python
>>> products.set_page(5, 50)
>>> for p in products.iterpage(True): print p.key(), p['ProductName'].value
... 51 Manjimup Dried Apples 52 Filo Mix 53 Perth Pasties 54 Tourtière 55 Pâté chinois
>>> for p in products.iterpage(True): print p.key(), p['ProductName'].value
... 56 Gnocchi di nonna Alice 57 Ravioli Angelo 58 Escargots de Bourgogne 59 Raclette Courdavault 60 Camembert Pierrot
```

In some cases, the server will restrict the page size and fewer entities will be returned than expected, in these cases the skiptoken is used automatically when the next page is requested:

```python
>>> products.set_page(30, 50)
>>> for p in products.iterpage(True): print p.key(), p['ProductName'].value
... INFO:root:Sending request to services.odata.org INFO:root:GET /V2/Northwind/Northwind.svc/Products?$skip=50&$top=30 HTTP/1.1 INFO:root:Finished Response, status 200 51 Manjimup Dried Apples 52 Filo Mix 53 Perth Pasties ... [and so on]
```
Filtering Collections

By default, an entity collection contains all items in the entity set or, if the collection was obtained by navigation, all items linked to the entity by the property being navigated. Filtering a collection (potentially) selects a sub-set of these entities based on a filter expression.

Filter expressions are set using the `set_filter()` method of the collection. Once a filter is set, the dictionary methods, and `iterpage`, will only return entities that match the filter.

The easiest way to set a filter is to compile one directly from a string representation using OData’s query language. For example:

```python
>>> import pyslet.odata2.core as core
>>> filter = core.CommonExpression.from_str("substringof('one',Product Name)")
>>> products.set_filter(filter)
>>> for p in products.itervalues(): print p.key(), p['ProductName'].value
```

To remove a filter, set the filter expression to None:

```python
>>> products.set_filter(None)
```

Ordering Collections

Like OData and python dictionaries, this API does not specify a default order in which entities will be returned by the iterators. However, unlike python dictionaries you can control this order using an orderby option.

OrderBy expressions are set using the `set_orderby()` method of the collection. Once an order by expression is set, the dictionary methods, and `iterpage`, will return entities in the order specified.

The easiest way to define an ordering is to compile one directly from a string representation using OData’s query language. For example:
To remove an ordering, set the orderby expression to None:

```
>>> products.Orderby(None)
```

## Expand and Select

Expansion and selection are two interrelated concepts in the API. Expansion allows you to follow specified navigation properties retrieving the entities they link to in the same way that simple and complex property values are retrieved.

Expand options are represented by nested dictionaries of strings. For example, to expand the Supplier navigation property of Products you would use a dictionary like this:

```
expansion={'Supplier': None}
```

The value in the dictionary is either None, indicating no further expansion, or another dictionary specifying the expansion to apply to any linked Suppliers:

```
>>> products.set_expand({'Supplier': None}, None)
```

```python
>>> scones = products[21]  
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(21)?$expand=Supplier HTTP/1.1
INFO:root:Finished Response, status 200
>>> supplier=scones['Supplier'].get_entity()  
>>> for k, v in supplier.data_items(): print k, v.value
  ... SupplierID 8
```
A critical point to note is that applying an expansion to a collection means that linked entities are retrieved at the same time as the entity they are linked to and cached. In the example above, the get_entity call does not generate a call to the server. Compare this with the same code executed without the expansion:

```python
>>> products.set_expand(None, None)
>>> scones = products[21]
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(21) HTTP/1.1
INFO:root:Finished Response, status 200
>>> supplier = scones['Supplier'].get_entity()
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(21)/Supplier HTTP/1.1
INFO:root:Finished Response, status 200
```

The select option complements expansion, narrowing down the simple and complex properties that are retrieved from the data source. You specify a select option in a similar way, using nested dictionaries. Simple and complex properties must always map to None, for a more complex example with navigation properties see below. Suppose we are only interested in the product name:

```python
>>> products.set_expand(None, {'ProductName':None})
>>> scones = products[21]
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(21)?$select=ProductID→%2CPproductName HTTP/1.1
INFO:root:Finished Response, status 200
>>> for k, v in scones.data_items(): print k, v.value
...
ProductID 21
ProductName Sir Rodney's Scones
SupplierID None
CategoryID None
QuantityPerUnit None
UnitPrice None
UnitsInStock None
UnitsOnOrder None
ReorderLevel None
Discontinued None
```

In Pyslet, the values of the key properties are always retrieved, even if they are not selected. This is required to maintain the dictionary-like behaviour of the collection. An entity retrieved this way has NULL values for any properties that weren't retrieved. The `is_selected()` method allows you to determine if a value is NULL in the data source or NULL because it is not selected:

```python
>>> for k, v in scones.data_items():
...     if scones.is_selected(k): print k, v.value
...
```
The expand and select options can be combined in complex ways:

```python
>>> products.set_expand({'Supplier':None}, {'ProductName':None, 'Supplier':{'Phone':None}})

>>>
```

```python
>>> scones = products[21]
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(21)?$expand=Supplier&...
INFO:root:Finished Response, status 200

>>> supplier = scones['Supplier'].get_entity()
>>> for k, v in scones.data_items():
...   if scones.is_selected(k): print k, v.value
...
ProductID 21
ProductName Sir Rodney's Scones
>>> for k, v in supplier.data_items():
...   if supplier.is_selected(k): print k, v.value
...
SupplierID 8
Phone (161) 555-4448
```

## Entity Objects

Continuing further with the database analogy and Entity is like a single record.

Entity instances behave like a read-only dictionary mapping property names onto their values. The values are either SimpleValue, Complex or DeferredValue instances. All property values are created on construction and cannot be assigned. To update a SimpleValue, whether it is a direct child or part of a Complex value, use its `set_from_value()` method:

```python
entity['Name'].set_from_value("Steve")
entity['Address']['City'].set_from_value("Cambridge")
```

The following attributes are useful for consumers of the API (and should be treated as read only):

- **entity_set** The `EntitySet` to which this entity belongs.
- **type_def** The `EntityType` which defines this entity’s type.
- **exists** True if this entity exists in the collection, i.e., it was returned by one of the dictionary methods of an entity collection such as `itervalues` or `[key]` look-up.

The following methods are useful for consumers of the API:

- **key()** Returns the entity’s key, as a single python value or tuple in the case of compound keys
- **get_location()** Returns a `pyslet.rfc2396.URI` object that represents this entity:

```python
>>> print scones.get_location()
http://services.odata.org/V2/Northwind/Northwind.svc/Products(21)
```

- **data_keys()** Iterates over the simple and complex property names:
```python
>>> list(scones.data_keys())
[u'ProductID', u'ProductName', u'SupplierID', u'CategoryID', u'QuantityPerUnit', u'
→'UnitPrice', u'UnitsInStock', u'UnitsOnOrder', u'ReorderLevel', u'Discontinued']
```

**data_items()**  Iterates over tuples of simple and complex property (name,value) pairs. See above for examples of usage.

**is_selected()**  Tests if the given data property is selected.

**navigation_keys()**  Iterates over the navigation property names:

```python
>>> list(scones.navigation_keys())
[u'Category', u'Order_Details', u'Supplier']
```

**navigation_items()**  Iterates over tuples of navigation property (name,DeferredValue) pairs.

**is_navigation_property()**  Tests if a navigation property with the given name exists

The following methods can be used only on entities that exists, i.e., entities that have been returned from one of the collection’s dictionary methods:

**commit()**  Normally you’ll use the the update_entity method of an open EntityCollection but in cases where the originating collection is no longer open this method can be used as a convenience method for opening the base collection, updating the entity and then closing the collection collection again.

**delete()**  Deletes this entity from the base entity set. If you already have the base entity set open it is more efficient to use the `del` operator but if the collection is no longer open or the entity was obtained from a collection opened through navigation then this method can be used to delete the entity.

The following method can only be used on entities that don’t exist, i.e., entities returned from the collection’s new_entity or copy_entity methods that have not been inserted.

**set_key()**  Sets the entity’s key

### SimpleValue

Simple property values are represented by (sub-classes of) `SimpleValue`, they share a number of common methods:

**is_null()**  Returns True if this value is NULL. This method is also used by Python’s non-zero test so:

```python
if entity['Property']:
    print entity['Property'].value
# prints even if value is 0
```

will print the Property value of entity if it is non-NULL. In particular, it will print empty strings or other representations of zero. If you want to exclude these from the test you should test the value attribute directly:

```python
if entity['Property'].value:
    print entity['Property'].value
# will not print if value is 0
```

**set_from_value()**  Updates the value, coercing the argument to the correct type and range checking its value.

**SetFromSimpleValue()**  Updates the value from another SimpleValue, if the types match then the value is simply copied, otherwise the value is coerced using `set_from_value`.

**set_from_literal()**  Updates the value by parsing it from a (unicode) string. This is the opposite to the unicode function. The literal form is the form used when serializing the value to XML (but does not include XML character escaping).
**set_null()**  Updates the value to NULL

The value attribute is always an immutable value in python and so can be used as a key in your own dictionaries. The following list describes the mapping from the EDM-defined simple types to their corresponding native Python types.

- **Edm.Boolean:** one of the Python constants True or False

- **Edm.Byte, Edm.SByte, Edm.Int16, Edm.Int32:** int

- **Edm.Int64:** long

- **Edm.Double, Edm.Single:** python float

- **Edm.Decimal:** python Decimal instance (from the built-in decimal module)

- **Edm.DateTime, Edm.DateTimeOffset:** py:class:pyslet.iso8601.TimePoint instance

  This is a custom object in Pyslet, see Working with Dates for more information.

- **Edm.Time:** py:class:pyslet.iso8601.Time instance

  Early versions of the OData specification incorrectly mapped this type to the XML Schema duration. The use of a Time object to represent it, rather than a duration, reflects this correction.

  See Working with Dates for more information.

- **Edm.Binary:** raw string

- **Edm.String:** unicode string

- **Edm.Guid:** Python UUID instance (from the built-in uuid module)

**Complex**

Complex values behave like dictionaries of data properties. They do not have keys or navigation properties. They are never NULL, is_null and the Python non-zero test will always return True.

**set_null()**  Although a Complex value can never be NULL, this method will set all of its data properties (recursively if necessary) to NULL

**DeferredValue**

Navigation properties are represented as DeferredValue instances. All deferred values can be treated as an entity collection and opened in a similar way to an entity set:

```python
>>> sconeSuppliers=scones['Supplier'].open()
```

```python
>>> for s in sconeSuppliers.itervalues(): print s['CompanyName'].value
```

... INFO:root:Sending request to services.odata.org

INFO:root:GET /V2/Northwind/Northwind.svc/Products(21)/Supplier HTTP/1.1

INFO:root:Finished Response, status 200

Specialty Biscuits, Ltd.

```python
>>>```

For reading, a collection opened from a deferred value behaves in exactly the same way as a collection opened from a base entity set. However, for writing there are some difference described above in **Entity Collections.**

If you use the dictionary methods to update the collection the changes are made straight away by accessing the data source directly. If you want to make a number of changes simultaneously, or you want to link entities to entities that don’t yet exist, then you should use the bind_entity method described below instead. This method defers the changes until the parent entity is updated (or inserted, in the case of non-existent entities.)
Read-only attributes useful to data consumers:

- **name** The name of the navigation property
- **from_entity** The parent entity of this navigation property
- **p_def** The `NavigationProperty` that defines this navigation property in the model.
- **isRequired** True if the target of this property has multiplicity 1, i.e., it is required.
- **isCollection** True if the target of this property has multiplicity *
- **isExpanded** True if this navigation property has been expanded. Expanded navigation keep a cached version of the target collection. Although you can open it and use it in the same way any other collection the values returned are returned from the cache and not by accessing the data source.

Methods useful to data consumers:

- **open()** Returns an `pyslet.odata2.csd1.EntityCollection` object that can be used to access the target entities.
- **get_entity()** Convenience method that returns the entity that is the target of the link when the target has multiplicity 1 or 0..1. If no entity is linked by the association then None is returned.
- **bind_entity()** Marks the target entity for addition to this navigation collection on next update or insert. If this navigation property is not a collection then the target entity will replace any existing target of the link.
- **target()** The target entity set of this navigation property.

### Working with Dates

In the EDM there are two types of date, DateTime and DateTimeOffset. The first represents a time-point in an implicit zone and the second represents a time-point with the zone offset explicitly set.

Both types are represented by the custom `pyslet.iso8601.TimePoint` class in Pyslet.

**time module from build 0.4.20140217 onwards**

Interacting with Python’s time module is done using the struct_time type, or lists that have values corresponding to those in struct_time:

```python
>>> import time
>>> orders = c.feeds['Orders'].open()
>>> orders.set_page(5)
>>> top = list(orders.iterpage())
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Orders?$skip=0&$top=5 HTTP/1.1
INFO:root:Finished Response, status 200
>>> print top[0]['OrderDate'].value
1996-07-04T00:00:00
>>> t = [None]*9
>>> top[0]['OrderDate'].value.update_struct_time(t)
>>> t
[1996, 7, 4, 0, 0, 0, 3, 186, -1]
>>> time.strftime("%a, %d %b %Y %H:%M:%S",t)
'Thu, 04 Jul 1996 00:00:00'
```

You can set values obtained from the time module in a similar way:

```python
>>> import pyslet.iso8601 as iso
>>> t = time.gmtime(time.time())
>>> top[0]['OrderDate'].set_from_value(iso.TimePoint.from_struct_time(t))
```
But if you just want a timestamp use one of the built-in factory methods:

```python
>>> top[0]['OrderDate'].set_from_value(iso.TimePoint.from_now_utc())
>>> print top[0]['OrderDate'].value
2014-02-17T21:56:23
```

In future versions, look out for better support for datetime and calendar module conversion methods.

### Working with Media Resources

OData is based on Atom and the Atom Publishing Protocol (APP) and inherits the concept of media resources and media link entries from those specifications.

In OData, an entity can be declared as a media link entry indicating that the main purpose of the entity is to hold a media stream. If the entity with the following URL is a media link entry:

```
http://host/Documents(123)
```

then the following URL provides access to the associated media resource:

```
http://host/Documents(123)/$value
```

In the DAL this behaviour is modelled by operations on the `collection` containing the entities. The methods you’ll use are:

- `is_medialink_collection()` Returns True if the entities are media link entries
- `read_stream()` Reads information about a stream, optionally copying the stream’s data to a file-like object.
- `new_stream()` Creates a new media resource, copying the stream’s data from a file-like object.

  This method implicitly creates an associated media link entry and returns the resulting `Entity` object. By its nature, APP does not guarantee the URL that will be used to store a posted resource. The implication for OData is that you can’t specify the key that will be used for the media resource’s entry, though this method does allow you to supply a hint.

- `update_stream()` Updates a media resource, copying the stream’s new data from a file-like object.

  If a collection is a collection of media link entries then the behaviour of :py:meth:`-pyslet.odata2.core.EntityCollection.insert_entity` is modified as entities are created implicitly when a new stream is added to the collection. In this case, `insert_entity` creates an empty stream of type application/octet-stream and then merges the property values from the entity being inserted into the new media link entry created for the stream.

### 4.2 OData Providers

The approach to writing a data access layer (DAL) taken by Pyslet is to use the Entity Data Model (EDM), and the extensions defined by OData, and to encapsulate them in an API defined by a set of abstract Python classes. The `Data Consumers` section goes through this API from the point of view of the consumer and provides a good primer for understanding what is required from a provider.

Pyslet includes three derived classes that implement the API in a variety of different storage scenarios:
1. OData Client - an implementation of the DAL that makes calls over the web to an OData server. Defined in the module `pyslet.odatav2.client` and used in the examples in the section Data Consumers.

2. In-memory data service - an implementation of the DAL that stores all entities and associations in python dictionaries. Defined in the module `pyslet.odatav2.memds`.

3. SQL data service - an implementation of the DAL that maps on to python’s database API. Defined in the module `pyslet.odatav2.sqlds`. In practice, the classes defined by this module will normally need to be sub-classed to deal with database-specific issues but a full implementation for SQLite is provided and a quick look at the source code for that should give you courage to tackle any modifications necessary for your favourite database. Using this DAL API is much easier than having to do these tweaks when they are distributed throughout your code in embedded SQL-statements.

A high-level plan for writing an OData provider would therefore be:

0. Identify the underlying DAL class that is most suited to your needs or, if there isn’t one, create a new DAL implementation using the existing implementations as a guide.

1. Create a metadata document to describe your data model

2. Write a test program that uses the DAL classes directly to validate that your model and the DAL implementation are working correctly

3. Create `pyslet.odata2.server.Server` that is bound to your model test it with an OData client to ensure that it works as expected.

4. Finally, create a sub-class of the server with any specific customisations needed by your application: mainly to implement your applications authentication and authorization model. (For a read-only service there may be nothing to do here.)

Of course, if all you want to do is use these interfaces as a DAL in your own application you can stop at item 3 above.

4.2.1 Sample Project: InMemory Data Service

The sample code for this service is in the samples/memcache directory in the Pyslet distribution.

This project demonstrates how to construct a simple OData service based on the `InMemoryEntityContainer` class. We don’t need any customisations, this class does everything we need ‘out of the box’.

**Step 1: Creating the Metadata Model**

Unlike other frameworks for implementing OData services Pyslet starts with the metadata model, it is not automatically generated: you must write it yourself!

Fortunately, there are plenty of examples you can use as a template. In this sample project we’ll write a very simple memory cache capable of storing a key-value pair. Here’s our data model:

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
  <edmx:DataServices m:DataServiceVersion="2.0">
      com/ado/2006/04/edm">
      <EntityContainer Name="MemCache" m:IsDefaultEntityContainer="true">
        <EntitySet Name="KeyValuePairs" EntityType="MemCacheSchema.KeyValuePair"/>
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```
Our model has one defined EntityType called `KeyValuePair` and one EntitySet called `KeyValuePairs` in a container called `MemCache`. The idea behind the model is that each key-value pair is inserted with an expires time, after which it is safe to clean it up.

For simplicity, we’ll save this model to a file and load it from the file when our script starts up. Here’s the source code:

```python
import pyslet.odata2.metadata as edmx

def load_metadata():
    """Loads the metadata file from the current directory."""
    doc = edmx.Document()
    with open('MemCacheSchema.xml', 'rb') as f:
        doc.read(f)
    return doc

def test_data(mem_cache):
    with mem_cache.open() as collection:
        for i in range(26):
            e = collection.new_entity()
            e.set_key(str(i))
            e['Value'].set_from_value(character(0x41 + i))
            e['Expires'].set_from_value(iso.TimePoint.from_unix_time(time.time() + 10 * i))
            collection.insert_entity(e)

def test_model():
    """Read and write some key value pairs"""
    doc = load_metadata()
    InMemoryEntityContainer(doc.root.DataServices['MemCacheSchema.MemCache'])
    test_data(mem_cache)
    with mem_cache.open() as collection:
```

The metadata module contains a Document object and the definitions of the elements in the edmx namespace that enable us to read the XML file.

**Step 2: Test the Model**

Let’s write a simple test function to test our model:

```python
def test_data(mem_cache):
    with mem_cache.open() as collection:
        for i in range(26):
            e = collection.new_entity()
            e.set_key(str(i))
            e['Value'].set_from_value(character(0x41 + i))
            e['Expires'].set_from_value(iso.TimePoint.from_unix_time(time.time() + 10 * i))
            collection.insert_entity(e)

def test_model():
    """Read and write some key value pairs"""
    doc = load_metadata()
    InMemoryEntityContainer(doc.root.DataServices['MemCacheSchema.MemCache'])
    test_data(mem_cache)
    with mem_cache.open() as collection:
```
for e in collection.itervalues():
    output("%s: %s (expires %s)\n" % (e['Key'].value, e['Value'].value, str(e['Expires'].value)))

Our function comes in two parts (for reasons that will become clear later). The first function takes an EntitySet object and creates 26 key-value pairs with increasing expiry times.

The main function loads the metadata model, creates the InMemoryEntityContainer object, calls the first function to create the test data and then opens the KeyValueCollection object itself to check that everything is in order. The function output() is just a Python 3 compatibility function (contrast with the builtin 'input') that allows us to write text to standard output. Here's the output from a sample run:

```
>>> import memcache
>>> memcache.test_model()
24: Y (expires 2014-02-17T22:26:21)
20: U (expires 2014-02-17T22:25:41)
22: W (expires 2014-02-17T22:26:01)
23: X (expires 2014-02-17T22:26:11)
 0: A (expires 2014-02-17T22:22:21)
 2: C (expires 2014-02-17T22:22:41)
 4: E (expires 2014-02-17T22:23:01)
 9: J (expires 2014-02-17T22:23:51)
 8: I (expires 2014-02-17T22:23:41)
11: L (expires 2014-02-17T22:24:11)
10: K (expires 2014-02-17T22:24:01)
12: M (expires 2014-02-17T22:24:21)
15: P (expires 2014-02-17T22:24:51)
14: O (expires 2014-02-17T22:24:41)
17: R (expires 2014-02-17T22:25:11)
16: Q (expires 2014-02-17T22:25:01)
```

It is worth pausing briefly here to look at the InMemoryEntityContainer object. When we construct this object we pass in the EntityContainer and it creates all the necessary storage for the EntitySets (and AssociationSets, if required) that it contains. It also binds internal implementations of the EntityCollection object to the model so that, in future, the EntitySet can be opened using the same API described previously in Data Consumers. From this point on we don’t need to refer to the container again as we can just open the EntitySet directly from the model. That object is the heart of our application, blink and you’ve missed it.

**Step 4: Link the Data Source to the OData Server**

OData runs over HTTP so we need to assign a service root URL for the server to run on. We define a couple of constants to help with this:

```
SERVICE_PORT = 8080
SERVICE_ROOT = "http://localhost:%s/" % SERVICE_PORT
```
We're also going to use a separate thread to run the server, a global variable helps here. We're using Python's WSGI interface for the server which requires a callable object to handle requests. The \texttt{Server} object implements callable behaviour to enable this:

```python
import logging, threading
from wsgiref.simple_server import make_server

cache_app = None  #: our Server instance

def run_cache_server():
    """Starts the web server running""
    server = make_server('', SERVICE_PORT, cache_app)
    logging.info("Starting HTTP server on port \%i..." % SERVICE_PORT)
    # Respond to requests until process is killed
    server.serve_forever()

The final part of server implementation involves loading the model, creating the server object and then spawning the server thread:

```python
def main():
    """Executed when we are launched""
    doc = load_metadata()
    InMemoryEntityContainer(doc.root.DataServices['MemCacheSchema.MemCache'])
    server = Server(serviceRoot=SERVICE_ROOT)
    server.set_model(doc)
    # The server is now ready to serve forever
    global cache_app
    cache_app = server
    t = threading.Thread(target=run_cache_server)
    t.setDaemon(True)
    t.start()
    logging.info("MemCache starting HTTP server on \%s" % SERVICE_ROOT)
```

The Server object just takes the serviceRoot as a parameter on construction and has a \texttt{set_model()} method which is used to assign the metadata document to it. That's all you need to do to create it, it uses the same API described in \textit{Data Consumers} to consume the data source and expose it via the OData protocol.

At this stage we can test it via the terminal and a browser:

```
>>> import memcache
>>> memcache.main()
```

At this point the server is running in a separate thread, listening on port 8080. A quick check from the browser shows this to be the case, when I hit \url{http://localhost:8080/KeyValuePairs} Firefox recognises that the document is an Atom feed and displays the feed title. The page source shows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
  <id>http://localhost:8080/KeyValuePairs</id>
  <updated>2014-02-17T22:41:51Z</updated>
  <link href="http://localhost:8080/KeyValuePairs" rel="self"/>
</feed>
```

Looks like it is working!
Step 5: Customise the Server

We don’t need to do much to customise our server, we’ll assume that it is only ever going to be exposed to clients we trust and so authentication is not required or will be handled by some intermediate proxy.

However, we do want to clean up expired entries automatically. Let’s add one last function to our code:

```
CLEANUP_SLEEP=10

def cleanup_forever(mem_cache):
    """Runs a loop continuously cleaning up expired items""
    now = edm.DateTimeValue()
    expires = core.PropertyExpression("Expires")
    t = core.LiteralExpression(now)
    filter = core.BinaryExpression(core.Operator.lt)
    filter.operands.append(expires)
    filter.operands.append(t)
    while True:
        now.set_from_value(iso.TimePoint.from_now_utc())
        logging.info("Cleanup thread running at %s", str(now.value))
        with mem_cache.open() as cacheEntries:
            cacheEntries.set_filter(filter)
            expired_list = list(cacheEntries)
            if expired_list:
                logging.info("Cleaning %i cache entries", len(expired_list))
                for expired in expired_list:
                    del cacheEntries[expired]
                cacheEntries.set_filter(None)
                logging.info("Cleanup complete, %i cache entries remain", len(cacheEntries))
        time.sleep(CLEANUP_SLEEP)
```

This function starts by building a filter expression manually. Filter expressions are just simple trees of expression objects. We start with a PropertyExpression that references a property named Expires and a literal expression with a date-time value. DateTimeValue is just a sub-class of SimpleValue which was introduced in Data Consumers. Previously we’ve only seen simple values that are part of an entity but in this case we create a standalone value to use in the expression. Finally, the filter expression is created as a BinaryExpression using the less than operator and the operands appended. The resulting expression tree looks like this:

Each time around the loop we can just update the value of the literal expression with the current time.

This function takes an EntitySet as a parameter so we can open it to get the collection and then apply the filter. Once filtered, all matching cache entries are loaded into a list before being deleted from the collection, one by one.

Finally, we remove the filter and report the number of remaining entries before sleeping ready for the next run.

We’ll call this function right after main, so we’ve got one thread running the server and the main thread running the cleanup loop.

Now we can test, we start by firing up our server application:

```
$ ./memcache.py
INFO:root:Cleanup thread running at 2014-02-17T23:03:34
INFO:root:Cleanup complete, 0 cache entries remain
```

4.2. OData Providers
Unfortunately, we need more than a simple browser to test the application properly. We want to know that the key value pairs are being created properly and for that we need a client capable of writing to the service. Fortunately, Pyslet has an OData consumer, so we open the interpreter in a new terminal and start interacting with our server:

```python
>>> from pyslet.odata2.client import Client
>>> c = Client("http://localhost:8080/")
```

As soon as we start the client our server registers hits:

```
INFO:root:Cleanup thread running at 2014-02-17T23:06:34
INFO:root:Cleanup complete, 0 cache entries remain
```

Entering the data manually would be tedious but we already wrote a suitable function for adding test data. Because both the data source and the OData client adhere to the same API we can simply pass the EntitySet to our test_data function:

```python
>>> import memcache

>>> memcache.test_data(c.feeds['KeyValuePairs'])
```

As we do this, the server window goes crazy as each of the POST requests comes through:

```
INFO:root:Cleanup thread running at 2014-02-17T23:08:14
INFO:root:Cleanup complete, 0 cache entries remain
127.0.0.1 - - [17/Feb/2014 23:08:23] "POST /KeyValuePairs HTTP/1.1" 201 717
... [and so on]
127.0.0.1 - - [17/Feb/2014 23:08:24] "POST /KeyValuePairs HTTP/1.1" 201 720
INFO:root:Cleanup thread running at 2014-02-17T23:08:24
INFO:root:Cleaning 1 cache entries
INFO:root:Cleanup complete, 19 cache entries remain
127.0.0.1 - - [17/Feb/2014 23:08:24] "POST /KeyValuePairs HTTP/1.1" 201 720
```

We can then watch the data gradually decay as each entry times out in turn. We can easily repopulate the cache, this time let’s catch it in a browser by navigating to:

```
http://localhost:8080/KeyValuePairs('25')?$format=json
```

The result is:

```
```
We can pick the value out directly with a URL like:

http://localhost:8080/KeyValuePairs('25')/Value/$value

This returns the simple string ‘Z’.

Conclusion

It is easy to write an OData server using Pyslet!

4.2.2 A SQL-Backed Data Service

The sample code for this service is in the samples directory in the Pyslet distribution.

This project demonstrates how to construct a simple OData service based on the SQLiteEntityContainer class.

We don’t need any customisations, this class does everything we need ‘out of the box’. Although we use SQLite by default, an implementation is also provided using the MySQLdb adaptor. If you want to use a database other than these you will need to create a new implementation of the generic SQLEntityContainer. See the reference documentation for sqlds for details on what is involved. You shouldn’t have to change much!

Step 1: Creating the Metadata Model

If you haven’t read the Sample Project: InMemory Data Service yet it is a good idea to do that to get a primer on how providers work. The actual differences between writing a SQL-backed service and one backed by the in-memory implementation are minimal. I haven’t repeated code here if it is essentially the same as the code shown in the previous example, but remember that the full working source is available in the samples directory.

For this project, I’ve chosen to write an OData service that exposes weather data for my home town of Cambridge, England. The choice of data set is purely because I have access to over 340,000 data points stretching back to 1995 thanks to the excellent Weather Station website run by the University of Cambridge’s Digital Technology Group: http://www.cl.cam.ac.uk/research/dtg/weather/

We start with our metadata model, which we write by hand. There are two entity sets. The first contains the actual data readings from the weather station and the second contains notes relating to known inaccuracies in the data. I’ve included a navigation property so that it is easy to see which note, if any, applies to a data point.

Here’s the model:

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
 <edmx:DataServices m:DataServiceVersion="2.0">
   <EntityContainer Name="CambridgeWeather" m:IsDefaultEntityContainer="true">
    <EntitySet Name="DataPoints" EntityType="WeatherSchema.DataPoint"/>
    <EntitySet Name="Notes" EntityType="WeatherSchema.Note"/>
    <AssociationSet Name="DataPointNotes" Association="WeatherSchema.DataPointNote"/>
  </Schema>
 </edmx:DataServices>
</edmx:Edmx>
```
I’ve added two feed customisations to this model. The TimePoint field of the data point will be echoed in the Atom ‘updated’ field and the Temperature field will become the Atom title. This will make my OData service more interesting to look at in a standard browser.

As before, we’ll save the model to a file and load it when our script starts up.

To link the model to a SQLite database back-end we need to create an instance of `SQLiteEntityContainer`: 

```xml
<Association Name="DataPointNote">
  <End Role="DataPoint" Type="WeatherSchema.DataPoint" Multiplicity="*"/>
  <End Role="Note" Type="WeatherSchema.Note" Multiplicity="0..1"/>
</Association>
```
def make_container(doc, drop=False, path=SAMPLE_DB):
    if drop and os.path.isfile(path):
        os.remove(path)
    create = not os.path.isfile(path)
    container = SQLiteEntityContainer(
        file_path=path,
    if create:
        container.create_all_tables()

This function handles the only SQL-specific part of our project. When we create a SQLite container we have to pass two keyword arguments: rather than just the container definition as we did for the in-memory implementation. We don’t need to return a value because the SQL implementation is bound to the model that was passed in doc.

The code above automatically creates the tables if the database doesn’t exist yet. This is fine if you are starting from scratch but if you want to expose an existing database you’ll need to work backwards from your existing schema when creating the model. Anyway, letting Pyslet create your SQL tables for you neglects your DBA who will almost certainly want to create indexes to optimise performance and tweak the model to get the best out of your platform. The automatically generated SQL script is supposed to be a starting point, not the complete solution.

For example, the data set I used for this project has over 300,000 records in it. At the end of this exercise I had an OData server capable of serving this information from a SQLite database but example URLs were taking 10s or more on my laptop to load. I created an index on the Temperature column using the SQLite command line and the page load times were instantaneous:

```
sqlite> create index TIndex ON DataPoints(Temperature);
```

## Modelling an Existing Database

For simple data properties it should be fairly easy to map to the EDM. Here is the way Pyslet maps simple types in the EDM to SQL types:

<table>
<thead>
<tr>
<th>EDM Type</th>
<th>SQL Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edm.Binary</td>
<td>BINARY(MaxLength) if FixedLength specified</td>
</tr>
<tr>
<td>Edm.Binary</td>
<td>VARBINARY(MaxLength) if no FixedLength</td>
</tr>
<tr>
<td>Edm.Boolean</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>Edm.Byte</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Edm.DateTime</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>Edm.DateTimeOffset</td>
<td>CHARACTER(20), ISO 8601 string representation is used</td>
</tr>
<tr>
<td>Edm.Decimal</td>
<td>DECIMAL(Precision,Scale), defaults 10,0</td>
</tr>
<tr>
<td>Edm.Double</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Edm.Guid</td>
<td>BINARY(16)</td>
</tr>
<tr>
<td>Edm.Int16</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Edm.Int32</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Edm.Int64</td>
<td>BIGINT</td>
</tr>
<tr>
<td>Edm.SByte</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Edm.Single</td>
<td>REAL</td>
</tr>
<tr>
<td>Edm.String</td>
<td>CHAR(MaxLength) or VARCHAR(MaxLength)</td>
</tr>
<tr>
<td>Edm.String</td>
<td>NCHAR(MaxLength) or NVARCHAR(MaxLength) if Unicode=&quot;true&quot;</td>
</tr>
<tr>
<td>Edm.Time</td>
<td>TIME</td>
</tr>
</tbody>
</table>
Navigation properties, and complex properties do not map as easily but they can still be modelled. To start with, look at the way the SQLite implementation turns our model into a SQL CREATE TABLE statement:

```python
>>> import weather
>>> doc=weather.load_metadata()
>>> weather.make_container(doc)
>>> print dataPoints.create_table_query()[0]
CREATE TABLE "DataPoints" ("TimePoint" TIMESTAMP NOT NULL,
"Temperature" REAL, "Humidity" SMALLINT, "DewPoint" REAL, "Pressure"
SMALLINT, "WindSpeed" REAL, "WindDirection" TEXT, "WindSpeedMax"
REAL, "SunRainStart" REAL, "Sun" REAL, "Rain" REAL,
"DataPointNotes_ID" INTEGER, PRIMARY KEY ("TimePoint"), CONSTRAINT
"DataPointNotes" FOREIGN KEY ("DataPointNotes_ID") REFERENCES
"Notes"("ID")
```

After all the data properties there's an additional property called DataPointNotes_ID which is a foreign key into the Notes table. This was created automatically to model the association set that links the two EntitySets in the container.

Pyslet generates foreign keys for the following types of association:

<table>
<thead>
<tr>
<th>Type</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1 to 1</td>
<td>With UNIQUE and NOT NULL constraints</td>
</tr>
<tr>
<td>* to 1</td>
<td>With a NOT NULL constraint only</td>
</tr>
<tr>
<td>* to 0..1</td>
<td>No additional constraints</td>
</tr>
</tbody>
</table>

When these relationships are reversed the foreign key is of course created in the target table.

What if your foreign key has a different name, say, NoteID? Pyslet gives you the chance to override all name mappings. To fix up this part of the model you need to create a derived class of the base class `SQLEntityContainer` and override the `mangle_name()` method.

In this case, the method would have been called like this:

```python
quotedName=container.mangle_name((u"DataPoints",u"DataPointNotes",u"ID"))
```

There is a single argument consisting of a tuple. The first item is the name of the EntitySet (SQL TABLE) and the subsequent items complete a kind of 'path' to the value. Foreign keys have a path comprising of the AssociationSet name followed by the name of the key field in the target EntitySet. The default implementation just joins the path with an underscore character. The method must return a suitably quoted value to use for the column name. To complete the example, here is how our subclass might implement this method to ensure that the foreign key is called 'NoteID' instead of 'DataPointNotes_ID':

```python
def mangle_name(self,source_path):
    if source_path==(u"DataPoints",u"DataPointNotes",u"ID"):
        return self.quote_identifier(u'NoteID')
    else:
        return super(MyCustomerContainer,self).mangle_name(source_path)
```

You may be wondering why we don’t expose the foreign key field in the model. Some libraries might force you to expose the foreign key in order to expose the navigation property but Pyslet takes the opposite approach. The whole point of navigation properties is to hide away details like foreign keys. If you really want to access the value you can always use an expansion and select the key field in the target entity. Exposing it in the source entity just tempts you in to writing code that ‘knows’ about your model for example, if we had exposed the foreign key in our example as a simple property we might have been tempted to do something like this:
When we should be doing something like this:

```python
note = data_point['Note'].get_entity()
if note is not None:
    # do something with the note
```

Complex types are handled in the same way as foreign keys, the path being comprised of the name(s) of the complex field(s) terminated by the name of a simple property. For example, if you have a complex type called Address and two properties of type Address called “Home” and “Work” you might end up with SQL that looked like this:

```sql
CREATE TABLE Employee (  ...
    Home_Street NVARCHAR(50),
    Home_City NVARCHAR(50),
    Home_Phone NVARCHAR(50),
    Work_Street NVARCHAR(50),
    Work_City NVARCHAR(50),
    Work_Phone NVARCHAR(50)
    ...
)
```

You often see SQL written like this anyway so if you want to tweak the mapping to put a Complex type in your model you can.

Finally, we need to deal with the symmetric relationships, 1 to 1 and * to *. These are modelled by separate tables. 1 to 1 relationships are best avoided, the advantages over combining the two entities into a single larger entity are marginal given OData’s $select option which allows you to pick a subset of the fields anyway. If you have them in your SQL schema already you might consider creating a view to combine them before attempting to map them to the metadata model.

Either way, both types of symmetric relationships get mapped to a table with the name of the AssociationSet. There are two sets of foreign keys, one for each of the EntitySets being joined. The paths are rather complex and are explained in detail in `SQLAssociationCollection`.

**Step 2: Test the Model**

Before we add the complication of using our model with a SQL database, let’s test it out using the same in-memory implementation we used before:

```python
def dry_run():
    doc = load_metadata()
    container = InMemoryEntityContainer(doc.root.DataServices['WeatherSchema.CambridgeWeather'])
    load_data(weather_data, SAMPLE_DIR)
    load_notes(weather_notes, 'weathernotes.txt', weather_data)
```

SAMPLE_DIR here is the name of a directory containing data from the weather station. The implementation of the load_data function is fairly ordinary, parsing the daily text files from the station and adding them to the DataPoints entity set.
The implementation of the load_notes function is more interesting as it demonstrates use of the API for binding entities together using navigation properties:

```python
def load_notes(weather_notes, file_name, weatherData):
    with open(file_name, 'r') as f:
        id = 1
        with weather_notes.open() as collection, weatherData.open() as data:
            while True:
                line = f.readline()
                if len(line) == 0:
                    break
                elif line[0] == '#':
                    continue
                noteWords = line.split()
                if noteWords:
                    note = collection.new_entity()
                    note['ID'].set_from_value(id)
                    start = iso.TimePoint(
                        date=iso.Date.from_str(noteWords[0]),
                        time=iso.Time(hour=0, minute=0,
                        →second=0))
                    note['StartDate'].set_from_value(start)
                    end = iso.TimePoint(
                        date=iso.Date.from_str(noteWords[1]).
                        →offset(days=1),
                        →second=0))
                    note['EndDate'].set_from_value(end)
                    note['Details'].set_from_value(string.
                        →join(noteWords[2:], ', '))
                    collection.insert_entity(note)
                    # now find the data points that match
                    data.set_filter(core.CommonExpression.from_
                        →str("TimePoint ge datetime'"%s
                        " and TimePoint lt datetime'"%s")%(unicode(start),
                        →unicode(end))))
                    for data_point in data.values():
                        data_point['Note'].bind_entity(note)
                        data.update_entity(data_point)
                        id = id + 1
        with weather_notes.open() as collection:
            collection.set_orderby(core.CommonExpression.orderby_from_str(
                'StartDate desc'))
            for e in collection.itervalues():
                with e['DataPoints'].open() as affectedData:
                    print "%s-%s: %s (%i data points affected)"
                    % (unicode(e['StartDate'].value),
                    →value, len(affectedData)),
```

The function opens collections for both Notes and DataPoints. For each uncommented line in the source file it creates a new Note entity, then, it adds a filter to the collection of data points that narrows down the collection to all the data points affected by the note and then iterates through them binding the note to the data point and updating the entity (to commit the change to the data source). Here’s a sample of the output on a dry-run of a small sample of the data from November 2007:

```
2007-12-25T00:00:00-2008-01-03T00:00:00: All sensors inaccurate (0 data points affected)
2007-11-01T00:00:00-2007-11-23T00:00:00: rain sensor over reporting rainfall following malfunction (49 data points affected)
```

Chapter 4. The Open Data Protocol (OData)
You may wonder why we use the `values` function, rather than `itervalues` in the loop that updates the data points. `itervalues` would certainly have been more efficient but, just like native Python dictionaries, it is a bad idea to modify the data source when iterating as unpredictable things may happen. The concept is extended by this API to cover the entire container: a thread should not modify the container while iterating through a collection.

Of course, this API has been designed for parallel use so there is always the chance that another thread or process is modifying the data source outside of your control. Behaviour in that case is left to be implementation dependent - storage engines have widely differing policies on what to do in these cases.

If you have large amounts of data to iterate through you should consider using `list(collection.iterpage(True))` instead. For a SQL data source this has the disadvantage of executing a new query for each page rather than spooling data from a single SELECT but it provides control over page size (and hence memory usage in your client) and is robust to modifications.

As an aside, if you change the call from `values` to `itervalues` in the sample you may well discover a bug in the SQLite driver in Python 2.7. The bug means that a commit on a database connection while you are fetching data on another cursor causes subsequent data access commands to fail. It's a bit technical, but the details are here: [http://bugs.python.org/issue10513](http://bugs.python.org/issue10513)

Having tested the model using the in-memory provider we can implement a full test using the SQL back-end we created in `make_container` above. This test function prints the 30 strongest wind gusts in the database, along with any linked note:

```python
def test_model(drop=False):
    doc=load_metadata()
    container=make_container(doc,drop)
    if drop:
        load_data(weatherData,SAMPLE_DIR)
        load_notes(weather_notes,'weathernotes.txt',weatherData)
    with weatherData.open() as collection:
        collection.set_orderby(core.CommonExpression.orderby_from_str('WindSpeedMax desc'))
        collection.set_page(30)
        for e in collection.iterpage():
            note=e['Note'].get_entity()
            if e['WindSpeedMax'] and e['Pressure']:
                print "%s: Pressure %i mb, max wind speed %0.1f knots (%0.1f mph); %s"%(unicode(e['TimePoint'].value),
                e['Pressure'].value,e['WindSpeedMax'].value*1.15078,
                note['Details'] if note is not None else '')
```

Here's some sample output:

```
>>> weather.test_model()
2002-10-27T10:30:00: Pressure 988mb, max wind speed 74.0 knots (85.2 mph);
2004-03-20T15:30:00: Pressure 993mb, max wind speed 72.0 knots (82.9 mph);
2007-01-18T14:30:00: Pressure 984mb, max wind speed 70.0 knots (80.6 mph);
... [ and so on ]
...
2007-01-11T10:30:00: Pressure 998mb, max wind speed 58.0 knots (66.7 mph);
2007-01-18T07:30:00: Pressure 980mb, max wind speed 58.0 knots (66.7 mph);
1996-02-18T04:30:00: Pressure 998mb, max wind speed 56.0 knots (64.4 mph); humidity and dewpoint readings may be inaccurate, particularly high humidity readings
```

4.2. OData Providers
2002-10-27T13:00:00: Pressure 996mb, max wind speed 56.0 knots (64.4 mph);
2004-01-31T17:30:00: Pressure 983mb, max wind speed 56.0 knots (64.4 mph);

Notice that the reading from 1996 has a related note.

**Step 4: Link the Data Source to the OData Server**

This data set is designed to be updated by some offline process that polls the weather station for the latest readings and adds them to the database behind the scenes. Unlike the memory-cache example, the OData interface should be read-only so we use the `ReadOnlyServer` sub-class of the OData server:

```python
def run_weather_server(weather_app=None):
    """Starts the web server running""
    server=make_server('',SERVICE_PORT,weather_app)
    logging.info("HTTP server on port %i running"%SERVICE_PORT)
    # Respond to requests until process is killed
    server.serve_forever()

def main():
    """Executed when we are launched""
    doc=load_metadata()
    container=make_container(doc)
    server=ReadOnlyServer(serviceRoot=SERVICE_ROOT)
    server.SetModel(doc)
    t=threading.Thread(target=run_weather_server,kwargs={'weather_app':server})
    t.setDaemon(True)
    t.start()
    logging.info("Starting HTTP server on %s"%SERVICE_ROOT)
    t.join()
```

Once the script is running we test in a browser. I’ve loaded the full data set into the server, how many data points? Here’s how we can find out, in our browser we go to:

```
http://localhost:8080/DataPoints/$count
```

The result is 325213. Firefox recognises that the feeds are in Atom format and renders the feed customisations we made earlier.
When we access this page with logging turned up to INFO we get the following output on the console, interspersed with the simple HTTP server output:

```
INFO:root:SELECT COUNT(*) FROM "DataPoints"; []
127.0.0.1 - - [21/Feb/2014 22:57:01] "GET /DataPoints/$count HTTP/1.1" 200 6
→"WindSpeed", "WindDirection", "WindSpeedMax", "SunRainStart", "Sun", "Rain", 
→"Temperature" AS o_1, "TimePoint" FROM "DataPoints" ORDER BY o_1 DESC, "TimePoint" 
→ASC; []
→$top=30 HTTP/1.1" 200 31006
```

You may wonder what those square brackets are doing at the end of the SQL statements. They’re actually used for logging the parameter values when the query has been parameterised. If we add a filter you’ll see what they do:

```
```

And here’s the output on the console:

```
→"WindSpeed", "WindDirection", "WindSpeedMax", "SunRainStart", "Sun", "Rain", 
→"Temperature" AS o_1, "TimePoint" FROM "DataPoints" WHERE ("Temperature" > ?) ORDER 
→BY o_1 DESC, "TimePoint" ASC; [-100]
127.0.0.1 - - [21/Feb/2014 16:35:09] "GET /DataPoints?$filter=Temperature%20gt%20-100&
→$orderby=Temperature%20desc&$top=30 HTTP/1.1" 200 31006
```

Yes, all Pyslet queries are fully parameterized for security and performance!
4.2.3 Sample Project: Custom Data Service

The sample code for this service is in the samples/fsodata directory in the Pyslet distribution: fsodata.py

This project demonstrates how to construct a simple OData service based on a custom EntityContainer class. It also demonstrates how to handle media streams in your own data sources.

Although OData is often talked about as the ODBC of the web there is no reason why your data has to be in a database format to be exposed by OData...

Step 0: Create the DAL implementation

If your data source is in a general form then you will want to create general classes derived from `pyslet.odata2.core.EntityCollection` and `pyslet.odata2.core.NavigationCollection`. For example, suppose you want to expose data stored in a ‘Unix’ database accessed using one of Python’s dbm modules. You could write a general implementation that maps this DAL API to the dbm interface. This is similar to the approach taken with the SQL classes, they are written using Python’s DB API enabling a wide variety of SQL databases to be exposed through OData with little or no extra work required for a specific data set.

On the other hand, if your datasource is fairly specific to a particular application you might create specific implementations of these classes that are tied to the entities in your model.

In this project, we’ll take the latter approach and so defer discussion of the implementation details until we’ve constructed the model.

Step 1: Creating the Metadata Model

For small amounts of data, the basic OData classes already supplied do almost everything you need. In this example we’ll expose information about the files and directories in a designated part of the file system for an application like a blog or a simple file sharing site. We’ll assume that there aren’t too many files and that walking the tree is a relatively painless operation to perform.

As before, we start with our metadata model, which we write by hand. There is just one entity set: Files. It has two navigation properties that are defined by a single parent/child association.

Here’s the model:

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<edmx:Edmx Version="1.0"
 <edmx:DataServices m:DataServiceVersion="2.0">
  <Schema Namespace="FSSchema"
   xmlns="http://schemas.microsoft.com/ado/2006/04/edm">
   <EntityContainer Name="FS" m:IsDefaultEntityContainer="true">
    <EntitySet Name="Files" EntityType="FSSchema.File"/>
    <AssociationSet Name="Directories"
     Association="FSSchema.Directory">
     <End Role="Parent" EntitySet="Files"/>
     <End Role="Child" EntitySet="Files"/>
    </AssociationSet>
   </EntityContainer>
   <EntityType Name="File" m:HasStream="true">
    <Key>
     <PropertyRef Name="path"/>
    </Key>
    <Property Name="path" Type="Edm.String" Nullable="false"/>
   </EntityType>
  </Schema>
 </edmx:DataServices>
</edmx:Edmx>
```
I've added two feed customisations to this model. The last modified date of the file will be echoed in the Atom 'updated' field and the file’s name will become the Atom title. This will make my OData service more interesting to look at in a standard browser.

Finally, we want to actually download these files so I've added the HasStream attribute to the EntityType declaration. The idea is that using the $value path option in the URL will allow you to download the contents of the file.

As before, we'll save the model to a file and load it when our script starts up. This model is fsschema.xml in the samples directory.

**Step 0: Revisited**

Now we have our metadata model specified we can start implementing the classes that will enable it. The keys in our entities are pseudo-paths to the files within a special directory using `/` as a separator, for example `/dirA/dirB/file.txt`.

We start with a constant to specify the BASE_PATH and two functions, one that turns our path 'keys' into file-system absolute paths and one that reverses the transformation. I won’t repeat the code for these functions here as they can be found in the sample code under the names fspath_to_path and path_to_fspath, but their main job is to ensure that symbolic links and all files and directories with names starting '.' are hidden from the service and that no nefarious OData queries can circumvent the restrictions on the exposed directory.

Given an absolute file system path we can now write a function that will fill in the details for an entity. Notice the last thing it does is set the entity’s `exists` flag to True indicating that the entity represents a real object in our exposed directory:

```python
def fspath_to_entity(fspath, e):
    path = fspath_to_path(fspath)
```

4.2. OData Providers
Pyslet Documentation, Release 0.7.20180305

e['path'].set_from_value(path)
if path == '/':
e['name'].set_from_value('/')
else:
e['name'].set_from_value(path.split('/')[-1])
if os.path.isfile(fspath):
e['isDirectory'].set_from_value(False)
try:
info = os.lstat(fspath)
e['size'].set_from_value(info.st_size)
e['lastAccess'].set_from_value(info.st_atime)
e['lastModified'].set_from_value(info.st_mtime)
except IOError:
# just leave the information as NULLs
pass
elif os.path.isdir(fspath):
e['isDirectory'].set_from_value(True)
else:
raise ValueError
e.exists = True

Armed with this utility function we derive a class from pyslet.odata2.core.EntityCollection and bind
it to our metadata model when the script starts up. We’ll look at the details of this class later but let’s start with the
declaration:
import pyslet.odata2.core as odata
class FSCollection(odata.EntityCollection):
""" this is our custom collection class
... more details below"""

Let’s look at the first part of the load_metadata function which is called on script start-up:
import pyslet.odata2.metadata as edmx
def load_metadata(
path=os.path.join(os.path.split(__file__)[0], 'fsschema.xml')):
"""Loads the metadata file from the script directory."""
doc = edmx.Document()
with open(path, 'rb') as f:
doc.read(f)
# next step is to bind our model to it
container = doc.root.DataServices['FSSchema.FS']
container['Files'].bind(FSCollection)
# ... more initialisation stuff here

The critical step here is the last line where we bind our custom collection class to the ‘Files’ entity set. From this point
on, calls to the DAL API for the File entity set will be routed to our collection class, not the default implementation.
What do we need to do to handle them?
Writing our Custom Entity Collection
The basic pyslet.odata2.csdl.EntityCollection class documents the key methods we must override.
Our implementation is made a little simpler because we don’t need to override the __init__ method. In fact, it is
enough to override just a single method to get our custom provider working: itervalues. There’s a catch though,
itervalues must iterate through all the entities in the collection honouring any filter, ordering and expand rules that are
152

Chapter 4. The Open Data Protocol (OData)


in effect. This sounds like a lot of work but the basic implementation has helper methods that can be used to wrap a simpler implementation.

We start by defining a generator function that yields all the entities in the collection, in no particular order:

```python
def generate_entities(self):
    """List all the files in our file system

    The first item yielded is a dummy value with path /"
    e = self.new_entity()
    e['path'].set_from_value('/')
    e['name'].set_from_value('/')
    e['isDirectory'].set_from_value(True)
    e.exists = True
    yield e
    for dirpath, dirnames, filenames in os.walk(BASE_PATH):
        for d in dirnames:
            fspath = os.path.join(dirpath, d)
            e = self.new_entity()
            try:
                fspath_to_entity(fspath, e)
                yield e
            except ValueError:
                # unexpected but ignore
                continue
        for f in filenames:
            fspath = os.path.join(dirpath, f)
            e = self.new_entity()
            try:
                fspath_to_entity(fspath, e)
                yield e
            except ValueError:
                # unexpected but ignore
                continue
```

We use the built-in os.walk generator and the helper function fspath_to_entity that we defined earlier. Notice how we use the `new_entity()` method to create an instance and then pass it to fspath_to_entity to get it filled in with the details. The first entity, corresponding to the root of our exposed directory, is created by hand for simplicity.

We can now use this generator, combined with the wrapper methods defined by the base class for `itervalues`:

```python
def itervalues(self):
    return self.order_entities(
        self.expand_entities(self.filter_entities(
            self.generate_entities())))
```

Our generator function is passed to filter_entities which iterates through our generator yielding only the entities that match the filter. Similarly, this filtered iterable is then iterated by the expand_entities method to implement the expand and select rules. Finally, the resulting generator is wrapped by the order_entities method which sorts them according to the orderby rules. This last step does nothing if there is no orderby option in effect but if there is it is a bit wasteful because the iterator will be turned into a list before it is sorted, causing all entities to be loaded into memory. See Big vs Small Data for advice on dealing with this issue.

With `itervalues` defined our provider should now be working. The navigation properties are not bound yet so they’ll yield nothing but the basic Files feed should be returning all the eligible files in the BASE_PATH directory.

Before we pack up and commit our changes though we need to revisit the advice in the base class. Although functional, our collection is very inefficient when someone uses direct key lookup. Essentially, we’re iterating through the entire collection every time, just to find a matching key. We SHOULD override `__getitem__()` to improve our code:
def __getitem__(self, path):
    """Get just a single file, by path""
    try:
        fspath = path_to_fspath(path)
        e = self.new_entity()
        fspath_to_entity(fspath, e)
        if self.check_filter(e):
            if self.expand or self.select:
                e.expand(self.expand, self.select)
                return e
            else:
                raise KeyError("Filtered path: %s" % path)
        except ValueError:
            raise KeyError("No such path: %s" % path)
    except ValueError:
        raise KeyError("No such path: %s" % path)

The code is pretty simple, we convert the path 'key' into a full file system path and then return just that entity. Our path_to_fspath method takes care of raising KeyError for us if the path doesn’t correspond to an object that exists in the directory we’re exposing. fspath_to_entity raises ValueError if the file system path turns out not to belong to a regular file or directory so we catch this and raise KeyError there too.

Notice that the value returned by key lookup must still honour any filter in place. We use the base class method check_filter to help us implement this requirement. Similarly for set_expand.

The final suggestion for improvement is to override the __len__ method in order to provide a more efficient implementation for determining the number of entities in the collection. Unfortunately, in this case we don’t really have a better method than iterating through them all so we skip that part.

### Dealing With Navigation

To make our example more interesting, I’ve defined two navigation properties that enable you to use OData to traverse the file system by navigating up to a File’s parent directory or down to the files and sub-directories it contains. The implementations are similar but we have to define two separate classes derived from pyslet.odata2.core.NavigationCollection and we have to use the attribute from_entity which contains the entity we are navigating from:

class FSChildren(odata.NavigationCollection):
    # itervalues defined as before

    def generate_entities(self):
        """List all the children of an entity""
        path = self.from_entity['path'].value
        fspath = path_to_fspath(path)
        if os.path.isdir(fspath):
            for filename in os.listdir(fspath):
                child_fspath = os.path.join(fspath, filename)
                try:
                    e = self.new_entity()
                    fspath_to_entity(child_fspath, e)
                    yield e
                except ValueError:
                    # skip this one
                    continue

    # __getitem__ omitted for brevity...
Notice in the second class that navigation properties are always defined in terms of collections, even if they are only supposed to yield a maximum of one item as is the case here with navigation to the parent directory.

To make these navigation classes active we have to bind them in a similar way to the way we bound the main collection class, here’s the rest of the load_metadata function we defined earlier:

```python
container['Files'].bind_navigation('Files', FSChildren)
container['Files'].bind_navigation('Parent', FSParent)
```

### Adding Support for Streams

To access the contents of the file we need to implement support for the stream methods on the base collection. These methods are only supported (and needed) on base collections, not on navigation collections. As a result, we’ll add them to our FSCollection class.

To support reading streams you need to support two new methods, read_stream and read_stream_close. These methods are very similar, they just provide different approaches to obtaining the data. read_stream pushes the data by writing it to a file you pass in as a parameter and read_stream_close pulls the stream, returning a generator that iterates over the data and closing the collection when the iteration terminates. This second form is used by the OData server as it is more compatible with the way the WSGI framework expects to consume data.

The stream methods use a very simple class `StreamInfo` to return some basic information about the stream such as the content type, the size and modification time. The content type is required, everything else is optional:

```python
def _get_path_info(self, path):
    try:
        e = self[path]
        fspath = path_to_fspath(path)
        if os.path.isdir(fspath):
```
This method returns a tuple of the native file system path and the basic information about the stream. For directories, we return a zero-length text/plain stream, for files we use an internally defined map_extension function to look up the file extension in a simple dictionary.

The type is an instance of `pyslet.http.params.MediaType` which is a class wrapper for content types, you can create you own very simply by passing the type and subtype as strings:

```python
type = params.MediaType('image', 'gif')
```

or, if you have untrusted input, by creating an instance from a string:

```python
type = params.MediaType.from_str('text/html; name=index.htm; charset=utf-8')
```

To generate the data we use another private method:

```python
def _generate_file(self, fspath, close_it=False):
    try:
        with open(fspath, 'rb') as f:
            data = ''
            while True:
                data = f.read(io.DEFAULT_BUFFER_SIZE)
                if not data:
                    # EOF
                    break
                else:
                    yield data
    finally:
        if close_it:
            self.close()
```

This is a generator method that yields the data in chunks. When the iteration is complete (or destroyed) the collection can be closed and cleaned up automatically by passing True for close_it.

Armed with these two methods we can finish our implementation by providing implementations of the two required methods for media stream support:

```python
def read_stream(self, path, out=None):
    fspath, sinfo = self._get_path_info(path)
    if out is not None and sinfo.size:
```
Step 2: Test the Model

Testing our model is fairly easy, I loaded a couple of files and a directory into the BASE_PATH and then ran this session from the interpreter:

```python
>>> from pyslet.py2 import output
>>> import fsodata
doc = fsodata.load_metadata()
container = doc.root.DataServices['FSSchema_FS']
collection = container['Files'].open()
>>> for path in collection: output(str(path) + "\n")
... /dtest/tmp.txt
>>> for f in collection.itervalues():
...     print f['path'].value, str(f['lastModified'].value)
... / None
/dtest None
/tmp.txt 2014-07-29T10:02:21
/dtest/tmp.txt 2014-07-29T10:23:18
>>> info, gen = collection.read_stream_close('/tmp.txt')
>>> info.size
6
>>> str(info.type)
'text/plain'
>>> for data in gen: output(data.decode('ascii'))
... Hello
```

Step 3: Link the Data Source to the OData Server

This step is almost identical to previous examples.

Once the script is running we can test in a browser:
Note: Sharp eyed readers might notice the difference in the time values displayed by the browser and those displayed by the interpreter session above. It is is worth drilling down a little into EDM’s DateTime type to explain. This type has fallen out of favour in OData version 4 but the idea of storing a date time value in an unspecified local time can be meaningful, even if the UTC time it represents varies depending on the location, daylight savings and so on. Indeed, this abstract concept is the one we use in day-to-day life all the time!

In this case, the dates returned by os.stat are elapsed seconds from the epoch, they are not really expressed in any particular time zone but bear in mind that their meaning doesn’t change when the clocks change. This elapsed time is passed directly to the DateTime class where it is treated as a ‘unix’ time, in effect ensuring that our lastModified dates are always stored in UTC (but with no explicit UTC offset).

The catch comes when we publish our information as an Atom feed using OData. There’s a slight oversight in the OData specification here because Atom insists that the updated time of an entry has a date with a timezone. When serialising the entity in Atom format Pyslet assumes that DateTime values are in UTC (which is correct in this case). Firefox, when it renders the feed, is smart enough to convert these updated times into local times for my system (which at the time was running in UTC+01:00).
Big vs Small Data

Real applications will probably want to expose more data than our simple example. How you do this depends on your data source. The worst case scenario for the implementation shown here is the use of orderby. When orderby is in effect all entities are iterated over and cached in memory before being sorted. A close second is a filter that misses all or most entities in a collection as, again, these filters will cause our method to iterate through all the entities even if iterpage is used to implement restrictions on the amount of data returned.

If your data source has its own query language then you should consider writing something that translates the OData query into the query language of your data source. This is the approach taken by the SQL-based examples.

If, on the other hand, your data source doesn’t have a good query language then you could expose it using a minimal OData implementation (such as the one given here) and then use the same schema to create a SQL-backed service. Pulling the data from your data source through the API and pushing it into the SQL-backed service would be fairly trivial and could be done as a periodic synchronization process. This works even better if you have a last modified field on your entities that you can use to filter out the unchanged ones, as our simple implementation of itervalues won’t cause the collection to be loaded into memory for a filter alone.

Finally, if periodic synchronization is not good enough to reflect the dynamic nature or your (unqueryable) data source then you will want to think about some type of intelligent caching to reduce the impact of worst case OData queries. You might think about simply disabling $orderby and $filter options (which is perfectly OK in OData). You can do that by overriding the set_orderby() and set_filter() methods, raising NotImplementedError.

4.2.4 Which DAL Implementation?

Transient Data

If your data is relatively small and transient then you could use the in memory implementation of the DAL API directly. This is the easiest route to creating a new OData provider as you won’t need to override any of the implementations.

Look at the example project Sample Project: InMemory Data Service to see how easy it is to create a useful in-memory key-value store.

SQL

If your data is currently in a SQL database, or if you intend to write a read-only data source and you could easily put your data into a SQL database, then you should use the Python DB ABI-based implementation as a starting point.

If your data is in a database other than a SQLite database you will have to provide a few tweaks by deriving a new class from SQLEntityContainer. This can’t be helped, the DB API does a good job at dealing with most issues, such as variation in parameterization conventions and expected data types, but SQL connection parameters and the occasional differences in the SQL syntax mean there is likely to be a small amount of work to do.

A look at the customisations required for SQLiteEntityContainer where a handful of methods have had to be overridden should point the way. You may want to override the default SQLEntityCollection object too where functions and operators from the the expression language can be mapped on to parameterized SQL queries.

Once you have a class that can connect to your chosen database move on to A SQL-Backed Data Service.

Customer Provider

Writing a customer provider isn’t as hard as you might think, provided your data set is of a manageable size then you can use the built-in behaviour of the base classes to take care of almost all the API’s needs. You just need to expose the entity values themselves by implementing a couple of methods!
Look at the example project Sample Project: Custom Data Service to see how you can write a simple application that exposes a download-directory to the web using OData (providing a little more metadata than is easily obtainable from plain HTTP.)

**An OData Proxy**

Finally, the OData client implementation of the DAL API opens the possibility of writing an OData proxy server. Why would you do this?

One of the big challenges for the OData protocol is web-security in the end user’s browser. By supporting JSON over the wire OData sends out a clear signal that using it directly from a Javascript on a web page should be possible. But in practice, this only works well for unauthenticated (and hence read-only) OData services. If you want to write more exciting applications you leave yourself open to all manner of browser-based attacks that could expose your data to unauthorised bad guys. To mitigate these risks browsers are increasingly locking down the browser to make it harder for cross-site exploits to happen, which is a good thing. The downside is that it makes it harder for your web-application to talk to an OData server unless they are both hosted on the same domain.

An OData proxy can be co-located with your application to overcome this problem. A dumb proxy is probably best implemented by the web-server, rather than a full-blown web application but the classes defined in this package are a good starting point for writing a more intelligent proxy such as one that checks for a valid session with your application before proxying the request.

The implementation isn’t trivial because the identities of the entities created by the client (as reported by get_location()) are the URLs of the entities as they appear in the remote data service whereas the OData proxy needs to serve up entities with identities with URLs that appear under its service root. As a result, you need to create a copy of the client’s model and implement proxy classes that implement the API by pulling and pushing entities into the client. This isn’t as much work as it sounds and you probably want to do it anyway so that your proxy can add value, such as hiding parts of the model that shouldn’t be proxied, adding constraints for authorisation, etc.

I’m the process of developing a set of proxy classes to act as a good starting point for this type of application. Watch this space, or reach out to me via the Pyslet home page.

### 4.3 OData Reference

The basic API for the DAL is defined by the Entity Data Model (EDM) defined in pyslet.odata2.csdl, which is extended by some core OData-specific features defined in pyslet.odata2.core and pyslet.odata2.metadata. With these three modules it is possible to create derived classes that implement the Data Access Layer API in a variety of different storage scenarios.

#### 4.3.1 Entity Data Model (EDM)

This module defines functions and classes for working with data based on Microsoft’s Entity Data Model (EDM) as documented by the Conceptual Schema Definition Language and associated file format: http://msdn.microsoft.com/en-us/library/dd541474.aspx

The classes in this model fall in to two categories. The data classes represent the actual data objects, like simple and complex values, entities and collections. The metadata classes represent the elements of the metadata model like entity types, property definitions, associations, entity sets and so on. The metadata elements have direct XML representations, the data classes do not.
Data Model

```python
class pyslet.odata2.csdl.EntityCollection(entity_set, **kwargs)
Bases: pyslet.odata2.csdl.DictionaryLike, pyslet.pep8.PEP8Compatibility
```

Represents a collection of entities from an `EntitySet`.

To use a database analogy, `EntitySet`’s are like tables whereas `EntityCollections` are more like the database cursors that you use to execute data access commands. An entity collection may consume physical resources (like a database connection) and so should be closed with the `close()` method when you’re done.

Entity collections support the context manager protocol in python so you can use them in with statements to make clean-up easier:

```python
with entity_set.open() as collection:
    if 42 in collection:
        print "Found it!"
```

The close method is called automatically when the with statement exits.

Entity collections also behave like a python dictionary of `Entity` instances keyed on a value representing the Entity’s key property or properties. The keys are either single values (as in the above code example) or tuples in the case of compound keys. The order of the values in the tuple is taken from the order of the `PropertyRef` definitions in the metadata model. You can obtain an entity’s key from the `Entity.key()` method.

When an `EntityCollection` represents an entire entity set you cannot use dictionary assignment to modify the collection. You must use `insert_entity()` instead where the reasons for this restriction are expanded on.

For consistency with python dictionaries the following statement is permitted, though it is effectively a no-operation:

```python
etColl[key]=entity
```

The above statement raises `KeyError` if `entity is not` a member of the entity set. If `key` does not match the entity’s key then `ValueError` is raised.

Although you can’t add an entity with assignment you can delete an entity with the delete operator:

```python
del etColl[key]
```

Deletes the entity with `key` from the entity set.

These two operations have a different meaning when a collection represents the subset of entities obtained through navigation. See `NavigationCollection` for details.

**Notes for data providers**

Derived classes MUST call super in their `__init__` method to ensure the proper construction of the parent collection class. The proper way to do this is:

```python
class MyCollection(EntityCollection):
    def __init__(self, paramA, paramsB, **kwargs):
        # paramA and paramB are examples of how to
        # consume private keyword arguments in this
        # method so that they aren't passed on to the
        # next __init__
        super(MyCollection, self).__init__(**kwargs)
```

All collections require a named `entity_set` argument, an `EntitySet` instance from which all entities in the collection are drawn.
Derived classes MUST also override `itervalues()`. The implementation of `itervalues` must return an iterable object that honours the value of the expand query option, the current filter and the orderby rules.

Derived classes SHOULD also override `__getitem__()` and `__len__()` as the default implementations are very inefficient, particularly for non-trivial entity sets.

Writeable data sources must override `__delitem__`.

If a particular operation is not supported for some data-service specific reason then `NotImplementedError` must be raised.

Writeable entity collections SHOULD override `clear()` as the default implementation is very inefficient.

```python
entity_set = None
    the entity set from which the entities are drawn
expand = None
    the expand query option in effect
select = None
    the select query option in effect
filter = None
    a filter or None for no filter (see `check_filter()`)
orderby = None
    a list of orderby rules or None for no ordering
skip = None
    the skip query option in effect
top = None
    the top query option in effect
topmax = None
    the provider-enforced maximum page size in effect
inlinecount = None
    True if inlinecount option is in effect

The inlinecount option is used to alter the representation of the collection and, if set, indicates that the `__len__` method will be called before iterating through the collection itself.
```

get_location()
    Returns the location of this collection as a `URI` instance.
    By default, the location is given as the location of the `entity_set` from which the entities are drawn.

get_title()
    Returns a user recognisable title for the collection.
    By default this is the fully qualified name of the entity set in the metadata model.

```

set_expand(expand, select=None)
    Sets the expand and select query options for this collection.
    The expand query option causes the named navigation properties to be expanded and the associated entities to be loaded in to the entity instances before they are returned by this collection.

expand
    A dictionary of expand rules. Expansions can be chained, represented by the dictionary entry also being a dictionary:
```
The select query option restricts the properties that are set in returned entities. The select option is a similar dictionary structure, the main difference being that it can contain the single key ‘*’ indicating that all data properties are selected.

**select_keys()**
Sets the select rule to select the key property/properties only.
Any expand rule is removed.

**expand_entities(entity_iterable)**
Utility method for data providers.
Given an object that iterates over all entities in the collection, returns a generator function that returns expanded entities with select rules applied according to expand and select rules.
Data providers should use a better method of expanded entities if possible as this implementation simply iterates through the entities and calls `Entity.expand()` on each one.

**set_filter(filter)**
Sets the filter object for this collection.
See `check_filter()` for more information.

**filter_entities(entity_iterable)**
Utility method for data providers.
Given an object that iterates over all entities in the collection, returns a generator function that returns only those entities that pass through the current filter object.
Data providers should use a better method of filtering entities if possible as this implementation simply iterates through the entities and calls `check_filter()` on each one.

**check_filter(entity)**
Checks `entity` against the current filter object and returns True if it passes.
This method is really a placeholder. Filtering is not covered in the CSDL model itself but is a feature of the OData `pyslet.odata2.core` module.
See `pyslet.odata2.core.EntityCollectionMixin.check_filter()` for more. The implementation in the case class simply raises `NotImplementedError` if a filter has been set.

**set_orderby(orderby)**
Sets the orderby rules for this collection.

**orderby** A list of tuples, each consisting of:

```
(an order object as used by :py:meth:`calculate_order_key`, 1 | -1 )
```

**calculate_order_key(entity, order_object)**
Given an entity and an order object returns the key used to sort the entity.
This method is really a placeholder. Ordering is not covered in the CSDL model itself but is a feature of the OData `pyslet.odata2.core` module.
See `pyslet.odata2.core.EntityCollectionMixin.calculate_order_key()` for more. The implementation in the case class simply raises `NotImplementedError`.
order_entities (entity_iterable)
Utility method for data providers.

Given an object that iterates over the entities in random order, returns a generator function that returns the same entities in sorted order (according to the orderby object).

This implementation simply creates a list and then sorts it based on the output of calculate_order_key() so is not suitable for use with long lists of entities. However, if no ordering is required then no list is created.

set_inlinecount (inlinecount)
Sets the inline count flag for this collection.

new_entity()
Returns a new py:class:Entity instance suitable for adding to this collection.

The data properties of the entity are set to null, not to their default values, even if the property is marked as not nullable.

The entity is not considered to exist until it is actually added to the collection. At this point we deviate from dictionary-like behaviour. Instead of using assignment you must call insert_entity():

```python
e=collection.new_entity()
e["ID"]=1000
e["Name"]="Fred"
assert 1000 not in collection
collection[1000]=e # raises KeyError
```

The correct way to add the entity is:

```python
collection.insert_entity(e)
```

The first block of code is prone to problems as the key 1000 may violate the collection’s key allocation policy so we raise KeyError when assignment is used to insert a new entity to the collection. This is consistent with the concept behind OData and Atom where new entities are POSTed to collections and the ID and resulting entity are returned to the caller on success because the service may have modified them to satisfy service-specific constraints.

copy_entity (entity)
Creates a new entity copying the value from entity

The key is not copied and is initially set to NULL.

insert_entity (entity)
Inserts entity into this entity set.

After a successful call to insert_entity:

1. entity is updated with any auto-generated values such as an autoincrement correct key.
2. exists is set to True for entity

Data providers must override this method if the collection is writable.

If the call is unsuccessful then entity should be discarded as its associated bindings may be in a misleading state (when compared to the state of the data source itself).

A general ConstraintError will be raised when the insertion violates model constraints (including an attempt to create two entities with duplicate keys).

update_entity (entity, merge=True)
Updates entity which must already be in the entity set.
The optional merge parameter can be used to force replace semantics instead of the default merge. When merging, any unselected data properties are left unchanged. With merge=False unselected data properties are replaced with their default values as defined by the underlying container. You will have to read back the entity (without a select filter) to obtain those defaults as the values in the entity objects.

Data providers must override this method if the collection is writable.

**update_bindings** *(entity)*

Iterates through the `Entity.navigation_items()` and generates appropriate calls to create/update any pending bindings.

Unlike the `commit()` method, which updates all data and navigation values simultaneously, this method can be used to selectively update just the navigation properties.

**set_page** *(top, skip=0, skiptoken=None)*

Sets the page parameters that determine the next page returned by `iterpage()`.

The skip and top query options are integers which determine the number of entities returned (top) and the number of entities skipped (skip).

`skiptoken` is an opaque token previously obtained from a call to `next_skiptoken()` on a similar collection which provides an index into collection prior to any additional skip being applied.

**set_topmax** *(topmax)*

Sets the maximum page size for this collection.

Data consumers should use `set_page()` to control paging, however data providers can use this method to force the collection to limit the size of a page to at most topmax entities. When topmax is in force and is less than the top value set in `set_page()`, `next_skiptoken()` will return a suitable value for identifying the next page in the collection immediately after a complete iteration of `iterpage()`.

Provider enforced paging is optional, if it is not supported `NotImplementedError` must be raised.

**iterpage** *(set_next=False)*

Returns an iterable subset of the values returned by `itervalues()`.

The subset is defined by the top, skip and skiptoken values set with `set_page()`.

If `set_next` is True then the page is automatically advanced so that the next call to `iterpage` iterates over the next page.

Data providers should override this implementation for a more efficient implementation. The default implementation simply wraps `itervalues()`.

**next_skiptoken**()

Following a complete iteration of the generator returned by `iterpage()`, this method returns the skip-token which will generate the next page or None if all requested entities were returned.

**itervalues**()

Iterates over the collection.

The collection is filtered as defined by `set_filter()` and sorted according to any rules defined by `set_orderby()`.

Entities are also expanded and selected according to the rules defined by `set_expand`.

Data providers must override this implementation which, by default, returns no entities (simulating an empty collection).

**CopyEntity** *(args, **kwargs)*

Deprecated equivalent to `copy_entity()`

**Expand** *(args, **kwargs)*

Deprecated equivalent to `set_expanded()`
Filter(*args, **kwargs)
Deprecated equivalent to `set_filter()`

OrderBy(*args, **kwargs)
Deprecated equivalent to `set_orderby()`

SelectKeys(*args, **kwargs)
Deprecated equivalent to `select_keys()`

SetInlineCount(*args, **kwargs)
Deprecated equivalent to `set_inlinecount()`

TopMax(*args, **kwargs)
Deprecated equivalent to `set_topmax()`

class pyslet.odata2.csdl.Entity(entity_set)
Bases: pyslet.py2.SortableMixin, pyslet.odata2.csdl.TypeInstance

Represents a single instance of an EntityType.

Entity instance must only be created by data providers, a child class may be used with data provider-specific functionality. Data consumers should use the `EntityCollection.new_entity()` or `EntityCollection.copy_entity` methods to create instances.

- entity_set is the entity set this entity belongs to

Entity instances extend `TypeInstance`'s dictionary-like behaviour to include all properties. As a result the dictionary values are one of `SimpleValue`, `Complex` or `DeferredValue` instances.

Property values are created on construction and cannot be assigned directly. To update a simple value use the value's `SimpleValue.set_from_value()` method:

```python
e['Name'].set_from_value("Steve")
# update simple property Name
e['Address']['City'].set_from_value("Cambridge")
# update City in complex property Address
```

A simple valued property that is NULL is still a `SimpleValue` instance, though it will behave as 0 in tests:

```python
e['Name'].set_from_value(None)  # set to NULL
if e['Name']:
    print "Will not print!"
```

Navigation properties are represented as `DeferredValue` instances. A deferred value can be opened in a similar way to an entity set:

```python
# open the collection obtained from navigation property Friends
with e['Friends'].open() as friends:
    # iterate through all the friends of entity e
    for friend in friends:
        print friend['Name']
```

A convenience method is provided when the navigation property points to a single entity (or None) by definition:

```python
mum=e['Mother'].get_entity()  # may return None
```

In the EDM one or more properties are marked as forming the entity’s key. The entity key is unique within the entity set. On construction, an Entity instance is marked as being ‘non-existent’, `exists` is set to False. This is consistent with the fact that the data properties of an entity are initialised to their default values, or NULL if there is no default specified in the model. Entity instances returned as values in collection objects have exists set to True.
Entities from the same entity set can be compared (unlike Complex instances), comparison is done by \texttt{key()}. Therefore, two instances that represent that same entity will compare equal.

If an entity does not exist, open will fail if called on one of its navigation properties with \texttt{NonExistentEntity}.

You can use \texttt{is_entity_collection()} to determine if a property will return an \texttt{EntityCollection} without the cost of accessing the data source itself.

\begin{verbatim}
exists = None
whether or not the instance exists in the entity set

selected = None
the set of selected property names or None if all properties are selected

\_\_i\_t\_e\_r\_\_\(()\)
Iterates over the property names, including the navigation properties.

Unlike native Python dictionaries, the order in which the properties are iterated over is defined. The regular property names are yielded first, followed by the navigation properties. Within these groups properties are yielded in the order they were declared in the metadata model.

\textbf{data\_keys()} 
Iterates through the names of this entity’s data properties only

The order of the names is always the order they are defined in the metadata model.

\textbf{data\_items()} 
Iterator that yields tuples of (key,value) for this entity’s data properties only.

The order of the items is always the order they are defined in the metadata model.

\textbf{merge(fromvalue)} 
Sets this entity’s value from \textit{fromvalue} which must be a \texttt{TypeInstance} instance. In other words, it may be either an Entity or a Complex value.

There is no requirement that \textit{fromvalue} be of the same type, but it must be broadly compatible, which is defined as:

\begin{quote}
Any named property present in both the current value and \textit{fromvalue} must be of compatible types.
\end{quote}

Any named property in the current value which is not present in \textit{fromvalue} is left unchanged by this method.

Null values in \textit{fromvalue} are not copied.

\textbf{navigation\_keys()} 
Iterates through the names of this entity’s navigation properties only.

The order of the names is always the order they are defined in the metadata model.

\textbf{navigation\_items()} 
Iterator that yields tuples of (key,deferred value) for this entity’s navigation properties only.

The order of the items is always the order they are defined in the metadata model.

\textbf{check\_navigation\_constraints(\textit{ignore\_end}=None)} 
For entities that do not yet exist, checks that each of the required navigation properties has been bound (with \texttt{DeferredValue.bind\_entity()}).

If a required navigation property has not been bound then \texttt{NavigationConstraintError} is raised.

If the entity already exists, \texttt{EntityExists} is raised.
For data providers, `ignore_end` may be set to an association set end bound to this entity’s entity set. Any violation of the related association is ignored.

**is_navigation_property**(name)

Returns true if name is the name of a navigation property, False otherwise.

**is_entity_collection**(name)

Returns True if name is the name of a navigation property that points to an entity collection, False otherwise.

**commit**()

Updates this entity following modification.

You can use select rules to provide a hint about which fields have been updated. By the same logic, you cannot update a property that is not selected!

The default implementation opens a collection object from the parent entity set and calls `EntityCollection.update_entity()`.

**delete**()

Deletes this entity from the parent entity set.

The default implementation opens a collection object from the parent entity set and uses the del operator.

Data providers must ensure that the entity’s `exists` flag is set to False after deletion.

**key**()

Returns the entity key as a single python value or a tuple of python values for compound keys.

The order of the values is always the order of the PropertyRef definitions in the associated EntityType’s `key`.

**set_key**(key)

Sets this entity’s key from a single python value or tuple.

The entity must be non-existent or `EntityExists` is raised.

**auto_key**(base=None)

Sets the key to a random value

**base** An optional key suggestion which can be used to influence the choice of automatically generated key.

**key_dict**()

Returns the entity key as a dictionary mapping key property names onto `SimpleValue` instances.

**expand**(expand, select=None)

Expands and selects properties of the entity according to the given `expand` and `select` rules (if any).

Data consumers will usually apply expand rules to a collection which will then automatically ensure that all entities returned by the collection have been expanded.

If, as a result of `select`, a non-key property is unselected then its value is set to NULL. (Properties that comprise the key are never NULL.)

If a property that is being expanded is also subject to one or more selection rules these are passed along with any chained expand method call.

The selection rules in effect are saved in the `select` member and can be tested using `is_selected()`.

**is_selected**(name)

Returns true if the property `name` is selected in this entity.

You should not rely on the value of a unselected property, in most cases it will be set to NULL.
**etag()**
Returns a list of EDMValue instance values to use for optimistic concurrency control or None if the entity does not support it (or if all concurrency tokens are NULL or unselected).

**etag_values()**
Returns a list of EDMValue instance values that may be used for optimistic concurrency control. The difference between this method and `etag()` is that this method returns all values even if they are NULL or unselected. If there are no concurrency tokens then an empty list is returned.

**generate_ctoken()**
Returns a hash object representing this entity’s value.

The hash is a SHA256 obtained by concatenating the literal representations of all data properties (strings are UTF-8 encoded) except the keys and properties which have Fixed concurrency mode.

**set_concurrency_tokens()**
A utility method for data providers.

Sets all `etag_values()` using the following algorithm:

1. Binary values are set directly from the output of `generate_ctoken()`
2. String values are set from the hexdigest of the output `generate_ctoken()`
3. Integer values are incremented.
4. `DateTime` and `DateTimeOffset` values are set to the current time in UTC (and nudged by 1s if necessary)
5. Guid values are set to a new random (type 4) UUID.

Any other type will generate a `ValueError`.

**etag_is_strong()**
Tests the strength of this entity’s etag

Defined by RFC2616:

A "strong entity tag" MAY be shared by two entities of a resource only if they are equivalent by octet equality.

The default implementation returns False which is consistent with the implementation of `generate_ctoken()` as that does not include the key fields.

**CheckNavigationConstraints**(*args, **kwargs*)
Deprecated equivalent to `check_navigation_constraints()`

**DataKeys**(*args, **kwargs*)
Deprecated equivalent to `data_keys()`

**Delete**(*args, **kwargs*)
Deprecated equivalent to `delete()`

**ETag**(*args, **kwargs*)
Deprecated equivalent to `etag()`

**ETagIsStrong**(*args, **kwargs*)
Deprecated equivalent to `etag_is_strong()`

**ETagValues**(*args, **kwargs*)
Deprecated equivalent to `etag_values()`

**Expand**(*args, **kwargs*)
Deprecated equivalent to `expand()`
**IsEntityCollection** (*args, **kwargs)
Deprecated equivalent to `is_entity_collection()`

**IsNavigationProperty** (*args, **kwargs)
Deprecated equivalent to `is_navigation_property()`

**KeyDict** (*args, **kwargs)
Deprecated equivalent to `key_dict()`

**NavigationItems** (*args, **kwargs)
Deprecated equivalent to `navigation_items()`

**NavigationKeys** (*args, **kwargs)
Deprecated equivalent to `navigation_keys()`

**Selected** (*args, **kwargs)
Deprecated equivalent to `is_selected()`

**SetConcurrencyTokens** (*args, **kwargs)
Deprecated equivalent to `set_concurrency_tokens()`

```python
class pyslet.odata2.csdl.SimpleValue(p_def=None):
    Bases: pyslet.py2.UnicodeMixin, pyslet.odata2.csdl.EDMValue

    An abstract class that represents a value of a simple type in the EDMModel.
    This class is not designed to be instantiated directly, use one of the factory methods in EDMValue to construct
    one of the specific child classes.

    type_code = None
    the SimpleType code

    mtype = None
    an optional pyslet.http.params.MediaType representing this value

    value = None
    The actual value or None if this instance represents a NULL value
    The python type used for value depends on type_code as follows:

    Edm.Boolean: one of the Python constants True or False
    Edm.Int64: int (Python 2: long)
    Edm.Double, Edm.Single: python float
    Edm.Decimal: python Decimal instance (from decimal module)
    Edm.DateTime, Edm.DateTimeOffset: py:class:pyslet.iso8601.TimePoint instance
    Edm.Time: py:class:pyslet.iso8601.Time instance (not a Duration, note corrected v2 specification of OData)
    Edm.Binary: binary string
    Edm.String: character string (unicode in Python 2)
    Edm.Guid: python UUID instance (from uuid module)

    For future compatibility, this attribute should only be updated using set_from_value() or one of the
    other related methods.

    simple_cast(type_code)
    Returns a new SimpleValue instance created from type_code
```
The value of the new instance is set using \texttt{\textbf{cast}}()

\textbf{cast (target\_value)}

Updates and returns \texttt{target\_value} a \texttt{SimpleValue} instance.

The value of \texttt{target\_value} is replaced with a value cast from this instance’s value.

If the types are incompatible a \texttt{TypeError} is raised, if the values are incompatible then \texttt{ValueError} is raised.

NULL values can be cast to any value type.

\textbf{set\_from\_simple\_value (new\_value)}

The reverse of the \texttt{\textbf{cast}}() method, sets this value to the value of \texttt{new\_value} casting as appropriate.

\textbf{\_eq\_(other)}

Instances compare equal only if they are of the same type and have values that compare equal.

\textbf{\_unicode\_()}\nonumber

Formats this value into its literal form.

NULL values cannot be represented in literal form and will raise \texttt{ValueError}.

\textbf{set\_from\_literal (value)}

Decodes a value from the value’s literal form.

You can get the literal form of a value using the \texttt{unicode} function.

\textbf{set\_null ()}

Sets the value to NULL

\textbf{set\_from\_value (new\_value)}

Sets the value from a python variable coercing \texttt{new\_value} if necessary to ensure it is of the correct type for the value’s \texttt{\textbf{type\_code}}.

\textbf{set\_random\_value (base=None)}

Sets a random value based

\texttt{base} a \texttt{SimpleValue} instance of the same type that may be used as a base or stem or the random value generated or may be ignored, depending on the value type.

\textbf{classmethod copy (value)}

Constructs a new \texttt{SimpleValue} instance by copying \texttt{value}

\textbf{IsNull (*args, **kwargs)}

Deprecated equivalent to \texttt{is\_null ()}

\textbf{SetFromSimpleValue (*args, **kwargs)}

Deprecated equivalent to \texttt{\textbf{set\_from\_simple\_value} ()}

\textbf{class \texttt{pyslet.odata2.csdl.NumericValue}}(\texttt{p\_def=None})

\textbf{Bases: \texttt{pyslet.odata2.csdl.SimpleValue}}

An abstract class that represents all numeric simple values.

The literal forms of numeric values are parsed in a two-stage process. Firstly the utility class \texttt{Parser} is used to obtain a numeric tuple and then the value is set using \texttt{set\_from\_numeric\_literal ()}

All numeric types may have their value set directly from int, (Python 2 long,) float or Decimal.

Integer representations are rounded towards zero using the python int (or Python 2 long) functions when necessary.

\textbf{set\_to\_zero ()}

Set this value to the default representation of zero
set_from_numeric_literal(num)
    Decodes a value from a numeric tuple as returned by `Parser.parse_numeric_literal()`.

SetFromLiteral(*args, **kwargs)
    Deprecated equivalent to `set_from_literal()`

SetFromNumericLiteral(*args, **kwargs)
    Deprecated equivalent to `set_from_numeric_literal()`

SetToZero(*args, **kwargs)
    Deprecated equivalent to `set_to_zero()`

class pyslet.odata2.csdl.FloatValue(p_def=None)
    Bases: pyslet.odata2.csdl.NumericValue

    Abstract class that represents one of Edm.Double or Edm.Single.
    Values can be set from int, (Python 2: long,) float or Decimal.
    There is no hard-and-fast rule about the representation of float in Python and we may refuse to accept values that fall within the accepted ranges defined by the CSDL if float cannot hold them. That said, you won’t have this problem in practice.
    The derived classes `SingleValue` and `DoubleValue` only differ in the Max value used when range checking.
    Values are formatted using Python’s default string conversion.

**Primitive SimpleTypes**

Simple values can be created directly using one of the type-specific classes below.

class pyslet.odata2.csdl.BinaryValue(p_def=None)
    Bases: pyslet.odata2.csdl.SimpleValue

    Represents a `SimpleValue` of type Edm.Binary.
    Binary literals allow content in the following form:

    \[ [A-Fa-f0-9][A-Fa-f0-9]* \]

    Binary values can be set from any Python type, though anything other than a binary string is set to its pickled representation. There is no reverse facility for reading an object from the pickled value.

class pyslet.odata2.csdl.BooleanValue(p_def=None)
    Bases: pyslet.odata2.csdl.SimpleValue

    Represents a simple value of type Edm.Boolean
    Boolean literals are one of:

    `true` | `false`

    Boolean values can be set from their Python equivalents and from any int, (Python 2 long,) float or Decimal where the non-zero test is used to set the value.

class pyslet.odata2.csdl.ByteValue(p_def=None)
    Bases: pyslet.odata2.csdl.NumericValue

    Represents a simple value of type Edm.Byte
    Byte literals must not have a sign, decimal point or exponent.
Byte values can be set from an int, (Python 2: long,) float or Decimal

class pyslet.odata2.csdl.DateTimeValue (p_def=None)
Bases: pyslet.odata2.csdl.SimpleValue

Represents a simple value of type Edm.DateTime

DateTime literals allow content in the following form:

```
yyyy-mm-ddThh:mm[:ss[.fffffff]]
```

DateTime values can be set from an instance of iso8601.TimePoint or type int, (Python 2: long,) float, Decimal or the standard Python date.datetime and date.date instances. In the case of date.date, the new value represents midnight at the beginning of the specified day.

Any zone specifier is ignored. There is no conversion to UTC, the value simply becomes a local time in an unspecified zone. This is a weakness of the EDM, it is good practice to limit use of the DateTime type to UTC times.

When set from a numeric value, the value must be non-negative. Unix time is assumed. See the from_unix_time() factory method of TimePoint for information.

If a property definition was set on construction then the defined precision is used when representing the value as a character string. For example, if the property has precision 3 then the output of the string conversion will appear in the following form:

```
1969-07-20T20:17:40.000
```

class pyslet.odata2.csdl.DateTimeOffsetValue (p_def=None)
Bases: pyslet.odata2.csdl.SimpleValue

Represents a simple value of type Edm.DateTimeOffset

DateTimeOffset literals are defined in terms of the XMLSchema lexical representation.

DateTimeOffset values can be set from an instance of iso8601.TimePoint or type int, (Python 2: long,) float or Decimal.

TimePoint instances must have a zone specifier. There is no automatic assumption of UTC.

When set from a numeric value, the value must be non-negative. Unix time in UTC assumed. See the from_unix_time() factory method of TimePoint for information.

If a property definition was set on construction then the defined precision is used when representing the value as a character string. For example, if the property has precision 3 then the output of the string conversion will appear in the following form:

```
1969-07-20T15:17:40.000-05:00
```

It isn’t completely clear if the canonical representation of UTC using ‘Z’ instead of an offset is intended or widely supported so we always use an offset:

```
1969-07-20T20:17:40.000+00:00
```

class pyslet.odata2.csdl.DecimalValue (p_def=None)
Bases: pyslet.odata2.csdl.NumericValue

Represents a simple value of type Edm.Decimal

Decimal literals must not use exponent notation and there must be no more than 29 digits to the left and right of the decimal point.

Decimal values can be set from int, (Python 2: long,) float or Decimal values.
class pyslet.odata2.csdl.DoubleValue (p_def=None)
Bases: pyslet.odata2.csdl<FloatValue

Represents a simple value of type Edm.Double

Max = 1.7976931348623157e+308
the largest positive double value

This value is set dynamically on module load, theoretically it may be set lower than the maximum allowed by the specification if Python’s native float is of insufficient precision but this is unlikely to be an issue.

MaxD = Decimal('1.79769313486E+308')
the largest positive double value converted to decimal form

class pyslet.odata2.csdl(GuidValue (p_def=None)
Bases: pyslet.odata2.csdl.SimpleValue

Represents a simple value of type Edm.Guid

Guid literals allow content in the following form: dddddddd-dddd-dddd-dddd-dddddddddddd where each d represents [A-Fa-f0-9].

Guid values can also be set directly from either binary or hex strings. Binary strings must be of length 16 and are passed as raw bytes to the UUID constructor, hexadecimal strings must be of length 32 characters. (In Python 2 both str and unicode types are accepted as hexadecimal strings, the length being used to determine if the source is a binary or hexadecimal representation.)

class pyslet.odata2.csdl.Int16Value (p_def=None)
Bases: pyslet.odata2.csdl.NumericValue

Represents a simple value of type Edm.Int16

class pyslet.odata2.csdl.Int32Value (p_def=None)
Bases: pyslet.odata2.csdl.NumericValue

Represents a simple value of type Edm.Int32

class pyslet.odata2.csdl.Int64Value (p_def=None)
Bases: pyslet.odata2.csdl.NumericValue

Represents a simple value of type Edm.Int64

class pyslet.odata2.csdl.SByteValue (p_def=None)
Bases: pyslet.odata2.csdl.NumericValue

Represents a simple value of type Edm.SByte

class pyslet.odata2.csdl.SingleValue (p_def=None)
Bases: pyslet.odata2.csdl.FloatValue

Represents a simple value of type Edm.Single

Max = 3.4028234663852886e+38
the largest positive single value

This value is set dynamically on module load, theoretically it may be set lower than the maximum allowed by the specification if Python’s native float is of insufficient precision but this is very unlikely to be an issue unless you’ve compiled Python on in a very unusual environment.

MaxD = Decimal('3.40282346639E+38')
the largest positive single value converted to Decimal

set_from_numeric_literal (num)
Decodes a Single value from a Numeric literal.
class pyslet.odata2.csdl.StringValue (p_def=None)
    Bases: pyslet.odata2.csdl.SimpleValue
    Represents a simple value of type Edm.String"
    The literal form of a string is the string itself.
    Values may be set from any string or object which supports conversion to character string.

class pyslet.odata2.csdl.TimeValue (p_def=None)
    Bases: pyslet.odata2.csdl.SimpleValue

Complex Types

class pyslet.odata2.csdl.Complex (p_def=None)
    Bases: pyslet.odata2.csdl.EDMValue, pyslet.odata2.csdl.TypeInstance
    Represents a single instance of a ComplexType.
    is_null ()
        Complex values are never NULL
    set_null ()
        Sets all simple property values to NULL recursively
    set_default_value ()
        Sets all simple property values to defaults recursively
    merge (new_value)
        Sets this value from new_value which must be a Complex instance.
        There is no requirement that new_value is of the same type, but it must be broadly compatible, which is defined as:
            Any named property present in both the current value and new_value must be of compatible types.
            Any named property in the current value which is not present in new_value is left unchanged by this method.
            Null values are not merged.
    IsNull (*args, **kwargs)
        Deprecated equivalent to is_null ()

Navigation: Deferred Values

class pyslet.odata2.csdl.DeferredValue (name, from_entity)
    Bases: pyslet.pep8.MigratedClass
    Represents the value of a navigation property.
    name = None
        the name of the associated navigation property
    from_entity = None
        the entity that contains this value
    p_def = None
        the definition of the navigation property
**isRequired** = None
True if this deferred value represents a (single) required entity

**isCollection** = None
True if this deferred value represents a collection

**isExpanded** = None
True if this deferred value has been expanded.

An expanded navigation property will return a read-only `ExpandedEntityCollection` when `open()` is called.

**bindings** = None
The list of entity instances or keys to bind to `from_entity` when it is inserted or next updated.

**target** ()
Returns the target entity set of this navigation (without opening the collection).

**get_entity** ()
Returns a single entity instance or None.

If this deferred value represents an entity collection then `NavigationError` is raised.

**open** ()
Opens the collection associated with this navigation property.

Returns an `EntityCollection` instance which must be closed when it is no longer needed. This is best achieved with the Python with statement using the collection’s context-manager behaviour. For example:

```python
with customer['Orders'].open() as orders:
    # do something with the orders
```

**set_expansion_values** (**values**)
Sets the expansion of this deferred value

**values** A list of `Entity` instances.

No call to the underlying data-layer is made, it is assumed that values is an appropriate representation of the data that would be obtained by executing:

```python
with self.open() as coll:
    return coll.values()
```

The purpose of this method is to allow the re-use of a value list that has been obtained previously without having to consult the data source again.

**set_expansion** (**expanded**)
Sets the expansion for this deferred value to the `ExpandedEntityCollection` given.

If `expanded` is None then the expansion is removed and future calls to `OpenCollection()` will yield a (dynamically created) entity collection.

**expand_collection** (**expand**, **select**)
A convenience function of use to data providers.

Expands this navigation property, further expanding the resulting collection of entities using the given `expand` and `select` options (see `EntityCollection.set_expand()` for details).

**bind_entity** (**target**)
Binds a `target` entity to this navigation property.

**target** is either the entity you’re binding or its key in the target entity set. For example:
customer['Orders'].bind(1)

binds the entity represented by ‘customer’ to the Order entity with key 1.

Just as for updates to data property values, the binding information is saved and acted upon when the entity is next updated or, for non-existent entities, inserted into the entity set.

If you attempt to bind to a target entity that doesn’t exist the target entity will be created automatically when the source entity is updated or inserted.

**check_navigation_constraint()**
Checks if this navigation property **isRequired** and raises **NavigationConstraintError** if it has not been bound with **bind_entity()**.

This method is only intended to be called on non-existent entities.

**update_bindings()**
Iterates through **bindings** and generates appropriate calls to update the collection.

Unlike the parent Entity’s **Entity.commit()** method, which updates all data and navigation values simultaneously, this method can be used to selectively update a single navigation property.

**clear_bindings()**
Removes any (unsaved) entity bindings from this navigation property.

**BindEntity** (*args, **kwargs)
Deprecated equivalent to **bind_entity()**

**ClearBindings** (*args, **kwargs)
Deprecated equivalent to **clear_bindings()**

**GetEntity** (*args, **kwargs)
Deprecated equivalent to **get_entity()**

**OpenCollection** (*args, **kwargs)
Deprecated equivalent to **open()**

**SetExpansion** (*args, **kwargs)
Deprecated equivalent to **set_expansion()**

**Target** (*args, **kwargs)
Deprecated equivalent to **target()**

---

**class** pyslet.odata2.csd1.NavigationCollection (from_entity, name, **kwargs)
**Bases:** pyslet.odata2.csd1.EntityCollection

Represents the collection of entities returned by a navigation property.

These collections behave in the same way as entity collections opened from the base **EntitySet** with the following exceptions:

etColl[key]=entity

Adds a link to **entity** from the source entity used to open the navigation collection. If **key** does not match **entity**’s key then **ValueError** is raised. The entity must already exist and be a member of the base entity set, otherwise **KeyError** is raised.

This class is used even if the navigation property is declared to return a single entity, rather than a collection. In this case assignment will only work if the collection is currently empty. To **replace** an existing link use **replace()**.

del etColl[key]
Deletes the link from the source entity to the entity with key. If no such link exists, KeyError is raised.

This behaviour differs from the base EntityCollection behaviour where the del operator removes the entity completely from the entity container. In this case the entity still exists in the parent entity set, only the link is removed.

Notes for data providers

On construction:

• entity_set is the entity set containing the target entities, the collection behaves like a subset of this entity set. It is passed to super

Named arguments specific to this class:

• from_entity is the source entity being navigated
• name is the name of the navigation property being navigated

Writeable collections must override the __setitem__() method.

name = None
    the name of the navigation property

from_entity = None
    the source entity

from_end = None
    the AssociationSetEnd that represents the source of this association

p_def = None
    the navigation property’s definition in the metadata model

insert_entity(entity)
    Inserts a new entity into the target entity set and simultaneously creates a link to it from the source entity.

replace(entity)
    This method replaces all links with a link to the single item, entity. If the collection was empty then this is
equivalent to __setitem__(entity.key(), entity).

    Although for some collections this is equivalent to clear() followed by __setitem__, this method must
be used to combine these operations into a single call when the collection is required to contain exactly
one link at all times.

class pyslet.odata2.csdl.ExpandedEntityCollection(entity_list, **kwargs)
Bases: pyslet.odata2.csdl.NavigationCollection

A special sub-class of NavigationCollection used when a navigation property has been expanded.

An expanded entity collection is a read-only, cached view of the entities linked from the source entity.

Warning: although you may apply a filter and orderby rules to an expanded collection these are evaluated on the
local copy and are not passed to the data source. As a result, there may be differences in the way these options
behave due to different expression semantics.

Note for data providers:

The named argument entity_list passed to this constructor is a simple python list of the entities the expanded
collection contains. Internally a dictionary of the entities is built to speed up access by key.
Supporting Classes

class pyslet.odata2.csdl.EDMValue(p_def=None)
    Bases: pyslet.py2.BoolMixin, pyslet.pep8.PEP8Compatibility
    Abstract class to represent a value in the EDMModel.
    This class is used to wrap or ‘box’ instances of a value. In particular, it can be used in a context where that value
    can have either a simple or complex type.
    EDMValue instances are treated as being non-zero if is_null() returns False.

    p_def = None
        An optional Property instance from the metadata model defining this value’s type

    is_null()
        Returns True if this object is Null.

classmethod from_property(p_def)
    Constructs an instance of the correct child class of EDMValue to represent a value defined by Property
    instance p_def.
    We support a special case for creating a type-less NULL. If you pass None for p_def then a type-less
    SimpleValue is instantiated.

classmethod from_type(type_code)
    Constructs an instance of the correct child class of EDMValue to represent an (undeclared) simple value
    of SimpleType type_code.

classmethod from_value(value)
    Constructs an instance of the correct child class of EDMValue to hold value.
    value may be any of the types listed in SimpleValue.

classmethod NewSimpleValue(*args, **kwargs)
    Deprecated equivalent to from_type()

classmethod NewSimpleValueFromValue(*args, **kwargs)
    Deprecated equivalent to from_value()

classmethod NewValue(*args, **kwargs)
    Deprecated equivalent to from_property()

class pyslet.odata2.csdl.TypeInstance(type_def=None)
    Bases: pyslet.odata2.csdl.DictionaryLike, pyslet.pep8.PEP8Compatibility
    Abstract class to represents a single instance of a ComplexType or EntityType.
    Behaves like a read-only dictionary mapping property names onto EDMValue instances. (You can change the
    value of a property using the methods of EDMValue and its descendants.)
    Unlike regular Python dictionaries, iteration over the of keys in the dictionary (the names of the properties) is
    always done in the order in which they are declared in the type definition.

    type_def = None
        the definition of this type

    AddProperty(*args, **kwargs)
        Deprecated equivalent to add_property()
Metadata Model

class pyslet.odata2.csd1.CSDLElement(parent, name=None)
  Bases: pyslet.xml.namespace.NSElement
  All elements in the metadata model inherit from this class.

class pyslet.odata2.csd1.Schema(parent)
  Bases: pyslet.odata2.csd1.NameTableMixin, pyslet.odata2.csd1.CSDLElement
  Represents the Edm root element.
  Schema instances are based on NameTableMixin allowing you to look up the names of declared Associations, ComplexTypes, EntityTypes, EntityContainers and Functions using dictionary-like methods.

  name = None
    the declared name of this schema

  Association = None
    a list of Association instances

  ComplexType = None
    a list of ComplexType instances

  EntityType = None
    a list of EntityType instances

class pyslet.odata2.csd1.EntityContainer(parent)
  Bases: pyslet.odata2.csd1.NameTableMixin, pyslet.odata2.csd1.CSDLElement
  Models an entity container in the metadata model.
  An EntityContainer inherits from NameTableMixin to enable it to behave like a scope. The EntitySet instances and AssociationSet instances it contains are declared within the scope.

  name = None
    the declared name of the container

  Documentation = None
    the optional Documentation

  EntitySet = None
    a list of EntitySet instances

  AssociationSet = None
    a list of AssociationSet instances

  find_entitysets(entity_type)
    Returns a list of all entity sets with a given type

      entity_type
        An EntityType instance.

    Returns an empty list if no declared EntitySets have this type.

class pyslet.odata2.csd1.EntitySet(parent)
  Bases: pyslet.odata2.csd1.CSDLElement
  Represents an EntitySet in the metadata model.

  name = None
    the declared name of the entity set

  entityType = None
    the name of the entity type of this set’s elements
entityType = None
the EntityType of this set’s elements

keys = None
a list of the names of this entity set’s keys in their declared order

navigation = None
a mapping from navigation property names to AssociationSetEnd instances

linkEnds = None
A mapping from AssociationSetEnd instances that reference this entity set to navigation property names (or None if this end of the association is not bound to a named navigation property)

unboundPrincipal = None
An AssociationSetEnd that represents our end of an association with an unbound principal or None if all principals are bound.

What does that mean? It means that there is an association set bound to us where the other role has a multiplicity of 1 (required) but our entity type does not have a navigation property bound to the association. As a result, our entities can only be created by a deep insert from the principal (the entity set at the other end of the association).

Clear as mud? An example may help. Suppose that each Order entity must have an associated Customer but (perhaps perversely) there is no navigation link from Order to Customer, only from Customer to Order. For the Order entity, the Customer is the principal as Orders can only exist when they are associated with a Customer.

Attempting to create an Order in the base collection of Orders will always fail:

```python
with Orders.open() as collection:
    order=collection.new_entity()
    # set order fields here
    collection.insert_entity(order)
    # raises ConstraintError as order is not bound to a customer
```

Instead, you have to create new orders from a Customer entity:

```python
with Customers.open() as collectionCustomers:
    # get the existing customer
    customer=collectionCustomers['ALFKI']
    with customer['Orders'].open() as collectionOrders:
        # create a new order
        order=collectionOrders.new_entity()
        # ... set order details here
        collectionOrders.insert_entity(order)
```

You can also use a deep insert:

```python
with Customers.open() as collectionCustomers, Orders.open() as collectionOrders:
    customer=collectionCustomers.new_entity()
    # set customer details here
    order=collectionOrders.new_entity()
    # set order details here
    customer['Orders'].bind_entity(order)
    collectionCustomers.insert_entity(customer)
```

For the avoidance of doubt, an entity set can’t have two unbound principals because if it did you would never be able to create entities in it!
Documentation = None
    the optional Documentation

get_fqname()
    Returns the fully qualified name of this entity set.

get_location()
    Returns a pyslet.rfc2396.URI instance representing the location for this entity set.

set_location()
    Sets the location of this entity set
    Resolves a relative path consisting of:

    [ EntityContainer.name '.' ] name

    The resolution of URIs is done in accordance with the XML specification, so is affected by any xml:base attributes set on parent elements or by the original base URI used to load the metadata model. If no base URI can be found then the location remains expressed in relative terms.

get_key(keylike)
    Extracts a key from a keylike argument

    keylike A value suitable for using as a key in an EntityCollection based on this entity set.

    Keys are represented as python values (as described in SimpleValue) or as tuples of python values in the case of compound keys. The order of the values in a compound key is the order in which the Key properties are defined in the corresponding EntityType definition.

    If keylike is already in the correct format for this entity type then it is returned unchanged.

    If the key is single-valued and keylike is a tuple containing a single value then the single value is returned without the tuple wrapper.

    If keylike is a dictionary, or an Entity instance, which maps property names to values (or to SimpleValue instances) the key is calculated from it by extracting the key properties. As a special case, a value mapped with a dictionary key of the empty string is assumed to be the value of the key property for an entity type with a single-valued key, but only if the key property’s name is not itself in the dictionary.

    If keylike cannot be turned in to a valid key the KeyError is raised.

extract_key(keyvalue)
    Extracts a key value from keylike.

    Unlike get_key, this method attempts to convert the data in keyvalue into the correct format for the key. For compound keys keyvalue must be a suitable list or tuple or compatible iterable supporting the len method. Dictionaries are not supported.

    If keyvalue cannot be converted into a suitable representation of the key then None is returned.

key_dict(key)
    Given a key from this entity set, returns a key dictionary.

    The result is a mapping from named properties to SimpleValue instances. The property name is always used as the key in the mapping, even if the key refers to a single property. This contrasts with get_key_dict().

get_key_dict(key)
    Given a key from this entity set, returns a key dictionary.

    The result is a mapping from named properties to SimpleValue instances. As a special case, if a single property defines the entity key it is represented using the empty string, not the property name.
bind(binding, **kws)
   Binds this entity set to a collection class

   binding Must be a class (or other callable) that returns an EntityCollection instance, by default we are bound to the default EntityCollection class which behaves like an empty collection.

   kws A python dict of named arguments to pass to the binding callable

open()
   Opens this entity set

   Returns an EntityCollection instance suitable for accessing the entities themselves.

bind_navigation(name, binding, **kws)
   Binds the navigation property name.

   binding Must be a class (or other callable) that returns a NavigationCollection instance. By default we are bound to the default NavigationCollection class which behaves like an empty collection.

   kws A python dict of named arguments to pass to the binding callable

open_navigation(name, source_entity)
   Opens a navigation collection

   Returns a NavigationCollection instance suitable for accessing the entities obtained by navigating from source_entity, an Entity instance, via the navigation property with name.

get_target(name)
   Returns the target entity set of navigation property name

get_multiplicity(name)
   Gets the multiplicities of a named navigation properly

   Returns the Multiplicity of both the source and the target of the named navigation property, as a tuple, for example, if customers is an entity set from the sample OData service:

   customers.get_multiplicity['Orders'] == (Multiplicity.ZeroToOne, Multiplicity.Many)

is_entity_collection(name)
   Tests the multiplicity of a named navigation property

   Returns True if more than one entity is possible when navigating the named property.

BindNavigation(*args, **kwars)
   Deprecated equivalent to bind_navigation()

GetFQName(*args, **kwars)
   Deprecated equivalent to get_fqname()

GetKey(*args, **kwars)
   Deprecated equivalent to get_key()

GetKeyDict(*args, **kwars)
   Deprecated equivalent to get_key_dict()

GetLocation(*args, **kwars)
   Deprecated equivalent to get_location()

IsEntityCollection(*args, **kwars)
   Deprecated equivalent to is_entity_collection()

KeyKeys(*args, **kwars)
   Deprecated equivalent to key_keys()
NavigationMultiplicity(*args, **kwargs)
    Deprecated equivalent to get_multiplicity()

NavigationTarget(*args, **kwargs)
    Deprecated equivalent to get_target()

OpenCollection(*args, **kwargs)
    Deprecated equivalent to open()

OpenNavigation(*args, **kwargs)
    Deprecated equivalent to open_navigation()

SetLocation(*args, **kwargs)
    Deprecated equivalent to set_location()

class pyslet.odata2.csd1.AssociationSet(parent)
    Bases: pyslet.odata2.csd1.CSDLElement

    Represents an association set in the metadata model.

    The purpose of the association set is to bind the ends of an association to entity sets in the container.

    Contrast this with the association element which merely describes the association between entity types.

    At first sight this part of the entity data model can be confusing but imagine an entity container that contains
    two entity sets that have the same entity type. Any navigation properties that reference this type will need to be
    explicitly bound to one or other of the entity sets in the container.

    As an aside, it isn’t really clear if the model was intended to be used this way. It may have been
    intended that the entity type in the definition of an entity set should be unique within the scope of the
    entity container.

    name = None
        the declared name of this association set

    associationName = None
        the name of the association definition

    association = None
        the Association definition

    Documentation = None
        the optional Documentation

class pyslet.odata2.csd1.AssociationSetEnd(parent)
    Bases: pyslet.py2.SortableMixin, pyslet.odata2.csd1.CSDLElement

    Represents the links between two entity sets

    The get_qualified_name() method defines the identity of this element. The built-in Python hash function
    returns a hash based on this value and the associated comparison functions are also implemented enabling these
    elements to be added to ordinary Python dictionaries.

    Oddly, role names are sometimes treated as optional but it can make it a challenge to work out which end of the
    association is which when we are actually using the model if one or both are missing. The algorithm we use is
    to use role names if either are given, otherwise we match the entity types. If these are also identical then the
    choice is arbitrary. To prevent confusion missing role names are filled in when the metadata model is loaded.

    name = None
        the role-name given to this end of the link

    entitySetName = None
        name of the entity set this end links to
entity_set = None
    EntitySet this end links to

associationEnd = None
    AssociationEnd that defines this end of the link

otherEnd = None
    the other AssociationSetEnd of this link

Documentation = None
    the optional Documentation

get_qualified_name()
    A utility function to return a qualified name.
    The qualified name comprises the name of the parent AssociationSet and the role name.

GetQualifiedName(*args, **kwargs)
    Deprecated equivalent to get_qualified_name()

class pyslet.odata2.csd1.Type(parent)
    Bases: pyslet.odata2.csd1.NameTableMixin, pyslet.odata2.csd1.CSDElement

    An abstract class for both Entity and Complex types.

    Types inherit from NameTableMixin to allow them to behave as scopes in their own right. The named properties are declared in the type’s scope enabling you so use them as dictionaries to look up property definitions.

    Because of the way nested scopes work, this means that you can concatenate names to do a deep look up, for example, if Person is a defined type:

    Person['Address']['City'] is Person['Address.City']

    name = None
        the declared name of this type

    baseType = None
        the name of the base-type for this type

    Property = None
        a list of Property

    get_fqname()
        Returns the full name of this type
        Includes the schema namespace prefix.

    GetFQName(*args, **kwargs)
        Deprecated equivalent to get_fqname()

class pyslet.odata2.csd1.EntityType(parent)
    Bases: pyslet.odata2.csd1.Type

    Models the key and the collection of properties that define a set of Entity

    Key = None
        the Key

    validate_expansion(expand, select)
        A utility method for data providers.
        Checks the expand and select options, as described in EntityCollection.set_expand() for validity raising ValueError if they violate the OData specification.

        Specifically the following are checked:
1. That “*” only ever appears as the last item in a select path
2. That nothing appears after a simple property in a select path
3. That all names are valid property names
4. That all expanded names are those of navigation properties

```python
ValidateExpansion(*args, **kwargs)
```

Deprecated equivalent to `validate_expansion()`

```python
class pyslet.odata2.csdl.Key(parent)
    Bases: pyslet.odata2.csdl.CSDLElement
    Models the key fields of an EntityType

    PropertyRef = None
        a list of `PropertyRef`

class pyslet.odata2.csdl.PropertyRef(parent)
    Bases: pyslet.odata2.csdl.CSDLElement
    Models a reference to a single property within a Key.

    name = None
        the name of this (key) property

    property = None
        the `Property` instance of this (key) property

    update_type_refs(scope, stop_on_errors=False)
        Sets property

class pyslet.odata2.csdl.Property(parent)
    Bases: pyslet.odata2.csdl.CSDLElement
    Models a property of an EntityType or ComplexType.

    name = None
        the declared name of the property

    type = None
        the name of the property’s type

    simpleTypeCode = None
        one of the `SimpleType` constants if the property has a simple type

    complexType = None
        the associated `ComplexType` if the property has a complex type

    nullable = None
        if the property may have a null value

    defaultValue = None
        a string containing the default value for the property or None if no default is defined

    maxLength = None
        the maximum length permitted for property values

    fixedLength = None
        a boolean indicating that the property must be of length `maxLength`
```
**precision** = None
   a positive integer indicating the maximum number of decimal digits (decimal values)

**scale** = None
   a non-negative integer indicating the maximum number of decimal digits to the right of the point

**unicode** = None
   a boolean indicating that a string property contains unicode data

**Documentation** = None
   the optional Documentation

class pyslet.odata2.csd1.ComplexType (parent)
   Bases: pyslet.odata2.csd1.Type

   Models the collection of properties that define a Complex value.
   This class is a trivial sub-class of Type

class pyslet.odata2.csd1.NavigationProperty (parent)
   Bases: pyslet.odata2.csd1.CSDLElement

   Models a navigation property of an EntityType.

   **name** = None
      the declared name of the navigation property

   **fromRole** = None
      the name of this link’s source role

   **toRole** = None
      the name of this link’s target role

   **from_end** = None
      the AssociationEnd instance representing this link’s source

   **to_end** = None
      the AssociationEnd instance representing this link’s target

   **ambiguous** = None
      flag set if Association is ambiguous within the parent EntityType, backLink will never be set!

   **backLink** = None
      the NavigationProperty that provides the back link (or None, if this link is one-way)

class pyslet.odata2.csd1.Association (parent)
   Bases: pyslet.odata2.csd1.NameTableMixin, pyslet.odata2.csd1.CSDLElement

   Models an association.
   This class inherits from NameTableMixin to enable it to behave like a scope in its own right. The contained AssociationEnd instances are declared in the association scope by role name.

   **name** = None
      the name declared for this association

   **Documentation** = None
      the optional Documentation

   **AssociationEnd** = None
      a list of AssociationEnd instances

   **get_fqname** ()
      Returns the full name of this association
The result includes the schema namespace prefix.

```
GetFQName(*args, **kwargs)
```

Deprecated equivalent to `get_fqname()`

```python
class pyslet.odata2.csdl.AssociationEnd (parent)
    Bases: pyslet.odata2.csdl.CSDLElement

Models one end of an Association.

We define a hash method to allow AssociationEnds to be used as keys in a dictionary.

```
name = None
    the role-name given to this end of the link

type = None
    name of the entity type this end links to

type = None
    name of the entity type this end links to

type = None
    EntityType this end links to

multiplicity = None
    aMultiplicity constant

otherEnd = None
    the other AssociationEnd of this link
```

```
get_qualified_name()
```

A utility function to return a qualified name.

The qualified name comprises the name of the parent Association and the role name.

```python
class pyslet.odata2.csdl.Documentation (parent)
    Bases: pyslet.odata2.csdl.CSDLElement

Used to document elements in the metadata model
```

### Misc Definitions

```python
pyslet.odata2.csdl.validate_simple_identifier(identifier)
```

Validates a simple identifier, returning the identifier unchanged or raising ValueError.

```python
class pyslet.odata2.csdl.SimpleType
    Bases: pyslet.xml.xsdatatypes.EnumerationNoCase

SimpleType defines constants for the core data types defined by CSDL
```

```python
SimpleType.Boolean
SimpleType.DEFAULT == None
```

For more methods see `Enumeration`

The canonical names for these constants uses the Edm prefix, for example, “Edm.String”. As a result, the class has attributes of the form “SimpleType.Edm.Binary” which are inaccessible to python unless getattr is used. To workaround this problem (and because the Edm. prefix seems to be optional) we also define aliases without the Edm. prefix. As a result you can use, e.g., SimpleType.Int32 as the symbolic representation in code but the following are all True:

```python
SimpleType.from_str("Edm.Int32") == SimpleType.Int32
SimpleType.from_str("Int32") == SimpleType.Int32
SimpleType.to_str(SimpleType.Int32) == "Edm.Int32"
```
A python dictionary that maps a type code (defined by the types module) to a constant from this class indicating a safe representation in the EDM. For example:

```python
SimpleType.PythonType[int]==SimpleType.Int64
```

class pyslet.odata2.csd1.ConcurrencyMode

Bases: pyslet.xml.xsdatatypes EnumerationNoCase

ConcurrencyMode defines constants for the concurrency modes defined by CSDL

```python
ConcurrencyMode.Fixed
ConcurrencyMode.DEFAULT == ConcurrencyMode.none
```

Note that although ‘Fixed’ and ‘None’ are the correct values lower-case aliases are also defined to allow the value ‘none’ to be accessible through normal attribute access. In most cases you won’t need to worry as a test such as the following is sufficient:

```python
if property.concurrencyMode==ConcurrencyMode.Fixed:
    # do something with concurrency tokens
```

For more methods see `Enumeration`

`pyslet.odata2.csd1.maxlength_from_str(value)`

Decodes a maxLength value from a character string.

“The maxLength facet accepts a value of the literal string “max” or a positive integer with value ranging from 1 to 2^31”

The value ‘max’ is returned as the value `MAX`

`pyslet.odata2.csd1.maxlength_to_str(value)`

Encodes a maxLength value as a character string.

we define the constant `MAX` to represent the special ‘max’ value of maxLength

class pyslet.odata2.csd1.Multiplicity

Defines constants for representing association end multiplicities.

`pyslet.odata2.csd1.multiplicity_from_str(src)`

Decodes a Multiplicity value from a character string.

The valid strings are “0..1”, “1” and “*”

`pyslet.odata2.csd1.multiplicity_to_str(value)`

Encodes a Multiplicity value as a character string.

class pyslet.odata2.csd1.Parser(source)

Bases: pyslet.unicode5.BasicParser

A CSDL-specific parser, mainly for decoding literal values of simple types.

The individual parsing methods may raise ValueError in cases where parsed value has a value that is out of range.

`parse_binary_literal()`

Parses a binary literal, returning a binary string

`parse_boolean_literal()`

Parses a boolean literal returning True, False or None if no boolean literal was found.
parse_byte_literal()
Parses a byteLiteral, returning a python integer.

We are generous in what we accept, ignoring leading zeros. Values outside the range for byte return None.

parse_datetime_literal()
Parses a DateTime literal, returning a pyslet.iso8601.TimePoint instance.

Returns None if no DateTime literal can be parsed. This is a generous way of parsing iso8601-like values, it accepts omitted zeros in the date, such as 4-7-2001.

parse_guid_literal()
Parses a Guid literal, returning a UUID instance from the uuid module.

Returns None if no Guid can be parsed.

parse_numeric_literal()
Parses a numeric literal returning a named tuple of strings:

```
( sign, ldigits, rdigits, expSign, edigits )
```

An empty string indicates a component that was not present except that rdigits will be None if no decimal point was present. Likewise, edigits may be None indicating that no exponent was found.

Although both ldigits and rdigits can be empty they will never both be empty strings. If there are no digits present then the method returns None, rather than a tuple. Therefore, forms like “E+3” are not treated as being numeric literals whereas, perhaps oddly, 1E+ is parsed as a numeric literal (even though it will raise ValueError later when setting any of the numeric value types).

Representations of infinity and not-a-number result in ldigits being set to ‘inf’ and ‘nan’ respectively. They always result in rdigits and edigits being None.

parse_time_literal()
Parses a Time literal, returning a pyslet.iso8601.Time instance.

Returns None if no Time literal can be parsed. This is a generous way of parsing iso8601-like values, it accepts omitted zeros in the leading field, such as 7:45:00.

Utility Classes

These classes are not specific to the EDM but are used to support the implementation. They are documented to allow them to be reused in other modules.

class pyslet.odata2.csdl.NameTableMixin
   Bases: pyslet.odata2.csdl.DictionaryLike

   A mix-in class to help other objects become named scopes.

   Using this mix-in the class behaves like a read-only named dictionary with string keys and object values. If the dictionary contains a value that is itself a NameTableMixin then keys can be compounded to look-up items in sub-scopes.

   For example, if the name table contains a value with key “X” that is itself a name table containing a value with key “Y” then both “X” and “X.Y” are valid keys, the latter performing a ‘deep lookup’ in the nested scope.

   name = None
      the name of this name table (in the context of its parent)

   nameTable = None
      a dictionary mapping names to child objects
`__getitem__`(key)
Looks up `key` in `nameTable` and, if not found, in each child scope with a name that is a valid scope prefix of `key`. For example, if `key` is “My.Scope.Name” then a child scope with name “My.Scope” would be searched for “Name” or a child scope with name “My” would be searched for “Scope.Name”.

`__iter__()`
Yields all keys defined in this scope and all compounded keys from nested scopes. For example, a child scope with name “My.Scope” which itself has a child “Name” would generate two keys: “My.Scope” and “My.Scope.Name”.

`__len__()`
Returns the number of keys in this scope including all compounded keys from nested scopes.

`declare`(value)
Declares a value in this named scope.

`value` must have a name attribute which is used to declare it in the scope; duplicate keys are not allowed and will raise `DuplicateKey`.

Values are always declared in the top-level scope, even if they contain the compounding character ‘.’, however, you cannot declare “X” if you have already declared “X.Y” and vice versa.

`undeclare`(value)
Removes a value from the named scope.

Values can only be removed from the top-level scope.

**class** `pyslet.odata2.csd1.DictionaryLike`

**Bases:** `object`

An abstract class for behaving like a dictionary.

Python 3 note: the dictionary interface has changed in Python 3 with the introduction of the dictionary view object and the corresponding change in behaviour of the keys, values and items methods. This class has not changed so is currently Python 2 dictionary like only. It is envisaged that when Pyslet is extended to include support for OData 4 a more Python3-friendly class will be used.

Derived classes must override `__iter__()` and `__getitem__()` and if the dictionary is writable `__setitem__()` and probably `__delitem__()` too. These methods all raise :NotImplementedError by default.

Derived classes should also override `__len__()` and `clear()` as the default implementations are inefficient.

A note on thread safety. Unlike native Python dictionaries, DictionaryLike objects can not be treated as thread safe for updates. The implementations of the read-only methods (including the iterators) are designed to be thread safe so, once populated, they can be safely shared. Derived classes should honour this contract when implementing `__iter__()`, `__getitem__()` and `__len__()` or clearly document that the object is not thread-safe at all.

Finally, one other difference worth noting is touched on in a comment from the following question on Stack Overflow: [http://stackoverflow.com/questions/3358770/python-dictionary-is-thread-safe](http://stackoverflow.com/questions/3358770/python-dictionary-is-thread-safe)

This question is about whether a dictionary can be modified during iteration. Although not typically a thread-safety issue the commenter says:

I think they are related. What if one thread iterates and the other modifies the dict?

To recap, native Python dictionaries limit the modifications you can make during iteration, quoting from the docs:

The dictionary `p` should not be mutated during iteration. It is safe (since Python 2.1) to modify the values of the keys as you iterate over the dictionary, but only so long as the set of keys does not change.
You should treat DictionaryLike objects with the same respect but the behaviour is not defined at this abstract class level and will vary depending on the implementation. Derived classes are only dictionary-like, they are not actually Python dictionaries!

```python
__getitem__(key)
Implements self[key]
This method must be overridden to make a concrete implementation

__setitem__(key, value)
Implements assignment to self[key]
This method must be overridden if you want your dictionary-like object to be writable.

__delitem__(key)
Implements del self[key]
This method should be overridden if you want your dictionary-like object to be writable.

__iter__()
Returns an object that implements the iterable protocol on the keys
This method must be overridden to make a concrete implementation

__len__()
Implements len(self)
The default implementation simply counts the keys returned by __iter__ and should be overridden with a more efficient implementation if available.

__contains__(key)
Implements: key in self
The default implementation uses __getitem__ and returns False if it raises a KeyError.

iterkeys()
Returns an iterable of the keys, simple calls __iter__

itervalues()
Returns an iterable of the values.
The default implementation is a generator function that iterates over the keys and uses __getitem__ to yield each value.

keys()
Returns a list of keys.
This is a copy of the keys in no specific order. Modifications to this list do not affect the object. The default implementation uses iterkeys()

values()
Returns a list of values.
This is a copy of the values in no specific order. Modifications to this list do not affect the object. The default implementation uses itervalues()

iteritems()
Returns an iterable of the key,value pairs.
The default implementation is a generator function that uses __iter__() and __getitem__ to yield the pairs.

items()
Returns a list of key,value pair tuples.
```
This is a copy of the items in no specific order. Modifications to this list do not affect the object. The default implementation uses `iteritems`.

**has_key**(key)
Equivalent to: `key in self`

**get**(key, default=None)
Equivalent to: `self[key]` if key in self else default.
Implemented using `__getitem__`

**setdefault**(key, value=None)
Equivalent to: `self[key]` if key in self else value; ensuring `self[key]=value`
Implemented using `__getitem__` and `__setitem__`.

**pop**(key, value=None)
Equivalent to: `self[key]` if key in self else value; ensuring key not in self.
Implemented using `__getitem__` and `__delitem__`.

**clear**()
Removes all items from the object.

The default implementation uses `keys()` and deletes the items one-by-one with `__delitem__`. It does this to avoid deleting objects while iterating as the results are generally undefined. A more efficient implementation is recommended.

**popitem**()
Equivalent to: `self[key]` for some random key; removing key.

This is a rather odd implementation but to avoid iterating over the whole object we create an iterator with `__iter__`, use `__getitem__` once and then discard it. If an object is found we use `__delitem__` to delete it, otherwise `KeyError` is raised.

**bigclear**()
Removes all the items from the object (alternative for large dictionary-like objects).

This is an alternative implementation more suited to objects with very large numbers of keys. It uses `popitem()` repeatedly until `KeyError` is raised. The downside is that `popitem` creates (and discards) one iterator object for each item it removes. The upside is that we never load the list of keys into memory.

**copy**()
Makes a shallow copy of this object.

This method must be overridden if you want your dictionary-like object to support the copy operation.

**update**(items)
Iterates through `items` using `__setitem__` to add them to the set.

**weakref**
list of weak references to the object (if defined)

### Exceptions

**class** `pyslet.odata2.csd1.NonExistentEntity`
**Bases:** `pyslet.odata2.csd1.EDMError`

Raised when attempting to perform a restricted operation on an entity that doesn’t exist yet. For example, getting the value of a navigation property.
class pyslet.odata2.csdl.EntityExists
    Bases: pyslet.odata2.csdl.EDMError
    Raised when attempting to perform a restricted operation on an entity that already exists. For example, inserting it into the base collection.

class pyslet.odata2.csdl.ConstraintError
    Bases: pyslet.odata2.csdl.EDMError
    General error raised when a constraint has been violated.

class pyslet.odata2.csdl.NavigationError
    Bases: pyslet.odata2.csdl.ConstraintError
    Raised when attempting to perform an operation on an entity and a violation of a navigation property’s relationship is encountered. For example, adding multiple links when only one is allowed or failing to add a link when one is required.

class pyslet.odata2.csdl.ConcurrencyError
    Bases: pyslet.odata2.csdl.ConstraintError
    Raised when attempting to perform an update on an entity and a violation of a concurrency control constraint is encountered.

class pyslet.odata2.csdl.ModelIncomplete
    Bases: pyslet.odata2.csdl.ModelError
    Raised when a model element has a missing reference.
    For example, an EntitySet that is bound to an undeclared : EntityType.

class pyslet.odata2.csdl.ModelConstraintError
    Bases: pyslet.odata2.csdl.ModelError
    Raised when an issue in the model other than completeness prevents an action being performed.
    For example, an entity type that is dependent on two unbound principals (so can never be inserted).

class pyslet.odata2.csdl.DuplicateName
    Bases: pyslet.odata2.csdl.ModelError
    Raised by NameTableMixin when attempting to declare a name in a context where the name is already declared.
    This might be raised if your metadata document incorrectly defines two objects with the same name in the same scope, for example

class pyslet.odata2.csdl.IncompatibleNames
    Bases: pyslet.odata2.csdl.DuplicateName
    A special type of DuplicateName exception raised by NameTableMixin when attempting to declare a name which might hide, or be hidden by, another name already declared.
    CSDL’s definition of SimpleIdentifier allows ‘.’ to be used in names but also uses it for qualifying names. As a result, it is possible to define a scope with a name like “My.Scope” which precludes the later definition of a scope called simply “My” (and vice versa).

class pyslet.odata2.csdl.InvalidMetadataDocument
    Bases: pyslet.odata2.csdl.ModelError
    Raised by general CSDL model violations.

class pyslet.odata2.csdl.EDMError
    Bases: exceptions.Exception

General exception for all CSDL model errors.

Constants

```python
```

Namespace to use for CSDL elements

### 4.3.2 OData Core Classes

This module extends the definitions in `pyslet.odata2.csdl` with OData-specific functions and classes. In most cases you won’t need to worry about which layer of the model a definition belongs to. Where a class is derived from one in the parent EDM the same name is used, therefore most of the time you should look to include items from the core module rather than from the base csdl module.

**Data Model**

```python
class pyslet.odata2.core.EntityCollection(entity_set, **kwargs)
    Bases: pyslet.odata2.csdl.EntityCollection

    EntityCollections that provide OData-specific options

    Our definition of EntityCollection is designed for use with Python’s diamond inheritance model. We inherit directly from the basic `pyslet.odata2.csdl.EntityCollection` object, providing additional methods that support the expression model defined by OData, media link entries and JSON encoding.

    `get_next_page_location()`
    Returns the location of this page of the collection
    The result is a `rfc2396.URI` instance.

    `new_entity()`
    Returns an OData aware instance

    `is_medialink_collection()`
    Returns True if this is a collection of Media-Link Entries

    `new_stream(src, sinfo=None, key=None)`
    Creates a media resource.

    `src` A file-like object from which the stream’s data will be read.

    `sinfo` A `StreamInfo` object containing metadata about the stream. If the `size` field of `sinfo` is set then at most `sinfo.size` bytes are read from `src`. Otherwise `src` is read until the end of the file.

    `key` The key associated with the stream being written. This value is taken as a suggestion for the key to use, its use is not guaranteed. The key actually used to store the stream can be obtained from the resulting entity.

    Returns the media-link entry `Entity`

    `update_stream(src, key, sinfo=None)`
    Updates an existing media resource.

    The parameters are the same as `new_stream()` except that the key must be present and must be an existing key in the collection.

    `read_stream(key, out=None)`
    Reads a media resource.
```
key  The key associated with the stream being read.

out  An optional file like object to which the stream’s data will be written. If no output file is provided
    then no data is written.

The return result is the StreamInfo class describing the stream.

read_stream_close (key)
Creates a generator for a media resource.

key  The key associated with the stream being read.

The return result is a tuple of the StreamInfo class describing the stream and a generator that yields the
stream’s data.

The collection is closed by the generator when the iteration is complete (or when the generator is de-
stroyed).

calculate_order_key (entity, order_object)
Evaluates order_object as an instance of py:class:CommonExpression.

class pyslet.odata2.core.Entity (entity_set)
Bases: pyslet.odata2.csdl.Entity
We override Entity in order to provide OData serialisation.

link_json ()
Returns a JSON-serialised link to this entity

class pyslet.odata2.core.StreamInfo (type=MediaType('application', 'octet-stream', {}), cre-
ated=None, modified=None, size=None)
Bases: object
Represents information about a media resource stream.

- **type**: None
  - The media type, a `MediaType` instance

- **created**: None
  - The optional creation time, a fully specified `TimePoint` instance that includes a zone

- **modified**: None
  - The optional modification time, a fully specified `TimePoint` instance that includes a zone

- **size**: None
  - The size of the stream (in bytes), None if not known

- **md5**: None
  - The 16 byte binary MD5 checksum of the stream, None if not known

### Navigation: Deferred Values

**class** `pyslet.odata2.core.NavigationCollection(from_entity, name, **kwargs)`

- **Bases**: `pyslet.odata2.core.EntityCollection`, `pyslet.odata2.csdl.NavigationCollection`

NavigationCollections that provide OData-specific options.

This class uses Python’s diamond inheritance model.

![Diagram showing method resolution order]

This allows us to inherit from both the OData-specific form of EntityCollection and NavigationCollection. This is illustrated in the above diagram which shows the method resolution order reading from the bottom of the diagram. The default object is omitted.

This technique is repeated in specific implementations of the API where common collection behaviour is implemented in a class that inherits from `EntityCollection` and then mixed in to a new class derived from `NavigationCollection`.

**expand_collection()**
- Return an expanded version of this collection
  - Returns an instance of an OData-specific `ExpandedEntityCollection`.

**get_location()**
- Returns the location of this collection as a `rfc2396.URI` instance.
  - We override the location based on the source entity set + the fromKey.

**class** `pyslet.odata2.coreExpandedEntityCollection(entity_list, **kwargs)`

- **Bases**: `pyslet.odata2.core.EntityCollection`, `pyslet.odata2.csdlExpandedEntityCollection`

Expanded collections with OData-specific behaviour.

This class uses diamond inheritance in a similar way to `NavigationCollection`.

---

4.3. OData Reference
4.3.3 OData Metadata Classes

This module defines sub-classes of those defined in the EDM that include special handling of the OData defined metadata attributes.

EDM Elements

Feed Customisation

class pyslet.odata2.metadata.EntityType(parent)
Bases: pyslet.odata2.csdl.EntityType, pyslet.odata2.metadata.FeedCustomisationMixin

Supports feed customisation behaviour of EntityTypes

get_source_path()  
Returns the source path

This result is read from the FC_SourcePath attribute. It is a list of property names that represents a path into the entity or None if there is no source path set.

has_stream()  
Returns true if this is a media link resource.

Read from the HasStream attribute. The default is False.

GetSourcePath(*args, **kwargs)  
Deprecated equivalent to get_source_path()

HasStream(*args, **kwargs)  
Deprecated equivalent to has_stream()

class pyslet.odata2.metadata.Property(parent)
Bases: pyslet.odata2.csdl.Property, pyslet.odata2.metadata.FeedCustomisationMixin

Supports feed customisation behaviour of Properties

get_mime_type()  
Returns the media type of a property

The result is read from the MimeType attribute. It is a MediaType instance or None if the attribute is not defined.

GetMimeType(*args, **kwargs)  
Deprecated equivalent to get_mime_type()

class pyslet.odata2.metadata.FeedCustomisationMixin
Bases: pyslet.pep8.MigratedClass

Utility class used to add common feed customisation attributes

get_target_path()  
Returns the target path for an element

The result is a list of qualified element names, that is, tuples of (namespace,name). The last name may start with ‘@’ indicating an attribute rather than an element.

Feed customisations are declared using the FC_TargetPath attribute. Returns None if there is no target path declared.
keep_in_content ()
Returns true if a property value should be kept in the content
This is indicated with the FC_KeepInContent attribute. If the attribute is missing then False is returned, so
properties with custom paths default to being omitted from the properties list.

get_fc_ns_prefix ()
Returns the custom namespace mapping to use.
The value is read from the FC_NsPrefix attribute. The result is a tuple of: (prefix, namespace uri).
If no mapping is specified then (None, None) is returned.

GetFCNsPrefix (*args, **kwargs)
Deprecated equivalent to get_fc_ns_prefix()

GetTargetPath (*args, **kwargs)
Deprecated equivalent to get_target_path()

KeepInContent (*args, **kwargs)
Deprecated equivalent to keep_in_content()

Entity Containers

class pyslet.odata2.metadata.EntityContainer (parent)
Bases: pyslet.odata2.csdl.EntityContainer
Supports OData’s concept of the default container.

is_default_entity_container ()
Returns True if this is the default entity container
The value is read from the IsDefaultEntityContainer attribute. The default is False.

IsDefaultEntityContainer (*args, **kwargs)
Deprecated equivalent to is_default_entity_container()

class pyslet.odata2.metadata.EntitySet (parent)
Bases: pyslet.odata2.csdl.EntitySet

set_location ()
Overridden to add support for the default entity container
By default, the path to an EntitySet includes the name of the container it belongs to, e.g., My-
Database.MyTable. This implementation checks to see if we in the default container and, if so, omissions
the container name prefix before setting the location URI.

EDMX Elements

class pyslet.odata2.metadata.Document (**args)
Bases: pyslet.odata2.edmx.Document
Class for working with OData-specific metadata documents.
Adds namespace prefix declarations for the OData metadata and OData dataservices namespaces.

classmethod get_element_class (name)
Returns the class used to represent an element.
Overrides get_element_class() to use the OData-specific implementations of the edmxcsl classes
defined in this module.
**validate()**

Validates any declared OData extensions

Checks many of the requirements given in the specification and raises `InvalidMetadataDocument` if the tests fail.

Returns the data service version required to process the service or None if no data service version is specified.

**Validate(** *args, **kwargs*)

Deprecated equivalent to `validate()`

```python
class pyslet.odata2.metadata.DataServices(parent):
    Bases: pyslet.odata2.edmx.DataServices
    Adds OData specific behaviour
    defaultContainer = None
    the default entity container
    data_services_version()
        Returns the data service version
        Read from the DataServiceVersion attribute. Defaults to None.
    search_containers(name)
        Returns an entity set or service operation with name
        name must be of the form:
        
        ![entity container>.]<entity set, function or operation name>
        
        The entity container must be present unless the target is in the default container in which case it must not be present.
        
        If name can’t be found KeyError is raised.
    DataServicesVersion(**args, **kwargs)
        Deprecated equivalent to `data_services_version()`
    SearchContainers(**args, **kwargs)
        Deprecated equivalent to `search_containers()`
```

### 4.3.4 OData Client

**Overview**

Warning: this client doesn’t support certificate validation when accessing servers through https URLs. This feature is coming soon...

**Using the Client**

The client implementation uses Python’s logging module to provide logging, when learning about the client it may help to turn logging up to “INFO” as it makes it clearer what the client is doing. “DEBUG” would show exactly what is passing over the wire.:

```python
>>> import logging
>>> logging.basicConfig(level=logging.INFO)
```
To create a new client simply instantiate a Client object. You can pass the URL of the service root you wish to connect to directly to the constructor which will then call the service to download the list of feeds and the metadata document from which it will set the `Client.model`.

```python
>>> from pyslet.odata2.client import Client
>>> c=Client("http://services.odata.org/V2/Northwind/Northwind.svc/")
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/ HTTP/1.1
INFO:root:Finished Response, status 200
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/$metadata HTTP/1.1
INFO:root:Finished Response, status 200

```

The `Client.feeds` attribute is a dictionary mapping the exposed feeds (by name) onto `EntitySet` instances. This makes it easy to open the feeds as EDM collections. In your code you’d typically use the with statement when opening the collection but for clarity we’ll continue on the python command line:

```python
>>> products=c.feeds['Products'].open()
>>> for p in products: print p
...
1
2
3
... [and so on]
...
20
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products?$skiptoken=20 HTTP/1.1
INFO:root:Finished Response, status 200
21
22
23
... [and so on]
...
76
77
```

Note that products behaves like a dictionary, iterating through it iterates through the keys in the dictionary. In this case these are the keys of the entities in the collection of products. Notice that the client logs several requests to the server interspersed with the printed output. Subsequent requests use $skiptoken because the server is limiting the maximum page size. These calls are made as you iterate through the collection allowing you to iterate through very large collections.

The keys alone are of limited interest, let’s try a similar loop but this time we’ll print the product names as well:

```python
>>> for k,p in products.iteritems(): print k,p['ProductName'].value
...
```

4.3. OData Reference
Sir Rodney’s Scones sound interesting, we can grab an individual record in the usual way:

```python
>>> scones=products[21]
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(21) HTTP/1.1
INFO:root:Finished Response, status 200
>>> for k,v in scones.data_items(): print k,v.value
...
ProductID 21
ProductName Sir Rodney's Scones
SupplierID 8
CategoryID 3
QuantityPerUnit 24 pkgs. x 4 pieces
UnitPrice 10.0000
UnitsInStock 3
UnitsOnOrder 40
ReorderLevel 5
Discontinued False
```

Well, I’ve simply got to have some of these, let’s use one of the navigation properties to load information about the supplier:

```python
>>> supplier=scones['Supplier'].get_entity()
INFO:root:Sending request to services.odata.org
INFO:root:GET /V2/Northwind/Northwind.svc/Products(21)/Supplier HTTP/1.1
INFO:root:Finished Response, status 200
>>> for k,v in supplier.data_items(): print k,v.value
...
SupplierID 8
CompanyName Specialty Biscuits, Ltd.
ContactName Peter Wilson
ContactTitle Sales Representative
Address 29 King's Way
City Manchester
Region None
PostalCode M14 GSD
Country UK
Phone (161) 555-4448
Fax None
HomePage None
```

Attempting to load a non existent entity results in a KeyError of course:
Finally, when we’re done, it is a good idea to close the open collection:

```python
>>> products.close()
```

**Reference**

```python
class pyslet.odata2.client.Client(service_root=None, **kwargs)
    Bases: pyslet.rfc5023.Client
    An OData client.
    Can be constructed with an optional URL specifying the service root of an OData service. The URL is passed
directly to `load_service()`.

    service = None
        a `pyslet.rfc5023.Service` instance describing this service
    feeds = None
        a dictionary of feed titles, mapped to `csdl.EntitySet` instances
    model = None
        a `metadata.Edmx` instance containing the model for the service

load_service(service_root, metadata=None)
    Configures this client to use the service at `service_root`

    service_root A string or `pyslet.rfc2396.URI` instance. The URI may now point to a local file
        though this must have an xml:base attribute to point to the true location of the service as calls to the
        feeds themselves require the use of http(s).
    metadata (None) A `pyslet.rfc2396.URI` instance pointing to the metadata file. This is usually
derived automatically by adding $metadata to the service root but some services have inconsistent
metadata models. You can download a copy, modify the model and use a local copy this way instead,
e.g., by passing something like:

        `URI.from_path('metadata.xml')`

    If you use a local copy you must add an xml:base attribute to the root element indicating the true
    location of the $metadata file as the client uses this information to match feeds with the metadata
    model.

LoadService (*args, **kwargs)
    Deprecated equivalent to `load_service()`
```

4.3. OData Reference  203
Exceptions

class pyslet.odata2.client.ClientException
    Bases: exceptions.Exception
    
    Base class for all client-specific exceptions.

class pyslet.odata2.client.AuthorizationRequired
    Bases: pyslet.odata2.client.ClientException
    
    The server returned a response code of 401 to the request.

class pyslet.odata2.client.UnexpectedHTTPResponse
    Bases: pyslet.odata2.client.ClientException
    
    The server returned an unexpected response code, typically a 500 internal server error. The error message contains details of the error response returned.

4.3.5 An In-Memory Data Service

4.3.6 SQL Database-based Data Services

This module defines a general (but abstract) implementation of the EDM-based data-access-layer (DAL) using Python’s DB API: http://www.python.org/dev/peps/pep-0249/

It also contains a concrete implementation derived from the above that uses the standard SQLite module for storage. For more information about SQLite see: http://www.sqlite.org/

Data Access Layer API

Entity Containers

There are primarily two use cases here:

1. Create a derived class of $\texttt{SQLEntityContainer}$ to provide platform specific modifications to the way SQL queries are constructed and database connections are created and managed.

2. Create a derived class of $\texttt{SQLEntityContainer}$ to provide modified name mappings for a specific database and metadata model.

These two use cases can be supported through multiple (diamond) inheritance. This makes it easier for you to separate the code required. In practice, implementations for different database platforms are likely to be shared (perhaps as part of future releases of Pyslet itself) whereas modifications to the name mangler to map this API to an existing database will be project specific.

For example, to achieve platform specific modifications you’ll override $\texttt{SQLEntityContainer.get\_collection\_class()}$:

```python
class MyDBContainer(SQLEntityContainer):
    
    def get_collection_class(self):
        return MyDBEntityCollection
```

To achieve modified property name mappings you’ll override $\texttt{SQLEntityContainer.mangle\_name()}$:
class SouthwindDBContainer(SQLEntityContainer):
    def mangle_name(self, source_path):
        # do some custom name mangling here....

Normally, you’ll want to achieve both use cases, so to actually instantiate your database you’ll select the container class that represents the database platform and then combine it with the class that contains your data-specific modifications:

```python
import MyDB, Southwind
# easy to configure constants at the top of your script
DBCONTAINER_CLASS=MyDB.MyDBContainer
DBCONTAINER_ARGS={
    'username': 'southwind',
    'password': 'password'
}
MAX_CONNECTIONS=100

class SouthwindDB(Southwind.SouthwindDBContainer, DBCONTAINER_CLASS):
    pass

# .... load the metadata from disk and then do something like this
db=SouthwindDB(container=SouthwindMetadata, max_connections=MAX_CONNECTIONS, **DBCONTAINER_ARGS)
```

class pyslet.odata2.sqlds.SQLEntityContainer(
    container, dbapi, streamstore=None, max_connections=10,
    field_name_joiner='_', max_idle=None, **kwargs)

Bases: object

Object used to represent an Entity Container (aka Database).

Keyword arguments on construction:

**container** The `EntityContainer` that defines this database.

**streamstore** An optional `StreamStore` that will be used to store media resources in the container. If absent, media resources actions will generate `NotImplementedError`.

**dbapi** The DB API v2 compatible module to use to connect to the database.

This implementation is compatible with modules regardless of their thread-safety level (provided they declare it correctly!).

**max_connections** (optional) The maximum number of connections to open to the database. If your program attempts to open more than this number (defaults to 10) then it will block until a connection becomes free. Connections are always shared within the same thread so this argument should be set to the expected maximum number of threads that will access the database.

If using a module with thread-safety level 0 `max_connections` is ignored and is effectively 1, so use of the API is then best confined to single-threaded programs. Multi-threaded programs can still use the API but it will block when there is contention for access to the module and context switches will force the database connection to be closed and reopened.

**field_name_joiner** (optional) The character used by the name mangler to join compound names, for example, to obtain the column name of a complex property like “Address/City”. The default is “_”, resulting in names like “Address_City” but it can be changed here. Note: all names are quoted using `quote_identifier()` before appearing in SQL statements.
max_idle (optional)  The maximum number of seconds idle database connections should be kept open before they are cleaned by the `pool_cleaner()` method. The default is `None` which means that the pool_cleaner never runs. Any other value causes a separate thread to be created to run the pool cleaner passing the value of the parameter each time. The frequency of calling the pool_cleaner method is calculated by dividing max_idle by 5, but it never runs more than once per minute. For example, a setting of 3600 (1 hour) will result in a pool cleaner call every 12 minutes.

This class is designed to work with diamond inheritance and super. All derived classes must call `__init__` through super and pass all unused keyword arguments. For example:

```python
class MyDBContainer:
    def __init__(self, myDBConfig, **kwargs):
        super(MyDBContainer, self).__init__(**kwargs)
        # do something with myDBConfig....
```

```
streamstore = None
the EntityContainer
```

```
dbapi = None
the optional StreamStore
```

```
module_lock = None
the DB API compatible module
```

```
fk_table = None
A mapping from an entity set name to a FK mapping of the form:

```python
dict(<association set end>: (<nullable flag>, <unique keys flag>),...)
```

The outer mapping has one entry for each entity set (even if the corresponding foreign key mapping is empty).

Each foreign key mapping has one entry for each foreign key reference that must appear in that entity set’s table. The key is an `AssociationSetEnd` that is bound to the entity set (the other end will be bound to the target entity set). This allows us to distinguish between the two ends of a recursive association.

```
aux_table = None
A mapping from the names of symmetric association sets to a tuple of:

```python
dict(<entity set A>, <name prefix A>, <entity set B>,
     <name prefix B>, <unique keys>)
```

```
mangled_names = None
A mapping from source path tuples to mangled and quoted names to use in SQL queries. For example:

```python
{'Customer': '"Customer"'
('Customer', 'Address', 'City') : "Address_City"
('Customer', 'Orders') : "Customer_Orders"
```

Note that the first element of the tuple is the entity set name but the default implementation does not use this in the mangled name for primitive fields as they are qualified in contexts where a name clash is possible. However, mangled navigation property names do include the table name prefix as they used as pseudo-table names.

```
field_name_joiner = None
Default string used to join complex field names in SQL queries, e.g. Address_City
```

```
ro_names = None
The set of names that should be considered read only by the SQL insert and update generation code. The
```

206 Chapter 4. The Open Data Protocol (OData)
items in the set are source paths, as per `mangled_names`. The set is populated on construction using the `ro_name()` method.

**mangle_name(source_path)**

Mangles a source path into a quoted SQL name

This is a key extension point to use when you are wrapping an existing database with the API. It allows you to control the names used for entity sets (tables) and properties (columns) in SQL queries.

**source_path** A tuple or list of strings describing the path to a property in the metadata model.

For entity sets, this is a tuple with a single entry in it, the entity set name.

For data properties this is a tuple containing the path, including the entity set name e.g., (“Customers”, “Address”, “City”) for the City property in a complex property ‘Address’ in entity set “Customers”.

For navigation properties the tuple is the navigation property name prefixed with the entity set name, e.g., (“Customers”, “Orders”). This name is only used as a SQL alias for the target table, to remove ambiguity from certain queries that include a join across the navigation property. The mangled name must be distinct from the entity set name itself. from other such aliases and from other column names in this table.

Foreign key properties contain paths starting with both the entity set and the association set names (see `SQLForeignKeyCollection` for details) unless the association is symmetric, in which case they also contain the navigation property name (see `SQLAssociationCollection` for details of these more complex cases).

The default implementation strips the entity set name away and uses the default joining character to create a compound name before calling `quote_identifier()` to obtain the SQL string. All names are mangled once, on construction, and from then on looked up in the dictionary of mangled names.

If you need to override this method to modify the names used in your database you should ensure all other names (including any unrecognized by your program) are passed to the default implementation for mangling.

**ro_name(source_path)**

Test if a source_path identifies a read-only property

This is an additional extension point to use when you are wrapping an existing database with the API. It allows you to manage situations where an entity property has an implied value and should be treated read only.

There are two key use cases, auto-generated primary keys (such as auto-increment integer keys) and foreign keys which are exposed explicitly as foreign keys and should only be updated through an associated navigation property.

**source_path** A tuple or list of strings describing the path to a property in the metadata model. See `mangle_name()` for more information.

The default implementation returns False.

If you override this method you must ensure all other names (including any unrecognized by your program) are passed to the default implementation using super.

**source_path_generator(entity_set)**

Utility generator for source path tuples for entity_set

**get_collection_class()**

Returns the collection class used to represent a generic entity set.

Override this method to provide a class derived from `SQLEntityCollection` when you are customising this implementation for a specific database engine.
**get_symmetric_navigation_class()**
Returns the collection class used to represent a symmetric relation.

Override this method to provide a class derived from `SQLAssociationCollection` when you are customising this implementation for a specific database engine.

**get_fk_class()**
Returns the class used when the FK is in the source table.

Override this method to provide a class derived from `SQLForeignKeyCollection` when you are customising this implementation for a specific database engine.

**get_rk_class()**
Returns the class used when the FK is in the target table.

Override this method to provide a class derived from `SQLReverseKeyCollection` when you are customising this implementation for a specific database engine.

**create_all_tables(out=None)**
Creates all tables in this container.

    out  An optional file-like object. If given, the tables are not actually created, the SQL statements are written to this file instead.

Tables are created in a sensible order to ensure that foreign key constraints do not fail but this method is not compatible with databases that contain circular references though, e.g., Table A -> Table B with a foreign key and Table B -> Table A with a foreign key. Such databases will have to be created by hand. You can use the `create_table_query` methods to act as a starting point for your script.

**drop_all_tables(out=None)**
Drops all tables in this container.

Tables are dropped in a sensible order to ensure that foreign key constraints do not fail, the order is essentially the reverse of the order used by `create_all_tables()`.

**connection_stats()**
Returns information about the connection pool

    Returns a triple of:

    nlocked  the number of connections in use by all threads.

    nunlocked  the number of connections waiting

    nidle  the number of dead connections

Connections are placed in the ‘dead pool’ when unexpected lock failures occur or if they are locked and the owning thread is detected to have terminated without releasing them.

**pool_cleaner(max_idle=900.0)**
Cleans up the connection pool

    max_idle (float)  Optional number of seconds beyond which an idle connection is closed. Defaults to 10 times the `SQL_TIMEOUT`.

**open()**
Creates and returns a new connection object.

Must be overridden by database specific implementations because the underlying DB ABI does not provide a standard method of connecting.

**close_connection(connection)**
Calls the underlying close method.
break_connection \((connection)\)
Called when closing or cleaning up locked connections.

This method is called when the connection is locked (by a different thread) and the caller wants to force that thread to relinquish control.

The assumption is that the database is stuck in some lengthy transaction and that break_connection can be used to terminate the transaction and force an exception in the thread that initiated it - resulting in a subsequent call to release_connection() and a state which enables this thread to acquire the connection’s lock so that it can close it.

The default implementation does nothing, which might cause the close method to stall until the other thread relinquishes control normally.

close \((timeout=5)\)
Closes this database.

This method goes through each open connection and attempts to acquire it and then close it. The object is put into a mode that disables acquire_connection() (it returns None from now on).

\textbf{timeout} Defaults to 5 seconds. If connections are locked by other running threads we wait for those threads to release them, calling break_connection() to speed up termination if possible.

If None (not recommended!) this method will block indefinitely until all threads properly call release_connection().

Any locks we fail to acquire in the timeout are ignored and the connections are left open for the python garbage collector to dispose of.

quote_identifier \((identifier)\)
Given an \textbf{identifier} returns a safely quoted form of it.

By default we strip double quote and then use them to enclose it. E.g., if the string ‘Employee_Name’ is passed then the string ““Employee_Name”” is returned.

prepare_sql_type \((simple_value, params, nullable=None)\)
Given a simple value, returns a SQL-formatted name of its type.

Used to construct CREATE TABLE queries.

\textbf{simple_value} A \texttt{pyslet.odata2.csdl.SimpleValue} instance which should have been created from a suitable \texttt{pyslet.odata2.csdl.Property} definition.

\textbf{params} A \texttt{SQLParams} object. Not used, see prepare_sql_literal()

\textbf{nullable} Optional Boolean that can be used to override the nullable status of the associated property definition.

For example, if the value was created from an Int32 non-nullable property and has default value 0 then this might return the string ‘INTEGER NOT NULL DEFAULT 0’.

You should override this implementation if your database platform requires special handling of certain datatypes. The default mappings are given below.
<table>
<thead>
<tr>
<th>EDM Type</th>
<th>SQL Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edm.Binary</td>
<td>BINARY(MaxLength) if FixedLength specified</td>
</tr>
<tr>
<td>Edm.Binary</td>
<td>VARBINARY(MaxLength) if no FixedLength</td>
</tr>
<tr>
<td>Edm.Boolean</td>
<td>BOOLEAN</td>
</tr>
<tr>
<td>Edm.Byte</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Edm.DateTime</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>Edm.DateTimeOffset</td>
<td>CHARACTER(27), ISO 8601 string representation is used with micro second precision</td>
</tr>
<tr>
<td>Edm.Decimal</td>
<td>DECIMAL(Precision,Scale), defaults 10,0</td>
</tr>
<tr>
<td>Edm.Double</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Edm.Guid</td>
<td>BINARY(16)</td>
</tr>
<tr>
<td>Edm.Int16</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Edm.Int32</td>
<td>INTEGER</td>
</tr>
<tr>
<td>Edm.Int64</td>
<td>BIGINT</td>
</tr>
<tr>
<td>Edm.SByte</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>Edm.Single</td>
<td>REAL</td>
</tr>
<tr>
<td>Edm.String</td>
<td>CHAR(MaxLength) or VARCHAR(MaxLength)</td>
</tr>
<tr>
<td>Edm.String</td>
<td>NCHAR(MaxLength) or NVARCHAR(MaxLength) if Unicode=&quot;true&quot;</td>
</tr>
<tr>
<td>Edm.Time</td>
<td>TIME</td>
</tr>
</tbody>
</table>

Parameterized CREATE TABLE queries are unreliable in my experience so the current implementation of the native create_table methods ignore default values when calling this method.

**prepare_sql_value** *(simple_value)*

Returns a python object suitable for passing as a parameter

*simple_value* A *pyslet.odata2.csdl.SimpleValue* instance.

You should override this method if your database requires special handling of parameter values. The default implementation performs the following conversions

<table>
<thead>
<tr>
<th>EDM Type</th>
<th>Python value added as parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>None</td>
</tr>
<tr>
<td>Edm.Binary</td>
<td>(byte) string</td>
</tr>
<tr>
<td>Edm.Boolean</td>
<td>True or False</td>
</tr>
<tr>
<td>Edm.Byte</td>
<td>int</td>
</tr>
<tr>
<td>Edm.DateTime</td>
<td>Timestamp instance from DB API module</td>
</tr>
<tr>
<td>Edm.DateTimeOffset</td>
<td>string (ISO 8601 basic format)</td>
</tr>
<tr>
<td>Edm.Decimal</td>
<td>Decimal instance</td>
</tr>
<tr>
<td>Edm.Double</td>
<td>float</td>
</tr>
<tr>
<td>Edm.Guid</td>
<td>(byte) string</td>
</tr>
<tr>
<td>Edm.Int16</td>
<td>int</td>
</tr>
<tr>
<td>Edm.Int32</td>
<td>int</td>
</tr>
<tr>
<td>Edm.Int64</td>
<td>long</td>
</tr>
<tr>
<td>Edm.SByte</td>
<td>int</td>
</tr>
<tr>
<td>Edm.Single</td>
<td>float</td>
</tr>
<tr>
<td>Edm.String</td>
<td>(unicode) string</td>
</tr>
<tr>
<td>Edm.Time</td>
<td>Time instance from DB API module</td>
</tr>
</tbody>
</table>

**read_sql_value** *(simple_value, new_value)*

Updates *simple_value* from *new_value.*

*simple_value* A *pyslet.odata2.csdl.SimpleValue* instance.
new_value  A value returned by the underlying DB API, e.g., from a cursor fetch operation

This method performs the reverse transformation to prepare_sql_value() and may need to be over-
ridden to convert new_value into a form suitable for passing to the underlying set_from_value() method.

new_from_sql_value(sql_value)
Returns a new simple value with value sql_value

The return value is a pyslet.odata2.csdl.SimpleValue instance.

sql_value  A value returned by the underlying DB API, e.g., from a cursor fetch operation

This method creates a new instance, selecting the most appropriate type to represent sql_value. By default
pyslet.odata2.csdl.EDMValue.from_value() is used.

You may need to override this method to identify the appropriate value type.

prepare_sql_literal(value)
Formats a simple value as a SQL literal

Although SQL containers use parameterised queries for all INSERT, SELECT, UPDATE and DELETE
queries, CREATE TABLE queries are generally only created when the data model has been designed using
the OData data model and are intended to be exported as SQL scripts for review (and perhaps modification)
by a DBA prior to being run on a real database server as part of initially provisioning a running system. In
this case, default values in the data model must be inserted into the CREATE TABLE query itself and so a
method is provided for transforming values accordingly.

select_limit_clause(skip, top)
Returns a SELECT modifier to limit a query

See limit_clause() for details of the parameters.

Returns a tuple of:

skip  0 if the modifier implements this functionality. If it does not implement this function then the value
passed in for skip must be returned.

modifier A string modifier to insert immediately after the SELECT statement (must be empty or end with
a space).

For example, if your database supports the TOP keyword you might return:

{(skip, 'TOP %i' % top)}

This will result in queries such as:

SELECT TOP 10 FROM ....

More modern syntax tends to use a special limit clause at the end of the query, rather than a SELECT
modifier. The default implementation returns:

{(skip, '')}

...essentially doing nothing.

limit_clause(skip, top)
Returns a limit clause to limit a query

skip  An integer number of entities to skip

top  An integer number of entities to limit the result set of a query or None is no limit is desired.

Returns a tuple of:
skip  0 if the limit clause implements this functionality. If it does not implement this function then the value passed in for skip must be returned.

clause  A limit clause to append to the query. Must be empty or end with a space.

For example, if your database supports the MySQL-style LIMIT and OFFSET keywords you would return (for non-None values of top):

```
(0, 'LIMIT %i OFFSET %i' % (top, skip))
```

This will result in queries such as:

```
SELECT * FROM Customers LIMIT 10 OFFSET 20
```

More modern syntax tends to use a special limit clause at the end of the query, rather than a SELECT modifier. Such as:

```
(skip, 'FETCH FIRST %i ROWS ONLY ' % top)
```

This syntax is part of SQL 2008 standard but is not widely adopted and, for compatibility with existing external database implementation, the default implementation remains blank.

For an example of how to create a platform-specific implementation see SQLite below.

### Entity Collections

These classes are documented primarily to facilitate the creation of alternative implementations designed to run over other DB API based data layers. The documentation goes a bit further than is necessary to help promote an understanding of the way the API is implemented.

**class**  `pyslet.odata2.sqlds.SQLEntityCollection`  (`container`, **kwargs)

Bases: `pyslet.odata2.sqlds.SQLCollectionBase`

Represents a collection of entities from an EntitySet.

This class is the heart of the SQL implementation of the API, constructing and executing queries to implement the core methods from `pyslet.odata2.csd1.EntityCollection`.

**insert_entity**  (`entity`)

Inserts `entity` into the collection.

We override this method, rerouting it to a SQL-specific implementation that takes additional arguments.

**insert_entity_sql**  (`entity`, `from_end=None`, `fk_values=None`, `transaction=None`)

Inserts `entity` into the collection.

This method is not designed to be overridden by other implementations but it does extend the default functionality for a more efficient implementation and to enable better transactional processing. The additional parameters are documented here.

**from_end**  An optional `pyslet.odata2.csd1.AssociationSetEnd` bound to this entity set. If present, indicates that this entity is being inserted as part of a single transaction involving an insert or update to the other end of the association.

This suppresses any check for a required link via this association (as it is assumed that the link is present, or will be, in the same transaction).

**fk_values**  If the association referred to by `from_end` is represented by a set of foreign keys stored in this entity set’s table (see `SQLReverseKeyCollection`) then `fk_values` is the list of (mangled column name, value) tuples that must be inserted in order to create the link.
**transaction**  An optional transaction. If present, the connection is left uncommitted.

The method functions in three phases.

1. Process all bindings for which we hold the foreign key. This includes inserting new entities where deep inserts are being used or calculating foreign key values where links to existing entities have been specified on creation.

   In addition, all required links are checked and raise errors if no binding is present.

2. A simple SQL INSERT statement is executed to add the record to the database along with any foreign keys generated in (1) or passed in `fk_values`.

3. Process all remaining bindings. Although we could do this using the `update_bindings()` method of DeferredValue we handle this directly to retain transactional integrity (where supported).

   Links to existing entities are created using the `insert_link` method available on the SQL-specific `SQLNavigationCollection`.

   Deep inserts are handled by a recursive call to this method. After step 1, the only bindings that remain are (a) those that are stored at the other end of the link and so can be created by passing values for `from_end` and `fk_values` in a recursive call or (b) those that are stored in a separate table which are created by combining a recursive call and a call to `insert_link`.

Required links are always created in step 1 because the overarching mapping to SQL forces such links to be represented as foreign keys in the source table (i.e., this table) unless the relationship is 1-1, in which case the link is created in step 3 and our database is briefly in violation of the model. If the underlying database API does not support transactions then it is possible for this state to persist resulting in an orphan entity or entities, i.e., entities with missing required links. A failed `rollback()` call will log this condition along with the error that caused it.

**update_entity** *(entity, merge=True)*

Updates entity

This method follows a very similar pattern to `InsertMethod()`, using a three-phase process.

1. **Process all bindings for which we hold the foreign key.** This includes inserting new entities where deep inserts are being used or calculating foreign key values where links to existing entities have been specified on update.

2. **A simple SQL UPDATE statement is executed to update the** record in the database along with any updated foreign keys generated in (1).

3. **Process all remaining bindings while retaining transactional** integrity (where supported).

   Links to existing entities are created using the `insert_link` or `replace` methods available on the SQL-specific `SQLNavigationCollection`. The replace method is used when a navigation property that links to a single entity has been bound. Deep inserts are handled by calling `insert_entity_sql` before the link is created.

The same transactional behaviour as `insert_entity_sql()` is exhibited.

**update_link** *(entity, link_end, target_entity, no_replace=False, transaction=None)*

Updates a link when this table contains the foreign key

**entity**  The entity being linked from (must already exist)

**link_end**  The `AssociationSetEnd` bound to this entity set that represents this entity set’s end of the association being modified.

**target_entity**  The entity to link to or None if the link is to be removed.
no_replace If True, existing links will not be replaced. The affect is to force the underlying SQL query to include a constraint that the foreign key is currently NULL. By default this argument is False and any existing link will be replaced.

transaction An optional transaction. If present, the connection is left uncommitted.

delete_entity (entity, from_end=None, transaction=None)
Deletes an entity

Called by the dictionary-like del operator, provided as a separate method to enable it to be called recursively when doing cascade deletes and to support transactions.

from_end An optional AssociationSetEnd bound to this entity set that represents the link from which we are being deleted during a cascade delete.

The purpose of this parameter is prevent cascade deletes from doubling back on themselves and causing an infinite loop.

transaction An optional transaction. If present, the connection is left uncommitted.

delete_link (entity, link_end, target_entity, transaction=None)
Deletes the link between entity and target_entity

The foreign key for this link must be held in this entity set’s table.

entity The entity in this entity set that the link is from.

link_end The AssociationSetEnd bound to this entity set that represents this entity set’s end of the association being modified.

target_entity The target entity that defines the link to be removed.

transaction An optional transaction. If present, the connection is left uncommitted.

clear_links (link_end, target_entity, transaction=None)
Deletes all links to target_entity

The foreign key for this link must be held in this entity set’s table.

link_end The AssociationSetEnd bound to this entity set that represents this entity set’s end of the association being modified.

target_entity The target entity that defines the link(s) to be removed.

transaction An optional transaction. If present, the connection is left uncommitted.

create_table_query ()
Returns a SQL statement and params for creating the table.

create_table ()
Executes the SQL statement create_table_query ()

drop_table_query ()
Returns a SQL statement for dropping the table.

drop_table ()
Executes the SQL statement drop_table_query ()

class pyslet.odata2.sqlqs.SQLCollectionBase (container, **kwargs)
Bases: pyslet.odata2.core.EntityCollection
A base class to provide core SQL functionality.

Additional keyword arguments:

container A SQLEntityContainer instance.
On construction a data connection is acquired from *container*, this may prevent other threads from using the database until the lock is released by the `close()` method.

**DEFAULT_VALUE = True**

A boolean indicating whether or not the collection supports the syntax:

```sql
UPDATE "MyTable" SET "MyField"=DEFAULT
```

Most databases do support this syntax but SQLite does not. In cases where this is False, default values are set explicitly as they are defined in the metadata model instead. If True then the default values defined in the metadata model are ignored by the collection object.

**container = None**

the parent container (database) for this collection

**connection = None**

a connection to the database acquired with `SQLEntityContainer.acquire_connection()`

**close()**

Closes the cursor and database connection if they are open.

**set_page(top, skip=0, skiptoken=None)**

Sets the values for paging.

Our implementation uses a special format for *skiptoken*. It is a comma-separated list of simple literal values corresponding to the values required by the ordering augmented with the key values to ensure uniqueness.

For example, if `$orderby=A,B` on an entity set with key `K` then the skiptoken will typically have three values comprising the last values returned for `A,B` and `K` in that order. In cases where the resulting skiptoken would be unreasonably large an additional integer (representing a further skip) may be appended and the whole token expressed relative to an earlier skip point.

**reset_joins()**

Sets the base join information for this collection

**add_join(name)**

Adds a join to this collection

- **name** The name of the navigation property to traverse.

  The return result is the alias name to use for the target table.

  As per the specification, the target must have multiplicity 1 or 0..1.

**join_clause()**

A utility method to return the JOIN clause.

Defaults to an empty expression.

**where_clause(entity, params, use_filter=True, use_skip=False, null_cols=())**

A utility method that generates the WHERE clause for a query

- **entity** An optional entity within this collection that is the focus of this query. If not None the resulting WHERE clause will restrict the query to this entity only.

- **params** The `SQLParams` object to add parameters to.

- **use_filter** Defaults to True, indicates if this collection’s filter should be added to the WHERE clause.

- **use_skip** Defaults to False, indicates if the skiptoken should be used in the where clause. If True then the query is limited to entities appearing after the skiptoken’s value (see below).

- **null_cols** An iterable of mangled column names that must be NULL (defaults to an empty tuple). This argument is used during updates to prevent the replacement of non-NULL foreign keys.
The operation of the skiptoken deserves some explanation. When in play the skiptoken contains the last value of the order expression returned. The order expression always uses the keys to ensure unambiguous ordering. The clause added is best served with an example. If an entity has key K and an order expression such as “tolower(Name) desc” then the query will contain something like:

```
SELECT K, Name, DOB, LOWER(Name) AS o_1, K ...
WHERE (o_1 < ? OR (o_1 = ? AND K > ?))
```

The values from the skiptoken will be passed as parameters.

**where_entity_clause** *(where, entity, params)*

Adds the entity constraint expression to a list of SQL expressions.

- **where** The list to append the entity expression to.
- **entity** An expression is added to restrict the query to this entity

**where_skiptoken_clause** *(where, params)*

Adds the entity constraint expression to a list of SQL expressions.

- **where** The list to append the skiptoken expression to.

**set_orderby** *(orderby)*

Sets the orderby rules for this collection.

We override the default implementation to calculate a list of field name aliases to use in ordered queries. For example, if the orderby expression is “tolower(Name) desc” then each SELECT query will be generated with an additional expression, e.g.:

```
SELECT ID, Name, DOB, LOWER(Name) AS o_1 ...
ORDER BY o_1 DESC, ID ASC
```

The name “o_1” is obtained from the name mangler using the tuple:

```
(entity_set.name,'o_1')
```

Subsequent order expressions have names ‘o_2’, ‘o_3’, etc.

Notice that regardless of the ordering expression supplied the keys are always added to ensure that, when an ordering is required, a defined order results even at the expense of some redundancy.

**orderby_clause** *

A utility method to return the orderby clause.

- **params** The SQLParams object to add parameters to.

**orderby_cols** *(column_names, params, force_order=False)*

A utility to add the column names and aliases for the ordering.

- **column_names** A list of SQL column name/alias expressions
- **params** The SQLParams object to add parameters to.
- **force_order** Forces the addition of an ordering by key if an orderby expression has not been set.

**insert_fields** *(entity)*

A generator for inserting mangled property names and values.

- **entity** Any instance of Entity

The yielded values are tuples of (mangled field name, SimpleValue instance).

Read only fields are never generated, even if they are keys. This allows automatically generated keys to be used and also covers the more esoteric use case where a foreign key constraint exists on the primary key
(or part thereof) - in the latter case the relationship should be marked as required to prevent unexpected constraint violations.

Otherwise, only selected fields are yielded so if you attempt to insert a value without selecting the key fields you can expect a constraint violation unless the key is read only.

auto_fields(entity)
A generator for selecting auto mangled property names and values.

documentation: Entity

The yielded values are tuples of (mangled field name, SimpleValue instance).

Only fields that are read only are yielded with the caveat that they must also be either selected or keys. The purpose of this method is to assist with reading back automatically generated field values after an insert or update.

key_fields(entity)
A generator for selecting mangled key names and values.

documentation: Entity

The yielded values are tuples of (mangled field name, SimpleValue instance). Only the keys fields are yielded.

select_fields(entity, prefix=True)
A generator for selecting mangled property names and values.

documentation: Entity

The yielded values are tuples of (mangled field name, SimpleValue instance). Only selected fields are yielded with the caveat that the keys are always selected.

update_fields(entity)
A generator for updating mangled property names and values.

documentation: Entity

The yielded values are tuples of (mangled field name, SimpleValue instance).

Neither read only fields nor key fields are generated.

For SQL variants that support default values on columns natively unselected items are suppressed and are returned instead in name-only form by default_fields().

For SQL variants that don’t support default values, unselected items are yielded here but with either the default value specified in the metadata schema definition of the corresponding property or as NULL.

This method is used to implement OData’s PUT semantics. See merge_fields() for an alternative.

merge_fields(entity)
A generator for merging mangled property names and values.

documentation: Entity

The yielded values are tuples of (mangled field name, SimpleValue instance).

Neither read only fields, keys nor unselected fields are generated. All other fields are yielded implementing OData’s MERGE semantics. See update_fields() for an alternative.

default_fields(entity)
A generator for mangled property names.

documentation: Entity

4.3. OData Reference
The yielded values are the mangled field names that should be set to default values. Neither read only fields, keys nor selected fields are generated.

**stream_field** *(entity, prefix=True)*

Returns information for selecting the stream ID.

- **entity** Any instance of *Entity*

  Returns a tuples of (mangled field name, *SimpleValue* instance).

**sql_expression** *(expression, params, context='AND')*

Converts an expression into a SQL expression string.

- **expression** A *pyslet.odata2.core.CommonExpression* instance.

  A *SQLParams* object of the appropriate type for this database connection.

  A string containing the SQL operator that provides the context in which the expression is being converted, defaults to ‘AND’. This is used to determine if the resulting expression must be bracketed or not. See *sql_bracket()* for a useful utility function to illustrate this.

  This method is basically a grand dispatcher that sends calls to other node-specific methods with similar signatures. The effect is to traverse the entire tree rooted at *expression*.

  The result is a string containing the parameterized expression with appropriate values added to the *params* object in *the same sequence* that they appear in the returned SQL expression.

  When creating derived classes to implement database-specific behaviour you should override the individual evaluation methods rather than this method. All related methods have the same signature.

  Where methods are documented as having no default implementation, *NotImplementedError* is raised.

**sql_bracket** *(query, context, operator)*

A utility method for bracketing a SQL query.

- **query** The query string

  A string representing the SQL operator that defines the context in which the query is to placed.

  E.g., ‘AND’

  The dominant operator in the query.

  This method is used by operator-specific conversion methods. The query is not parsed, it is merely passed in as a string to be bracketed (or not) depending on the values of *context* and *operator*.

  The implementation is very simple, it checks the precedence of *operator* in *context* and returns *query* bracketed if necessary:

    ```
    collection.sql_bracket("Age+3","","*","+")=="(Age+3)"
    collection.sql_bracket("Age*3","","*","+")=="Age*3"
    ```

**sql_expression_member** *(expression, params, context)*

Converts a member expression, e.g., Address/City

  This implementation does not support the use of navigation properties but does support references to complex properties.

  It outputs the mangled name of the property, qualified by the table name.

**sql_expression_cast** *(expression, params, context)*

Converts the cast expression: no default implementation

**sql_expression_generic_binary** *(expression, params, context, operator)*

A utility method for implementing binary operator conversion.
The signature of the basic `sql_expression()` is extended to include an `operator` argument, a string representing the (binary) SQL operator corresponding to the expression object.

```python
sql_expression_mul(expression, params, context)
Converts the mul expression: maps to SQL “*”
```

```python
sql_expression_div(expression, params, context)
Converts the div expression: maps to SQL “/”
```

```python
sql_expression_mod(expression, params, context)
Converts the mod expression: no default implementation
```

```python
sql_expression_add(expression, params, context)
Converts the add expression: maps to SQL “+”
```

```python
sql_expression_sub(expression, params, context)
Converts the sub expression: maps to SQL “-“
```

```python
sql_expression_lt(expression, params, context)
Converts the lt expression: maps to SQL “<”
```

```python
sql_expression_gt(expression, params, context)
Converts the gt expression: maps to SQL “>”
```

```python
sql_expression_le(expression, params, context)
Converts the le expression: maps to SQL “<=”
```

```python
sql_expression_ge(expression, params, context)
Converts the ge expression: maps to SQL “>=”
```

```python
sql_expression_isof(expression, params, context)
Converts the isof expression: no default implementation
```

```python
sql_expression_eq(expression, params, context)
Converts the eq expression: maps to SQL “=”
```

```python
sql_expression_ne(expression, params, context)
Converts the ne expression: maps to SQL “<>”
```

```python
sql_expression_and(expression, params, context)
Converts the and expression: maps to SQL “AND”
```

```python
sql_expression_or(expression, params, context)
Converts the or expression: maps to SQL “OR”
```

```python
sql_expression_endswith(expression, params, context)
Converts the endswith function: maps to “op[0] LIKE ‘%’+op[1]”

This is implemented using the concatenation operator
```

```python
sql_expression_indexof(expression, params, context)
Converts the indexof method: maps to POSITION( op[0] IN op[1] )
```

```python
sql_expression_replace(expression, params, context)
Converts the replace method: no default implementation
```

```python
sql_expression_startswith(expression, params, context)
Converts the startswith function: maps to “op[0] LIKE op[1]+'%’”

This is implemented using the concatenation operator
```

```python
sql_expression_tolower(expression, params, context)
Converts the tolower method: maps to LOWER function
```
sql_expression_toupper (expression, params, context)
Converts the toupper method: maps to UCASE function

sql_expression_trim (expression, params, context)
Converts the trim method: maps to TRIM function

sql_expression_substring (expression, params, context)
Converts the substring method
maps to SUBSTRING( op[0] FROM op[1] [ FOR op[2] ] )

sql_expression_substringof (expression, params, context)
Converts the substringof function
maps to “op[1] LIKE ‘%’+op[0]‘%’”
To do this we need to invoke the concatenation operator.
This method has been poorly defined in OData with the parameters being switched between versions 2 and 3. It is being withdrawn as a result and replaced with contains in OData version 4. We follow the version 3 convention here of “first parameter in the second parameter” which fits better with the examples and with the intuitive meaning:

```
substringof(A,B) == A in B
```

sql_expression_concat (expression, params, context)
Converts the concat method: maps to ||

sql_expression_length (expression, params, context)
Converts the length method: maps to CHAR_LENGTH( op[0] )

sql_expression_year (expression, params, context)
Converts the year method: maps to EXTRACT(YEAR FROM op[0])

sql_expression_month (expression, params, context)
Converts the month method: maps to EXTRACT(MONTH FROM op[0])

sql_expression_day (expression, params, context)
Converts the day method: maps to EXTRACT(DAY FROM op[0])

sql_expression_hour (expression, params, context)
Converts the hour method: maps to EXTRACT(HOUR FROM op[0])

sql_expression_minute (expression, params, context)
Converts the minute method: maps to EXTRACT(MINUTE FROM op[0])

sql_expression_second (expression, params, context)
Converts the second method: maps to EXTRACT(SECOND FROM op[0])

sql_expression_round (expression, params, context)
Converts the round method: no default implementation

sql_expression_floor (expression, params, context)
Converts the floor method: no default implementation

sql_expression_ceiling (expression, params, context)
Converts the ceiling method: no default implementation

class pyslet.odata2.sqlds.SQLNavigationCollection (aset_name, **kwargs)
Bases: pyslet.odata2.sqlds.SQLCollectionBase, pyslet.odata2.core.NavigationCollection
Abstract class representing all navigation collections.
Additional keyword arguments:

**aset_name** The name of the association set that defines this relationship. This additional parameter is used by the name mangler to obtain the field name (or table name) used for the foreign keys.

**insert_link** *(entity, transaction=None)*

Inserts a link to *entity* into this collection.

**transaction** An optional transaction. If present, the connection is left uncommitted.

**replace_link** *(entity, transaction=None)*

Replaces all links with a single link to *entity*.

**transaction** An optional transaction. If present, the connection is left uncommitted.

**delete_link** *(entity, transaction=None)*

A utility method that deletes the link to *entity* in this collection.

This method is called during cascaded deletes to force-remove a link prior to the deletion of the entity itself.

**transaction** An optional transaction. If present, the connection is left uncommitted.

```python
class pyslet.odata2.sqlds.SQLForeignKeyCollection(**kwargs)

Bases: pyslet.odata2.sqlds.SQLNavigationCollection

The collection of entities obtained by navigation via a foreign key

This object is used when the foreign key is stored in the same table as *from_entity*. This occurs when the relationship is one of:

- 0..1 to 1
- Many to 1
- Many to 0..1

The name mangler looks for the foreign key in the field obtained by mangling:

**(entity set name, association set name, key name)**

For example, suppose that a link exists from entity set Orders[*] to entity set Customers[0..1] and that the key field of Customer is “CustomerID”. If the association set that binds Orders to Customers with this link is called OrdersToCustomers then the foreign key would be obtained by looking up:

```python
('Orders', 'OrdersToCustomers', 'CustomerID')
```

By default this would result in the field name:

```python
'OrdersToCustomers_CustomerID'
```

This field would be looked up in the ‘Orders’ table. The operation of the name mangler can be customised by overriding the *SQLEntityContainer.mangle_name()* method in the container.

**reset_joins()**

Overridden to provide an inner join to *from_entity’s* table.

The join clause introduces an alias for the table containing *from_entity*. The resulting join looks something like this:

```sql
SELECT ... FROM Customers
INNER JOIN Orders AS nav1 ON
    Customers.CustomerID=nav1.OrdersToCustomers_CustomerID
```
... WHERE nav1.OrderID = ?;

The value of the OrderID key property in from_entity is passed as a parameter when executing the expression.

In most cases, there will be a navigation properly bound to this association in the reverse direction. For example, to continue the above example, Orders to Customers might be bound to a navigation property in the reverse direction called, say, 'AllOrders' in the target entity set.

If this navigation property is used in an expression then the existing INNER JOIN defined here is used instead of a new LEFT JOIN as would normally be the case.

```
where_clause(entity, params, use_filter=True, use_skip=False)
```

Adds the constraint for entities linked from from_entity only.

We continue to use the alias set in the join_clause() where an example WHERE clause is illustrated.

```
class pyslet.odata2.sqllds.SQLReverseKeyCollection(**kwargs)
    Bases: pyslet.odata2.sqllds.SQLNavigationCollection

    The collection of entities obtained by navigation to a foreign key

    This object is used when the foreign key is stored in the target table. This occurs in the reverse of the cases
    where SQLReverseKeyCollection is used, i.e:

    1 to 0..1 1 to Many 0..1 to Many

    The implementation is actually simpler in this direction as no JOIN clause is required.

    where_clause(entity, params, use_filter=True, use_skip=False)
    Adds the constraint to entities linked from from_entity only.

    delete_link(entity, transaction=None)
    Called during cascaded deletes.

    This is actually a no-operation as the foreign key for this association is in the entity’s record itself and will
    be removed automatically when entity is deleted.

    clear_links(transaction=None)
    Deletes all links from this collection’s from_entity

    transaction  An optional transaction. If present, the connection is left uncommitted.
```

```
class pyslet.odata2.sqllds.SQLAssociationCollection(**kwargs)
    Bases: pyslet.odata2.sqllds.SQLNavigationCollection

    The collection obtained by navigation using an auxiliary table

    This object is used when the relationship is described by two sets of foreign keys stored in an auxiliary table. This occurs mainly when the link is Many to Many but it is also used for 1 to 1 relationships. This last use may seem odd but it is used to represent the symmetry of the relationship. In practice, a single set of foreign keys is likely to exist in one table or the other and so the relationship is best modelled by a 0..1 to 1 relationship even if the intention is that the records will always exist in pairs.

    The name of the auxiliary table is obtained from the name mangler using the association set’s name. The keys use a more complex mangled form to cover cases where there is a recursive Many to Many relation (such as a social network of friends between User entities). The names of the keys are obtained by mangling:

    ( association set name, target entity set name,
      navigation property name, key name )
```
An example should help. Suppose we have entities representing sports Teams(TeamID) and sports Players(PlayerID) and that you can navigate from Player to Team using the “PlayedFor” navigation property and from Team to Player using the “Players” navigation property. Both navigation properties are collections so the relationship is Many to Many. If the association set that binds the two entity sets is called PlayersAndTeams then the auxiliary table name will be mangled from:

\[ ('PlayersAndTeams') \]

and the fields will be mangled from:

\[ ('PlayersAndTeams', 'Teams', 'PlayedFor', 'TeamID')
   ('PlayersAndTeams', 'Players', 'Players', 'PlayerID') \]

By default this results in column names ‘Teams_PlayedFor_TeamID’ and ‘Players_Players_PlayerID’. If you are modelling an existing database then ‘TeamID’ and ‘PlayerID’ on their own are more likely choices. You would need to override the 

\[ SQLEntityContainer.mangle_name() \]

method in the container to catch these cases and return the shorter column names.

Finally, to ensure the uniqueness of foreign key constraints, the following names are mangled:

\[ 
( association set name, association set name, 'fkA')
( association set name, association set name, 'fkB')
\]

Notice that the association set name is used twice as it is not only defines the scope of the name but must also be incorporated into the constraint name to ensure uniqueness across the entire database.

**reset_joins()**

Overridden to provide an inner join to the aux table.

If the Customer and Group entities are related with a Many-Many relationship called Customers_Groups, the resulting join looks something like this (when the from_entity is a Customer):

```sql
SELECT ... FROM Groups
INNER JOIN Customers_Groups ON
   Groups.GroupID = Customers_Groups.Groups_MemberOf_GroupID
...
WHERE Customers_Groups.Customers_Members_CustomerID = ?;
```

The value of the CustomerID key property in from_entity is passed as a parameter when executing the expression.

**add_join(name)**

Overridden to provide special handling of navigation

In most cases, there will be a navigation property bound to this association in the reverse direction. For Many-Many relations this can’t be used in an expression but if the relationship is actually 1-1 then we would augment the default INNER JOIN with an additional INNER JOIN to include the whole of the from_entity. (Normally we’d think of these expressions as LEFT joins but we’re navigating back across a link that points to a single entity so there is no difference.)

To illustrate, if Customers have a 1-1 relationship with PrimaryContacts through a Customers_PrimaryContacts association set then the expression grows an additional join:

```sql
SELECT ... FROM PrimaryContacts
INNER JOIN Customers_PrimaryContacts ON
   PrimaryContacts.ContactID = Customers_PrimaryContacts.PrimaryContacts_Contact_ContactID
INNER JOIN Customers AS nav1 ON
   Customers_PrimaryContacts.Customers_Customer_CustmerID = nav1.CustomerID
```
This is a cumbersome query to join two entities that are supposed to have a 1-1 relationship, which is one of the reasons why it is generally better to pick on side of the relationship or other and make it 0..1 to 1 as this would obviate the auxiliary table completely and just put a non-NULL, unique foreign key in the table that represents the 0..1 side of the relationship.

```sql
Customers.CustomerID
...
WHERE Customers_PrimaryContacts.Customers_Customer_CustomerID = ?;
```

**where_clause** *(entity, params, use_filter=True, use_skip=False)*

Provides the `from_entity` constraint in the auxiliary table.

**insert_entity** *(entity)*

Rerouted to a SQL-specific implementation

**insert_entity_sql** *(entity, transaction=None)*

Inserts `entity` into the base collection and creates the link.

This is always done in two steps, bound together in a single transaction (where supported). If this object represents a 1 to 1 relationship then, briefly, we'll be in violation of the model. This will only be an issue in non-transactional systems.

**delete_link** *(entity, transaction=None)*

Called during cascaded deletes to force-remove a link prior to the deletion of the entity itself.

This method is also re-used for simple deletion of the link in this case as the link is in the auxiliary table itself.

**clear_links** *(transaction=None)*

Deletes all links from this collection's `from_entity`

- **transaction** An optional transaction. If present, the connection is left uncommitted.

**classmethod clear_links_unbound** *(container, from_end, from_entity, transaction)*

Special class method for deleting all the links from `from_entity`

This is a class method because it has to work even if there is no navigation property bound to this end of the association.

- **container** The `SQLEntityContainer` containing this association set.
- **from_end** The `AssociationSetEnd` that represents the end of the association that `from_entity` is bound to.
- **from_entity** The entity to delete links from
- **transaction** The current transaction (required)

This is a class method because it has to work even if there is no navigation property bound to this end of the association. If there was a navigation property then an instance could be created and the simpler `clear_links()` method used.

**classmethod create_table_query** *(container, aset_name)*

Returns a SQL statement and params to create the auxiliary table.

This is a class method to enable the table to be created before any entities are created.

**classmethod create_table** *(container, aset_name)*

Executes the SQL statement `create_table_query()`

**classmethod drop_table_query** *(container, aset_name)*

Returns a SQL statement to drop the auxiliary table.
**classmethod drop_table** *(container, aset_name)*

Executes the SQL statement `drop_table_query()`

**SQLite**

This module also contains a fully functional implementation of the API based on the sqlite3 module. The first job with any SQL implementation is to create a base collection class that implements any custom expression handling.

In the case of SQLite we override a handful of the standard SQL functions only. Notice that this class is derived from `SQLCollectionBase`, an abstract class. If your SQL platform adheres to the SQL standard very closely, or you are happy for SQL-level errors to be generated when unsupported SQL syntax is generated by some filter or orderby expressions then you can skip the process of creating customer collection classes completely.

```python
class pyslet.odata2.sqllds.SQLiteEntityCollectionBase(container, **kwargs)
Bases: pyslet.odata2.sqllds.SQLCollectionBase

Base class for SQLite SQL custom mappings.

This class provides some SQLite specific mappings for certain functions to improve compatibility with the OData expression language.

**DEFAULT_VALUE = False**

SQLite does not support setting a value = DEFAULT

**sql_expression_substring** *(expression, params, context)*

Converts the substring method

maps to `substr( op[0], op[1] [, op[2] ] )`

Few databases seem to actually support the standard syntax using FROM ... [FOR ...]. SQLite is no exception.

**sql_expression_length** *(expression, params, context)*

Converts the length method: maps to `length( op[0] )`

**sql_expression_year** *(expression, params, context)*

Converts the year method

maps to `CAST(strftime('%Y',op[0]) AS INTEGER)`

**sql_expression_month** *(expression, params, context)*

Converts the month method

maps to `CAST(strftime('%m',op[0]) AS INTEGER)`

**sql_expression_day** *(expression, params, context)*

Converts the day method

maps to `CAST(strftime('%d',op[0]) AS INTEGER)`

**sql_expression_hour** *(expression, params, context)*

Converts the hour method

maps to `CAST(strftime('%H',op[0]) AS INTEGER)`

**sql_expression_minute** *(expression, params, context)*

Converts the minute method

maps to `CAST(strftime('%M',op[0]) AS INTEGER)`

**sql_expression_second** *(expression, params, context)*

Converts the second method

maps to `CAST(strftime('%S',op[0]) AS INTEGER)`
sql_expression_tolower(expression, params, context)
Converts the tolower method
maps to lower(op[0])

sql_expression_toupper(expression, params, context)
Converts the toupper method
maps to upper(op[0])

To ensure that our custom implementations are integrated in to all the collection classes we have to create specific classes for all collection types. These classes have no implementation!

class pyslet.odata2.sqlds.SQLiteEntityCollection(container, **kwargs)
Bases: pyslet.odata2.sqlds.SQLiteEntityCollectionBase, pyslet.odata2.sqlds.SQLEntityCollection
SQLite-specific collection for entity sets

class pyslet.odata2.sqlds.SQLiteForeignKeyCollection(**kwargs)
Bases: pyslet.odata2.sqlds.SQLiteEntityCollectionBase, pyslet.odata2.sqlds.SQLForeignKeyCollection
SQLite-specific collection for navigation from a foreign key

class pyslet.odata2.sqlds.SQLiteReverseKeyCollection(**kwargs)
Bases: pyslet.odata2.sqlds.SQLiteEntityCollectionBase, pyslet.odata2.sqlds.SQLReverseKeyCollection
SQLite-specific collection for navigation to a foreign key

class pyslet.odata2.sqlds.SQLiteAssociationCollection(**kwargs)
Bases: pyslet.odata2.sqlds.SQLiteEntityCollectionBase, pyslet.odata2.sqlds.SQLAssociationCollection
SQLite-specific collection for symmetric association sets

Finally, we can override the main container class to provide a complete implementation of our API using the sqlite3 module.

class pyslet.odata2.sqlds.SQLiteEntityContainer(file_path, **kwargs)
Bases: pyslet.odata2.sqlds.SQLEntityContainer
sqlite_options={},
Creates a container that represents a SQLite database.

Additional keyword arguments:

file_path The path to the SQLite database file.

sqlite_options A dictionary of additional options to pass as named arguments to the connect method. It defaults to an empty dictionary, you won’t normally need to pass additional options and you shouldn’t change the isolation_level as the collection classes have been designed to work in the default mode. Also, check_same_thread is forced to False, this is poorly documented but we only do it so that we can close a connection in a different thread from the one that opened it when cleaning up.

For more information see sqlite3

All other keyword arguments required to initialise the base class must be passed on construction except dbapi which is automatically set to the Python sqlite3 module.

get_collection_class()
Overridden to return SQLiteEntityCollection
get_symmetric_navigation_class()
  Overridden to return SQLiteAssociationCollection

get_fk_class()
  Overridden to return SQLiteForeignKeyCollection

get_rk_class()
  Overridden to return SQLiteReverseKeyCollection

open()
  Calls the underlying connect method.
  
  Passes the file_path used to construct the container as the only parameter. You can pass the string ':memory:' to create an in-memory database.
  
  Other connection arguments are not currently supported, you can derive a more complex implementation by overriding this method and (optionally) the __init__ method to pass in values for .

break_connection (connection)
  Calls the underlying interrupt method.

close_connection (connection)
  Calls the underlying close method.

prepare_sql_type (simple_value, params, nullable=None)
  Performs SQLite custom mappings

<table>
<thead>
<tr>
<th>EDM Type</th>
<th></th>
<th>SQLite Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edm.Binary</td>
<td></td>
<td>BLOB</td>
</tr>
<tr>
<td>Edm.Decimal</td>
<td></td>
<td>TEXT</td>
</tr>
<tr>
<td>Edm.Guid</td>
<td></td>
<td>BLOB</td>
</tr>
<tr>
<td>Edm.String</td>
<td></td>
<td>TEXT</td>
</tr>
<tr>
<td>Edm.Time</td>
<td></td>
<td>REAL</td>
</tr>
<tr>
<td>Edm.Int64</td>
<td></td>
<td>INTEGER</td>
</tr>
</tbody>
</table>

The remainder of the type mappings use the defaults from the parent class.

prepare_sql_value (simple_value)
  Returns a python value suitable for passing as a parameter.

  We inherit most of the value mappings but the following types have custom mappings.

<table>
<thead>
<tr>
<th>EDM Type</th>
<th>Python value added as parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edm.Binary</td>
<td>buffer object</td>
</tr>
<tr>
<td>Edm.Decimal</td>
<td>string representation obtained with str()</td>
</tr>
<tr>
<td>Edm.Guid</td>
<td>buffer object containing bytes representation</td>
</tr>
<tr>
<td>Edm.Time</td>
<td>value of pyslet.iso8601.Time.get_total_seconds()</td>
</tr>
</tbody>
</table>

Our use of buffer type is not ideal as it generates warning when Python is run with the -3 flag (to check for Python 3 compatibility) but it seems unavoidable at the current time.

read_sql_value (simple_value, new_value)
  Reverses the transformation performed by prepare_sql_value

new_from_sql_value (sql_value)
  Returns a new simple value instance initialised from sql_value

  Overridden to ensure that buffer objects returned by the underlying DB API are converted to strings. Otherwise sql_value is passed directly to the parent.

4.3. OData Reference 227
**prepare_sql_literal** *(value)*

Formats a simple value as a SQL literal

Overridden for custom SQLite mappings.

**Utility Classes**

Some miscellaneous classes documented mainly to make the implementation of the collection classes easier to understand.

```python
class pyslet.odata2.sqllds.SQLTransaction(container, connection)
    Bases: object

Class used to model a transaction.

Python’s DB API uses transactions by default, hiding the details from the caller. Essentially, the first execute call on a connection issues a BEGIN statement and the transaction ends with either a commit or a rollback. It is generally considered a bad idea to issue a SQL command and then leave the connection with an open transaction.

The purpose of this class is to help us write methods that can operate either as a single transaction or as part of sequence of methods that form a single transaction. It also manages cursor creation and closing and logging.

Essentially, the class is used as follows:
```
t = SQLTransaction(db_container, db_connection)
try:
    t.begin()
    t.execute("UPDATE SOME_TABLE SET SOME_COL='2'")
    t.commit()
except Exception as e:
    t.rollback(e)
finally:
    t.close(e)
```

The transaction object can be passed to a sub-method between the begin and commit calls provided that method follows the same pattern as the above for the try, except and finally blocks. The object keeps track of these ‘nested’ transactions and delays the commit or rollback until the outermost method invokes them.

```python
api = None
    the database module
c
connection = None
    the database connection
cursor = None
    the database cursor to use for executing commands
no_commit = None
    used to manage nested transactions
query_count = None
    records the number of successful commands
commit()
    Ends this transaction with a commit
    Nested transactions do nothing.
rollback *(err=None, swallow=False)*
    Calls the underlying database connection rollback method.
    Nested transactions do not rollback the connection, they do nothing except re-raise err (if required).
```
If rollback is not supported the resulting error is absorbed.

**err** The exception that triggered the rollback. If not None then this is logged at INFO level when the rollback succeeds.

If the transaction contains at least one successfully executed query and the rollback fails then *err* is logged at ERROR rather than INFO level indicating that the data may now be in violation of the model.

**swallow** A flag (defaults to False) indicating that *err* should be swallowed, rather than re-raised.

**close()**
Closes this transaction after a rollback or commit.

Each call to `begin()` MUST be balanced with one call to `close()`.

```python
class pyslet.odata2.sqlds.SQLParams
    Bases: object

    An abstract class used to build parameterized queries.

    Python’s DB API supports three different conventions for specifying parameters and each module indicates the
    convention in use. The SQL construction methods in this module abstract away this variability for maximum
    portability using different implementations of the basic SQLParams class.

    **add_param(value)**
    Adds a value to this set of parameters

    Returns the string to include in the query in place of this value.

    **value:** The native representation of the value in a format suitable for passing to the underlying DB API.

    **classmethod escape_literal(literal)**
    Escapes a literal string, returning the escaped version

    This method is only used to escape characters that are interpreted specially by the parameter substitution
    system. For example, if the parameters are being substituted using python’s `%` operator then the ‘%’ sign
    needs to be escaped (by doubling) in the output.

    This method has nothing to do with turning python values into SQL escaped literals, that task is always
    deferred to the underlying DB module to prevent SQL injection attacks.

    The default implementation does nothing, in most cases that is the correct thing to do.
```

```python
class pyslet.odata2.sqlds.QMarkParams
    Bases: pyslet.odata2.sqlds.SQLParams

    A class for building parameter lists using ‘?’ syntax.

class pyslet.odata2.sqlds.NumericParams
    Bases: pyslet.odata2.sqlds.SQLParams

    A class for building parameter lists using ‘:1’, ‘:2’,... syntax

class pyslet.odata2.sqlds.NamedParams
    Bases: pyslet.odata2.sqlds.SQLParams

    A class for building parameter lists using ‘:A’, ‘:B’,... syntax

    Although there is more freedom with named parameters, in order to support the ordered lists of the other formats
    we just invent parameter names using ‘:p0’, ‘:p1’, etc.
```
Misc Definitions

```python
pyslet.odata2.sqlds.SQL_TIMEOUT = 90

int(x=0) -> int or long
int(x, base=10) -> int or long

Convert a number or string to an integer, or return 0 if no arguments are given. If x is floating point, the
conversion truncates towards zero. If x is outside the integer range, the function returns a long instead.

If x is not a number or if base is given, then x must be a string or Unicode object representing an integer literal
in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults
to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>>
int('0b100', base=0) 4
```

class pyslet.odata2.sqlds.UnparameterizedLiteral(value)

Bases: pyslet.odata2.core.LiteralExpression

Class used as a flag that this literal is safe and does not need to be parameterized.

This is used in the query converter to prevent things like this happening when the converter itself constructs a
LIKE expression:

```
"name" LIKE %?+??; params=["", "Smith", "]
```

```python
pyslet.odata2.sqlds.SQLOperatorPrecedence = {'>=': 4, '<>': 4, '<=': 4, 'AND': 2, 'LIKE': 4,

Look-up table for SQL operator precedence calculations.

The keys are strings representing the operator, the values are integers that allow comparisons for operator prece-
dence. For example:

```
SQLOperatorPrecedence['+']<SQLOperatorPrecedence['*']
SQLOperatorPrecedence['<']<=SQLOperatorPrecedence['>']
```
```
class pyslet.odata2.sqlds.DummyLock

Bases: object

An object to use in place of a real Lock, can always be acquired
```

Exceptions

```python
class pyslet.odata2.sqlds.DatabaseBusy

Bases: pyslet.odata2.sqlds.SQLException

Raised when a database connection times out.
```

class pyslet.odata2.sqlds.SQLException

Bases: exceptions.Exception

Base class for all module exceptions.

4.3.7 OData Server Reference
This sub-package defines functions and classes for working with HTTP as defined by RFC2616: http://www.ietf.org/rfc/rfc2616.txt and RFC2617: http://www.ietf.org/rfc/rfc2617.txt

The purpose of this module is to expose some of the basic constructs (including the synax of protocol components) to allow them to be used normatively in other contexts. The module also contains a functional HTTP client designed to support non-blocking and persistent HTTP client operations.

5.1 HTTP Client

5.1.1 Sending Requests

Here is a simple example of Pyslet’s HTTP support in action from the python interpreter:

```python
>>> import pyslet.http.client as http
>>> c = http.Client()
>>> r = http.ClientRequest('http://odata.pyslet.org')
>>> c.process_request(r)
>>> r.response.status
200
>>> print r.response.get_content_type()
text/html; charset=UTF-8
>>> print r.response.entity_body.getvalue()
<html>
<head><title>Pyslet Home</title></head>
<body>
<p><a href="http://qtimigration.googlecode.com/"><img src="logoc-large.png" width="1024"></a></p>
</body>
</html>
```
In its simplest form there are three steps required to make an HTTP request, firstly you need to create a Client object.
The purpose of the Client object is sending requests and receiving responses. The second step is to create a ClientRequest object describing the request you want to make. For a simple GET request you only need to specify the URL. The third step is to instruct the Client to process the request. Once this method returns you can examine the request’s associated response. The response’s entity body is written to a StringIO object by default.

The request and response objects are both derived classes of a basic HTTP Message class. This class has methods for getting and setting headers. You can use the basic get_header() and set_header() to set headers from strings or, where provided, you can use special wrapper methods such as get_content_type() to get and set headers using special-purpose class objects that represent parsed forms of the expected value. In the case of Content-Type headers the result is a MediaType() object. Providing these special object types is one of the main reasons why Pyslet’s HTTP support is different from other clients. By exposing these structures you can reuse HTTP concepts in other contexts, particularly useful when other technical specifications make normative references to them.

Here is a glimpse of what you can do with a parsed media type, continuing the above example:

```python
>>> type = r.response.get_content_type()
>>> type
MediaType('text','html', {'charset': ('charset', 'UTF-8')})
>>> type.type
'text'
>>> type.subtype
'html'
>>> type['charset']
'UTF-8'
>>> type['name']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "pyslet/http/params.py", line 382, in __getitem__
      repr(key))
KeyError: "MediaType instance has no parameter 'name'
```

There are lots of other special get_ and set_ methods on the Message, Request and Response objects.

### 5.1.2 Pipelining

One of the use cases that Pyslet’s HTTP client is designed to cover is reusing an HTTP connection to make multiple requests to the same host. The example above takes care to close the Client object when we’re done because otherwise it would leave the connection to the server open ready for another request.

### 5.1.3 Reference

The client module imports the grammar, params, messages and auth modules and these can therefore be accessed using a single import in your code. For example:

```python
import pyslet.http.client as http
```


```python
class pyslet.http.client.Client(max_connections=100, ca_certs=None, timeout=None, max_inactive=None)
```
An HTTP client

**Note:** In Pyslet 0.4 and earlier the name HTTPRequestManager was used, this name is still available as an alias for Client.

The object manages the sending and receiving of HTTP/1.1 requests and responses respectively. There are a number of keyword arguments that can be used to set operational parameters:

- **max_connections** The maximum number of HTTP connections that may be open at any one time. The method `queue_request()` will block (or raise `RequestManagerBusy`) if an attempt to queue a request would cause this limit to be exceeded.

- **timeout** The maximum wait time on the connection. This is not the same as a limit on the total time to receive a request but a limit on the time the client will wait with no activity on the connection before assuming that the server is no longer responding. Defaults to None, no timeout.

- **max_inactive** The maximum time to keep a connection inactive before terminating it. By default, HTTP connections are kept open when the protocol allows. These idle connections are kept in a pool and can be reused by any thread. This is useful for web-service type use cases (for which Pyslet has been optimised) but it is poor practice to keep these connections open indefinitely and anyway, most servers will hang up after a fairly short period of time anyway.

  If not None, this setting causes a cleanup thread to be created that calls the `idle_cleanup()` method periodically passing this setting value as its argument.

- **ca_certs** The file name of a certificate file to use when checking SSL connections. For more information see [http://docs.python.org/2.7/library/ssl.html](http://docs.python.org/2.7/library/ssl.html)

  In practice, there seem to be serious limitations on SSL connections and certificate validation in Python distributions linked to earlier versions of the OpenSSL library (e.g., Python 2.6 installed by default on OS X and Windows).

**Warning:** By default, ca_certs is optional and can be passed as None. In this mode certificates will not be checked and your connections are not secure from man in the middle attacks. In production use you should always specify a certificate file if you expect to use the object to make calls to https URLs.

Although max_connections allows you to make multiple connections to the same host+port the request manager imposes an additional restriction. Each thread can make at most 1 connection to each host+port. If multiple requests are made to the same host+port from the same thread then they are queued and will be sent to the server over the same connection using HTTP/1.1 pipelining. The manager (mostly) takes care of the following restriction imposed by RFC2616:

Clients SHOULD NOT pipeline requests using non-idempotent methods or non-idempotent sequences of methods

In other words, a POST (or CONNECT) request will cause the pipeline to stall until all the responses have been received. Users should beware of non-idempotent sequences as these are not automatically detected by the manager. For example, a GET,PUT sequence on the same resource is not idempotent. Users should wait for the GET request to finish fetching the resource before queuing a PUT request that overwrites it.

In summary, to take advantage of multiple simultaneous connections to the same host+port you must use multiple threads.

**ConnectionClass**

alias of `Connection`
httpUserAgent = None
The default User-Agent string to use, defaults to a string derived from the installed version of Pyslet, e.g.:

```python
pyslet 0.5.20140727 (http.client.Client)
```

classmethod get_server_certificate_chain(url, method=None, options=None)
Returns the certificate chain for an https URL

**url** A `URI` instance. This must use the https scheme or ValueError will be raised.

**method** (SSL.TLSv1_METHOD) The SSL method to use, one of the constants from the pyOpenSSL module.

**options** (None) The SSL options to use, as defined by the pyOpenSSL module. For example, SSL.OP_NO_SSLv2.

This method requires pyOpenSSL to be installed, if it isn’t then a RuntimeError is raised.

The address and port is extracted from the URL and interrogated for its certificate chain. No validation is performed. The result is a string containing the concatenated PEM format certificate files. This string is equivalent to the output of the following UNIX command:

```bash
echo | openssl s_client -showcerts -connect host:port 2>&1 | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p'
```

The purpose of this method is to provide something like the ssh-style trust whereby you can download the chain the first time you connect, store it to a file and then use that file for the ca_certs argument for SSL validation in future.

If the site certificate changes to one that doesn’t validate to a certificate in the same chain then the SSL connection will fail.

As this method does no validation there is no protection against a man-in-the-middle attack when you use this method. You should only use this method when you trust the machine and connection you are using or when you have some other way to independently verify that the certificate chain is good.

queue_request (request, timeout=None)
Starts processing an HTTP request

**request** A `messages.Request` object.

**timeout** Number of seconds to wait for a free connection before timing out. A timeout raises `RequestManagerBusy`

None means wait forever, 0 means don’t block.

The default implementation adds a User-Agent header from `httpUserAgent` if none has been specified already. You can override this method to add other headers appropriate for a specific context but you must pass this call on to this implementation for proper processing.

active_count ()
Returns the total number of active connections.

thread_active_count ()
Returns the total number of active connections associated with the current thread.

thread_task (timeout=None)
Processes all connections bound to the current thread then blocks for at most timeout (0 means don’t block) while waiting to send/receive data from any active sockets.

Each active connection receives one call to `Connection.connection_task()` There are some situations where this method may still block even with timeout=0. For example, DNS name resolution and SSL handshaking. These may be improved in future.
Returns True if at least one connection is active, otherwise returns False.

```python
thread_loop(timeout=60)
```
Repeatedly calls `thread_task()` until it returns False.

```python
process_request(request, timeout=60)
```
Process an `messages.Message` object.

The request is queued and then `thread_loop()` is called to exhaust all HTTP activity initiated by the current thread.

```python
idle_cleanup(max_inactive=15)
```
Cleans up any idle connections that have been inactive for more than `max_inactive` seconds.

```python
active_cleanup(max_inactive=90)
```
Clean up active connections that have been inactive for more than `max_inactive` seconds.

This method can be called from any thread and can be used to remove connections that have been abandoned by their owning thread. This can happen if the owning thread stops calling `thread_task()` leaving some connections active.

Inactive connections are killed using `Connection.kill()` and then removed from the active list. Should the owning thread wake up and attempt to finish processing the requests a socket error or `messages.HTTPException` will be reported.

```python
close()
```
Closes all connections and sets the manager to a state where new connections cannot be created. Active connections are killed, idle connections are closed.

```python
add_credentials(credentials)
```
Adds a `pyslet.http.auth.Credentials` instance to this manager.

Credentials are used in response to challenges received in HTTP 401 responses.

```python
remove_credentials(credentials)
```
Removes credentials from this manager.

```python
credentials
```

If the credentials can’t be found then they are silently ignored as it is possible that two threads may independently call the method with the same credentials.

```python
dnslookup(host, port)
```
Given a host name (string) and a port number performs a DNS lookup using the native `socket.getaddrinfo` function. The resulting value is added to an internal dns cache so that subsequent calls for the same host name and port do not use the network unnecessarily.

If you want to flush the cache you must do so manually using `flush_dns()`.

```python
flush_dns()
```
Flushes the DNS cache.

```python
find_credentials(challenge)
```
Searches for credentials that match `challenge`

```python
find_credentials_by_url(url)
```
Searches for credentials that match `url`

```python
class pyslet.http.client.ClientRequest(url, method='GET', res_body=None, protocol=<pyslet.http.params.HTTPVersion object>, auto_redirect=True, max_retries=3, min_retry_time=5, **kwargs)
```

5.1. HTTP Client
Bases: `pyslet.http.messages.Request`

Represents an HTTP request.

To make an HTTP request, create an instance of this class and then pass it to an `Client` instance using either `Client.queue_request()` or `Client.process_request()`.

**url** An absolute URI using either http or https schemes. A `pyslet.rfc2396.URI` instance or an object that can be passed to its constructor.

And the following keyword arguments:

**method** A string. The HTTP method to use, defaults to “GET”

**entity_body** A string or stream-like object containing the request body. Defaults to None meaning no message body. For stream-like objects the tell and seek methods must be supported to enable resending the request if required.

**res_body** A stream-like object to write data to. Defaults to None, in which case the response body is returned as a string in the `res_body`.

**protocol** An `params.HTTPVersion` object, defaults to `HTTPVersion(1,1)`

**autoredirect** Whether or not the request will follow redirects, defaults to True.

**max_retries** The maximum number of times to attempt to resend the request following an error on the connection or an unexpected hang-up. Defaults to 3, you should not use a value lower than 1 because, when pipelining, it is always possible that the server has gracefully closed the socket and we won’t notice until we’ve sent the request and get 0 bytes back on recv. Although ‘normal’ this scenario counts as a retry.

**manager = None**
the `Client` object that is managing us

**connection = None**
the `Connection` object that is currently sending us

**status = None**
the status code received, 0 indicates a failed or unsent request

**error = None**
If status == 0, the error raised during processing

**scheme = None**
the scheme of the request (http or https)

**hostname = None**
the hostname of the origin server

**port = None**
the port on the origin server

**url = None**
the full URL of the requested resource

**res_body = None**
the response body received (only used if not streaming)

**auto_redirect = None**
whether or not auto redirection is in force for 3xx responses

**max_retries = None**
the maximum number of retries we’ll attempt

**response = None**
the associated `ClientResponse`
send_pipe = None
the send pipe to use on upgraded connections

recv_pipe = None
the recv pipe to use on upgraded connections

set_url(url)
Sets the URL for this request

This method sets the Host header and the following local attributes: scheme, hostname, port and request_uri.

can_retry()
Returns True if we reconnect and retry this request

set_client(client)
Called when we are queued for processing.

client an Client instance

connect(connection, send_pos)
Called when we are assigned to an HTTPConnection"

connection A Connection object

send_pos The position of the sent bytes pointer after which this request has been (or at least has started to be) sent.

disconnect(send_pos)
Called when the connection has finished sending us

This may be before or after the response is received and handled!

send_pos The number of bytes sent on this connection before the disconnect. This value is compared with the value passed to connect() to determine if the request was actually sent to the server or abandoned without a byte being sent.

For idempotent methods we lose a life every time. For non-idempotent methods (e.g., POST) we do the same except that if we’ve been (at least partially) sent then we lose all lives to prevent “indeterminate results”.

finished()
Called when we have a final response and have disconnected from the connection. There is no guarantee that the server got all of our data, it might even have returned a 2xx series code and then hung up before reading the data, maybe it already had what it needed, maybe it thinks a 2xx response is more likely to make us go away. Whatever. The point is that you can’t be sure that all the data was transmitted just because you got here and the server says everything is OK

class pyslet.http.client.ClientResponse(request, **kwargs)
Bases: pyslet.http.messages.Response

handle_headers()
Hook for response header processing.

This method is called when a set of response headers has been received from the server, before the associated data is received! After this call, recv will be called zero or more times until handle_message or handle_disconnect is called indicating the end of the response.

Override this method, for example, if you want to reject or invoke special processing for certain responses (e.g., based on size) before the data itself is received. To abort the response, close the connection using Connection.request_disconnect().

Override the Finished() method instead to clean up and process the complete response normally.
**handle_message**

Hook for normal completion of response

**handle_disconnect** *(err)*

Hook for abnormal completion of response

Called when the server disconnects before we’ve completed reading the response. Note that if we are reading forever this may be expected behaviour and *err* may be None.

We pass this information on to the request.

### 5.1.4 Exceptions

**class** *pyslet.http.client.RequestManagerBusy*

**Bases:** *pyslet.http.messages.HTTPException*

The HTTP client is busy

Raised when attempting to queue a request and no connections become available within the specified timeout.

### 5.2 HTTP Authentication

Pyslet’s http sub-package contains an auth module that supports RFC2617. This module adds core support for HTTP’s simple challenge-response authentication.

#### 5.2.1 Adding Credentials to an HTTP Request

The simplest way to do basic authentication is to simply add a preformatted Authorization header to each request. For example, if you need to send a request to a website and you know that it requires you to pass your basic auth credentials you could just do something like this:

```python
import pyslet.http.client as http
client = http.Client()
request = http.ClientRequest('https://www.example.com/mypage')
request.set_header('Authorization', 'Basic Sm9ob1NtaXR0OnNlY3J1dA==')
client.process_request(request)
```

Calculating the correctly formatted value for the Authorization header can be simplified by creating a `BasicCredentials` object:

```python
import pyslet.http.auth as auth
credentials = auth.BasicCredentials()
credentials.userid = "JohnSmith"
credentials.password = "secret"
str(credentials)
'BASIC Sm9ob1NtaXR0OnNlY3J1dA="'
```

As you can see, the credentials object takes care of the syntax for you. The userid and password are character strings but you should be aware that only characters in the ISO-8859-1 character set can be used in user names and passwords.

If you don’t want to add the Authorization header yourself you can delegate responsibility to the http client itself. Before you do that though you have to add an additional piece of information to your credentials objects: the protection space. A protection space is simply the combination of the http scheme (http/https), the host and any optional port information. You can calculate the protection space associated with a URL using Pyslet’s URI object:
Notice that the `get_canonical_root()` method takes care of figuring out default ports and removing the path for you so you can get the protection space for any http-based URL. By setting the protectionSpace attribute on the BasicCredentials object you tell the client which sites it should offer the credentials to:

```python
credentials.protectionSpace = uri
c.add_credentials(credentials)
r = http.ClientRequest('https://www.example.com/mypage')
c.process_request(r)
```

The HTTP client has a credential store and an add_credentials method. Once added, the following happens when a 401 response is received:

1. The client iterates through any received challenges
2. Each challenge is matched against the stored credentials
3. If matching credentials are found then an Authorization header is added and the request resent
4. If the request receives another 401 response indicating that the attempt to authenticate failed then the credentials are removed from the store and we go back to (1)

This process terminates when there are no more credentials that match any of the challenges or when a code other than 401 is received.

If the matching credentials are BasicCredentials (and that’s the only type Pyslet supports out of the box), then some additional logic gets activated on success. RFC 2617 says that for basic authentication, a challenge implies that all paths “at or deeper than the depth of the last symbolic element in the path field” fall into the same protection space. Therefore, when credentials are used successfully, Pyslet adds the path to the credentials using BasicCredentials.add_success_path. Next time a request is sent to a URL on the same server with a path that meets this criterium the Authorization header will be added automatically without waiting for a 401 challenge response.

You can simulate this behaviour yourself if you want to pre-empt a 401 response completely. You just need to add a suitable path to the credentials before you add them to the client. So if you know your credentials are good for everything in /website/~user/ you could continue the above code like this:

```python
credentials.add_success_path('/website/~user/')
```

That last slash is really important, if you leave it off it will add everything in ‘/website/’ to your protection space which is probably not what you want.

### 5.2.2 Class Reference

```python
class pyslet.http.auth.Credentials
    Bases: pyslet.http.params.Parameter
```

An abstract class that represents a set of HTTP authentication credentials.

Instances are typically created and then added to a request manager object using `add_credentials()` for matching against HTTP authorization challenges.

The built-in `str` function can be used to format instances according to the grammar defined in the specification.
scheme_class = {'basic': <class 'pyslet.http.auth.BasicCredentials'>}
A dictionary mapping lower-case auth schemes onto the special classes used to represent their credential messages

classmethod register(scheme, credential_class)
Registers a class to represent credentials in scheme

scheme A string representing an auth scheme, e.g., ‘Basic’. The string is converted to lower-case before it is registered to allow for case insensitive look-up.

credential_class A class derived from Credentials that is used to represent a set of base Credentials in that scheme. The class must have a from_words() class method.

If a class has already been registered for the scheme it is replaced. The mapping is kept in the scheme_class dictionary.

base = None
The base credentials

By default, new instances are ‘base’ credentials and this attribute will be None. When credentials are traded in response to a challenge they become session credentials and base contains the original ‘base’ credentials object from which the proffered credentials were derived.

scheme = None
the authentication scheme

protectionSpace = None
the protection space in which these credentials should be used.

The protection space is a pyslet.rfc2396.URI instance reduced to just the the URL scheme, host-name and port.

realm = None
the realm in which these credentials should be used.

The realm is a simple string as returned by the HTTP server. If None then these credentials will be used for any realm within the protection space.

match_challenge(challenge)
Returns True if these credentials can be used in response to challenge.

challenge A Challenge instance

The match is successful if the authentication scheme, the protection space and the realms match the corresponding values in the challenge.

test_url(url)
Returns True if these credentials can be used peremptorily when making a request to url.

url A pyslet.rfc2396.URI instance.
The default implementation always returns False.

classmethod from_str(source)
Constructs a Credentials instance from an HTTP formatted string.

get_response(challenge=None)
Creates a new set of credentials from a challenge

challenge (None) A Challenge instance containing a challenge that has been received from the server. If these credentials are being used pre-emptively (based on the target URL) then challenge will be None.
This is an abstract method, by default None is returned.

For base credentials, this method must return a new credentials object to be used in a new authentication session or None if these credentials cannot be used in response to challenge. If there is no challenge, return a new credentials object that can be used pre-emptively.

For session credentials (instances returned by a previous call) this method must return a new credentials object, the same credentials object with updated state or None if the challenge indicates that the authentication session has failed. (If there is no challenge, the credentials should be returned unchanged.)

In all cases, the returned object must have base set to the original base credentials used to initiate the authentication session.

class pyslet.http.auth.BasicCredentials
Bases: pyslet.http.auth.Credentials
test_url(url)
   Given a URI instance representing an absolute URI, checks if these credentials contain a matching protection space and path prefix.
test_path(path)
   Returns True if there is a path prefix that matches path
add_success_path(path)
   Updates credentials based on success at path
   path A string of octets representing the path that these credentials have been used for with a successful result.

This method implements the requirement that paths “at or deeper than the depth of the last symbolic element in the path field” should be treated as being part of the same protection space.

The path is reduced to a path prefix by removing the last symbolic element and then it is tested against existing prefixes to ensure that the most general prefix is being stored, for example, if path is “/website/document” it will replace any existing prefixes of the form “/website/folder.” with the common prefix “/website”.

class pyslet.http.auth.Challenge(scheme, *params)
Bases: pyslet.http.params.Parameter

Represents an HTTP authentication challenge.

Instances are created from a scheme and a variable length list of 3-tuples containing parameter (name, value, qflag) values. The types of the items are as follows:

   name A character string containing the token that names the parameter
   value A binary string representing the parameter value.
   qflag a boolean indicating that the value must always be quoted, even if it is a valid token.

All Challenges require a realm parameter, if omitted a realm of “Default” is used.

Instances behave like read-only lists of (name,value) pairs implementing len, indexing and iteration in the usual way. Instances also support basic key lookup of parameter names by implementing __contains__ and __getitem__ (which returns the parameter value and raises KeyError for undefined parameters). Name lookup handles case sensitivity by looking first for a case-sensitive match and then for a case insensitive match. Instances are not truly dictionary like.

scheme_class = {'basic': <class 'pyslet.http.auth.BasicChallenge'>}

A dictionary mapping lower-case auth schemes onto the special classes used to represent their challenge messages

5.2. HTTP Authentication
can_parse = True

A scheme-specific flag indicating whether or not the scheme is parsable. Some schemes use the WWW-
Authenticate header but do not adhere to the syntax past the scheme name.

classmethod register(scheme, challenge_class)

Registers a class to represent a challenge in scheme

scheme A string representing an auth scheme, e.g., ‘Basic’. The string is converted to lower-case before
it is registered to allow for case insensitive look-up.

challenge_class A class derived from Challenge that is used to represent a Challenge issued using that
scheme

If a class has already been registered for the scheme it is replaced. The mapping is kept in the
scheme_class dictionary.

scheme = None

the name of the schema

protectionSpace = None

An optional protection space indicating the scope of this challenge. When the HTTP client receives a
challenge this is set to the URL representing the scheme/host/port of the server.

classmethod from_str(source)

Creates a Challenge from a source string.

classmethod list_from_str(source)

Creates a list of Challenges from a source string.

class pyslet.http.auth.BasicChallenge(*params)

Bases: pyslet.http.auth.Challenge

Represents an HTTP Basic authentication challenge.

5.3 HTTP Messages

This modules defines objects that represent the values of HTTP messages and message headers and a special-purpose
parser for parsing them from strings of octets.

5.3.1 Messages

class pyslet.http.messages.Request(**kwargs)

Bases: pyslet.http.messages.Message

method = None

the http method, always upper case, e.g., ‘POST’

request_uri = None

the request uri as it appears in the start line

response = None

the associated response

send_start()

Returns the start-line for this message

send_transferlength()

Adds request-specific processing for transfer-length
Request messages that must not have a message body are automatically detected and will raise an exception if they have a non-None body.

Request messages that may have a message body but have a transfer-length of 0 bytes will have a Content-Length header of 0 added if necessary

get_start()  
Returns the start line

is_idempotent()  
Returns True if this is an idempotent request

extract_authority()  
Extracts the authority from the request

If the request_uri is an absolute URL then it is updated to contain the absolute path only and the Host header is updated with the authority information (host[:port]) extracted from it, otherwise the Host header is read for the authority information. If there is no authority information in the request None is returned.

If the url contains user information it raises NotImplementedError

get_accept()  
Returns an AcceptList instance or None if no “Accept” header is present.

set_accept(accept_value)  
Sets the “Accept” header, replacing any existing value.

accept_value A AcceptList instance or a string that one can be parsed from.

get_accept_charset()  
Returns an AcceptCharsetList instance or None if no “Accept-Charset” header is present.

set_accept_charset(accept_value)  
Sets the “Accept-Charset” header, replacing any existing value.

accept_value A AcceptCharsetList instance or a string that one can be parsed from.

get_accept_encoding()  
Returns an AcceptEncodingList instance or None if no “Accept-Encoding” header is present.

set_accept_encoding(accept_value)  
Sets the “Accept-Encoding” header, replacing any existing value.

accept_value A AcceptEncodingList instance or a string that one can be parsed from.

get_cookie()  
Reads the ‘Cookie’ header(s)

Returns a dictionary of cookies. If there are multiple values for a cookie the dictionary value is a set, otherwise it is a string.

set_cookie(cookie_list)  
Set a “Set-Cookie” header

cookie_list a list of cookies such as would be returned by pyslet.http.cookie.CookieStore.search().

If cookie list is None the Cookie header is removed.

class pyslet.http.messages.Response(request=None, **kwargs)  
Bases: pyslet.http.messages.Message


A dictionary mapping status code integers to their default message defined by RFC2616
send_start ()
    Returns the start-line for this message

get_accept_ranges ()
    Returns an AcceptRanges instance or None if no “Accept-Ranges” header is present.

set_accept_ranges (accept_value)
    Sets the “Accept-Ranges” header, replacing any existing value.

    accept_value  A AcceptRanges instance or a string that one can be parsed from.

get_age ()
    Returns an integer or None if no “Age” header is present.

set_age (age)
    Sets the “Age” header, replacing any existing value.

    age  an integer or long value or None to remove the header

get_etag ()
    Returns a EntityTag instance parsed from the ETag header or None if no “ETag” header is present.

set_etag (etag)
    Sets the “ETag” header, replacing any existing value.

    etag  a EntityTag instance or None to remove any ETag header.

get_location ()
    Returns a pyslet.rfc2396.URI instance created from the Location header.
    If no Location header was present None is returned.

set_location (location)
    Sets the Location header

    location:  a pyslet.rfc2396.URI instance or a string from which one can be parsed. If None, the
                Location header is removed.

get_www_authenticate ()
    Returns a list of Challenge instances.

    If there are no challenges an empty list is returned.

set_www_authenticate (challenges)
    Sets the “WWW-Authenticate” header, replacing any exsiting value.

    challenges  a list of Challenge instances

get_set_cookie ()
    Reads all ‘Set-Cookie’ headers

    Returns a list of Cookie instances

set_set_cookie (cookie, replace=False)
    Set a “Set-Cookie” header

    cookie  a Cookie instance

    replace=True  Remove all existing cookies from the response

    replace=False  Add this cookie to the existing cookies in the response (default value)

    If called multiple times the header value will become a list of cookie values. No folding together is
    performed.

    If cookie is None all Set-Cookie headers are removed, implying replace mode.
class pyslet.http.messages.Message (entity_body=None, protocol=<pyslet.http.params.HTTPVersion object>, send_stream=None, recv_stream=None)

Bases: pyslet.pep8.PEP8Compatibility, object

An abstract class to represent an HTTP message.

The methods of this class are thread safe, using a lock to protect all access to internal structures.

The generic syntax of a message involves a start line, followed by a number of message headers and an optional message body.

**entity_body** The optional entity_body parameter is a byte string containing the entity body, a file like object or object derived from io.RawIOBase. There are restrictions on the use of non-seekable streams, in particular the absence of a working seek may affect redirects and retries.

There is a subtle difference between passing None, meaning no entity body and an empty string ''. The difference is that an empty string will generate a Content-Length header indicating a zero length message body when the message is sent, whereas None will not. Some message types are not allowed to have an entity body (e.g., a GET request) and these messages must not have a message body (even a zero length one) or an error will be raised.

File-like objects do not generate a Content-Length header automatically as there is no way to determine their size when sending, however, if a Content-Length header is set explicitly then it will be used to constrain the amount of data read from the entity_body.

**GENERAL_HEADERS** = {'transfer-encoding': 'Transfer-Encoding', 'connection': 'Connection', 'upgrade': 'Upgrade', 'pragma': 'Pragma', 'cache-control': 'Cache-Control', 'date': 'Date', 'warning': 'Warning', 'via': 'Via', 'trailer': 'Trailer'}

a mapping from lower case header name to preferred case name

**MAX_READAHEAD** = 131072

A constant used to control the maximum read-ahead on an entity body’s stream. Entity bodies of undetermined length that exceed this size cannot be sent in requests to HTTP/1.0 server.

**lock = None**

the lock used to protect multi-threaded access

**got_headers = None**

boolean indicating that all headers have been received

**keep_alive = None**

by default we’ll keep the connection alive

**set_protocol** (version)

Sets the protocol

version An params.HTTPVersion instance or a string that can be parsed for one.

**clear_keep_alive**()

Clears the keep_alive flag on this message

The flag always starts set to True and cannot be set once cleared.

**start_sending** (protocol=<pyslet.http.params.HTTPVersion object>)

Starts sending this message

**protocol** The protocol supported by the target of the message, defaults to HTTP/1.1 but can be overridden when the recipient only supports HTTP/1.0. This has the effect of suppressing some features.

The message is sent using the send_ family of methods.

**send_start**()

Returns the start-line for this message
send_header()  
Returns a data string ready to send to the server

send_transferlength()  
Calculates the transfer length of the message
It will read the Transfer-Encoding or Content-Length headers to determine the length.
If the length of the entity body is known, this method will verify that it matches the Content-Length or set
that header’s value accordingly.
If the length of the entity body is not known, this method will set a Transfer-Encoding header.

abort_sending()  
Aborts sending the message body
Called after start_sending, this method attempts to abort the sending the message body and returns the
(approximate) number of bytes that will be returned by future calls to send_body. (Ignoring chunk boundaries.)
Messages that are already complete will return 0.
Messages that are using chunked transfer encoding can be aborted and will return 0 indicating that the next
chunk returned by send_body() will be the trailing chunk.
Messages that are not using chunked transfer encoding cannot be aborted and will return the number of
bytes remaining or -1 if this cannot be determined (the latter case is only possible when the message body
will be terminated by a connection close and so only applies to responses).
This method has a very special use case. In cases where a server rejects a request before reading the entire
message body the client may attempt to abort the sending of the body without closing the connection. The
only way to do this is to truncate a body being sent with chunked encoding. You might wonder why a
client would go to such lengths to keep the connection open. The answer is NTLM which authenticates a
connection, so a large POST that gets an early 401 response must be retried on the same connection. This
can only be done if the message boundaries are well defined. There’s a good discussion of the issue at
https://curl.haxx.se/mail/lib-2004-08/0002.html

send_body()  
Returns (part of) the message body
Returns an empty string when there is no more data to send.
Returns None if the message is read blocked.

start_receiving()  
Starts receiving this message
The message is received using the recv_mode() and recv() methods.

RECV_HEADERS = -3  
recv_mode constant for a set of header lines terminated by CRLF, followed by a blank line.

RECV_LINE = -2  
recv_mode constant for a single CRLF terminated line

RECV_ALL = -1  
recv_mode constant for unlimited data read

recv_mode()  
Indicates the type of data expected during recv
The result is interpreted as follows, using the recv_mode constants defined above:
RECV_HEADERS this message is expecting a set of headers, terminated by a blank line. The next call to recv must be with a list of binary CRLF terminated strings the last of which must the string CRLF only.

RECV_LINE this message is expecting a single terminated line. The next call to recv must be with a binary string representing a single terminated line.

integer > 0 the minimum number of bytes we are waiting for when data is expected. The next call to recv must be with a binary string of up to but not exceeding integer number of bytes

0 we are currently write-blocked but may need more data, the next call to recv must pass None to give the message time to write out existing buffered data.

RECV_ALL we want to read until the connection closes, the next call to recv must be with a binary string. The string can be of any length but an empty string signals the end of the data.

None the message is not currently in receiving mode, calling recv will raise an error.

recv_start (start_line)
Receives the start-line

Implemented differently for requests and responses.

handle_headers()
Hook for processing the message headers

This method is called after all headers have been received but before the message body (if any) is received. Derived classes should always call this implementation first (using super) to ensure basic validation is performed on the message before the body is received.

The default implementation sets got_headers to True.

recv_transferlength()
Called to calculate the transfer length when receiving

The values of transferlength and transferchunked are set by this method. The default implementation checks for a Transfer-Encoding header and then a Content-Length header in that order.

If it finds neither then behaviour is determined by the derived classes Request and Response which wrap this implementation.

RFC2616:

If a Transfer-Encoding header field is present and has any value other than “identity”, then the transfer-length is defined by use of the “chunked” transfer-coding, unless the message is terminated by closing the connection

This is a bit weird, if I have a non-identity value which fails to mention ‘chunked’ then it seems like I can’t imply chunked encoding until the connection closes. In practice, when we handle this case we assume chunked is not being used and read until connection close.

handle_message()
Hook for processing the message

This method is called after the entire message has been received, including any chunk trailer.

get_headerlist()
Returns all header names

The list is alphabetically sorted and lower-cased.

has_header (field_name)
True if this message has a header with field_name

5.3. HTTP Messages
get_header \((field\_name, list\_mode=False)\)

Returns the header with \(field\_name\) as a string.

- \*list\_mode=False* In this mode, get_header always returns a single binary string, this isn’t always what you want as it automatically ‘folds’ multiple headers with the same name into a string using ‘,’ as a separator.

- \*list\_mode=True* In this mode, get_header always returns a list of binary strings.

If there is no header with \(field\_name\) then None is returned in both modes.

set_header \((field\_name, field\_value, append\_mode=False)\)

Sets the header with \(field\_name\) to the string \(field\_value\).

- If \(field\_value\) is None then the header is removed (if present).

- If a header already exists with \(field\_name\) then the behaviour is determined by \(append\_mode\):

- \*append\_mode==True* \(field\_value\) is joined to the existing value using ‘,’ as a separator.

- \*append\_mode==False* (Default) \(field\_value\) replaces the existing value.

get_allow()

Returns an Allow instance or None if no “Allow” header is present.

set_allow (allowed)

Sets the “Allow” header, replacing any existing value.

- \(allowed\) A Allow instance or a string that one can be parsed from.

- If allowed is None any existing Allow header is removed.

get_authorization()

Returns a Credentials instance.

If there are no credentials None returned.

set_authorization (credentials)

Sets the “Authorization” header

- \(credentials\) a Credentials instance

get_cache_control()

Returns an CacheControl instance or None if no “Cache-Control” header is present.

set_cache_control (cc)

Sets the “Cache-Control” header, replacing any existing value.

- \(cc\) A CacheControl instance or a string that one can be parsed from.

- If \(cc\) is None any existing Cache-Control header is removed.

get_connection()

Returns a set of connection tokens from the Connection header.

If no Connection header was present an empty set is returned. All tokens are returned as lower case.

set_connection (connection_tokens)

Set the Connection tokens from an iterable set of connection_tokens

If the list is empty any existing header is removed.

get_content_encoding()

Returns a list of lower-cased content-coding tokens from the Content-Encoding header.

If no Content-Encoding header was present an empty list is returned.
Content-codings are always listed in the order they have been applied.

**set_content_encoding** *(content_codings)*

Sets the Content-Encoding header from a an iterable list of *content-coding* tokens. If the list is empty any existing header is removed.

**get_content_language** ()

Returns a list of *LanguageTag* instances from the Content-Language header

If no Content-Language header was present an empty list is returned.

**set_content_language** *(lang_list)*

Sets the Content-Language header from a an iterable list of *LanguageTag* instances.

**get_content_length** ()

Returns the integer size of the entity from the Content-Length header

If no Content-Length header was present None is returned.

**set_content_length** *(length)*

Sets the Content-Length header from an integer or removes it if *length* is None.

**get_content_location** ()

Returns a *pyslet.rfc2396.URI* instance created from the Content-Location header.

If no Content-Location header was present None is returned.

**set_content_location** *(location)*

Sets the Content-Location header from location, a *pyslet.rfc2396.URI* instance or removes it if *location* is None.

**get_content_md5** ()

Returns a 16-byte binary string read from the Content-MD5 header or None if no Content-MD5 header was present.

The result is suitable for comparing directly with the output of the Python’s MD5 digest method.

**set_content_md5** *(digest)*

Sets the Content-MD5 header from a 16-byte binary string returned by Python’s MD5 digest method or similar. If digest is None any existing Content-MD5 header is removed.

**get_content_range** ()

Returns a *ContentRange* instance parsed from the Content-Range header.

If no Content-Range header was present None is returned.

**set_content_range** *(range)*

Sets the Content-Range header from range, a *ContentRange* instance or removes it if *range* is None.

**get_content_type** ()

Returns a *MediaType* instance parsed from the Content-Type header.

If no Content-Type header was present None is returned.

**set_content_type** *(mtype=None)*

Sets the Content-Type header from mtype, a *MediaType* instance, or removes it if *mtype* is None.

**get_date** ()

Returns the value of the Date header.

The return value is a *params.FullDate* instance. If no Date header was present None is returned.

**set_date** *(date=None)*

Sets the value of the Date header

### 5.3. HTTP Messages
date a params.FullDate instance or None to remove the Date header.

To set the date header to the current date use:

```python
set_date(params.FullDate.from_now_utc())
```

get_last_modified()  
Returns the value of the Last-Modified header

The result is a params.FullDate instance. If no Last-Modified header was present None is returned.

set_last_modified(date=None)  
Sets the value of the Last-Modified header field

date a FullDate instance or None to remove the header

To set the Last-Modified header to the current date use:

```python
set_last_modified(params.FullDate.from_now_utc())
```

get_transfer_encoding()  
Returns a list of params.TransferEncoding

If no TransferEncoding header is present None is returned.

set_transfer_encoding(field_value)  
Set the Transfer-Encoding header

field_value A list of params.TransferEncoding instances or a string from which one can be parsed. If None then the header is removed.

set_upgrade(protocols)  
Sets the “Upgrade” header, replacing any existing value.

protocols An iterable list of params.ProductToken instances.

In addition to setting the upgrade header this method ensures that “upgrade” is present in the Connection header.

5.3.2 General Header Types

class pyslet.http.messages.CacheControl(*args)  
Bases: pyslet.http.params.Parameter

Represents the value of a Cache-Control general header.

Instances are immutable, they are constructed from a list of arguments which must not be empty. Arguments are treated as follows:

string a simple directive with no parameter

2-tuple of string and non-tuple a directive with a simple parameter

2-tuple of string and tuple a directive with a quoted list-style parameter

Instances behave like read-only lists implementing len, indexing and iteration in the usual way. Instances also support basic key lookup of directive names by implementing __contains__ and __getitem__ (which returns None for defined directives with no parameter and raises KeyError for undefined directives). Instances are not truly dictionary like.

classmethod from_str(source)  
Create a Cache-Control value from a source string.
5.3.3 Request Header Types

class pyslet.http.messages.AcceptList (*args)
    Bases: object

    Represents the value of an Accept header

    The built-in str function can be used to format instances according to the grammar defined in the specification.

    Instances are immutable, they are constructed from one or more AcceptItem instances. There are no comparison methods.

    Instances behave like read-only lists implementing len, indexing and iteration in the usual way.

    select_type (mtype_list)
        Returns the best match from mtype_list, a list of media-types

        In the event of a tie, the first item in mtype_list is returned.

classmethod from_str (source)
    Create an AcceptList from a source string.

class pyslet.http.messages.MediaRange (type='*', subtype='*', parameters={})
    Bases: pyslet.http.params.MediaType

    Represents an HTTP media-range.

    Quoting from the specification:

        “Media ranges can be overridden by more specific media ranges or specific media types. If more than one media range applies to a given type, the most specific reference has precedence.”

    We override the base class ordering so that MediaRange instances sort according to these rules. The following media ranges would be sorted in the order shown:

    1. image/png
    2. image/*
    3. text/plain; charset=utf-8
    4. text/plain
    5. text/*
    6. */*

    If we have two rules with identical precedence then we sort them alphabetically by type; sub-type and ultimately alphabetically by parameters

classmethod from_str (source)
    Creates a media-range from a source string.

    Unlike the parent media-type we ignore all spaces.

match_media_type (mtype)
    Tests whether a media-type matches this range.

    mtype A MediaType instance to be compared to this range.

    The matching algorithm takes into consideration wild-cards so that */* matches all types, image/* matches any image type and so on.

    If a media-range contains parameters then each of these must be matched exactly in the media-type being tested. Parameter names are treated case-insensitively and any additional parameters in the media type are ignored. As a result:
text/plain does not match the range application/myapp;charset=utf-8

application/myapp; charset=utf-8; option=on does match the range application/myapp; option=on

```python
class pyslet.http.messages.AcceptItem (range=MediaType('*', '*'), qvalue=1.0, extensions={})
   Bases: pyslet.http.params.SortableParameter

   Represents a single item in an Accept header

   Accept items are sorted by their media ranges. Equal media ranges sort by descending qvalue, for example:

   text/plain;q=0.75 < text/plain;q=0.5

   Extension parameters are ignored in all comparisons.

   range = None
      the MediaRange instance that is acceptable

   q = None
      the q-value (defaults to 1.0)

  classmethod from_str (source)
      Creates a single AcceptItem instance from a source string.
```

```python
class pyslet.http.messages.AcceptCharsetItem (token='*', qvalue=1.0)
   Bases: pyslet.http.messages.AcceptToken

   Represents a single item in an Accept-Charset header
```

```python
class pyslet.http.messages.AcceptCharsetList (*args)
   Bases: pyslet.http.messages.AcceptTokenList

   Represents an Accept-Charset header

   ItemClass
      alias of AcceptCharsetItem

   select_token (token_list)
      Overridden to provide default handling of iso-8859-1
```

```python
class pyslet.http.messages.AcceptEncodingItem (token='*', qvalue=1.0)
   Bases: pyslet.http.messages.AcceptToken

   Represents a single item in an Accept-Encoding header
```

```python
class pyslet.http.messages.AcceptEncodingList (*args)
   Bases: pyslet.http.messages.AcceptTokenList

   Represents an Accept-Encoding header

   ItemClass
      alias of AcceptEncodingItem

   select_token (token_list)
      Overridden to provide default handling of identity
```

```python
class pyslet.http.messages.AcceptLanguageItem (token='*', qvalue=1.0)
   Bases: pyslet.http.messages.AcceptToken

   Represents a single item in an Accept-Language header.
```

```python
class pyslet.http.messages.AcceptLanguageList (*args)
   Bases: pyslet.http.messages.AcceptTokenList

   Represents an Accept-Language header
```
**ItemClass**  
the class used to create items in this token list  
alias of `AcceptLanguageItem`

```python
select_token(token_list)
```
Remapped to `select_language()`

```python
class pyslet.http.messages.AcceptToken(token='*', qvalue=1.0)
```
Bases: `pyslet.http.params.SortableParameter`

Represents a single item in a token-based Accept-* header

AcceptToken items are sorted by their token, with wild cards sorting behind specified tokens. Equal values sort by descending qvalue, for example:

iso-8859-2;q=0.75 < iso-8859-2;q=0.5

```python
token = None
```
the token that is acceptable or “*” for any token

```python
q = None
```
the q-value (defaults to 1.0)

```python
classmethod from_str(source)
```
Creates a single AcceptToken instance from a `source` string.

```python
class pyslet.http.messages.AcceptTokenList(*args)
```
Bases: `pyslet.http.params.Parameter`

Represents the value of a token-based Accept-* header

Instances are immutable, they are constructed from one or more `AcceptToken` instances. There are no comparison methods.

Instances behave like read-only lists implementing len, indexing and iteration in the usual way.

**ItemClass**  
the class used to create new items in this list  
alias of `AcceptToken`

```python
select_token(token_list)
```
Returns the best match from `token_list`, a list of tokens.

In the event of a tie, the first item in `token_list` is returned.

```python
classmethod from_str(source)
```
Create an AcceptTokenList from a `source` string.

### 5.3.4 Response Header Types

```python
class pyslet.http.messages.AcceptRanges(*args)
```
Bases: `pyslet.http.params.SortableParameter`

Represents the value of an Accept-Ranges response header.

Instances are immutable, they are constructed from a list of string arguments. If the argument list is empty then a value of “none” is assumed.

Instances behave like read-only lists implementing len, indexing and iteration in the usual way. Comparison methods are provided.

---

5.3. HTTP Messages  

253
classmethod from_str(source)
Create an AcceptRanges value from a source string.

5.3.5 Entity Header Types

class pyslet.http.messages.Allow(*args)
Bases: pyslet.http.params.SortableParameter
Represents the value of an Allow entity header.
Instances are immutable, they are constructed from a list of string arguments which may be empty.
Instances behave like read-only lists implementing len, indexing and iteration in the usual way. Comparison methods are provided.

classmethod from_str(source)
Create an Allow value from a source string.

is_allowed(method)
Tests if method is allowed by this value.

class pyslet.http.messages.ContentRange(first_byte=None, last_byte=None, total_len=None)
Bases: object
Represents a single content range
first_byte Specifies the first byte of the range
last_byte Specifies the last byte of the range
total_len Specifies the total length of the entity
With no arguments an invalid range representing an unsatisfied range request from an entity of unknown length is created.
If first_byte is specified on construction last_byte must also be specified or TypeError is raised.
The built-in str function can be used to format instances according to the grammar defined in the specification.
Instances are immutable.

first_byte = None
first byte in the range

last_byte = None
last byte in the range

classmethod from_str(source)
Creates a single ContentRange instance from a source string.

is_valid()
Returns True if this range is valid, False otherwise.
A valid range is any non-empty byte range wholly within the entity described by the total length. Unsatisfied content ranges are treated as invalid.

5.3.6 Parsing Header Values

In most cases header values will be parsed automatically when reading them from messages. For completeness a header parser is exposed to enable you to parse these values from more complex strings.
class pyslet.http.messages.HeaderParser (source, ignore_sp=True)


A special parser for parsing HTTP headers from TEXT

In keeping with RFC2616 all parsing is done on binary strings. See base class for more information.

require_media_range ()

Parses a MediaRange instance.

Raises BadSyntax if no media-type was found.

require_accept_item ()

Parses an AcceptItem instance

Raises BadSyntax if no item was found.

require_accept_list ()

Parses an AcceptList instance

 Raises BadSyntax if no valid items were found.

require_accept_token (cls=pyslet.http.messages.AcceptToken)

Parses a single AcceptToken instance

Raises BadSyntax if no item was found.

cls  An optional sub-class of AcceptToken to create instead.

require_accept_token_list (cls=pyslet.http.messages.AcceptTokenList)

Parses a list of token-based accept items

Returns an AcceptTokenList instance. If no tokens were found then an empty list is returned.

cls  An optional sub-class of AcceptTokenList to create instead.

require_contentrange ()

Parses a ContentRange instance.

require_product_token_list ()

Parses a list of product tokens

Returns a list of params.ProductToken instances. If no tokens were found then an empty list is returned.

5.3.7 Exceptions

class pyslet.http.messages.HTTPException

Bases: exceptions.Exception

Class for all HTTP message-related errors.

5.4 HTTP Protocol Parameters

5.4.1 URLs

class pyslet.http.params.HTTPURL (octets='http://localhost/')

Bases: pyslet.rfc2396.ServerBasedURL

Represents http URLs
canonicalize()

Returns a canonical form of this URI

This method is almost identical to the implementation in ServerBasedURL except that a missing path is replaced by '/' in keeping with rules for making HTTP requests.


Represents https URLs

DEFAULT_PORT = 443

the default HTTPS port

5.4.2 Parameters

This module defines classes and functions for handling basic parameters used by HTTP. Refer to Section 3 of RFC2616 for details.

The approach taken by this module is to provide classes for each of the parameter types. Most classes have a class method `from_str` which returns a new instance parsed from a string and performs the reverse transformation to the `to_bytes` function. In all cases, string arguments provided on construction should be binary strings, not character strings.

Instances are generally immutable objects which is consistent with them representing values of parameters in the protocol. They can be used as values in dictionaries (`__hash__` is defined) and comparison methods are also provided, including inequalities where a logical ordering exists.

class pyslet.http.params.Parameter(Bases: object

Abstract base class for HTTP Parameters

Provides conversion to strings based on the `to_bytes()` method. In Python 2, also provides conversion to the unicode string type. In Python 3, implements `__bytes__` to enable use of `bytes(parameter)` which becomes portable as in Python 2 `__str__` is mapped to `to_bytes` too.

The HTTP grammar and the parsers and classes that implement it all use binary strings but usage of byte values outside those of the US ASCII codepoints is discouraged and unlikely to be portable between systems.

When required, Pyslet converts to character strings using the ISO-8859-1 codec. This ensures that the conversions never generate unicode decoding errors and is consistent with the text of RFC2616.

As the purpose of these modules is to provide a way to use HTTP constructs in other contexts too, parameters use `character` strings where possible. Therefore, if an attribute must represent a token then it is converted to a character string and `must` therefore be compared using character strings and not binary strings. For example, see the type and subtype attributes of `MediaType`. Similarly where tokens are passed as arguments to constructors these must also be character strings.

Where an attribute may be, or may contain, a value that would be represented as a quoted string in the protocol then it is stored as a binary string. You need to take particular care with parameter lists as the parameter names are tokens so are character strings but the parameter values are binary strings. The distinction is lost in Python 2 but the following code snippet will behave unexpectedly in Python 3 so for future compatibility it is better to make usage explicit now:

```python
Python 2.7

>>> from pyslet.http.params import MediaType
```
```python
globalMediaTypes = [
    "text/plain; charset=utf-8",
    "application/json; charset=utf-8",
    "image/png",
    "image/jpeg",
]
```

Python 3.5
```python
>>> from pyslet.http.params import MediaType
>>> t = MediaType.from_str("text/plain; charset=utf-8")
>>> "Yes" if t["charset"] == 'utf-8' else "No"
'Yes'
>>> "Yes" if t["charset"] == b'utf-8' else "No"
'Yes'
```

Such values may be set using character strings, in which case ISO-8859-1 is used to encode them.

**classmethod bstr(arg)**

Returns arg as a binary string

**classmethod bparameters(parameters)**

Ensures parameter values are binary strings

**to_bytest()**

Returns a binary string representation of the parameter

This method should be used in preference to str for compatibility with Python 3.

**class pyslet.http.params.SortableParameter**

Bases: `pyslet.py2SortableMixin, pyslet.http.params.Parameter`

Base class for sortable parameters

Inherits from SortableMixin allowing sorting to be implemented using a class-specific sortkey method implementation.

A __hash__ implementation that calls sortkey is also provided to enable instances to be used as dictionary keys.

**otherkey(other)**

Overridden to provide comparison with strings.

If other is of either character or binary string types then it is passed to the classmethod from_str which is assumed to return a new instance of the same class as self which can then be compared by the return value of sortkey.

This enables comparisons such as the following:

```python
>>> t = MediaType.from_str("text/plain")
>>> t == "text/plain"
True
>>> t > "image/png"
True
>>> t < "video/mp4"
True
```

**class pyslet.http.params.HTTPVersion(major=1, minor=None)**

Bases: `pyslet.http.params.SortableParameter`

Represents the HTTP Version.

**major** The (optional) major version as an int
The (optional) minor version as an int

The default instance, HTTPVersion(), represents HTTP/1.1

HTTPVersion objects are sortable (such that 1.1 > 1.0 and 1.2 < 1.25).

On conversion to a string the output is of the form:

HTTP/<major>.<minor>

For convenience, the constants HTTP_1p1 and HTTP_1p0 are provided for comparisons, e.g.:

```python
if HTTPVersion.from_str(version_str) < HTTP_1p1:
    # do something to support a legacy system...
```

major = None

major protocol version (read only)

minor = None

minor protocol version (read only)

```python
classmethod from_str(source)

Constructs an HTTPVersion object from a string.
```

```python
class pyslet.http.params.FullDate(src=None, date=None, time=None)


A special sub-class for HTTP-formatted dates

We extend the basic ISO TimePoint, mixing in the Parameter base class and providing an implementation of to_bytes.

The effect is to change the way instances are formatted while retaining other timepoint features, including comparisons. Take care not to pass an instance as an argument where a plain TimePoint is expected as unexpected formatting errors could result. You can always wrap an instance to convert between the two types:

```python
>>> from pyslet.iso8601 import TimePoint
>>> from pyslet.http.params import FullDate

>>> eagle = TimePoint.from_str('1969-07-20T15:17:40-05:00')
>>> print eagle
1969-07-20T15:17:40-05:00
>>> eagle = FullDate(eagle)
>>> print eagle
Sun, 20 Jul 1969 20:17:40 GMT
>>> eagle = TimePoint(eagle)
>>> print eagle
1969-07-20T15:17:40-05:00
```

Notice that when formatting the date is always expressed in GMT as per the recommendation in the HTTP specification.

```python
classmethod from_http_str(source)

Returns an instance parsed from an HTTP formatted string

There are three supported formats as described in the specification:

>>> "Sun, 06 Nov 1994 08:49:37 GMT"
>>> "Sunday, 06-Nov-94 08:49:37 GMT"
>>> "Sun Nov 6 08:49:37 1994"
```

to_bytes()

Formats the instance according to RFC 1123
The format is as follows:

```
Sun, 06 Nov 1994 08:49:37 GMT
```

This format is also described in RFC2616 in the production rfc1123-date.

```python
class pyslet.http.params.TransferEncoding (token='chunked', parameters={})
```

Bases: `pyslet.http.params.SortableParameter`

Represents an HTTP transfer-encoding.

- **token** The transfer encoding identifier, defaults to “chunked”
- **parameters** A parameter dictionary mapping parameter names to tuples of strings: (parameter name, parameter value)

When sorted, the order in which parameters were parsed is ignored. Instances are supported first by token and then by alphabetical parameter name/value pairs.

```python
token = None
```

the lower-cased transfer-encoding token (defaults to “chunked”)

```python
parameters = None
```

declared extension parameters

```python
classmethod from_str (source)
```

Parses the transfer-encoding from a `source` string.

If the encoding is not parsed correctly `BadSyntax` is raised.

```python
classmethod list_from_str (source)
```

Creates a list of transfer-encodings from a string

Transfer-encodings are comma-separated

```python
class pyslet.http.params.Chunk (size=0, extensions=None)
```

Bases: `pyslet.http.params.SortableParameter`

Represents an HTTP chunk header

- **size** The size of this chunk (defaults to 0)
- **extensions** A parameter dictionary mapping parameter names to tuples of strings: (chunk-ext-name, chunk-ext-val)

For completeness, instances are sortable by size and then by alphabetical parameter name, value pairs.

```python
size = None
```

the chunk-size

```python
classmethod from_str (source)
```

Parses the chunk header from a `source` string of `TEXT`.

If the chunk header is not parsed correctly `BadSyntax` is raised. The header includes the chunk-size and any chunk-extension parameters but it does not include the trailing CRLF or the chunk-data

```python
class pyslet.http.params.MediaType (type='application', subtype='octet-stream', parameters={})
```

Bases: `pyslet.http.params.SortableParameter`

Represents an HTTP media-type.

The built-in `str` function can be used to format instances according to the grammar defined in the specification.

- **type** The type code string, defaults to ‘application’
subtype  The sub-type code, defaults to ‘octet-stream’

parameters  A dictionary such as would be returned by \texttt{grammar.WordParser.parse.parameters()} containing the media type’s parameters.

Instances are immutable and support parameter value access by lower-case key (as a character string), returning the corresponding value or raising KeyError. E.g., mtype[‘charset’]

Instances also define comparison methods and a hash implementation. Media-types are compared by (lower case) type, subtype and ultimately parameters.

\texttt{classmethod from\_str(source)}

Creates a media-type from a \texttt{source} string.

Enforces the following rule from the specification:

Linear white space (LWS) MUST NOT be used between the type and subtype, nor between an attribute and its value

The source may be either characters or bytes. Character strings must consist of iso-8859-1 characters only and should be plain ascii.

\texttt{class \textcolor{red}{pyslet.http.params.ProductToken(token=None, version=None)}}

\texttt{Bases: pyslet.http.params.SortableParameter}

Represents an HTTP product token.

The comparison operations use a more interesting sort than plain text on version in order to provide a more intuitive ordering. As it is common practice to use dotted decimal notation for versions (with some alphanumeric modifiers) the version string is exploded (see \texttt{explode()}) internally on construction and this exploded value is used in comparisons. The upshot is that version 1.0.3 sorts before 1.0.10 as you would expect and 1.0a < 1.0 < 1.0.3a3 < 1.0.3a20 < 1.0.3b1 < 1.0.3; there are limits to this algorithm. 1.0dev > 1.0b1 even though it looks like it should be the other way around. Similarly 1.0-live < 1.0-prod etc.

You shouldn’t use this comparison as a definitive way to determine that one release is more recent or up-to-date than another unless you know that the product in question uses a numbering scheme compatible with these rules. On the other hand, it can be useful when sorting lists for human consumption.

\texttt{token = None}

the product’s token

\texttt{version = None}

the product’s version

\texttt{classmethod explode(version)}

Returns an exploded version string.

Version strings are split by dot and then by runs of non-digit characters resulting in a list of tuples. Numbers that have modified are treated as if they had a ~ suffix. This ensures that when sorting, 1.0 > 1.0a (i.e., qualifiers indicate earlier releases, ~ being the ASCII character with the largest codepoint).

Examples will help:

\begin{verbatim}
explode("2.15")==((2, "~"), (15, "~"))
explode("2.17b3")==((2, "~"), (17, "b", 3, "~"))
explode("2.b3")==((2, "~"), (-1, "b", 3, "~"))
\end{verbatim}

Note that a missing leading numeric component is treated as -1 to force “a3” to sort before “0a3”.

\texttt{classmethod from\_str(source)}

Creates a product token from a \texttt{source} string.
**classmethod list_from_str**(source)

Creates a list of product tokens from a source string.

Individual tokens are separated by white space.

**class** pyslet.http.params.LanguageTag**(primary, *subtags)**

**Bases:** pyslet.http.params.SortableParameter

Represents an HTTP language-tag.

Language tags are compared by lower casing all components and then sorting by primary tag, then by each sub-tag. Note that en sorts before en-US.

**partial_match**(range)

True if this tag is a partial match against range

**range** A tuple of lower-cased subtags. An empty tuple matches all instances.

For example:

```python
lang=LanguageTag("en",("US","Texas"))
lang.partial_match(())==True
lang.partial_match(("en",))==True
lang.partial_match(("en","us"))==True
lang.partial_match(("en","us","texas"))==True
lang.partial_match(("en","gb"))==False
lang.partial_match(("en","us","tex"))==False
```

**classmethod from_str**(source)

Creates a language tag from a source string.

Enforces the following rules from the specification:

White space is not allowed within the tag

**class** pyslet.http.params.EntityTag**(tag, weak=True)**

**Bases:** pyslet.http.params.SortableParameter

Represents an HTTP entity-tag.

**tag** The opaque tag

**weak** A boolean indicating if the entity-tag is a weak or strong entity tag. Defaults to True.

Instances are compared by tag and then, if the tags match, by wheather the tag is weak or not.

**weak = None**

True if this is a weak tag

**tag = None**

the opaque tag

**classmethod from_str**(source)

Creates an entity-tag from a source string.

---

**Parsing Parameter Values**

In most cases parameter values will be parsed directly by the class methods provided in the parameter types themselves. For completeness a parameter parser is exposed to enable you to parse these values from more complex strings.

---

5.4. HTTP Protocol Parameters 261
class pyslet.http.params.ParameterParser(source, ignore_sp=True)

    An extended parser for parameter values

    This parser defines attributes for dealing with English date names that are useful beyond the basic parsing
    functions to allow the formatting of date information in English regardless of the locale.

    require_http_version()
        Parses an HTTPVersion instance
        Returns an HTTPVersion instance.

    wkday = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
        A list of English day-of-week abbreviations: wkday[0] == "Mon", etc.

    weekday = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
        A list of English day-of-week full names: weekday[0] == "Monday"

        A list of English month names: month[0] == "Jan", etc.

    require_fulldate()
        Parses a FullDate instance.
        Returns a FullDate instance or raises BadSyntax if none is found.

    parse_delta_seconds()
        Parses a delta-seconds value, see WordParser.parse_integer()

    parse_charset()
        Parses a charset, see WordParser.parse_tokenlower()

    parse_content_coding()
        Parses a content-coding, see WordParser.parse_tokenlower()

    require_transfer_encoding()
        Parses a TransferEncoding instance

    require_chunk()
        Parses a chunk header
        Returns a Chunk instance.

    require_media_type()
        Parses a MediaType instance.

    require_product_token()
        Parses a ProductToken instance.
        Raises BadSyntax if no product token was found.

    parse_qvalue()
        Parses a qvalue returning a float
        Returns None if no qvalue was found.

    require_language_tag()
        Parses a language tag returning a LanguageTag instance. Raises BadSyntax if no language tag was found.

    require_entity_tag()
        Parses an entity-tag returning a EntityTag instance. Raises BadSyntax if no language tag was found.
5.5 HTTP Grammar

5.5.1 Using the Grammar

The functions and data definitions here are exposed to enable normative use in other modules. Use of the grammar itself is typically through use of a parser. There are two types of parser, an OctetParser that is used for parsing raw strings (or octets, represented by bytes in Python) such as those obtained from the HTTP connection itself and a WordParser that tokenizes the input string first and then provides a higher-level word-based parser.

class pyslet.http.grammar.OctetParser(source)
    Bases: pyslet.unicode5.BasicParser

A special purpose parser for parsing HTTP productions

Strictly speaking, HTTP operates only on bytes so the parser is always set to binary mode. However, as a concession to the various normative references to HTTP in other specifications where character strings are parsed they will be accepted provided they only contain US ASCII characters.

parse_lws()
    Parses a single instance of the production LWS

    The return value is the LWS string parsed or None if there is no LWS.

parse_onetext (unfold=False)
    Parses a single TEXT instance.

    unfold  Pass True to replace folding LWS with a single SP. Defaults to False.

    Parses a single byte or run of LWS matching the production TEXT. The return value is either:

    1 a single byte of TEXT (not a binary string) excluding the LWS characters

    2 a binary string of LWS

    3 None if no TEXT was found

    You may find the utility function pyslet.py2.is_byte() useful to distinguish cases 1 and 2 correctly in both Python 2 and Python 3.

parse_text (unfold=False)
    Parses TEXT

    unfold  Pass True to replace folding LWS with a single SP. Defaults to False.

    Parses a run of characters matching the production TEXT. The return value is the matching TEXT as a binary string (including any LWS) or None if no TEXT was found.

parse_token()
    Parses a token.

    Parses a single instance of the production token. The return value is the matching token as a binary string or None if no token was found.

parse_comment (unfold=False)
    Parses a comment.

    unfold  Pass True to replace folding LWS with a single SP. Defaults to False.

    Parses a single instance of the production comment. The return value is the entire matching comment as a binary string (including the brackets, quoted pairs and any nested comments) or None if no comment was found.
**parse_cctext** *(unfold=False)*

Parses cctext.

- **unfold** Pass True to replace folding LWS with a single SP. Defaults to False.

Parses a run of characters matching the production cctext. The return value is the matching cctext as a binary string (including any LWS) or None if no cctext was found.

The original text of RFC2616 is ambiguous in the definition of cctext but the later errata corrected this to exclude the backslash byte ($5C$) so we stop if we encounter one.

**parse_quoted_string** *(unfold=False)*

Parses a quoted-string.

- **unfold** Pass True to replace folding LWS with a single SP. Defaults to False.

Parses a single instance of the production quoted-string. The return value is the entire matching string (including the quotes and any quoted pairs) or None if no quoted-string was found.

**parse_qdtext** *(unfold=False)*

Parses qdtext.

Parses a run of characters matching the production qdtext. The return value is the matching qdtext string (including any LWS) or None if no qdtext was found.

If **unfold** is True then any folding LWS is replaced with a single SP. It defaults to False

Although the production for qdtext would include the backslash character we stop if we encounter one, following the RFC2616 errata instead.

**parse_quoted_pair**

Parses a single quoted-pair.

The return value is the matching binary string including the backslash so it will always be of length 2 or None if no quoted-pair was found.

---

**class** `pyslet.http.grammar.WordParser` *(source, ignore_sp=True)*

Bases: `pyslet.unicode5.ParserMixin`

A word-level parser and tokeniser for the HTTP grammar.

- **source** The binary string to be parsed into words. It will normally be valid TEXT but it can contain control characters if they are escaped as part of a comment or quoted string. For compatibility, character strings are accepted provided they only contain US ASCII characters

- **ignore_sp** (defaults to True) LWS is unfolded automatically. By default the parser ignores spaces according to the rules for implied LWS in the specification and neither SP nor HT will be stored in the word list. If you set **ignore_sp** to False then LWS is not ignored and each run of LWS is returned as a single SP in the word list.

The source is parsed completely into words on construction using `OctetParser`. If the source contains a CRLF (or any other non-TEXT bytes) that is not part of a folding or escape sequence it raises `ParserError`.

For the purposes of this parser, a word may be either a single byte (in which case it is a separator or SP, note that HT is never stored in the word list) or a binary string, in which case it is a token, a comment or a quoted string.

Warning: in Python 2 a single byte is indistinguishable from a binary string of length 1.

Methods follow the same pattern as that described in the related `pyslet.unicode5.BasicParser` using `match_`, `parse_` and `require_` naming conventions. It also includes the `pyslet.unicode5.ParseMixin` class to enable the convenience methods for converting between look-ahead and non-look-ahead parsing modes.

- **pos = None** a pointer to the current word in the list
the_word = None
    the current word or None

setpos(pos)
    Sets the current position of the parser.
    Example usage for look-ahead:

```
# wp is a WordParser instance
savepos=wp.pos
try:
    # parse a token/sub-token combination
    token=wp.require_token()
    wp.require_separator(byte('/'))
    subtoken=wp.require_token()
    return token,subtoken
except BadSyntax:
    wp.setpos(savepos)
    return None,
```

parser_error(production=None)
    Raises an error encountered by the parser
    See BadSyntax for details.
    If production is None then the previous error is re-raised. If multiple errors have been raised previously the one with the most advanced parser position is used. The operation is similar to `pyslet.unicode5.BasicParser.parser_error()`.
    To improve the quality of error messages an internal record of the starting position of each word is kept (within the original source).
    The position of the parser is always set to the position of the error raised.

match_end()
    True if all of words have been parsed

peek()
    Returns the next word
    If there are no more words, returns None.

parse_word()
    Parses any word from the list
    Returns the word parsed or None if the parser was already at the end of the word list.

parse_word_as_bstr()
    Parses any word from the list
    Returns a binary string representing the word. In cases where the next work is a separator it converts the word to a binary string (in Python 2 this is a noop) before returning it.

is_token()
    Returns True if the current word is a token

parse_token()
    Parses a token from the list of words
    Returns the token or None if the next word was not a token. The return value is a binary string. This is consistent with the use of this method for parsing tokens in contexts where a token or a quoted string may be present.
parse_tokenlower()
Returns a lower-cased token parsed from the word list
Returns None if the next word was not a token. Unlike parse_token() the result is a character string.

parse_tokenlist()
Parses a list of tokens
Returns the list or [] if no tokens were found. Lists are defined by RFC2616 as being comma-separated.
Note that empty items are ignored, so strings such as “x,y” return just [“x”, “y”].
The list of tokens is returned as a list of character strings.

require_token(expected=’token’)
Returns the current token or raises BadSyntax
expected the name of the expected production, it defaults to “token”.

is_integer()
Returns True if the current word is an integer token

parse_integer()
Parses an integer token from the list of words
Return the integer’s value or None.

require_integer(expected=’integer’)
Parses an integer or raises BadSyntax
expected can be set to the name of the expected object, defaults to “integer”.

is_hexinteger()
Returns True if the current word is a hex token

parse_hexinteger()
Parses a hex integer token from the list of words
Return the hex integer’s value or None.

require_hexinteger(expected=’hex integer’)
Parses a hex integer or raises BadSyntax
expected can be set to the name of the expected object, defaults to “hex integer”.

is_separator(sep)
Returns True if the current word matches sep

parse_separator(sep)
Parses a sep from the list of words.
Returns True if the current word matches sep and False otherwise.

require_separator(sep, expected=None)
Parses sep or raises BadSyntax
sep A separator byte (not a binary string).
expected can be set to the name of the expected object

is_quoted_string()
Returns True if the current word is a quoted string.

parse_quoted_string()
Parses a quoted string from the list of words.
Returns the decoded value of the quoted string or None.
parse_sp()

Parses a SP from the list of words.

Returns True if the current word is a SP and False otherwise.

parse_parameters(parameters, ignore_allsp=True, case_sensitive=False, qmode=None)

Parses a set of parameters

parameters  the dictionary in which to store the parsed parameters
ignore_allsp  a boolean (defaults to True) which causes the function to ignore all LWS in the word list. If
set to False then space around the ‘=’ separator is treated as an error and raises BadSyntax.
(case_sensitive  controls whether parameter names are treated as case sensitive, defaults to False.
qmode  allows you to pass a special parameter name that will terminate parameter parsing (without being
parsed itself). This is used to support headers such as the “Accept” header in which the parameter
called “q” marks the boundary between media-type parameters and Accept extension parameters.
Defaults to None

Updates the parameters dictionary with the new parameter definitions. The key in the dictionary is the
parameter name (converted to lower case if parameters are being dealt with case insensitively) and the
value is a 2-item tuple of (name, value) always preserving the original case of the parameter name.

Returns the parameters dictionary as the result. The method always succeeds as parameter lists can be
empty.

Compatibility warning: parameter names must be tokens and are therefore converted to character
strings. Parameter values, on the other hand, may be quoted strings containing characters from unknown character
sets and are therefore always represented as binary strings.

parse_remainder(sep=“”)

Parses the rest of the words

The result is a single string representing the remaining words joined with sep, which defaults to an empty
string.

Returns an empty string if the parser is at the end of the word list.

5.5.2 Basic Syntax

This section defines functions for handling basic elements of the HTTP grammar, refer to Section 2.2 of RFC2616 for
details.

The HTTP protocol only deals with octets so the following functions take a single byte as an argument and return True
if the byte matches the production and False otherwise. As a convenience they all accept None as an argument and
will return False.

A byte is defined as the type returned by indexing a binary string and is therefore an integer in the range 0..255 in
Python 3 and a single character string in Python 2.

pyslet.http.grammar.is_octet(b)

Returns True if a byte matches the production for OCTET.

pyslet.http.grammar.is_char(b)

Returns True if a byte matches the production for CHAR.

pyslet.http.grammar.is_upalpha(b)

Returns True if a byte matches the production for UPALPHA.

pyslet.http.grammar.is_loalpha(b)

Returns True if a byte matches the production for LOALPHA.
pyslet.http.grammar.is_alpha(b)
Returns True if a byte matches the production for ALPHA.

pyslet.http.grammar.is_digit(b)
Returns True if a byte matches the production for DIGIT.

pyslet.http.grammar.is_digits(src)
Returns True if all bytes match the production for DIGIT.

Empty strings return False

pyslet.http.grammar.is_ctl(b)
Returns True if a byte matches the production for CTL.

pyslet.http.grammar.is_separator(b)
Returns True if a byte is a separator

pyslet.http.grammar.is_hex(b)
Returns True if a byte matches the production for HEX.

The following constants are defined to speed up comparisons, in each case they are the byte (see above) corresponding to the syntax elements defined in the specification.

And similarly, these byte constants are not defined in the grammar but are useful for comparisons. Again they are the byte representing these separators and will have a different type in Python 2 and 3.

The following binary string constant is defined for completeness:

There are no special definitions for LWS and TEXT, these productions are handled by OctetParser

The following functions operate on binary strings. Note that in Python 2 a byte is also a binary string (of length 1) but in Python 3 a byte is not a valid string. Use pyslet.py2.byte_to_bstr() if you need to create a binary string from a single byte.

pyslet.http.grammar.is_hexdigits(src)
Returns True if all bytes match the production for HEX.

Empty strings return False

pyslet.http.grammar.check_token(t)
 Raises ValueError if t is not a valid token

  t A binary string, will also accept a single byte.

  Returns a character string representing the token on success.

pyslet.http.grammar.decode_quoted_string(qstring)
Decodes a quoted string, returning the unencoded string.

  Surrounding double quotes are removed and quoted bytes, bytes preceded by $5C (backslash), are unescaped.

  The return value is a binary string. In most cases you will want to decode it using the latin-1 (iso-8859-1) codec as that was the original intention of RFC2616 but in practice anything outside US ASCII is likely to be non-portable.

pyslet.http.grammar.quote_string(s, force=True)
Places a string in double quotes, returning the quoted string.

  force Always quote the string, defaults to True. If False then valid tokens are not quoted but returned as-is.

  This is the reverse of decode_quoted_string(). Note that only the double quote, and CTL characters other than SP and HT are quoted in the output.
5.5.3 Misc Functions

```python
def format_parameters(parameters):
    """
    Formats a dictionary of parameters
    This function is suitable for formatting parameter dictionaries parsed by WordParser.
    ParseParameters. These dictionaries are key/value pairs where the keys are character
    strings and the values are binary strings.
    Parameter values are quoted only if their values require it, that is, only if their values are not
    valid tokens.
    """
```

5.5.4 Exceptions

```python
class BadSyntax(production='', parser=None):
    """
    Raised by the WordParser whenever a syntax error is encountered by the parsers. Note that
    tokenization errors are raised separately during construction itself.
    production The name of the production being parsed. (Defaults to an empty string.)
    parser The WordParser instance raising the error (optional)
    BadSyntax is a subclass of ValueError.
    """
```

5.6 HTTP Cookies

This module contains classes for handling Cookies, as defined by RFC6265 HTTP State Management Mechanism

5.6.1 Client Scenarios

By default, Pyslet’s HTTP client does not support cookies. Adding support, if you want it, is done with the
CookieStore class. All you need to do is create an instance and add it to the client before processing any requests:

```python
from pyslet.http.client import http

client = http.Client()
cookie_store = http.cookie.CookieStore()
client.set_cookie_store(cookie_store)
```

Support for cookies is then transparently added to each request.

By default, the CookieStore object does not support domain cookies because it doesn’t know which domains are
effectively top level domains (TLDs) so treats all domains as effective TLDs. Domain cookies can’t be stored for
TLDs as this would allow a website at www.exampleA.com to set or overwrite a cookie in the ‘com’ domain which
would then be sent to www.exampleB.com. There are lots of reasons why this is a bad idea, websites could disrupt
each others operation or worse, compromise security and user privacy.

For most applications you can fix this by creating exceptions for domains you want your client to trust. For example,
if you want to interact with www.example.com and www2.example.com you might want to allow domain cookies for
example.com, knowing that the effective TLD in this case is simply ‘com’.

```python
cookie_store.add_private_suffix('example.com')
```
If you want to emulate the behaviour of real browsers you will need to upload a proper database of effective TLDs. For more information see `CookieStore.fetch_public_suffix_list()` and `CookieStore.set_public_list()`. Be warned, the public suffix list changes routinely and you’ll want to ensure you have the latest values loaded.

### 5.6.2 Web Application Scenarios

If you are writing a web application you may want to handle cookies directly by adding response headers explicitly to a response object provided by your web framework.

There are two classes for representing cookie definitions, you should use the stricter `Section4Cookie` when creating cookies as this follows the recommended syntax in the RFC and will catch problems such as attempting to set a cookie value containing a comma. Although user agents are supposed to cope with such values some systems are now rejecting cookies that do not adhere to the stricter section 4 definitions.

The following code creates a cookie called SID with a maximum lifespan of 15 minutes:

```python
import pyslet.http.cookie as cookie
c = cookie.Section4Cookie("SID", "31d4d96e407aad42", max_age=15*60,
               path="/", http_only=True, secure=True)
print c
```

It outputs the text required to set the Set-Cookie header:

```
SID=31d4d96e407aad42; Path=/; Max-Age=900; Secure; HttpOnly
```

You may want to add additional attributes such as an expires time for backwards compatibility or a domain to allow the cookie to be sent to other websites in a shared domain. See `Cookie` for details.

### 5.6.3 Reference

```python
class pyslet.http.cookie.Cookie(name, value, path=None, domain=None, expires=None,
                   max_age=None, secure=False, http_only=False, extensions=None)
```

Bases: `pyslet.http.params.Parameter`

Represents the definition of a cookie

Where binary strings are required, character strings will be accepted and converted using UTF-8 but non-ASCII characters are not portable and should be avoided.

- **name** The name of the cookie as a binary string.
- **value** The value of the cookie as a binary string.
- **path** (optional) A character string containing the path of the cookie. If None then the ‘directory’ of the page that returned the cookie will be used by the client.
- **domain** (optional) A character string containing the domain of the cookie. If None then the host name of the server that returned the cookie will be used by the client and the cookie will be treated as ‘host only’.
- **expires** (optional) An `TimePoint` instance. If None then the cookie will be treated as a session cookie by the client.
- **max_age** (optional) An integer, the length of time before the cookie expires in seconds. Overrides the expires value. If None then the value of expires is used instead, if both are None then the cookie will be treated as a session cookie by the client.
secure (Default: False)  Whether or not the cookie should be exposed only over secure protocols, such as https.

http_only (Default: False)  Whether or not the cookie should be exposed only via the HTTP protocol. Recommended value: True!

extensions A list of binary strings containing attribute extensions. The strings should be of the form name=value but this is not enforced.

Instances can be converted to strings using the builtin str function and the output that results is a valid Set-Cookie header value.

name = None
     the cookie’s name

value = None
     the cookie’s name

path = None
     the cookie’s path

domain = None
     the cookie’s domain

secure = None
     the cookie’s secure flag

http_only = None
     the cookie’s httponly flag

creation_time = None
     the creation time of the cookie, initialised to the current time as returned by the builtin time.time function.

access_time = None
     the last access time of the cookie, initialised to the current time as returned by the builtin time.time function.

expires_time = None
     the expiry time of the cookie, as an integer compatible with the value returned by time.time

max_age = None
     the max_age value

expires = None
     the expires value as passed to the constructor, this is preserved and is used when serialising the definition even if Max-Age is also in effect. Some older clients may not support Max-Age and they will look at the Expires time instead.

classmethod bstr(arg)
     Overridden to use UTF-8 for binary encoding of arguments

     This method is used to convert arguments provided in constructors to the binary strings required by some attributes. The default implementation uses ISO-8859-1 in keeping with the general HTTP specification (although no encoding is portable across a wide range of browser and server implementations).

     We override it here for Cookies because the cookie specification hints that UTF-8 would be an appropriate choice for displaying binary information found in cookies.

classmethod cstr(value)
     Used to interpret binary strings in cookies

     This method can be used as a default way of interpreting binary information found in cookies. It tries to decode using UTF-8 but, if that fails, it reverts to the default HTTP encoding of ISO-8859-1. It is used sparingly, in most cases binary values are left uninterpreted but attributes such as the path and domain must be interpreted in relation to components of the URL and URLs use characters.

5.6. HTTP Cookies
For clarity, a domain name containing non-ASCII characters (U-labels) that has simply been UTF-8 encoded will be converted back to the original form (with U-labels) whereas the same domain correctly encoded in ACE format (xn-) will be unchanged by this decoding.

```
classmethod from_str(src)
Creates a new instance from a src string

The string is parsed using the generous parsing rules of Section 5 of the specification. Returns a new instance.

is_persistent()
Returns True if there is no expires time on this cookie.

The expires time is calculated from either the max_age or expires attributes.

is_hostonly()
Returns True if this cookie is 'host only'

In other words, it should only be sent to the host that set the cookie originally.

touch(now=None)
Updates the cookie’s last access time.

now (optional) Time value to use. This can be in the past or the future and improves performance when updating multiple cookies simultaneously.

expired(now=None)
Returns True if the cookie has expired

now (optional) Time value at which to test, this can be in the past or the future and is largely provided to aid testing and also to improve performance when a large number of cookies need to be tested sequentially.
```

class pyslet.http.cookie.Section4Cookie(*args, **kwargs)
Bases: pyslet.http.cookie.Cookie

Represents a strict cookie definition

The purpose of this class is wrap Cookie to enforce more validation rules on the definition to ensure that the cookie adheres to section 4 syntax, and not just the broader section 5 syntax.

Names are checked for token validity, values are checked against the syntax for cookie-value and the attributes are checked against the other constraints in the specification.

The built-in str function will return a string that is valid against the section 4 syntax.

```
classmethod from_str(src)
Creates a new instance from a src string

Overridden to provide stricter parsing. This may still appear more generous than expected because the strict syntax allows an unrestricted set of attribute extensions so unrecognised attributes will often be recorded but not in any useful way.
```

Client Support

User agents that support cookies are obliged to keep a cookie store in which cookies can be saved and retrieved keyed on their domain, path and cookie name.

Pyslet’s approach is to provide an in-memory store with nodes defined for each domain (host) that a cookie has been associated with or which is the target of a public or private suffix rule. Nodes are also created for any implied parent domains and the result is a tree-like structure of dictionaries that can be quickly searched for each request.
class pyslet.http.cookie.CookieStore
    Bases: object

    An object that provides in-memory storage for cookies.

    There are no initialisation options. By default, the cookie storage will refuse all ‘domain’ cookies. That is, cookies that have a domain attribute. If a domain cookie is received from a host that exactly matches its domain attribute then it is converted to a host-only cookie and is stored.

    This behaviour can be changed by adding exclusions (in the form of calls to add_private_suffix()) or by loading in a new public prefix database using set_public_list().

    set_cookie(url, c)
        Store a cookie.

        url A URI instance representing the resource that is setting the cookie.

        c A Cookie instance, typically parsed from a Set-Cookie header returned when requesting the resource at url.

        If the cookie can’t be set then CookieError is raised. Reasons why a cookie might be refused are a mismatch between a domain attribute and the url, or an attempt to set a cookie in a public domain, such as ‘co.uk’.

    search(url)
        Searches for cookies that match a resource

        url A URI instance representing the resource that we want to find cookies for.

        The return result is a sorted list of Cookie objects. The sort order is defined in the specification, longer paths are sorted first, otherwise older cookies are listed before newer ones.

        Expired cookies are automatically removed from the repository and all cookies returned have their access time updated to the current time.

    expire_cookies(now=None, dnode=None)
        Expire stored cookies.

        now (optional) The time at which to expire the cookies, defaults to the current time. This can be used to expire cookies based on some past or future point.

        Iterates through all stored cookies and removes any that have expired.

    end_session(now=None, dnode=None)
        Expire all session cookies.

        now (optional) The time at which to expire cookies. See expire_cookies() for details.

        Iterates through all stored cookies and removes any session cookies in addition to any that have expired.

    add_public_suffix(suffix)
        Marks a domain suffix as being public.

        suffix A string: a public suffix, may contain wild-card characters to match any entire label, for example: “.uk”, “.tokyo.jp”, “com”

        Once a domain suffix is marked as being public future cookies will not be stored against that suffix (except in the unusual case where a cookie is ‘host only’ and the host name is a public suffix).

    add_private_suffix(suffix)
        Marks a domain suffix as being private.

        suffix A string: a public suffix, may contain wild-card characters to match any entire label, for example: “example.co.uk”, “*.tokyo.jp”, “com”
This method is required to override an existing public rule, thereby ensuring that future cookies can be stored against domains matching this suffix.

**classmethod fetch_public_suffix_list** *(fpath, src='https://publicsuffix.org/list/effective_tld_names.dat', overwrite=False)*

Fetches the public suffix list and saves to fpath

- **fpath** A local file path to save the file in
- **src** A string or **URI** instance pointing at the file to retrieve. It default to the data file https://publicsuffix.org/list/effective_tld_names.dat
- **overwrite** *(Default: False)* A flag to force an overwrite of an existing file at fpath, by default, if fpath already exists this method returns without doing anything.

**set_public_list** *(black_list, tld_depth=1)*

Loads a new public suffix list

- **black_list** A string containing a list of public suffixes in the format defined by: https://publicsuffix.org/list/
- **tld_depth** *(Default: 1)* The depth of domain that will be automatically treated as public. The default is 1, meaning that all top-level domains will be treated as public.

This methods loads data from a public list using calls to **add_public_prefix()** and **add_private_prefix()**, the latter being for exclusion rules.

If you use the full list published by the Public Suffix List project it is safe to use the default tld_depth value of 1:

https://publicsuffix.org/list/effective_tld_names.dat

If you want to load a much smaller list then you should focus on a large value for tld_depth (255 for example) and documenting exclusions only. For example:

```python
// Exclusion list
// Accept domain cookies for example.com, example.co.uk
!example.com
!example.co.uk
```

**test_public_domain** *(domain_str)*

Test if a domain is public

- **domain_str** A domain string, e.g., “www.example.com”

Returns True if this domain is marked as public, False otherwise.

**get_registered_domain** *(domain_str, u_labels=False)*

Returns the publicly registered portion of a domain

- **domain_str** A domain string, e.g., “www.example.com”
- **u_labels** *(Default: False)* Flag indicating whether or not to return unicode labels instead of encoded ASCII Labels.

Compares this domain against the database of public domains and returns the publicly registered part of the domain. For example, www.example.com would typically return example.com and www.example.co.uk would typically return example.co.uk.

If domain_str is already a publicly registered domain then it returns None. If domain_str is itself None, None is also returned.

Initially, all domains are marked as public so this function will always return None. It is intended for use after a public list has been loaded, such as the public suffix list (see **set_public_list()**).
check_public_suffix(domain_str, match_str)
See Public Suffix Test Data for details.
http://mxr.mozilla.org/mozilla-central/source/network/test/unit/data/test_psl.txt?raw=1
Returns True if there is a match, False otherwise. Negative results are logged at ERROR level. Used for testing the public suffixes loaded with set_public_list().

Syntax
The following basic functions can be used to test characters against the syntax productions defined in the specification. In each case, if the argument is None then False is returned.

class pyslet.http.cookie.CookieParser(source)
General purpose class for parsing RFC6265 productions
Unlike the basic syntax functions these methods allow a longer string, such as that received from an HTTP header, to be parsed into its component parts.

Methods follow inherited naming conventions, require_ methods raise a ValueError if the production is not matched whereas parse_ methods optionally parse a production if it is present and return None if not present.

require_set_cookie_string(strict=False)
Parses the set-cookie-string production

strict (Default: False) Use the stricter section 4 syntax rules instead of the more permissive algorithm described in section 5.2

This is the format of the Set-Cookie header, it returns a Cookie instance or None if this cookie definition should be ignored.

require_cookie_string(strict=False)
Parses the value of a Cookie header.

strict (Default: False) Indicates if stricter section 4 parsing is required.

Returns a dictionary of values, the keys are the names of the cookies in the cookie string and the values are either strings or, in the case of multiply defined names, sets of strings. We use sets as the specification makes it clear that you should not rely on the order of such definitions. All strings (including cookie names) are binary strings.

require_name_value_pair()
Returns a (name, value) pair

Parsed according to the looser section 5 syntax so will allow almost anything as a name and value provided it has an ‘=’.

require_cookie_pair()
Returns a (name, value) pair parsed according to cookie-pair

Parsed according to the stricter section 4 syntax so will only accept valid tokens as names, the ‘=’ is required and the value must be parseable with require_cookie_value().

parse_cookie_pair()
See: require_cookie_pair()

If not parsed returns (None, None) rather than just None.

require_cookie_value()
Returns a cookie-value (binary) string.
Parsed according to the stricter section 4 syntax so will not allow whitespace, comma, semicolon or backslash characters and will only allow double-quote when it is used to completely “enclose” the value, in which case the double-quotes are still considered to be part of the value string.

**parse_cookie_av()**

Parses a cookie-av string.

This production is effectively the production for extension-av in the stricter section 4 syntax. Effectively it returns everything up to but not including the next ';' or CTL character.

It never returns None, if nothing is found it returns an empty string instead.

**parse_sane_cookie_date()**

Parses the sane-cookie-date production.

This is the stricter syntax defined in section 4. The returns result is a `FullDate` instance.

**parse_cookie_date_tokens()**

Parses a date-token-list

This uses the weak section 5.1 syntax

It never returns None, if there are no tokens then it returns an empty list. Delimiters are always discarded.

**require_cookie_date()**

Parses a date value

This uses the weak section 5.1 syntax and the algorithm described there. It absorbs almost all errors returning None if this date value should be ignored - but warnings are logged to alert you to the failure. The implications of replacing a date with None in this syntax are typically that a cookie that is supposed to be persistent become session only. However, if this was an attempt to remove a cookie with a very early date then the failure could cause more problems.

If successful, it returns a `FullDate` instance.

**Date and Time**

**pyslet.http.cookie.split_year(year_str)**

Parses a year from a binary string

Uses the generous rules in section 5.1 and returns a year value, adjusted using the 2-digit year algorithm documented there.

If a year value can’t be found ValueError is raised.

**pyslet.http.cookie.split_month(month_str)**

Parses a month from a string

Uses the generous rules in section 5.1 and returns a month value from 1 (January) to 12 (December).

If a month value can’t be found ValueError is raised.

**pyslet.http.cookie.split_day_of_month(dom_str)**

Parses a day-of-month from a binary string

Users the generous rules in section 5.1 and returns a single integer or raises ValueError if a valid day of month can’t be found.

**pyslet.http.cookie.split_time(time_str)**

Parses a time from a binary string

Users the generous rules in section 5.1 and returns a triple of hours, minutes, seconds. These values are unchecked!
If the time can’t be found `ValueError` is raised.

**Basic Syntax**

These functions follow the pattern of behaviour defined in the *HTTP Grammar* module, taking a *byte* as an argument. They will all return `False` if the argument is `None`.

```python
def is_delimiter(b):
    """Tests a character against the production delimiter"
    This production is from the weaker section 5 syntax of RFC6265.

def is_non_delimiter(b):
    """Tests a character against the production non-delimiter"
    The result differs from using `not is_delimiter` only in the handling of `None` which will return `False` when passed to either function.

def is_non_digit(b):
    """Tests a character against the production non-digit."

def is_cookie_octet(b):
    """Tests a byte against production cookie_octet"
```

**Domain Name Syntax**

```python
def domain_in_domain(subdomain, domain):
    """Returns `try` if `subdomain` is a sub-domain of `domain`.
    `subdomain` A reversed list of strings returned by `split_domain()`
    `domain` A reversed list of strings as returned by `split_domain()`
    For example:
    
    >>> domain_in_domain(['com', 'example'], ['com', 'example', 'www'])
    True
```

```python
def split_domain(domain_str, allow_wildcard=False):
    """Splits a domain string"
    `domain_str` A character string, or a UTF-8 encoded binary string.
    `allow_wildcard` (Default: `False`) Allows the use of a single ‘*’ character as a domain label for the purposes of parsing wildcard domain definitions.
    Returns a list of lower cased *ASCII* labels as character strings, converting U-Labels to ACE form (xn–) in the process. For example:
    
    >>> split_domain('example.COM')
    >>> ['example', 'com']
    >>> split_domain(u'xn--85x722f.com.cn')
    >>> ['xn--85x722f', 'com', 'cn']
    Raises `ValueError` if `domain_str` is not valid.
```

```python
def encode_domain(domain, allow_wildcard=False):
    """Returns domain correctly encoded as a binary string"
```

5.6. HTTP Cookies 277
**domain** A binary or character string containing a representation of a domain using either U-Labels or ACE form for non-ASCII characters.

**allow_wildcard** *(Default: False)* Allows the use of a single ‘*’ character as a domain label for the purposes of encoding wildcard domain definitions.

The result is a character string containing only ASCII encoded characters.

```python
pyslet.http.cookie.is_ldh_label(label)
```
Tests a binary string against the definition of LDH label

LDH Label is defined in RFC5890 as being the classic label syntax defined in RFC1034 and updated in RFC1123. To cut a long story short the update in question is described as follows:

One aspect of host name syntax is hereby changed: the restriction on the first character is relaxed to allow either a letter or a digit.

Although not spelled out there this would make the updated syntax:

```
<label> ::= <let-dig> \[ \[ <ldh-str> \] <let-dig> \]
<ldh-str> ::= <let-dig-hyp> | <let-dig-hyp> <ldh-str>
<let-dig-hyp> ::= <let-dig> | "-"
<let-dig> ::= <letter> | <digit>
```

```python
pyslet.http.cookie.is_rldh_label(label)
```
Tests a binary string against the definition of R-LDH label

As defined by RFC5890

Reserved LDH labels, known as “tagged domain names” in some other contexts, have the property that they contain “–” in the third and fourth characters but which otherwise conform to LDH label rules.

Non-Reserved LDH labels are the set of valid LDH labels that do not have “–” in the third and fourth positions.

Therefore you can test for a NR-LDH label simply by using the `not` operator.

```python
pyslet.http.cookie.is_a_label(label)
```
Test a binary string against the definition of A-label.

As defined by RFC5890

In fact, this function currently only tests for being an XN– label.

the class of labels that begin with the prefix “xn–” (case independent), but otherwise conform to the rules for LDH labels [is called “XN-labels”]…

The XN-labels that are valid Punycode output are known as “A-labels” if they also meet the other criteria for IDNA-validity

So bear in mind that (a) the remainder of the label may fail to decode properly when passed to the punycde algorithm and (b) even if it does decode it may result in a string that is not actually a valid U-Label.

**Exceptions**

```python
class pyslet.http.cookie.CookieError
```
Bases: exceptions.ValueError

Raised when an operation violates RFC6265 rules.
CHAPTER 6

Other Supporting Standards

The section contains modules that implement supporting specifications that are not specific, or at least have no special interest in the domains of Learning, Education and Training. In some cases modules provide utilities to help interface with Python’s standard libraries.

Contents:

6.1 WSGI Utilities

This module defines special classes and functions to make it easier to write applications based on the WSGI specification.

6.1.1 Overview

WSGI applications are simple callable objects that take two arguments:

```
result = application(environ, start_response)
```

In these utility classes, the arguments are encapsulated into a special context object based on `WSGIContext`. The context object allows you to get and set information specific to handling a single request, it also contains utility methods that are useful for extracting information from the URL, headers and request body and, likewise, methods that are useful for setting the response status and headers. Even in multi-threaded servers, each context instance is used by a single thread.

The application callable itself is modeled by an instance of the class `WSGIApp`. The instance may be called by multiple threads simultaneously so any state stored in the application is shared across all contexts and threads.

Many of the app class’ methods take an abbreviated form of the WSGI callable signature:

```
result = wsgi_app.page_method(context)
```

In this pattern, `wsgi_app` is a `WSGIApp` instance and `page_method` is the name of some response generating method defined in it.
In practice, you’ll derive a class from WSGIApp for your application and, possibly, derive a class from WSGIContext too. In the latter case, you must set the class attribute `WSGIApp.ContextClass` to your custom context class before creating your application instance.

The lifecycle of a script that runs your application can be summed up:

1. Define your WSGIApp sub-class
2. Set the values of any class attributes that are specific to a particular runtime environment. For example, you’ll probably want to set the path to the `WSGIApp.settings_file` where you can provide other runtime configuration options.
3. Configure the class by calling the `WSGIApp.setup()` class method.
4. Create an instance of the class
5. Start handling requests!

Here’s an example:

```python
# Runtime configuration directives
#
#: path to settings file
SETTINGS_FILE = '/var/www/wsgi/data/settings.json'

# Step 1: define the WSGIApp sub-class
class MyApp(WSGIApp):
    """Your class definitions here""

    #: set class attributes to configured values
    @property
    def settings_file(self):
        return SETTINGS_FILE

    #: call setup to configure the application
    def setup(self):

    #: create an instance
    def run_server(self):

In the last step we call a run_server method which uses Python’s builtin HTTP/WSGI server implementation. This is suitable for testing an application but in practice you’ll probably want to deploy your application with some other WSGI driver, such as Apache and modwsgi

**Testing**

The core WSGIApp class has a number of methods that make it easy to test your application from the command line, using Python’s built-in support for WSGI. In the example above you saw how the run_server method can be used.

There is also a facility to launch an application from the command line with options to override several settings. You can invoke this behaviour simply be calling the main class method:

```python
from pyslet.wsgi import WSGIApp
class MyApp(WSGIApp):
    """Your class definitions here""
pass
```
This simple example is available in the samples directory. You can invoke your script from the command line, –help can be used to look at what options are available:

```bash
$ python samples/wsgi_basic.py --help
```

Options:
- `h, --help` show this help message and exit
- `v` increase verbosity of output up to 3x
- `p PORT, --port=PORT` port on which to listen
- `i, --interactive` Enable interactive prompt after starting server
- `--static=STATIC` Path to the directory of static files
- `--private=PRIVATE` Path to the directory for data files
- `--settings=SETTINGS` Path to the settings file

You could start a simple interactive server on port 8081 and hit it from your web browser with the following command:

```bash
$ python samples/wsgi_basic.py -ivvp8081
```

Typing ‘stop’ at the cmd prompt in interactive mode exits the server. Anything other than stop is evaluated as a python expression in the context of a method on your application object which allows you to interrogate you application while it is running:

```python
cmd: self
<__main__.MyApp object at 0x1004c2b50>
cmd: self.settings
{'WSGIApp': {'interactive': True, 'static': None, 'port': 8081, 'private': None, 'level': 20}}
```

If you include -vvv on the launch you’ll get full debugging information including all WSGI environment information and all application output logged to the terminal.

### 6.1.2 Handling Pages

To handle a page you need to register your page with the request dispatcher. You typically do this during `WSGIApp.init_dispatcher()` by calling `WSGIApp.set_method()` and passing a pattern to match in the path and a bound method:

```python
class MyApp (WSGIApp):
    def init_dispatcher(self):
        super(MyApp, self).init_dispatcher()
        self.set_method("/*", self.home)

    def home(self, context):
        data = "<html><head><title>Hello</title></head>" \
             "<body><p>Hello world!</p></body></html>"
```
In this example we registered our simple ‘home’ method as the handler for all paths. The star is used instead of a complete path component and represents a wildcard that matches any value. When used at the end of a path it matches any (possibly empty) sequence of path components.

### 6.1.3 Data Storage

Most applications will need to read from or write data to some type of data store. Pyslet exposes its own data access layer to web applications, for details of the data access layer see the OData section.

To associate a data container with your application simply derive your application from `WSGIDataApp` instead of the more basic WSGIApp.

You’ll need to supply a metadata XML document describing your data schema and information about the data source in the settings file.

The minimum required to get an application working with a sqlite3 database would be to a directory with the following layout:

```
settings.json
metadata.xml
data/
```

The settings.json file would contain:

```json
{
"WSGIApp": {
   "private": "data"
},
"WSGIDataApp": {
   "metadata": "metadata.xml"
}
}
```

If the settings file is in samples/wsgi_data your source might look this:

```python
from pyslet.wsgi import WSGIDataApp
class MyApp(WSGIDataApp):
    settings_file = 'samples/wsgi_data/settings.json'
    # method definitions as before
    if __name__ == '__main__':
        MyApp.main()
```

To create your database the first time you will either want to run a custom SQL script or get Pyslet to create the tables for you. With the script above both options can be achieved with the command line:

```
$ python samples/wsgi_data.py --create_tables --ivvp8081
```

This command starts the server as before but instructs it to create the tables in the database before running. Obviously you can only specify this option the first time!
Alternatively you might want to customise the table creation script, in which case you can create a pro-forma to edit using the –sqlout option instead:

```
$ python samples/wsgi_data.py --sqlout > wsgi_data.sql
```

### 6.1.4 Session Management

The `WSGIDataApp` is further extended by `SessionApp` to cover the common use case of needing to track information across multiple requests from the same user session.

The approach taken requires cookies to be enabled in the user’s browser. See `Problems with Cookies` below for details.

A decorator, `session_decorator()` is defined to make it easy to write (page) methods that depend on the existence of an active session. The session initiation logic is a little convoluted and is likely to involve at least one redirect when a protected page is first requested, but this all happens transparently to your application. You may want to look at overriding the `ctest_page()` and `cfail_page()` methods to provide more user-friendly messages in cases where cookies are blocked.

### CSRF

Hand-in-hand with session management is defence against cross-site request forgery (CSRF) attacks. Relying purely on a session cookie to identify a user is problematic because a third party site could cause the user’s browser to submit requests to your application on their behalf. The browser will send the session cookie even if the request originated outside one of your application’s pages.

POST requests that affect the state of the server or carry out some other action requiring authorisation must be protected. Requests that simply return information (i.e., GET requests) are usually safe, even if the response contains confidential information, as the browser prevents the third party site from actually reading the HTML. Be careful when returning data other than HTML though, for example, data that could be parsed as valid JavaScript will need additional protection. The importance of using HTTP request methods appropriately cannot be understated!

The most common pattern for preventing this type of fraud is to use a special token in POST requests that can’t be guessed by the third party and isn’t exposed outside the page from which the POSTed form is supposed to originate. If you decorate a page that is the target of a POST request (the page that performs the action) with the session decorator then the request will fail if a CSRF token is not included in the request. The token can be read from the session object and will need to be inserted into any forms in your application. You shouldn’t expose your CRSF token in the URL as that makes it vulnerable to being discovered, so don’t add it to forms that use the GET action.

Here’s a simple example method that shows the use of the session decorator:

```python
@session_decorator
def home(self, context):
    page = """<html><head><title>Session Page</title></head><body>
    <h1>Session Page</h1>
    $s
    </body></html>"
    with self.container['Sessions'].open() as collection:
        try:
            entity = collection[context.session.sid]
            user_name = entity['UserName'].value
        except KeyError:
            user_name = None
        if user_name:
            noform = """<p>Welcome: $s</p>""
            page = page % (noform % xml.EscapeCharData(user_name))
        else:
```

6.1. WSGI Utilities
form = """"<form method="POST" action="setname"> 
  <p>Please enter your name: <input type="text" name="name"/> 
  <input type="hidden" name="csrf_token" value="%s" /> 
  <input type="submit" value="Set"/></p> 
</form>"""
page = page % ( 
    form % (xml.EscapeCharData(self.csrf_token, True), 
            xml.EscapeCharData(context.session.sid, True)))
context.set_status(200)
return self.html_response(context, page)

We’ve added a simple database table to store the session data with the following entity:

```xml
<EntityType Name="Session">
  <Key>
    <PropertyRef Name="SessionID"/>
  </Key>
  <Property Name="SessionID" Type="Edm.String" MaxLength="64" Nullable="false"/>
  <Property Name="UserName" Type="Edm.String" Nullable="true" MaxLength="256" Unicode="true"/>
</EntityType>
```

Our database must also contain a small table used for key management, see below for information about encryption.

Our method reads the value of this property from the database and prints a welcome message if it is set. If not, it prints a form allowing you to enter your name. Notice that we must include a hidden field containing the CSRF token. The name of the token parameter is given in `SessionApp.csrf_token` and the value is read from the session object passed in the accompanying cookie - the browser should prevent third parties from reading the cookie’s value.

The action method that processes the form looks like this:

```python
@session_decorator
def setname(self, context):
    user_name = context.get_form_string('name')
    if user_name:
        with self.container['Sessions'].open() as collection:
            try:
                entity = collection[context.session.sid]
                entity['UserName'].set_from_value(user_name)
                collection.update_entity(entity)
            except KeyError:
                entity = collection.new_entity()
                entity['SessionID'].set_from_value(context.session.sid)
                entity['UserName'].set_from_value(user_name)
                collection.insert_entity(entity)
        return self.redirect_page(context, context.get_app_root())
```

A sample application containing this code is provided and can again be run from the command line:

```bash
$ python samples/wsgi/wsgi_session.py --create_tables -ivvp8081
```

problems with cookies

There has been significant uncertainty over the use of cookies with some browsers blocking them in certain situations and some users blocking them entirely. In particular, the E-Privacy Directive in the European Union has led to a spate of scrolling consent banners and pop-ups on website landing pages.
It is worth bearing in mind that use of cookies, as opposed to URL-based solutions or cacheable basic auth credentials, is currently considered more secure for passing session identifiers. When designing your application you need to balance the privacy rights of your users with the need to keep their information safe and secure. Indeed, the main provisions of the directive are about providing security of services. As a result, it is generally accepted that the use of cookies for tracking sessions is essential and does not require any special consent from the user.

By extending \texttt{WSGIDataApp} this implementation always persists session data on the server. This gets around most of the perceived issues with the directive and cookies but does not absolve you and your application of the need to obtain consent from a more general data protection perspective!

Perhaps more onerous, but less discussed, is the obligation to remove ‘traffic data’, sometimes referred to as metadata, about the transmission of a communication. For this reason, we don’t store the originating IP address of the session even though doing so might actually increase security. As always, it’s a balance.

Finally, by relying on cookies we will sometimes fall foul of browser attempts to automate the privacy preferences of their users. The most common scenario is when our application is opened in a frame within another application. In this case, some browsers will apply a much stricter policy on blocking cookies. For example, Microsoft’s Internet Explorer (from version 6) requires the implementation of the P3P standard for communicating privacy information. Although some sites have chosen to fake a policy to trick the browser into accepting their cookies this has resulted in legal action so is not to be recommended.


To maximise the chances of being able to create a session this class uses automatic redirection to test for cookie storage and a mechanism for transferring the session to a new window if it detects that cookies are blocked.

For a more detailed explanation of how this is achieved see my blog post \texttt{Putting Cookies in the Frame}

In many cases, once the application has been opened in a new window and the test cookie has been set successfully, future framed calls to the application will receive cookies and the user experience will be much smoother.

### 6.1.5 Encrypting Data

Sometimes you’ll want to encrypt sensitive data stored in a data store to prevent, say, a database administrator from being able to read it. This module provides a utility class called \texttt{AppCipher} which is designed to make this easier.

An \texttt{AppCipher} is initialised with a key. There are various strategies for storing keys for application use, in the simplest case you might read the key from a configuration file that is only available on the application server and not to the database administrator, say.

To assist with key management \texttt{AppCipher} will store old keys (suitably encrypted) in the data store using an entity with the following properties:

```xml
<EntityType Name="AppKey">
  <Key>
    <PropertyRef Name="KeyNum"/>
  </Key>
  <Property Name="KeyNum" Nullable="false" Type="Edm.Int32"/>
  <Property Name="KeyString" Nullable="false" Type="Edm.String" MaxLength="256" Unicode="false"/>
</EntityType>
```

SessionApp’s require an \texttt{AppCipher} to be specified in the settings and an \texttt{AppKeys} entity set in the data store to enable signing of the session cookie (to guard against cookie tampering).

The default implementation of \texttt{AppCipher} does not use any encryption (it merely obfuscates the input using base64 encoding) so to be useful you’ll need to use a class derived from \texttt{AppCipher}. If you have the \texttt{Pycrypto} module installed you can use the \texttt{AESAppCipher} class to use the AES algorithm to encrypt the data.

For details, see the reference section below.
6.1.6 Reference

class pyslet.wsgi.WSGIContext (environ, start_response, canonical_root=None)
Bases: object

A class used for managing WSGI calls

environ The WSGI environment

start_response The WSGI call-back

canonical_root A URL that overrides the automatically derived canonical root, see WSGIApp for more details.

This class acts as a holding place for information specific to each request being handled by a WSGI-based application. In some frameworks this might be called the request object but we already have requests modelled in the http package and, anyway, this holds information about the WSGI environment and the response too.

MAX_CONTENT = 65536
The maximum amount of content we’ll read into memory (64K)

environ = None
the WSGI environ

start_response_method = None
the WSGI start_response callable

status = None
the response status code (an integer), see set_status()

status_message = None
the response status message (a string), see set_status()

headers = None
a list of (name, value) tuples containing the headers to return to the client. name and value must be strings

set_status (code)
Sets the status of the response

code An HTTP integer response code.

This method sets the status_message automatically from the code. You must call this method before calling start_response.

add_header (name, value)
Adds a header to the response

name The name of the header (a string)

value The value of the header (a string)

start_response ()
Calls the WSGI start_response method

If the status has not been set a 500 response is generated. The status string is created automatically from status and status_message and the headers are set from headers.

The return value is the return value of the WSGI start_response call, an obsolete callable that older applications use to write the body data of the response.

If you want to use the exc_info mechanism you must call start_response yourself directly using the value of start_response_method

get_app_root ()
Returns the root of this application
The result is a `pyslet.rfc2396.URI` instance, It is calculated from the environment in the same way as `get_url()` but only examines the SCRIPT_NAME portion of the path.

It always ends in a trailing slash. So if you have a script bound to /script/myscript.py running over http on www.example.com then you will get:

```
http://www.example.com/script/myscript.py/
```

This allows you to generate absolute URLs by resolving them relative to the computed application root, e.g.:

```
URI.from_octets('images/counter.png').resolve(
    context.get_app_root())
```

would return:

```
http://www.example.com/script/myscript.py/images/counter.png
```

for the above example. This is preferable to using absolute paths which would strip away the SCRIPT_NAME prefix when used.

`get_url()`

Returns the URL used in the request

The result is a `pyslet.rfc2396.URI` instance, It is calculated from the environment using the algorithm described in URL Reconstruction section of the WSGI specification except that it ignores the Host header for security reasons.

Unlike the result of `get_app_root()` it doesn’t necessarily end with a trailing slash. So if you have a script bound to /script/myscript.py running over http on www.example.com then you may get:

```
http://www.example.com/script/myscript.py
```

A good pattern to adopt when faced with a missing trailing slash on a URL that is intended to behave as a ‘directory’ is to add the slash to the URL and use xml:base (for XML responses) or HTML’s <base> tag to set the root for relative links. The alternative is to issue an explicit redirect but this requires another request from the client.

This causes particular pain in OData services which frequently respond on the service script’s URL without a slash but generate incorrect relative links to the contained feeds as a result.

`get_query()`

Returns a dictionary of query parameters

The dictionary maps parameter names onto strings. In cases where multiple values have been supplied the values are comma separated, so a URL ending in ?option=Apple&option=Pear would result in the dictionary:

```
{‘option’: 'Apple,Pear'}
```

This method only computes the dictionary once, future calls return the same dictionary!

Note that the dictionary does not contain any cookie values or form parameters.

`get_content()`

Returns the content of the request as a string

The content is read from the input, up to CONTENT_LENGTH bytes, and is returned as a binary string. If the content exceeds `MAX_CONTENT` (default: 64K) then BadRequest is raised.
This method can be called multiple times, the content is only actually read from the input the first time. Subsequent calls return the same string.

This call cannot be called on the same context as `get_form()`, whichever is called first takes precedence. Calls to `get_content` after `get_form` return None.

`get_form()`
Returns a FieldStorage object parsed from the content.

The query string is excluded before the form is parsed as this only covers parameters submitted in the content of the request. To search the query string you will need to examine the dictionary returned by `get_query()` too.

This method can be called multiple times, the form is only actually read from the input the first time. Subsequent calls return the same FieldStorage object.

This call cannot be called on the same context as `get_content()`, whichever is called first takes precedence. Calls to `get_form` after `get_content` return None.

Warning: `get_form` will only parse the form from the content if the request method was POST!

`get_form_string(name, max_length=65536)`
Returns the value of a string parameter from the form.

`name` The name of the parameter

`max_length (optional, defaults to 64KB)` Due to an issue in the implementation of FieldStorage it isn’t actually possible to definitively tell the difference between a file upload and an ordinary input field. HTML5 clarifies the situation to say that ordinary fields don’t have a content type but FieldStorage assumes ‘text/plain’ in this case and sets the file and type attribute of the field anyway.

To prevent obtuse clients sending large files disguised as ordinary form fields, tricking your application into loading them into memory, this method checks the size of any file attribute (if present) against `max_length` before returning the field’s value.

If the parameter is missing from the form then an empty string is returned.

`get_form_long(name)`
Returns the value of a (long) integer parameter from the form.

`name` The name of the parameter

If the parameter is missing from the form then None is returned, if the parameter is present but is not a valid integer then `BadRequest` is raised.

`get_cookies()`
Returns a dictionary of cookies from the request

If no cookies were passed an empty dictionary is returned.

For details of how multi-valued cookies are handled see: `pyslet.http.cookie.CookieParser.request_cookie_string()`.

class `pyslet.wsgi.WSGIApp`
Bases: `pyslet.wsgi.DispatchNode`

An object to help support WSGI-based applications.

Instances are designed to be callable by the WSGI middle-ware, on creation each instance is assigned a random identifier which is used to provide comparison and hash implementations. We go to this trouble so that derived classes can use techniques like the functools lru_cache decorator in future versions.

`ContextClass` the context class to use for this application, must be (derived from) `WSGIContext`
alias of `WSGIContext`

**static_files = None**

The path to the directory for `static_files`. Defaults to None. An `pyslet.vfs.OSFilePath` instance.

**private_files = None**

Private data directory

An `pyslet.vfs.OSFilePath` instance.

The directory used for storing private data. The directory is partitioned into sub-directories based on the lower-cased class name of the object that owns the data. For example, if `private_files` is set to `/var/www/data` and you derive a class called `MyApp` from WSGIApp you can assume that it is safe to store and retrieve private data files from `/var/www/data/myapp`.

`private_files` defaults to None for safety. The current WSGIApp implementation does not depend on any private data.

**settings_file = None**

The path to the settings file. Defaults to None.

An `pyslet.vfs.OSFilePath` instance.

The format of the settings file is a json dictionary. The dictionary’s keys are class names that define a scope for class-specific settings. The key ‘WSGIApp’ is reserved for settings defined by this class. The defined settings are:

- **level (None)** If specified, used to set the root logging level, a value between 0 (NOTSET) and 50 (CRITICAL). For more information see python’s logging module.
- **port (8080)** The port number used by `run_server()`.
- **canonical_root (“http://localhost” or “http://localhost:<port>”)** The canonical URL scheme, host and (port if required) for the application. This value is passed to the context and used by `WSGIContext.get_url()` and similar methods in preference to the SERVER_NAME and SERVER_PORT to construct absolute URLs returned or recorded by the application. Note that the Host header is always ignored to prevent related security attacks.

If no value is given then the default is calculated taking in to consideration the port setting.

- **interactive (False)** Sets the behaviour of `run_server()`, if specified the main thread prompts the user with a command line interface allowing you to interact with the running server. When False, `run_server` will run forever and can only be killed by an application request that sets `stop` to True or by an external signal that kills the process.
- **static (None)** A URL to the static files (not a local file path). This will normally be an absolute path or a relative path. Relative paths are relative to the settings file in which the setting is defined. As URL syntax is used you must use the ‘/’ as a path separator and add proper URL-escaping. On Windows, UNC paths can be specified by putting the host name in the authority section of the URL.
- **private (None)** A URL to the private files. Interpreted as per the ‘static’ setting above.

**settings = None**

the class settings loaded from `settings_file` by `setup()`

**base = None**

the base URI of this class, set from the path to the settings file itself and is used to locate data files on the server. This is a `pyslet.rfc2396.FileURL` instance. Not to be confused with the base URI of resources exposed by the application this class implements!
**private_base = None**

the base URI of this class’ private files. This is set from the `private_files` member and is a `pyslet.rfc2396.FileURL` instance.

**content_type = {"ico": MediaType('image', 'vnd.microsoft.icon', {})}**

The mime type mapping table.

This table is used before falling back on Python’s built-in `guess_type` function from the mimetypes module. Add your own custom mappings here.

It maps file extension (without the dot) on to `MediaType` instances.

**MAX_CHUNK = 65536**

the maximum chunk size to read into memory when returning a (static) file. Defaults to 64K.

**js_origin = 0**

the integer millisecond time (since the epoch) corresponding to 01 January 1970 00:00:00 UTC the JavaScript time origin.

**clslock = <_RLock owner=None count=0>**

a threading.RLock instance that can be used to lock the class when dealing with data that might be shared amongst threads.

**classmethod main()**

Runs the application

Options are parsed from the command line and used to `setup()` the class before an instance is created and launched with `run_server()`.

**classmethod add_options(parser)**

Defines command line options.

- **parser** An OptionParser instance, as defined by Python’s built-in `optparse` module.

The following options are added to `parser` by the base implementation:

- **-v** Sets the logging level to WARNING, INFO or DEBUG depending on the number of times it is specified.Overrides the ‘level’ setting in the settings file.

- **-p, --port** Overrides the value of the ‘port’ setting in the settings file.

- **-i, --interactive** Overrides the value of the ‘interactive’ setting in the settings file.

- **--static** Overrides the value of `static_files`.

- **--private** Overrides the value of `private_files`.

- **--settings** Sets the path to the `settings_file`.

**classmethod setup(options=None, args=None, **kwargs)**

Perform one-time class setup

- **options** An optional object containing the command line options, such as an optparse.Values instance created by calling `parse_args` on the OptionParser instance passed to `add_options()`.

- **args** An optional list of positional command-line arguments such as would be returned from `parse_args` after the options have been removed.

All arguments are given as keyword arguments to enable use of super and diamond inheritance.

The purpose of this method is to perform any actions required to setup the class prior to the creation of any instances.
The default implementation loads the settings file and sets the value of `settings`. If no settings file can be found then an empty dictionary is created and populated with any overrides parsed from options.

Finally, the root logger is initialised based on the level setting.

Derived classes should always use super to call the base implementation before their own setup actions are performed.

classmethod resolve_setup_path( uri_path, private=False )
Resolves a settings-relative path

  uri_path  The relative URI of a file or directory.

  private (False)  Resolve relative to the private files directory

Returns `uri_path` as an OSFilePath instance after resolving relative to the settings file location or to the private files location as indicated by the private flag. If the required location is not set then `uri_path` must be an absolute file URL (starting with, e.g., `file:///`). On Windows systems the authority component of the URL may be used to specify the host name for a UNC path.

stop = None
flag: set to True to request `run_server()` to exit

id = None
a unique ID for this instance

init_dispatcher()
Used to initialise the dispatcher.

By default all requested paths generate a 404 error. You register pages during `init_dispatcher()` by calling `set_method()`. Derived classes should use super to pass the call to their parents.

set_method( path, method )
Registers a bound method in the dispatcher

  path  A path or path pattern

  method  A bound method or callable with the basic signature:

  
  result = method(context)

A star in the path is treated as a wildcard and matches a complete path segment. A star at the end of the path (which must be after a '/') matches any sequence of path segments. The matching sequence may be empty, in other words, “/images/*” matches “/images/”. In keeping with common practice a missing trailing slash is ignored when dispatching so “/images” will also be routed to a method registered with “/images/*” though if a separate registration is made for “/images” it will be matched in preference.

Named matches always take precedence over wildcards so you can register “/images/*” and “/images/count.png” and the latter path will be routed to its preferred handler. Similarly you can register “/background.png” and “/home/background.png” but remember the ‘*’ only matches a single path component! There is no way to match background.png in any directory.

call_wrapper( environ, start_response )
Alternative entry point for debugging

Although instances are callable you may use this method instead as your application’s entry point when debugging.

This method will log the environ variables, the headers output by the application and all the data (in quoted-printable form) returned at DEBUG level.
It also catches a common error, that of returning something other than a string for a header value or in the generated output. These are logged at ERROR level and converted to strings before being passed to the calling framework.

**static_page** *(context)*

Returns a static page

This method can be bound to any path using `set_method()` and it will look in the `static_files` directory for that file. For example, if `static_files` is “/var/www/html” and the PATH_INFO variable in the request is “/images/logo.png” then the path “/var/www/html/images/logo.png” will be returned.

There are significant restrictions on the names of the path components. Each component *must* match a basic label syntax (equivalent to the syntax of domain labels in host names) except the last component which must have a single ‘.’ separating two valid labels. This conservative syntax is designed to be safe for passing to file handling functions.

**file_response** *(context, file_path)*

Returns a file from the file system

*file_path* The system file path of the file to be returned as a `pyslet.vfs.OSFilePath` instance.

The Content-Length header is set from the file size, the Last-Modified date is set from the file’s st_mtime and the file’s data is returned in chunks of `MAX_CHUNK` in the response.

The status is *not* set and must have been set before calling this method.

**html_response** *(context, data)*

Returns an HTML page

*data* A string containing the HTML page data. This may be a unicode or binary string.

The Content-Type header is set to text/html (with an explicit charset if data is a unicode string). The status is *not* set and must have been set before calling this method.

**json_response** *(context, data)*

Returns a JSON response

*data* A string containing the JSON data. This may be a unicode or binary string (encoded with utf-8).

The Content-Type is set to “application/json”. The status is *not* set and must have been set before calling this method.

**text_response** *(context, data)*

Returns a plain text response

*data* A string containing the text data. This may be a unicode or binary string (encoded with US-ASCII).

The Content-Type is set to “text/plain” (with an explicit charset if a unicode string is passed). The status is *not* set and must have been set before calling this method.

**redirect_page** *(context, location, code=303)*

Returns a redirect response

*location* A URI instance or a string of octets.

*code* (303) The redirect status code. As a reminder the typical codes are 301 for a permanent redirect, a 302 for a temporary redirect and a 303 for a temporary redirect following a POST request. This latter code is useful for implementing the widely adopted pattern of always redirecting the user after a successful POST request to prevent browsers prompting for re-submission and is therefore the default.
This method takes care of setting the status, the Location header and generating a simple HTML redirection page response containing a clickable link to location.

`error_page(context, code=500, msg=None)`
Generates an error response

- **code (500)** The status code to send.
- **msg (None)** An optional plain-text error message. If not given then the status line is echoed in the body of the response.

**class** `pyslet.wsgi.WSGIDataApp(**kwargs)`

Bases: `pyslet.wsgi.WSGIApp`

Extends WSGIApp to include a data store

The key `‘WSGIDataApp’` is reserved for settings defined by this class in the settings file. The defined settings are:

- **container (None)** The name of the container to use for the data store. By default, the default container is used. For future compatibility you should not depend on using this option.

- **metadata (None)** URI of the metadata file containing the data schema. The file is assumed to be relative to the `settings_file`.

- **source_type (‘sqlite’)** The type of data source to create. The default value is sqlite. A value of `‘mysql’` select’s Pyslet’s mysqldbds module instead.

- **sqlite_path (‘database.sqlite3’)** URI of the database file. The file is assumed to be relative to the `private_files` directory, though an absolute path may be given.

- **dbhost (‘localhost’)** For mysql databases, the hostname to connect to.

- **dbname (None)** The name of the database to connect to.

- **dbuser (None)** The user name to connect to the database with.

- **dbpassword (None)** The password to use in conjunction with dbuser

- **keynum (‘0’)** The identification number of the key to use when storing encrypted data in the container.

- **secret (None)** The key corresponding to keynum. The key is read in plain text from the settings file and must be provided in order to use the `app_cipher` for managing encrypted data and secure hashing. Derived classes could use an alternative mechanism for reading the key, for example, using the `keyring` python module.

- **cipher (‘aes’)** The type of cipher to use. By default `AESAppCipher` is used which uses AES internally with a 256 bit key created by computing the SHA256 digest of the secret string. The only other supported value is ‘plaintext’ which does not provide any encryption but allows the app_cipher object to be used in cases where encryption may or may not be used depending on the deployment environment. For example, it is often useful to turn off encryption in a development environment!

- **when (None)** An optional value indicating when the specified secret comes into operation. The value should be a fully specified time point in ISO format with timezone offset, such as ‘2015-01-01T09:00:00-05:00’. This value is used when the application is being restarted after a key change, for details see `AppCipher.change_key()`.

The use of AES requires the PyCrypto module to be installed.

**classmethod add_options(parser)**

Adds the following options:

- **-s, --sqlout** print the suggested SQL database schema and then exit. The setting of –create is ignored.
--create_tables create tables in the database

-m, --memory Use an in-memory SQLite database. Overrides any source_type and encryption setting values. Implies --create_tables

metadata = None
the metadata document for the underlying data service

data_source = None
the data source object for the underlying data service the type of this object will vary depending on the source type. For SQL-type containers this will be an instance of a class derived from SQLEntityContainer

container = None
the entity container (cf database)

classmethod setup(options=None, args=None, **kwargs)
Adds database initialisation

Loads the metadata document. Creates the data_source according to the configured settings (creating the tables only if requested in the command line options). Finally sets the container to the entity container for the application.

If the -s or --sqlout option is given in options then the data source's create table script is output to standard output and sys.exit(0) is used to terminate the process.

classmethod new_app_cipher()
Creates an AppCipher instance

This method is called automatically on construction, you won't normally need to call it yourself but you may do so, for example, when writing a script that requires access to data encrypted by the application.

If there is no 'secret' defined then None is returned.

Reads the values from the settings file and creates an instance of the appropriate class based on the cipher setting value. The cipher uses the 'AppKeys' entity set in container to store information about expired keys. The AppKey entities have the following three properties:

KeyNum (integer key) The key identification number

KeyString (string) The encrypted secret, for example:

'1:OBimcmOesY0t021NuPXTpoMObOCs6v0Op11'

The number before the colon is the key identification number of the secret used to encrypt the string (and will always be different from the KeyNum field of course). The data after the colon is the base-64 encoded encrypted string. The same format is used for all data encrypted by AppCipher objects. In this case the secret was the word 'secret' and the algorithm used is AES.

Expires (DateTime) The UTC time at which this secret will expire. After this time a newer key should be used for encrypting data though this key may of course still be used for decrypting data.

app_cipher = None
the application's cipher, a AppCipher instance.

pyslet.wsgi.session_decorator(page_method)
Decorates a web method with session handling

page_method An unbound method with signature: page_method(obj, context) which performs the WSGI protocol and returns the page generator.

Our decorator just calls SessionContext.session_wrapper().
class **pyslet.wsgi.SessionContext** (environ, start_response, canonical_root=None)

Bases: **pyslet.wsgi.WSGIContext**

Extends the base class with a session object.

**session** = None

a session object, or None if no session available

**start_response**()

Saves the session cookie.

class **pyslet.wsgi.SessionApp**(**kwargs**)

Bases: **pyslet.wsgi.WSGIDataApp**

Extends WSGIDataApp to include session handling.

These sessions require support for cookies. The SessionApp class itself uses two cookies purely for session tracking.

The key ‘SessionApp’ is reserved for settings defined by this class in the settings file. The defined settings are:

**timeout** (600) The number of seconds after which an inactive session will time out and no longer be accessible to the client.

**cookie** (‘sid’) The name of the session cookie.

**cookie_test** (‘ctest’) The name of the test cookie. This cookie is set with a longer lifetime and acts both as a test of whether cookies are supported or not and can double up as an indicator of whether user consent has been obtained for any extended use of cookies. It defaults to the value ‘0’, indicating that cookies can be stored but that no special consent has been obtained.

**cookie_test_age** (8640000) The age of the test cookie (in seconds). The default value is equivalent to 100 days. If you use the test cookie to record consent to some cookie policy you should ensure that when you set the value you use a reasonable lifespan.

**csrftoken** (‘csrftoken’) The name of the form field containing the CSRF token

**classmethod setup**(options=None, args=None, **kwargs**)

Adds database initialisation

**csrf_token** = None

The name of our CSRF token

**ContextClass**

Extended context class

alias of **SessionContext**

**SessionClass**

The session class to use, must be (derived from) **Session**

alias of **CookieSession**

**init_dispatcher**()

Adds pre-defined pages for this application

These pages are mapped to /ctest and /wlaunch. These names are not currently configurable. See **ctest()** and **wlaunch()** for more information.

**session_wrapper**(context, page_method)

Called by the session_decorator

Uses **set_session()** to ensure the context has a session object. If this request is a POST then the form is parsed and the CSRF token checked for validity.
**set_session** *(context)*

Sets the session object in the context

The session is read from the session cookie, established and marked as being seen now. If no cookie is found a new session is created. In both cases a cookie header is set to update the cookie in the browser.

**set_session_cookie** *(context)*

Adds the session cookie to the response headers

The cookie is bound to the path returned by `WSGIContext.get_app_root()` and is marked as being http_only and is marked secure if we have been accessed through an https URL.

You won’t normally have to call this method but you may want to override it if your application wishes to override the cookie settings.

**clear_session_cookie** *(context)*

Removes the session cookie

**set_test_cookie** *(context, value='0')*

Adds the test cookie

**establish_session** *(context)*

Mark the session as established

This will update the session ID, override this method to update any data store accordingly if you are already associating protected information with the session to prevent it becoming orphaned.

**merge_session** *(context, merge_session)*

Merges a session into the session in the context

Override this method to update any data store. If you are already associating protected information with merge_session you need to transfer it to the context session.

The default implementation does nothing and merge_session is simply discarded.

**session_page** *(context, page_method, return_path)*

Returns a session protected page

**context** The `WSGIContext` object

**page_method** A function or *bound* method that will handle the page. Must have the signature:

```python
page_method(context)
```

and return the generator for the page as per the WSGI specification.

**return_path** A `pyslet.rfc2396.URI` instance pointing at the page that will be returned by page_method, used if the session is not established yet and a redirect to the test page needs to be implemented.

This method is only called *after* the session has been created, in other words, context.session must be a valid session.

This method either calls the page_method (after ensuring that the session is established) or initiates a redirection sequence which culminates in a request to return_path.

**ctest** *(context)*

The cookie test handler

This page takes three query parameters:

**return** The return URL the user originally requested

**s** The session that should be received in a cookie
sig  The session signature which includes the User-Agent at the end of the message.

framed (optional) An optional parameter, if present and equal to ‘1’ it means we’ve already attempted to load the page in a new window so if we still can’t read cookies we’ll return the cfail_page().

If cookies cannot be read back from the context this page will call the ctest_page() to provide an opportunity to open the application in a new window (or cfail_page() if this possibility has already been exhausted.

If cookies are successfully read, they are compared with the expected values (from the query) and the user is returned to the return URL with an automatic redirect. The return URL must be within the same application (to prevent ‘open redirect’ issues) and, to be extra safe, we change the user-visible session ID as we’ve exposed the previous value in the URL which makes it more liable to snooping.

ctest_page (context, target_url, return_url, s, sig)
Returns the cookie test page
Called when cookies are blocked (perhaps in a frame).

context  The request context

target_url  A string containing the base link to the wlaunch page. This page can opened in a new window (which may get around the cookie restrictions). You must pass the return_url and the sid values as the ‘return’ and ‘sid’ query parameters respectively.

return_url  A string containing the URL the user originally requested, and the location they should be returned to when the session is established.

s  The session

sig  The session signature

You may want to override this implementation to provide a more sophisticated page. The default simply presents the target_url with added “return”, “s” and “sig” parameters as a simple hypertext link that will open in a new window.

A more sophisticated application might render a button or a form but bear in mind that browsers that cause this page to load are likely to prevent automated ways of opening this link.

wlaunch (context)
Handles redirection to a new window

The query parameters must contain:

return  The return URL the user originally requested

s  The session that should also be received in a cookie

sig  The signature of the session, return URL and User-Agent

This page initiates the redirect sequence again, but this time setting the framed query parameter to prevent infinite redirection loops.

cfail_page (context)
Called when cookies are blocked completely.

The default simply returns a plain text message stating that cookies are blocked. You may want to include a page here with information about how to enable cookies, a link to the privacy policy for your application to help people make an informed decision to turn on cookies, etc.

check_redirect (context, target_path)
Checks a target path for an open redirect

target_path  A string or URI instance.
Returns True if the redirect is safe.

The test ensures that the canonical root of our application matches the canonical root of the target. In other words, it must have the same scheme and matching authority (host/port).

class pyslet.wsgi.AppCipher(key_num, key, key_set, when=None)

    A cipher for encrypting application data

    key_num  A key number
    key      A binary string containing the application key.
    key_set  An entity set used to store previous keys. The entity set must have an integer key property ‘KeyNum’ and a string field ‘KeyString’. The string field must be large enough to contain encrypted versions of previous keys.
    when (None)  A fully specified pyslet.iso8601.TimePoint at which time the key will become active. If None, the key is active straight away. Otherwise, the key_set is searched for a key that is still active and that key is used when encrypting data until the when time, at which point the given key takes over.

The object wraps an underlying cipher. Strings are encrypted using the cipher and then encoded using base64. The output is then prefixed with an ASCII representation of the key number (key_num) followed by a ‘:’. For example, if key_num is 7 and the cipher is plain-text (the default) then encrypt(“Hello”) results in:

"7:SGVsbG8="

When decrypting a string, the key number is parsed and matched against the key_num of the key currently in force. If the string was encrypted with a different key then the key_set is used to look up that key (which is itself encrypted of course). The process continues until a key encrypted with key_num is found.

The upshot of this process is that you can change the key associated with an application. See change_key() for details.

MAX_AGE = 100

the maximum age of a key, which is the number of times the key can be changed before the original key is considered too old to be used for decryption.

new_cipher(key)

    Returns a new cipher object with the given key

    The default implementation creates a plain-text ‘cipher’ and is not suitable for secure use of encrypt/decrypt but, with a sufficiently good key, may still be used for hashing.

change_key(key_num, key, when)

    Changes the key of this application.

    key_num  The number given to the new key, must differ from the last MAX_AGE key numbers.
    key      A binary string containing the new application key.
    when     A fully specified pyslet.iso8601.TimePoint at which point the new key will come into effect.

Many organizations have a policy of changing keys on a routine basis, for example, to ensure that people who have had temporary access to the key only have temporary access to the data it protects. This method makes it easier to implement such a policy for applications that use the AppCipher class.

The existing key is encrypted with the new key and a record is written to the key_set to record the existing key number, the encrypted key string and the when time, which is treated as an expiry time in this context.
This procedure ensures that strings encrypted with an old key can always be decrypted because the value of the old key can be looked up. Although it is encrypted, it will be encrypted with a new(er) key and the procedure can be repeated as necessary until a key encrypted with the newest key is found.

The key change process then becomes:

1. Start a utility process connected to the application’s entity container using the existing key and then call the change_key method. Pass a value for when that will give you time to reconfigure all AppCipher clients. Assuming the key change is planned, a time in hours or even days ahead can be used.

2. Update or reconfigure all existing applications so that they will be initialised with the new key and the same value for when next time they are restarted.

3. Restart/refresh all running applications before the change over time. As this does not need to be done simultaneously, a load balanced set of application servers can be cycled on a schedule to ensure continuous running.

Following a key change the entity container will still contain data encrypted with old keys and the architecture is such that compromise of a key is sufficient to read all encrypted data with that key and all previous keys. Therefore, changing the key only protects new data.

In situations where policy dictates a key change it might make sense to add a facility to the application for re-encrypting data in the data store by going through a read-decrypt/encrypt-write cycle with each protected data field. Of course, the old key could still be used to decrypt this information from archived backups of the data store. Alternatively, if the protected data is itself subject to change on a routine basis you may simply rely on the natural turnover of data in the application. The strategy you choose will depend on your application.

The MAX_AGE attribute determines the maximum number of keys that can be in use in the data set simultaneously. Eventually you will have to update encrypted data in the data store.

encrypt (data)

Encrypts data with the current key.

data A binary input string.

Returns a character string of ASCII characters suitable for storage.

decrypt (data)

Decrypts data.

data A character string containing the encrypted data

Returns a binary string containing the decrypted data.

sign (message)

Signs a message with the current key.

message A binary message string.

Returns a character string of ASCII characters containing a signature of the message. It is recommended that character strings are encoded using UTF-8 before signing.

check_signature (signature, message=None)

Checks a signature returned by sign

signature The ASCII signature to be checked for validity.

message A binary message string. This is optional, if None then the message will be extracted from the signature string (reversing ascii_sign).

On success the method returns the validated message (a binary string) and on failure it raises ValueError.
ascii_sign(message)
    Signs a message with the current key
    message  A binary message string
    The difference between ascii_sign and sign is that ascii_sign returns the entire message, including the
    signature, as a URI-encoded character string suitable for storage and/or transmission.
    The message is %-encoded (as implemented by pyslet.rfc2396.escape_data()). You may apply
    the corresponding unescape data function to the entire string to get a binary string that contains
    an exact copy of the original data.

class pyslet.wsgi.AESAppCipher(key_num, key, key_set, when=None)
    Bases: pyslet.wsgi.AppCipher
    A cipher object that uses AES to encrypt the data
    The Pycrypto module must be installed to use this class.
    The key is hashed using the SHA256 algorithm to obtain a 32 byte value for the AES key. The encrypted strings
    contain random initialisation vectors so repeated calls won’t generate the same encrypted values. The CFB
    mode of operation is used.

Utility Functions

pyslet.wsgi.generate_key(key_length=128)
    Generates a new key
    key_length  The minimum key length in bits. Defaults to 128.
    The key is returned as a sequence of 16 bit hexadecimal strings separated by ‘.’ to make them easier to read and
    transcribe into other systems.
pyslet.wsgi.key60(src)
    Generates a non-negative 60-bit long from a source string.
    src  A binary string.
    The idea behind this function is to create an (almost) unique integer from a given string. The integer can then
    be used as the key field of an associated entity without having to create foreign keys that are long strings. There
    is of course a small chance that two source strings will result in the same integer.
    The integer is calculated by truncating the SHA256 hexdigest to 15 characters (60-bits) and then converting to
    long. Future versions of Python promise improvements here, which would allow us to squeeze an extra 3 bits
    using int.from_bytes but alas, not in Python 2.x

Exceptions

If thrown while handling a WSGI request these errors will be caught by the underlying handlers and generate calls to
WSGIApp.error_page() with an appropriate 4xx response code.
class pyslet.wsgi.BadRequest
    Bases: exceptions.Exception
    An exception that will generate a 400 response code
class pyslet.wsgi.PageNotAuthorized
    Bases: pyslet.wsgi.BadRequest
    An exception that will generate a 403 response code
```python
class pyslet.wsgi.PageNotFound
    Bases: pyslet.wsgi.BadRequest
    An exception that will generate a 404 response code

class pyslet.wsgi.MethodNotAllowed
    Bases: pyslet.wsgi.BadRequest
    An exception that will generate a 405 response code
```

Other subclasses of Exception are caught and generate 500 errors:

```python
class pyslet.wsgi.SessionError
    Bases: exceptions.RuntimeError
    Unexpected session handling error
```

6.2 XML

This sub-package defines classes for working with XML documents. The version of the standard implemented is the Extensible Markup Language (Fifth Edition), for more info see: http://www.w3.org/TR/xml/

### 6.2.1 XML: Introduction

XML is an integral part of many standards for LET but Pyslet takes a slightly different approach from the pre-existing XML support in the Python language. XML elements are represented by instances of a basic `Element` class which can be used as a base class to customize document processing for specific types of XML document. It also allows these XML elements to ‘come live’ with additional methods and behaviours.

### 6.2.2 XML: Reference

#### Documents and Elements

```python
class pyslet.xml_structures.Node(parent=None)
    Bases: pyslet.py2.UnicodeMixin, pyslet.pep8.MigratedClass
    Base class for Element and Document shared attributes.
    XML documents are defined hierarchically, each element has a parent which is either another element or an XML document.

    get_children()
        Returns an iterator over this object’s children.

classmethod get_element_class(name)
    Returns a class object for representing an element
    name  a unicode string representing the element name.

    The default implementation returns None - for elements this has the effect of deferring the call to the parent document (where this method is overridden to return `Element`).
    This method is called immediately prior to `add_child()` and (when applicable) `get_child_class()`.
    The real purpose of this method is to allow an element class to directly control the way the name of a child element maps to the class used to represent it. You would normally override this method in the `Document` class.
```

6.2. XML
to map element names to classes but in some cases you may want to tweek the mapping at the individual element level. For example, if the same element name is used for two different purposes in the same XML document. Although confusing, this is allowed in XML schema.

get_child_class (stag_class)
Supports custom content model handling

stag_class  The class of an element that is about to be created in the current context with add_child() or the builtin str if data has been recieved in a context where only element content was expected.

This method is only called when the XMLParser.sgml_omit_tag option is in effect. It is called prior to add_child() and gives the context (the parent element or document) a chance to modify the child element that will be created or indicate the end of the current element through use of the OMITTAG feature of SGML.

It returns the class of an element whose start tag has been omitted from the the document and should be added at this point or None if stag_class implies the end of the current element and the end tag may be omitted.

Otherwise this method should return stag_class unchanged (the default implementation does this) indicating that the parser should proceed as normal. In the case of unexpected data this is treated as a validity error and handled according to the parser’s validity checking options.

Validation errors are dealt with by the parser or, where the model is encoded into the classes themselves, by :meth:`add_child` and not by this method which should never raise validation errors.

Although not necessary for true XML parsing this method allows us to support the parsing of XML-like documents that omit tags, such as HTML. For example, suppose we have the following document:

```
<title>My Blank HTML Page</title>
```

The parser would recognise the start tag for <title> and then call this method (on the HTML document) passing the pyslet.html.Title class. For HTML documents, this method always returns the pyslet.html401.HTML class (ignoring stag_class completely). The result is that an HTML element is opened instead and the parser tries again, calling this method for the new HTML element. That does not accept Title either and returns the pyslet.html.Head class. Finally, a Head element is opened and that will accept Title as a child so it returns stag_class unchanged and the parser continues having inferred the omitted tags: <html> and <head>.

add_child (child_class, name=None)
Returns a new child of the given class attached to this object.

child_class  A class (or callable) used to create a new instance of Element.

name  The name given to the element (by the caller). If no name is given then the default name for the child is used. When the child returned is an existing instance, name is ignored.

processing_instruction (target, instruction=")
Abstract method for handling processing instructions

By default, processing instructions are ignored.

get_base ()
Returns the base URI for a node

Abstract method, when used on a Document it returns the URI used to load the document, if known.

set_base (base)
Sets the base URI of a node.

base  A string suitable for setting xml:base or a pyslet.rfc2396.URI instance.
Abstract method. Changing the base effects the interpretation of all relative URIs in this node and its children.

**get_lang()**
Get the language of a node

Abstract method, when used on a `Document` it gets the default language to use in the absence of an explicit xml:lang value.

**set_lang(lang)**
Set the language of a node

`lang` A string suitable for setting the xml:lang attribute of an element.

Abstract method, when used on a `Document` it sets a default language to use in the absence of an explicit xml:lang value.

**get_space()**
Gets the space policy of a node

Abstract method, when used on a `Document` it gets the default space policy to use in the absence of an explicit xml:space value.

**class pyslet.xml.structures.Document**(root=None, base_uri=None, req_manager=None, **kws)

Base class for all XML documents.

With no arguments, a new Document is created with no base URI or root element.

**root** If root is a class object (descended from `Element`) it is used to create the root element of the document.

If root is an orphan instance of `Element` (i.e., it has no parent) is is used as the root element of the document and its `Element.attach_to_doc()` method is called.

**base_uri (aka baseURI for backwards compatibility)** See `set_base()` for more information
req_manager (aka reqManager for backwards compatibility) Sets the request manager object to use for future HTTP calls. Must be an instance of `pyslet.http.client.Client`.

```python
definition
  base_uri = None
  The base uri of the document (as an URI instance)
  lang = None
  The default language of the document (see `set_lang()`).
  declaration = None
  The XML declaration (or None if no XMLDeclaration is used)
  dtd = None
  The dtd associated with the document or None.
  root = None
  The root element or None if no root element has been created yet.
  ```

```python
def definition
  get_children()
  Yields the root element
  XMLParser (entity)
  Creates a parser for this document
    entity The entity to parse the document from
  This method allows some document classes to override the parser used to parse them. This method is only used when parsing existing document instances (see `read()` for more information).
  Classes that override this method may still register themselves with `register_doc_class()` but if they do then the default `XMLParser` object will be used as automatic detection of document class is done by the parser itself based on the information in the prolog (and/or first element).

```

```python
classmethod get_element_class(name)
  Defaults to returning `Element`.
  Derived classes override this method to enable the XML parser to create instances of custom classes based on the document context and element name.

add_child(child_class, name=None)
  Creates the root element of the document.
  If there is already a root element it is detached from the document first using `Element.detach_from_doc()`.
  Unlike `Element.add_child()` there are no model customization options. The root element is always found at `root`.

set_base(base_uri)
  Sets the base_uri of the document to the given URI.
  base_uri An instance of `pyslet.rfc2396.URI` or an object that can be passed to its constructor.
  Relative file paths are resolved relative to the current working directory immediately and the absolute URI is recorded as the document’s `base_uri`.

get_base()
  Returns a string representation of the document’s base_uri.

get_lang()
  Returns the default language for the document.
**set_lang** *(lang)*
Sets the default language for the document.

**get_space()**
Returns the default space policy for the document.

By default we return None, indicating that no policy is in force. Derived documents can overrrid this behaviour to return either “preserve” or “default” to affect space handling.

**validation_error** *(msg, element, data=None, aname=None)*
Called when a validation error is triggered.

*msg* contains a brief message suitable for describing the error in a log file.

*element* the element in which the validation error occurred

*data, aname* See *Element.validation_error()*.

Prior to raising *XMLValidityError* this method logs a suitable message at WARN level.

**register_element** *(element)*
Registers an element’s ID

If the element has an ID attribute it is added to the internal ID table. If the ID already exists *XMLIDClashError* is raised.

**unregister_element** *(element)*
Removes an elements ID

If the element has a uniquely defined ID it is removed from the internal ID table. Called prior to detaching the element from the document.

**get_element_by_id** *(id)*
Returns the element with a given ID

Returns None if the ID is not the ID of any element.

**get_unique_id** *(base_str=None)*
Generates a random element ID that is not yet defined

*base_str* A suggested prefix (defaults to None).

**read** *(src=None, **kws)*
Reads this document, parsing it from a source stream.

With no arguments the document is read from the *base_uri* which must have been specified on con-
struction or with a call to the *set_base()* method.

*src (defaults to None)* You can override the document’s base URI by passing a value for *src* which may
be an instance of *XMLEntity* or a file-like object suitable for passing to *read_from_stream()*.

**read_from_stream** *(src)*
Reads this document from a stream

*src* Any object that can be passed to *XMLEntity*’s constructor.

If you need more control, for example over encodings, you can create the entity yourself and use
*read_from_entity()* instead.

**read_from_entity** *(e)*
Reads this document from an entity

*e* An *XMLEntity* instance.

The document is read from the current position in the entity.
create (dst=None, **kws)
   Creates the Document.
   Outputs the document as an XML stream.

dst (defaults to None)  The stream is written to the base_uri by default but if the ‘dst’ argument is provided then it is written directly to there instead. dst can be any object that supports the writing of binary strings.

Currently only documents with file type baseURIs are supported. The file’s parent directories are created if required. The file is always written using the UTF-8 as per the XML standard.

generate_xml (escape_function=<function escape_char_data>, tab='\t', encoding='UTF-8')
   A generator that yields serialised XML.
   escape_function  The function that will be used to escape character data. The default is escape_char_data(). The alternate name escapeFunction is supported for backwards compatibility.
   tab (defaults to ‘t’)  Whether or not indentation will be used is determined by the tab parameter. If it is empty then no pretty-printing is performed, otherwise elements are indented (where allowed by their defining classes) for ease of reading.
   encoding (defaults to “UTF-8”)  The name of the character encoding to put in the XML declaration.
   Yields character strings, the first string being the XML declaration which always specifies the encoding UTF-8

write_xml (writer, escape_function=<function escape_char_data>, tab='\t')
   Writes serialized XML to an output stream
   writer  A file or file-like object operating in binary mode.
   The other arguments follow the same pattern as generate_xml() which this method uses to create the output which is always UTF-8 encoded.

update (**kws)
   Updates the Document.
   Update outputs the document as an XML stream. The stream is written to the base_uri which must already exist! Currently only documents with file type baseURIs are supported.

diff_string (other_doc, before=10, after=5)
   Compares XML documents
   other_doc  Another Document instance to compare with.
   before (default 10)  Number of lines before the first difference to output
   after (default 5)  Number of lines after the first difference to output
   The two documents are converted to character strings and then compared line by line until a difference is found. The result is suitable for logging or error reporting. Used mainly to make the output of unittests easier to understand.

Create (*args, **kwargs)
   Deprecated equivalent to create()    

DiffString (*args, **kwargs)
   Deprecated equivalent to diff_string()  

GenerateXML (*args, **kwargs)
   Deprecated equivalent to generate_xml()
GetElementByID (*args, **kwargs)
Deprecated equivalent to get_element_by_id()

GetUniqueID (*args, **kwargs)
Deprecated equivalent to get_unique_id()

Read (*args, **kwargs)
Deprecated equivalent to read()

ReadFromEntity (*args, **kwargs)
Deprecated equivalent to read_from_entity()

ReadFromStream (*args, **kwargs)
Deprecated equivalent to read_from_stream()

RegisterElement (*args, **kwargs)
Deprecated equivalent to register_element()

UnregisterElement (*args, **kwargs)
Deprecated equivalent to unregister_element()

Update (*args, **kwargs)
Deprecated equivalent to update()

ValidationError (*args, **kwargs)
Deprecated equivalent to validation_error()

WriteXML (*args, **kwargs)
Deprecated equivalent to write_xml()

class pyslet.xml.structures.Element (parent, name=None)

Bases: pyslet.xml.structures.Node

Base class that represents all XML elements.

This class is usually used only as a default to represent elements with unknown content models or that require no special processing. The power of Pyslet’s XML package comes when different classes are derived from this one to represent the different (classes of) elements defined by an application. These derived classes will normally some form of custom serialisation behaviour (see below).

Although derived classes are free to implement a wide range of python protocols they must always return True in truth tests. An implementation of __bool__ (Python 2, __nonzero__) is provided that does this. This ensures that derived classes are free to implement __len__ but bear in mind that an instance of a derived class for which __len__ returns 0 must still evaluate to True.

Elements compare equal if their names, attribute lists and canonical children all compare equal. No rich comparison methods are provided.

In addition to truth testing, custom attribute serialisation requires a custom implementation of __getattr__, see below for more details.

Elements are usually constructed by calling the parent element’s (or document’s) Node.add_child() method. When constructed directly, the constructor requires that the parent Node be passed as an argument. If you pass None then an orphan element is created (see attach_to_parent()).

Some aspects of the element’s XML serialisation behaviour are controlled by special class attributes that can be set on derived classes.

XMLNAME The default name of the element the class represents.

XMLCONTENT The default content model of the element; one of the ElementType constants.

You can customise attribute mappings using the following special class attributes.
**ID** The name of the ID attribute if the element has a unique ID. With this class attribute set, ID handling is automatic (see `set_id()` and py:attr:`id` below).

By default, attributes are simply stored as name/value character strings in an internal dictionary. It is often more useful to map XML attributes directly onto similarly named attributes of the instances that represent each element.

This mapping can be provided using class attributes of the form `XMLATTR_aname` where /aname/ is the name of the attribute as it would appear in the element’s tag. There are a number of forms of attribute mapping.

**XMLATTR_aname=**<string>

This form creates a simple mapping from the XML attribute ‘aname’ to a python attribute with a defined name. For example, you might want to create a mapping like this to avoid a python reserved word:

```xml
<element class="x"/>
```

This allows XML elements like this:

```
<element class="x"/>
```

To be parsed into python objects that behave like this:

```python
element.style_class="x"  # True
```

If an instance is missing a python attribute corresponding to a defined XML attribute, or it’s value has been set to None, then the XML attribute is omitted from the element’s tag when generating XML output.

**XMLATTR_aname=(<string>, decode_function, encode_function)**

More complex attributes can be handled by setting `XMLATTR_aname` to a tuple. The first item is the python attribute name (as above); the `decode_function` is a simple callable that takes a string argument and returns the decoded value of the attribute and the `encode_function` performs the reverse transformation.

The encode/decode functions can be None to indicate a no-operation.

For example, you might want to create an integer attribute using something like:

```xml
<!-- source XML -->
<element apples="5"/>
```

```python
# class attribute definition
XMLATTR_apples = ('n_apples', int, str)
```

```python
# the resulting object behaves like this...
element.n_apples == 5  # True
```

**XMLATTR_aname=(<string>, decode_function, encode_function, type)**

When XML attribute values are parsed from tags the optional `type` component of the tuple descriptor can be used to indicate a multi-valued attribute. For example, you might want to use a multi-valued mapping for XML attributes defined using one of the plural forms, IDREFS, ENTITIES and NMTOKENS.

If the `type` value is not None then the XML attribute value is first split by white-space, as per the XML specification, and then the decode function is applied to each resulting component. The instance attribute is then set depending on the value of `type`:
list The instance attribute becomes a list, for example:

```xml
<-- source XML -->
<element primes="2 3 5 7"/>
# class attribute definition
XMLATTR_primes = ('primes', int, str, list)
# resulting object behaves like this...
element.primes == [2, 3, 5, 7]  # True
```

dict The instance attribute becomes a dictionary mapping parsed values on to their frequency, for example:

```xml
<-- source XML -->
<element fruit="apple pear orange pear"/>
# class attribute definition
XMLATTR_fruit = ('fruit', None, None, dict)
# resulting object behaves like this...
element.fruit == {'apple': 1, 'orange': 1, 'pear': 2}
```

In this case, the decode function (if given) must return a hashable object!

When serialising to XML the reverse transformations are performed using the encode functions and the type (plain, list or dict) of the attribute’s current value. The declared multi-valued type is ignored. For dictionary values the order of the output values may not be the same as the order originally read from the XML input.

Warning: Empty lists and dictionaries result in XML attribute values that are present but with empty strings. If you wish to omit these attributes in the output XML you must set the attribute value to None.

Some element specifications define large numbers of optional attributes and it is inconvenient to write constructors to initialise these members in each instance and possibly wasteful of memory if a document contains large numbers of such elements.

To obviate the need for optional attributes to be present in every instance an implementation of `__getattr__` is provided that will ensure that `element.aname` returns None if ‘aname’ is the target of an attribute mapping rule, regardless of whether or not the attribute has actually been set for the instance.

Implementation note: internally, the `XMLATTR_*` descriptors are parsed into two mappings the first time they are needed. The forward map maps XML attribute names onto tuples of:

```python
(<python attribute name>, decode_function, type)
```

The reverse map maps python attribute names onto a tuple of:

```python
(<xml attribute name>, encode_function)
```

XML attribute names may contain many characters that are not legal in Python syntax but automated attribute processing is still supported for these attributes even though the declaration cannot be written into the class definition. Use the built-in function `setattr` immediately after the class is defined, for example:

```python
class MyElement(Element):
    pass

setattr(MyElement, 'XMLATTR_hyphen-attr', 'hyphen_attr')
```
XMLCONTENT = 2
We default to a mixed content model

set_xmlname(name)
Sets the name of this element

   name  A character string.

   You will not normally need to call this method, it is called automatically during child creation.

get_xmlname()
Returns the name of this element

   In the default implementation this is a simple character string.

get_document()
Returns the document that contains the element.

   If the element is an orphan, or is the descendent of an orphan then None is returned.

set_id(id)
Sets the id of the element

   The change is registered with the enclosing document. If the id is already taken then XMLIDClashError
   is raised.

classmethod mangle_aname(name)
Returns a mangled attribute name

   A mangled attribute name is simple name prefixed with “XMLATTR_”.

classmethod unmangle_aname(mname)
Returns an unmangled attribute name.

   If mname is not a mangled name, None is returned. A mangled attribute name starts with “XMLATTR_”.

get_attributes()
Returns a dict mapping attribute names onto values.

   Each attribute value is represented as a character string. Derived classes MUST override this method if
   they define any custom attribute mappings.

   The dictionary returned represents a copy of the information in the element and so may be modified by the
   caller.

set_attribute(name, value)
Sets the value of an attribute.

   name  The name of the attribute to set

   value  The value of the attribute (as a character string) or None to remove the attribute.

get_attribute(name)
Gets the value of a single attribute as a string.

   If the element has no attribute with name then KeyError is raised.

   This method searches the attribute mappings and will return attribute values obtained by encoding the
   associated objects according to the mapping.

is_valid_name(value)
Returns True if a character string is a valid NAME

   This test can be done standalone using the module function of the same name (this implementation defaults
to using that function). By checking validity in the context of an element derived classes may override this
test.
This test is used currently only used when checking IDs (see `set_id()`)

**is_empty()**

Whether this element *must* be empty.

If the class defines the `XMLCONTENT` attribute then the model is taken from there and this method returns True only if `XMLCONTENT` is `ElementType.EMPTY`.

Otherwise, the method defaults to False

**is_mixed()**

Whether or not the element *may* contain mixed content.

If the class defines the `XMLCONTENT` attribute then the model is taken from there and this method returns True only if `XMLCONTENT` is `ElementType.MIXED`.

Otherwise, the method defaults to True

**get_children()**

Returns an iterable of the element’s children.

This method iterates through the internal list of children only. Derived classes with custom models (i.e., those that define attributes to customise child element creation) MUST override this method.

Each child is either a character string or an instance of Element (or a derived class thereof). We do not represent comments, processing instructions or other meta-markup.

**get_canonical_children()**

Returns children with canonical white space

A wrapper for `get_children()` that returns an iterable of the element’s children canonicalized for white space as follows. We check the current setting of xml:space, returning the same list of children as `get_children()` if ‘preserve’ is in force. Otherwise we remove any leading space and collapse all others to a single space character.

**get_or_add_child(child_class)**

Returns the first child of type `child_class`

If there is no child of that class then a new child is added.

**add_child(child_class, name=None)**

Adds a new child of the given class attached to this element.

- `child_class` A class object (or callable) used to create a new instance.
- `name` The name given to the element (by the caller). If no name is given then the default name for the child is used. When the child returned is an existing instance, name is ignored.

By default, an instance of `child_class` is created and attached to the internal list of child elements.

Child creation can be customised to support a more natural mapping for structured elements as follows. Firstly, the name of `child_class` (not the element name) is looked up in the parent (self), if there is no match, the method resolution order is followed for `child_class` looking up the names of each base in turn until a matching attribute is found. If there are no matches then the default handling is performed.

Otherwise, the behaviour is determined by the matching attribute as follows.

1. **If the attribute is None then a new instance of `child_class`** is created and assigned to the attribute.
2. **If the attribute is a list then a new instance of `child_class`** is created and appended to the attribute’s value.
3. **Finally, if the attribute value is already an instance of `child_class`** it is returned unchanged.
4 Deprecated: A method attribute is called either without arguments (if the method name matches the child_class exactly) or with the child_class itself passed as an argument. It must return the new child element.

In summary, a new child is created and attached to the element’s model unless the model supports a single element of the given child_class and the element already exists (as evidenced by an attribute with the name of child_class or one of its bases), in which case the existing instance is returned.

remove_child(child)
Removes a child from this element’s children.

child An Element instance that must be a direct child. That is, one that would be yielded by get_children().

By default, we search the internal list of child elements.

For content model customisation we follow the same name matching conventions as for child creation (see add_child()). If a matching attribute is found then we process them as follows:

1 If the attribute’s value is child then it is set to None, if it is not child then XMLUnknownChild is raised.

2 If the attribute is a list then we remove child from the list. If child is not in the list XMLUnknownChild is raised.

3. If the attribute is None then we raise XMLUnknownChild.

find_children(child_class, child_list, max=None)
Finds children of a given class

Deprecated in favour of:

list(e.find_children_depth_first(child_class, False))

child_class A class object derived from Element. May also be a tuple as per the definition of the builtin isinstance function in python.

child_list A list. Matching children are appended to this.

max (defaults to None) Maximum number of children to match (None means no limit). This value is used to check against the length of child_list so any elements already present will count towards the total.

Nested matches are not included. In other words, if the model of child_class allows further elements of type child_class as children (directly or indirectly) then only the top-level match is returned. (Use find_children_depth_first() for a way to return recursive lists of matching children.)

The search is done depth first so children are returned in the logical order they would appear in the document.

find_children_breadth_first(child_class, sub_match=True, max_depth=1000, **kws)
Generates all children of a given class

child_class A class object derived from Element. May also be a tuple as per the definition of the builtin isinstance function in python.

sub_match (defaults to True) Matching elements are also scanned for nested matches. If False, only the outer-most matching element is returned.

max_depth Controls the maximum depth of the scan with level 1 indicating direct children only. It must be a positive integer and defaults to 1000.
Warning: to reduce memory requirements when searching large documents this method performs a two-pass scan of the element’s children, i.e., `get_children()` will be called twice.

Given that XML documents tend to be broader than they are deep `find_children_depth_first()` is a better method to use for general purposes.

**find_children_depth_first** *(child_class, sub_match=True, max_depth=1000, **kws)*

Generates all children of a given class

- **child_class**: A class object derived from `Element`. May also be a tuple as per the definition of the builtin `isinstance` function in python.

- **sub_match (defaults to True)**: Matching elements are also scanned for nested matches. If False, only the outer-most matching element is returned.

- **max_depth**: Controls the maximum depth of the scan with level 1 indicating direct children only. It must be a positive integer and defaults to 1000.

Uses a depth-first scan of the element hierarchy rooted at the current element.

**find_parent** *(parent_class)*

Finds the first parent of the given class.

- **parent_class**: A class object descended from `Element`.

Traverses the hierarchy through parent elements until a matching parent is found or returns None.

**attach_to_parent** *(parent)*

Called to attach an orphan element to a parent.

This method is not normally needed, when creating XML elements you would normally call `add_child()` on the parent which ensures that elements are created in the context of a parent node. The purpose of this method is to allow orphaned elements to be associated with a (new) parent. For example, after being detached from one element hierarchy and attached to another.

This method does not do any special handling of child elements, the caller takes responsibility for ensuring that this element will be returned by future calls to `parent.get_children()`. However, `attach_to_doc()` is called to ensure id registrations are made.

**attach_to_doc** *(doc=None)*

Called when the element is first attached to a document.

This method is not normally needed, when creating XML elements you would normally call `add_child()` on the parent which ensures that elements are created in the context of a containing document. The purpose of this method is to allow orphaned elements to be associated with a parent (document) after creation. For example, after being detached from one element hierarchy and attached to another (possibly in a different document).

The default implementation ensures that any ID attributes belonging to this element or its descendents are registered.

**detach_from_parent** *

Called to detach an element from its parent

The result is that this element becomes an orphan.

This method does not do any special handling of child elements, the caller takes responsibility for ensuring that this element will no longer be returned by future calls to the (former) parent’s `get_children()` method.

We do call `detach_from_doc()` to ensure id registrations are removed and `parent` is set to None.

**detach_from_doc** *(doc=None)*

Called when an element is being detached from a document.
The document the element is being detached from, if None then this is determined automatically.

The default implementation ensures that any ID attributes belonging to this element or its descendents are unregistered.

**add_data** *(data)*

Adds a character string to this element’s children.

This method raises a validation error if the element cannot take data children.

**content_changed** *

Notifies an element that its content has changed.

Called by the parser once the element’s attribute values and content have been parsed from the source. Can be used to trigger any internal validation required following manual changes to the element.

The default implementation tidies up the list of children reducing runs of data to a single unicode string to make future operations simpler and faster.

**generate_value**(ignore_elements=False)

Generates strings representing the element’s content

A companion method to get_value() which is useful when handling elements that contain a large amount of data. For more information see get_value().

**get_value**(ignore_elements=False)

Returns a single object representing the element’s content.

**ignore_elements** If True then any elements found in mixed content are ignored. If False then any child elements cause XMLMixedContentError to be raised.

The default implementation returns a character string and is only supported for elements where mixed content is permitted (is_mixed()). It uses generate_value() to iterate through the children.

If the element is empty an empty string is returned.

Derived classes may return more complex objects, such as values of basic python types or class instances that better represent the content of the element.

You can pass ignore_elements as True to override this behaviour in the unlikely event that you want:

```xml
<!-- elements like this... -->
<data>This is <em>the</em> value</data>

# to behave like this:
data.get_value(True) == "This is value"
```

**set_value**(value)

Replaces the content of the element.

**value** A character string used to replace the content of the element. Derived classes may support a wider range of value types, if the default implementation encounters anything other than a character string it attempts to convert it before setting the content.

The default implementation is only supported for elements where mixed content is permitted (see is_mixed()) and only affects the internally maintained list of children. Elements with more complex mixed models MUST override this method.

If value is None then the element becomes empty.

**reset**(reset_attrs=False)

Resets all children (and optionally attribute values).
reset_attrs Whether or not to reset attribute values too.

Called by the default implementation of set_value() with reset_atts=False, removes all children from the internally maintained list of children.

Called by the default implementation of add_child() with reset_atts=True when an existing element instance is being recycled (obviating the constructor). The default implementation removes only unmapped attribute values. Mapped attribute values are not reset.

Derived classes should call this method if they override the implementation of set_value().

Derived classes with custom content models, i.e., those that provide a custom implementation for get_children(), must override this method and treat it as an event associated with parsing the start tag of the element. (This method is also a useful signal for resetting an state used for validating custom content models.)

Required children should be reset and optional children should be orphaned using detach_from_parent() and any references to them in instance attributes removed. Failure to override this method will can result in the child elements accumulating from one read to the next.

validation_error (msg, data=None, aname=None)
Called when a validation error occurred in this element.

msg Message suitable for logging and reporting the nature of the error.

data The data that caused the error may be given in data.

aname The attribute name may also be given indicating that the offending data was in an attribute of the element and not the element itself.

The default implementation simply calls the containing Document’s Document. validation_error() method. If the element is an orphan then XMLValidityError is raised directly with msg.

static sort_names (name_list)
Sorts names in a predictable order

name_list A list of element or attribute names

The default implementation assumes that the names are strings or unicode strings so uses the default sort method.

deepcopy (parent=None)
Creates a deep copy of this element.

parent The parent node to attach the new element to. If it is None then a new orphan element is created.

This method mimics the process of serialisation and deserialisation (without the need to generate markup). As a result, element attributes are serialised and deserialised to strings during the copy process.

gt_base ()
Returns the value of the xml:base attribute as a string.

set_base (base)
Sets the value of the xml:base attribute from a string.

Changing the base of an element effects the interpretation of all relative URIs in this element and its children.

resolve_base ()
Returns the base of the current element.

The URI is calculated using any xml:base values of the element or its ancestors and ultimately relative to the base URI of the document itself.
If the element is not contained by a Document, or the document does not have a fully specified base_uri then the return result may be a relative path or even None, if no base information is available.

The return result is always None or a character string, such as would be obtained from the xml:base attribute.

**resolve_uri (uriref)**

Resolves a URI reference in the current context.

**uriref** A *pyslet.rfc2396.URI* instance or a string that one can be parsed from.

The argument is resolved relative to the xml:base values of the element's ancestors and ultimately relative to the document's base. The result may still be a relative URI, there may be no base set or the base may only be known in relative terms.

For example, if the Document was loaded from the URL:

```
http://www.example.com/images/catalog.xml
```

and *e* is an element in that document then:

```
e.resolve_uri('smiley.gif')
```

would return a URI instance representing the fully-specified URI:

```
http://www.example.com/images/smiley.gif
```

**relative_uri (href)**

Returns href expressed relative to the element’s base.

**href** A *pyslet.rfc2396.URI* instance or a string that one can be parsed from.

If href is already a relative URI then it is converted to a fully specified URL by interpreting it as being the URI of a file expressed relative to the current working directory.

For example, if the Document was loaded from the URL:

```
http://www.example.com/images/catalog.xml
```

and *e* is an element in that document then:

```
e.relative_uri('http://www.example.com/images/smiley.gif')
```

would return a URI instance representing relative URI:

```
'smiley.gif'
```

If the element does not have a fully-specified base URL then the result is a fully-specified URL itself.

**get_lang ()**

Returns the value of the xml:lang attribute as a string.

**set_lang (lang)**

Sets the value of the xml:lang attribute from a string.

See **resolve_lang ()** for how to obtain the effective language of an element.

**resolve_lang ()**

Returns the effective language for the current element.

The language is resolved using the xml:lang value of the element or its ancestors. If no xml:lang is in effect then None is returned.
get_space()

Gets the value of the xml:space attribute

set_space(space)

Sets the xml:space attribute

   space  A character string containing the new value or None to clear the attribute definition on this element.

resolve_space(space)

Returns the effective space policy for the current element.

   The policy is resolved using the value returned by get_space() on this element or its ancestors. If no space policy is in effect then None is returned.

can_pretty_print()

True if this element’s content may be pretty-printed.

   This method is used when formatting XML files to text streams. The output is also affected by the xml:space attribute. Derived classes can override the default behaviour.

   The difference between this method and the xml:space attribute is that this method indicates if white space can be safely added to the output to improve formatting by inserting line feeds to break it over multiple lines and to insert spaces or tab characters to indent tags.

   On the other hand, xml:space='preserve' indicates that white space in the original document must not be taken away. It therefore makes sense that if get_space() returns ‘preserve’ we will return False. Derived classes may consider providing an implementation of get_space that always return ‘preserve’ and using the default implementation of this method.

   This method will return False if one of the following is true:

   • the special attribute SGMLCDATA is present

   • the special content model attribute XMLCONTENT indicates that the element may contain mixed content (this is the default for generic instances of Element)

   • get_space() is set to ‘preserve’ (xml:space)

   • self.parent.can_pretty_print() returns False

   Otherwise we return True.

write_xml_attributes(attributes, escape_function=<function escape_char_data>, root=False, **kws)

Creates strings serialising the element’s attributes

   attributes  A list of character strings

   escape_function  The function that will be used to escape character data. The default is escape_char_data(). The alternate name escapeFunction is supported for backwards compatibility.

   root  Indicates if this element should be treated as the root element. By default there is no special action required but derived classes may need to generate additional attributes, such as those that relate to the namespaces or schema used by the element.

The attributes are generated as strings of the form ‘name=”value”’ with values escaped appropriately for serialised XML output. The attributes are always sorted into a predictable order (based on attribute name) to ensure that identical documents produce identical output.

generate_xml(escape_function=<function escape_char_data>, indent=”", tab=’\t’, root=False, **kws)

A generator that yields serialised XML.
**escape_function** The function that will be used to escape character data. The default is `escape_char_data()`. The alternate name `escapeFunction` is supported for backwards compatibility.

**indent (defaults to an empty string)** The string to use for passing any inherited indent, used in combination with the tab parameter for pretty printing. See below.

**tab (defaults to ‘t’)** Whether or not indentation will be used is determined by the tab parameter. If it is empty then no pretty-printing is performed for the element, otherwise the element will start with a line-feed followed by any inherited `indent` and finally followed by the content of `tab`. For example, if you prefer to have your XML serialised with a 4-space indent then pass `tab=' ' `.

If the element is in a context where pretty printing is not allowed (see `can_pretty_print()`) then `tab` is ignored.

**root (defaults to False)** Indicates if this is the root element of the document. See `write_xml_attributes()`.

Yields character strings.

**write_xml** *(writer, escape_function=<function escape_char_data>, indent='', tab='\t', root=False, **kws)*

Writes serialized XML to an output stream

**writer** A file or file-like object operating in binary mode.

The other arguments follow the same pattern as `generate_xml()` which this method uses to create the output which is always UTF-8 encoded.

**AddData** *(*args, **kwargs)*

Deprecated equivalent to `add_data()`

**AttachToDocument** *(*args, **kwargs)*

Deprecated equivalent to `attach_to_doc()`

**AttachToParent** *(*args, **kwargs)*

Deprecated equivalent to `attach_to_parent()`

**ContentChanged** *(*args, **kwargs)*

Deprecated equivalent to `content_changed()`

**Copy** *(*args, **kwargs)*

Deprecated equivalent to `deepcopy()`

**DeleteChild** *(*args, **kwargs)*

Deprecated equivalent to `remove_child()`

**DetachFromDocument** *(*args, **kwargs)*

Deprecated equivalent to `detach_from_doc()`

**DetachFromParent** *(*args, **kwargs)*

Deprecated equivalent to `detach_from_parent()`

**FindChildren** *(*args, **kwargs)*

Deprecated equivalent to `find_children()`

**FindChildrenBreadthFirst** *(*args, **kwargs)*

Deprecated equivalent to `find_children_breadth_first()`

**FindChildrenDepthFirst** *(*args, **kwargs)*

Deprecated equivalent to `find_children_depth_first()`

**FindParent** *(*args, **kwargs)*

Deprecated equivalent to `find_parent()`
GenerateXML(*args, **kwargs)
    Deprecated equivalent to generate_xml()

GetAttribute(*args, **kwargs)
    Deprecated equivalent to get_attribute()

GetAttributes(*args, **kwargs)
    Deprecated equivalent to get_attributes()

GetCanonicalChildren(*args, **kwargs)
    Deprecated equivalent to get_canonical_children()

GetDocument(*args, **kwargs)
    Deprecated equivalent to get_document()

GetValue(*args, **kwargs)
    Deprecated equivalent to get_value()

GetXMLName(*args, **kwargs)
    Deprecated equivalent to get_xmlname()

IsEmpty(*args, **kwargs)
    Deprecated equivalent to is_empty()

IsMixed(*args, **kwargs)
    Deprecated equivalent to is_mixed()

IsValidName(*args, **kwargs)
    Deprecated equivalent to is_valid_name()

classmethod MangleAttributeName(*args, **kwargs)
    Deprecated equivalent to mangle_aname()

PrettyPrint(*args, **kwargs)
    Deprecated equivalent to can_pretty_print()

RelativeURI(*args, **kwargs)
    Deprecated equivalent to relative_uri()

ResolveBase(*args, **kwargs)
    Deprecated equivalent to resolve_base()

ResolveLang(*args, **kwargs)
    Deprecated equivalent to resolve_lang()

ResolveURI(*args, **kwargs)
    Deprecated equivalent to resolve_uri()

setAttribute(*args, **kwargs)
    Deprecated equivalent to set_attribute()

setId(*args, **kwargs)
    Deprecated equivalent to set_id()

setSpace(*args, **kwargs)
    Deprecated equivalent to set_space()

setValue(*args, **kwargs)
    Deprecated equivalent to set_value()

setXMLName(*args, **kwargs)
    Deprecated equivalent to set_xmlname()
static SortNames(*args, **kwargs)
    Deprecated equivalent to sort_names()

classmethod UnmangleAttributeName(*args, **kwargs)
    Deprecated equivalent to unmangle_aname()

ValidationError(*args, **kwargs)
    Deprecated equivalent to validation_error()

WriteXML(*args, **kwargs)
    Deprecated equivalent to write_xml()

WriteXMLAttributes(*args, **kwargs)
    Deprecated equivalent to write_xml_attributes()

pyslet.xml.structures.map_class_elements(class_map, scope)
    Adds element name -> class mappings to class_map

    class_map A dictionary that maps XML element names onto class objects that should be used to represent them.

    scope A dictionary, or an object containing a __dict__ attribute, that will be scanned for class objects to add to the mapping. This enables scope to be a module. The search is not recursive, to add class elements from imported modules you must call map_class_elements for each module.

    Mappings are added for each class that is derived from Element that has an XMLNAME attribute defined. It is an error if a class is found with an XMLNAME that has already been mapped.

Exceptions

class pyslet.xml.structures.XMLMissingResourceError
    Bases: pyslet.xml.structures.XMLError
    Raised when an entity cannot be found (e.g., missing file).
    Also raised when an external entity reference is encountered but the opening of external entities is turned off.

class pyslet.xml.structures.XMLMissingLocationError
    Bases: pyslet.xml.structures.XMLError
    Raised when on create, read or update when base_uri is None

class pyslet.xml.structures.XMLUnsupportedSchemeError
    Bases: pyslet.xml.structures.XMLError
    Document.base_uri has an unsupported scheme

    Currently only file, http and https schemes are supported for open operations. For create and update operations, only file types are supported.

class pyslet.xml.structures.XMLUnexpectedHTTPResponse
    Bases: pyslet.xml.structures.XMLError
    Raised by Document.open_uri()

    The message contains the response code and status message received from the server.

Prolog and Document Type Declaration

class pyslet.xml.structures.XMLDTD
    Bases: pyslet.pep8.MigratedClass
An object that models a document type declaration.

The document type declaration acts as a container for the entity, element and attribute declarations used in a document.

```python
name = None
    The declared Name of the root element
parameter_entities = None
    A dictionary of XMLParameterEntity instances keyed on entity name.
general_entities = None
    A dictionary of XMLGeneralEntity instances keyed on entity name.
notations = None
    A dictionary of XMLNotation instances keyed on notation name.
element_list = None
    A dictionary of ElementType definitions keyed on the name of element.
attribute_lists = None
    A dictionary of dictionaries, keyed on element name. Each of the resulting dictionaries is a dictionary of XMLAttributeDefinition keyed on attribute name.
```

```python
declare_entity(entity)
    Declares an entity in this document.
    The same method is used for both general and parameter entities. The value of entity can be either an XMLGeneralEntity or an XMLParameterEntity instance.

get_parameter_entity(name)
    Returns the parameter entity definition matching name.
    Returns an instance of XMLParameterEntity. If no parameter has been declared with name then None is returned.

get_entity(name)
    Returns the general entity definition matching name.
    Returns an instance of XMLGeneralEntity. If no general has been declared with name then None is returned.
```

```python
declare_notation(notation)
    Declares a notation for this document.
    The value of notation must be a XMLNotation instance.

get_notation(name)
    Returns the notation declaration matching name.
    name The name of the notation to search for.
    Returns an instance of XMLNotation. If no notation has been declared with name then None is returned.
```

```python
declare_element_type(etype)
    Declares an element type.
    etype An ElementType instance containing the element definition.

get_element_type(element_name)
    Looks up an element type definition.
    element_name the name of the element type to look up
```
The method returns an instance of `ElementType` or None if no element with that name has been declared.

```python
declare_attribute(element_name, attr_def)
```
Declares an attribute.

- `element_name` the name of the element type which should have this attribute applied
- `attr_def` An `XMLAttributeDefinition` instance describing the attribute being declared.

```python
get_attribute_list(name)
```
Returns a dictionary of attribute definitions

- `name` The name of the element type to look up.
  If there are no attributes declared for this element type, None is returned.

```python
get_attribute_definition(element_name, attr_name)
```
Looks up an attribute definition.

- `element_name` the name of the element type in which to search
- `attr_name` the name of the attribute to search for.

The method returns an instance of `XMLAttributeDefinition` or None if no attribute matching this description has been declared.

```python
GetAttributeList(*args, **kwargs)
```
Deprecated equivalent to `get_attribute_list()`

```python
class pyslet.xml.structures.XMLDeclaration(version, encoding='UTF-8', standalone=False)
```
Bases: `pyslet.xml.structures.XMLTextDeclaration`

Represents a full XML declaration.

Unlike the parent class, `XMLTextDeclaration`, the version is required. `standalone` defaults to False as this is the assumed value if there is no standalone declaration.

```python
standalone = None
```
Whether an XML document is standalone.

```python
class pyslet.xml.structures.ElementType
```
Bases: `object`

Represents element type definitions.

```python
EMPTY = 0
```
Content type constant for EMPTY

```python
ANY = 1
```
Content type constant for ANY

```python
MIXED = 2
```
Content type constant for mixed content

```python
ELEMENT_CONTENT = 3
```
Content type constant for element content

```python
SGMLCDATA = 4
```
Additional content type constant for SGML CDATA

```python
entity = None
```
The entity in which this element was declared
name = None
    The name of this element

content_type = None
    The content type of this element, one of the constants defined above.

content_model = None
    A XMLContentParticle instance which contains the element’s content model or None in the case of
    EMPTY or ANY declarations.

particle_map = None
    A mapping used to validate the content model during parsing. It maps the name of the first child element
    found to a list of XMLNameParticle instances that can represent it in the content model. For more
    information see XMLNameParticle.particle_map.

build_model()
    Builds internal structures to support model validation.

is_deterministic()
    Tests if the content model is deterministic.
    For degenerate cases (elements declared with ANY or EMPTY) the method always returns True.

class pyslet.xml.structures.XMLContentParticle
    Bases: object

    An object for representing content particles.

    ZeroOrOne = 1
        Occurrence constant for ‘?’

    OneOrMore = 3
        Occurrence constant for ‘+’

occurrence = None
    One of the occurrence constants defined above.

build_particle_maps(exit_particles)
    Abstract method that builds the particle maps for this node or its children.
    For more information see XMLNameParticle.particle_map.

    Although only name particles have particle maps this method is called for all particle types to allow the
    model to be built hierarchically from the root out to the terminal (name) nodes. exit_particles provides a
    mapping to all the following particles outside the part of the hierarchy rooted at the current node that are
    directly reachable from the particles inside.

seek_particles(pmap)
    Adds all possible entry particles to pmap.

    Abstract method, pmap is a mapping from element name to a list of XMLNameParticles
    XMLNameParticle.

    Returns True if a required particle was added, False if all particles added are optional.

    Like build_particle_maps(), this method is called for all particle types. The mappings requested
    represent all particles inside the part of the hierarchy rooted at the current node that are directly reachable
    from the preceeding particles outside.

add_particles(src_map, pmap)
    A utility method that adds particles from src_map to pmap.

    Both maps are mappings from element name to a list of XMLNameParticles XMLNameParticle.
    All entries in src_map not currently in pmap are added.
is_deterministic(pmap)

A utility method for identifying deterministic particle maps.

A deterministic particle map is one in which each name maps uniquely to a single content particle. A non-deterministic particle map contains an ambiguity, for example ((b,d)|(b,e)). The particle map created by `seek_particles()` for the enclosing choice list would have two entries for ‘b’, one to map the first particle of the first sequence and one to the first particle of the second sequence.

Although non-deterministic content models are not allowed in SGML they are tolerated in XML and are only flagged as compatibility errors.

class pyslet.xml.structures.XMLNameParticle
    Bases: pyslet.xml.structures.XMLContentParticle

    Represents a content particle for a named element

    name = None
        the name of the element type that matches this particle

    particle_map = None
        Each `XMLNameParticle` has a particle map that maps the name of the ‘next’ element found in the content model to the list of possible `XMLNameParticle` that represent it in the content model.

        The content model can be traversed using `ContentParticleCursor`.

class pyslet.xml.structures.XMLChoiceList
    Bases: pyslet.xml.structures.XMLContentParticle

    Represents a choice list of content particles in the grammar

class pyslet.xml.structures.XMLSequenceList
    Bases: pyslet.xml.structures.XMLContentParticle

    Represents a sequence list of content particles in the grammar

class pyslet.xml.structures.XMLAttributeDefinition
    Bases: object

    Represents an Attribute declaration

    There is no special functionality provided by this class, instances hold the data members identified and the class defines a number of constants suitable for setting and testing them.

    Contents are defined using CAPS, mixed case versions are provided only for backwards compatibility.

    CDATA = 0
        Type constant representing CDATA

    ID = 1
        Type constant representing ID

    IDREF = 2
        Type constant representing IDREF

    IDREFS = 3
        Type constant representing IDREFS

    ENTITY = 4
        Type constant representing ENTITY

    ENTITIES = 5
        Type constant representing ENTITIES
NMTOKEN = 6
    Type constant representing NMTOKEN

NMTOKENS = 7
    Type constant representing NMTOKENS

NOTATION = 8
    Type constant representing NOTATION

ENUMERATION = 9
    Type constant representing an enumeration, not defined as a keyword in the specification but representing
    declarations that match production [59], Enumeration.

IMPLIED = 0
    Presence constant representing #IMPLIED

REQUIRED = 1
    Presence constant representing #REQUIRED

FIXED = 2
    Presence constant representing #FIXED

DEFAULT = 3
    Presence constant representing a declared default value. Not defined as a keyword but represents a decla-
    ration with a default value defined in production [60].

entity = None
    the entity in which this attribute was declared

ame = None
    the name of the attribute

type = None
    One of the above type constants

values = None
    An optional dictionary of values

defaultValue = None
    An optional default value

Physical Structures

class pyslet.xml.structures.XMLEntity(src=None, encoding=None, req_manager=None, **kws)

    Bases: pyslet.pep8.MigratedClass

    Represents an XML entity.

    This object serves two purposes, it acts as both the object used to store information about declared entities and
    also as a parser for feeding unicode characters to the main XMLParser.

    src May be a character string, a binary string, an instance of pyslet.rfc2396.URI, an instance of
    pyslet.http.client.ClientResponse or any object that supports file-like behaviour (seek and read).

    If provided, the corresponding open method is called immediately, see open_unicode(),
    open_string(), open_uri(), open_http_response() and open_file().

    encoding If src is not None then this value will be passed when opening the entity reader.

    req_manager If src is a URI, passed to open_uri()
XMLEntity objects act as context managers, hence it is possible to use:

```python
with XMLEntity(src=URI.from_octets('mydata.xml')) as e:
    # process the entity here, will automatically close
```

**location = None**

the location of this entity (used as the base URI to resolve relative links). A `pyslet.rfc2396.URI` instance.

**mimetype = None**

The mime type of the entity, if known, or None otherwise. A `pyslet.http.params.MediaType` instance.

**encoding = None**

the encoding of the entity (text entities), e.g., ‘utf-8’

**bom = None**

Flag to indicate whether or not the byte order mark was detected. If detected the flag is set to True. An initial byte order mark is not reported in `the_char` or by the `next_char()` method.

**the_char = None**

The character at the current position in the entity

**line_num = None**

The current line number within the entity (first line is line 1)

**line_pos = None**

the current character position within the entity (first char is 1)

**buff_text = None**

used by `XMLParser.push_entity()`

**chunk_size = 8192**

Characters are read from the data_source in chunks.

The default chunk size is set from `io.DEFAULT_BUFFER_SIZE`, typically 8KB.

In fact, in some circumstances the entity reader starts more cautiously. If the entity reader expects to read an XML or Text declaration, which may have an encoding declaration then it reads one character at a time until the declaration is complete. This allows the reader to change to the encoding in the declaration without causing errors caused by reading too many characters using the wrong codec. See `change_encoding()` and `keep_encoding()` for more information.

**get_name()**

Returns a name to represent this entity

The name is intended for logs and error messages. It defaults to the location if set.

**is_external()**

Returns True if this is an external entity.

The default implementation returns True if `location` is not None, False otherwise.

**open()**

Opens the entity for reading.

The default implementation uses `open_uri()` to open the entity from `location` if available, otherwise it raises `NotImplementedError`.

**is_open()**

Returns True if the entity is open for reading.
open_unicode (src)
Opens the entity from a unicode string.

open_string (src, encoding=None)
Opens the entity from a binary string.

src A binary string.
encoding The optional encoding is used to convert the string to unicode and defaults to None - meaning that the auto-detection method will be applied.

The advantage of using this method instead of converting the string to unicode and calling open_unicode() is that this method creates a unicode reader object to parse the string instead of making a copy of it in memory.

open_file (src, encoding='utf-8')
Opens the entity from a file.

src An existing (open) binary file.
encoding The optional encoding provides a hint as to the intended encoding of the data and defaults to UTF-8. Unlike other Open* methods we do not assume that the file is seekable however, you may set encoding to None for a seekable file thus invoking auto-detection of the encoding.

open_uri (src, encoding=None, req_manager=None, **kws)
Opens the entity from a URI.

src A pyslet.rfc2396.URI instance of either file, http or https schemes.
encoding The optional encoding provides a hint as to the intended encoding of the data and defaults to UTF-8. For http(s) resources this parameter is only used if the charset cannot be read successfully from the HTTP headers.
req_manager The optional req_manager allows you to pass an existing instance of pyslet.http.client.Client for handling URI with http or https schemes. (reqManager is supported for back-wards compatibility.)

open_http_response (src, encoding='utf-8')
Opens the entity from an HTTP response passed in src.

encoding The optional encoding provides a hint as to the intended encoding of the data and defaults to UTF-8. This parameter is only used if the charset cannot be read successfully from the HTTP response headers.

reset ()
Resets an open entity

The entity returns to the first character in the entity.

get_position_str ()
A short string describing the current position.

For example, if the current character is pointing to character 6 of line 4 then it will return the string ‘Line 4.6’

next_char ()
Advances to the next character in an open entity.

This method takes care of the End-of-Line handling rules for XML which force us to remove any CR characters and replace them with LF if they appear on their own or to silently drop them if they appear as part of a CR-LF combination.
auto_detect_encoding(src_file)
Auto-detects the character encoding

src_file A file object. The object must support seek and blocking read operations. If src_file has been
opened in text mode then no action is taken.

documentation

class pyslet.xml.structures.XMLDeclaredEntity(name=None, definition=None)

Abstract class representing a declared entity.

name An optional string used as the name of the entity
definition The definition of the entity is either a string or an instance of XMLEntityID, depending on
whether the entity is an internal or external entity respectively.

entity = None
the entity in which this entity was declared

text

name = None
the name passed to the constructor
definition = None
the definition passed to the constructor

get_name()
Human-readable name suitable for logging/error reporting.
Simply returns name
is_external()  
    Returns True if this is an external entity.

open()  
    Opens the entity for reading.

    External entities must be parsed for text declarations before the replacement text is encountered. This  
    requires a small amount of look-ahead which may result in some characters needing to be re-parsed. We  
    pass this to future parsers using buff_text.

class pyslet.xml.structures.XMLGeneralEntity(name=None, definition=None, notation=None)  
    Bases: pyslet.xml.structures.XMLDeclaredEntity  
    Represents a general entity.

    name  Optional name

    definition  An optional definition

    notation  An optional notation.

    notation = None  
        the notation name for external unparsed entities

get_name()  
    Formats the name as a general entity reference.

class pyslet.xml.structures.XMLParameterEntity(name=None, definition=None)  
    Bases: pyslet.xml.structures.XMLDeclaredEntity  
    Represents a parameter entity.

    name  An optional name

    definition  An optional definition.

    See base class for more information on the parameters.

open_as_pe()  
    Opens the parameter entity in the context of a DTD.

    This special method implements the rule that the replacement text of a parameter entity, when included as  
    a PE, must be enlarged by the attachment of a leading and trailing space.

get_name()  
    Formats the name as a parameter entity reference.

class pyslet.xml.structures.XMLExternalID(public=None, system=None)  
    Bases: object  
    Represents external references to entities.

    public  An optional public identifier

    system  An optional system identifier

    One (or both) of the identifiers should be provided.

get_location(base=None)  
    Get an absolute URI for the external entity.

    Returns a pyslet.rfc2396.URI resolved against base if applicable. If there is no system identifier  
    then None is returned.
class pyslet.xml.structures.XMLTextDeclaration

    version='1.0', encoding='UTF-8')
Bases: object

    Represents the text components of an XML declaration.

    Both version and encoding are optional, though one or other are required depending on the context in which the declaration will be used.

class pyslet.xml.structures.XMLNotation

    name, external_id
Bases: object

    Represents an XML Notation defined in Section 4.7

    name  The name of the notation

    external_id  A XMLExternalID instance in which one of public or system must be provided.

    name = None
        the notation name

    external_id = None
        the external ID of the notation (an XMLExternalID instance)

Syntax

White Space Handling

pyslet.xml.structures.is_s(c)

    Tests production [3] S

    Optimized for speed as this function is called a lot by the parser.

pyslet.xml.structures.collapse_space(data, smode=True, stest=<function is_s>)

    Returns data with all spaces collapsed to a single space.

    smode  Determines the fate of any leading space, by default it is True and leading spaces are ignored provided the string has some non-space characters.

    stest  You can override the test of what constitutes a space by passing a function for stest, by default we use is_s() and any value passed to stest should behave similarly.

    Note on degenerate case: this function is intended to be called with non-empty strings and will never return an empty string. If there is no data then a single space is returned (regardless of smode).

Names

pyslet.xml.structures.is_name_start_char(c)

    Tests if the character c matches production [4] NameStartChar.

pyslet.xml.structures.is_name_char(c)

    Tests production [4a] NameChar

pyslet.xml.structures.is_valid_name(name)

    Tests if name is a string matching production [5] Name

pyslet.xml.structures.is_reserved_name(name)

    Tests if name is reserved

    Names beginning with ‘xml’ are reserved for future standardization
Character Data and Markup

```
    escape_char_data(src, quote=False)
```

Returns a unicode string with XML reserved characters escaped. We also escape return characters to prevent them being ignored. If quote is True then the string is returned as a quoted attribute value.

```
    escape_char_data7(src, quote=False)
```

Escapes reserved and non-ASCII characters.

```
    src A character string
    quote (defaults to False) When True, will surround the output in either single or double quotes (preferred) depending on the contents of src.
```

Characters outside the ASCII range are replaced with character references.

CDATA Sections

```
    CDATA_START = u'<![CDATA[
    CDATA_END = u']]>'
```

character string constant for “<![CDATA[“ character string constant for “[>]”

```
    escape_cdsect(src)
    src A character string of data
```

Wraps a string in a CDATA section

Returns a character string enclosed in <![CDATA[ ]]> with ]]> replaced by the clumsy sequence: ]]>]]&gt;<![CDATA[

Degenerate case: an empty string is returned as an empty string

Exceptions

```
    class XMLLError
        Bases: exceptions.Exception
        Base class for all exceptions raised by this module.
    class XMLValidityError
        Bases: pyxlet.xml.structures.XMLLError
        Base class for all validation errors
```

Raised when a document or content model violates a validity constraint. These errors can be generated by the parser (for example, when validating a document against a declared DTD) or by Elements themselves when content is encountered that does not fit content model expected.

```
    class XMLIDClashError
        Bases: pyxlet.xml.structures.XMLValidityError
        A validity error caused by two elements with the same ID
    class XMLIDValueError
        Bases: pyxlet.xml.structures.XMLValidityError
        A validity error caused by an element with an invalid ID
```
ID attribute must satisfy the production for NAME.

```python
class pyslet.xml.structures.DuplicateXMLNAME
    Bases: pyslet.xml.structures.XMLError
    Raised by map_class_elements()
    Indicates an attempt to declare two classes with the same XML name.
```

```python
class pyslet.xml.structures.XMLAttributeSetter
    Bases: pyslet.xml.structures.XMLError
    Raised when a badly formed attribute mapping is found.
```

```python
class pyslet.xml.structures.XMLMixedContentError
    Bases: pyslet.xml.structures.XMLError
    Raised by Element.get_value()
    Indicates unexpected element children.
```

```python
class pyslet.xml.structures.XMLParentError
    Bases: pyslet.xml.structures.XMLError
    Raised by Element.attach_to_parent()
    Indicates that the element was not an orphan.
```

```python
class pyslet.xml.structures.XMLUnknownChild
    Bases: pyslet.xml.structures.XMLError
    Raised by Element.remove_child()
    Indicates that the child being removed was not found in the element’s content.
```

### 6.2.3 XML: Namespaces

#### Documents and Elements

```python
class pyslet.xml.namespace.NSNode
    Bases: pyslet.xml.structures.Node
    Base class for NSElement and Document shared attributes.
```

This class adds a number of method for managing the mapping between namespace prefixes and namespace URIs in both elements and in the document itself.

You don’t have to worry about using these, they are called automatically enabling the transparent serialisation of XML elements with appropriately defined namespace prefixes. You only need to use these method if you wish to customise the way the mapping is done. The most likely use case is simply to call `make_prefix()` at the document level to add an explicit declaration of any auxiliary namespaces, typically done by the `__init__` method on classes derived from NSDocument.

#### get_prefix(ns)

Returns the prefix assigned to a namespace

- **ns** The namespace URI as a character string.

Returns None if no prefix is currently in force for this namespace.

#### get_ns(prefix=“”)

Returns the namespace associated with prefix.

- **prefix** The prefix to search for, the empty string denotes the default namespace.
This method searches back through the hierarchy until it finds the namespace in force or returns None if no definition for this prefix can be found.

In the special case of prefix being ‘xml’ the XML namespace itself is returned. See XML_NAMESPACE.

```python
new_prefix (stem='ns')
Return an unused prefix

stem  The returned value will be of the form stem# where # is a number used in sequence starting with 1.
This argument defaults to ns so, by default, the prefixes ns1, ns2, ns3, etc. are used.
```

```python
make_prefix (ns, prefix=None)
Creates a new mapping for a namespace

ns  The namespace being mapped
prefix  The character string representing the namespace in qualified names (without the colon). This
parameter is optional, if no value is provided then a new randomly generated prefix is used using
new_prefix().

Note that an empty string denotes the default namespace, which will appear simply as xmlns=<ns> in
the element’s tag.
```

If the prefix has already been declared for this node then ValueError is raised.

```python
get_prefix_map ()
Returns the complete prefix to ns mapping in force

Combines the prefix mapping for this element with that of it’s parents. Returns a dictionary mapping prefix
strings to the URIs of the namespaces they represent.
```

```python
write_nsattrs (attributes, escape_function=<function escape_char_data>, root=False, **kws)
Adds strings representing any namespace attributes

See write_xml_attributes() for details of argument usage.
```

This method is defined for both NSDocument and NSElement and it prefixes the attribute list with any
XML namespace declarations that are defined by this node. If root is True then all namespace declarations
that are in force are written, not just those attached to this node. See get_prefix_map() for more
information.

```python
GetPrefix (*args, **kwargs)
Deprecated equivalent to get_prefix()
```

```python
MakePrefix (*args, **kwargs)
Deprecated equivalent to make_prefix()
```

```python
class pyslet.xml.namespace.NSDocument (root=None, base_uri=None, req_manager=None, **kws)
```

```python
default_ns = None
The default namespace for this document class
```

A special class attribute used to set the default namespace for elements created within the document that
are parsed without an effective namespace declaration. Set to None, but typically overridden by derived
classes.

```python
XMLParser (entity)
Namespace documents use the special XMLNSParser.
```

```python
classmethod get_element_class (name)
Returns a class object suitable for representing <name>
```

name is a tuple of (namespace, name), this overrides the behaviour of Document, in which name is a string.
The default implementation returns NSElement.

class pyslet.xml.namespace.NSElement (parent, name=\texttt{None})

Bases: pyslet.xml.namespace.NSNode, pyslet.xml.structures.Element

Element class used for namespace-aware elements.

Namespace aware elements have special handling for elements that contain namespace declarations and for handling qualified names. A qualified name is a name that starts with a namespace prefix followed by a colon, for example “md:name” might represent the ‘name’ element in a particular namespace indicated by the prefix ‘md’.

The same element could equally be encountered with a different prefix depending on the namespace declarations in the document. As a result, to interpret element (and attribute) names they must be expanded.

An expanded name is represented as a 2-tuple consisting of two character strings, the first is a URI of a namespace (used only as an identifier, the URI does not have to be the URI of an actual resource). The second item is the element name defined within that namespace.

In general, when dealing with classes derived from NSElement you should use expanded names wherever you would normally use a plain character string. For example, the class attribute XMLNAME, used by derived classes to indicate the default name to use for the element the class represents must be an expanded name:

\begin{verbatim}
class MyElement (NSElement):
    XMLNAME = (\texttt{'http://www.example.com/namespace'}, \texttt{'MyElement'})
\end{verbatim}

Custom attribute mappings use special class attributes with names starting with XMLATTR, and this mechanism cannot be extended to use the expanded names. As a result these mappings can only be used for attributes that have no namespace. In practice this is not a significant limitation as attributes are usually defined this way in XML documents. Note that the special XML attributes (which appear to be in the namespace implicitly declared by the prefix “xml:”) should be referenced using the special purpose get/set methods provided.

\begin{verbatim}
set_xmlname (name)
    Sets the name of this element

    Overridden to support setting the name from either an expanded name or an unqualified name (in which case the namespace is set to None).

get_xmlname ()
    Returns the name of this element

    For classes derived from NSElement this is always an expanded name (even if the first component is None, indicating that the namespace is not known).

classmethod mangle_aname (name)
    Returns a mangled attribute name

    Custom setters are enabled only for attributes with no namespace. For attributes from other namespaces the default processing defined by the Element’s set_attribute/get_attribute(s) implementation is used.

classmethod unmangle_aname (mname)
    Overridden to return an expanded name.

    Custom attribute mappings are only supported for attributes with no namespace.

set_attribute (name, value)
    Sets the value of an attribute.

    Overridden to allow attributes to be set using either expanded names (2-tuples) or unqualified names (character strings).
\end{verbatim}
Implementation notes: for elements descended from NSElement all attributes are stored using expanded names internally. The method `unmangle_name()` is overridden to return a 2-tuple to make their 'no namespace' designation explicit.

This method also catches the new namespace prefix mapping for the element which is placed in a special attribute by `XMLNSParser.parse_nsattrs()` and updates the element's namespace mappings accordingly.

```python
get_attribute(name)
```

Gets the value of an attribute.

Overridden to allow attributes to be got using either expanded names (2-tuples) or unqualified names (character strings).

```python
pyslet.xml.namespace.map_class_elements(class_map, scope, ns_alias_table=None)
```

Adds element name -> class mappings to class_map

- **class_map** A dictionary that maps XML element expanded names onto class objects that should be used to represent them.
- **scope** A dictionary, or an object containing a `__dict__` attribute, that will be scanned for class objects to add to the mapping. This enables scope to be a module. The search is not recursive, to add class elements from imported modules you must call `map_class_elements` for each module.
- **ns_alias_table** Used to create multiple mappings for selected element classes based on namespace aliases. It is a dictionary mapping a canonical namespace to a list of aliases. For example, if:

```python
```

An element class with:

```python
XMLNAME = ('http://www.example.com/schema-v3', 'data')
```

would then be used by the parser to represent the `<data>` element in the v1, v2 and v3 schema variants.

The scope is searched for classes derived from `NSElement` that have an XMLNAME attribute defined. It is an error if a class is found with an XMLNAME that has already been mapped.

**Backwards Compatibility**

```python
class pyslet.xml.namespace.XMLNSDocument
```

Alias for `NSDocument`

```python
class pyslet.xml.namespace.XMLNSElement
```

Alias for `NSElement`

**Namespace URIs**

```python
pyslet.xml.namespace.XML_NAMESPACE = 'http://www.w3.org/XML/1998/namespace'
```

URI string constant for the special XML namespace

```python
pyslet.xml.namespace.XMLNS_NAMESPACE = 'http://www.w3.org/2000/xmlns/'
```

URI string constant for the special XMLNS namespace

```python
pyslet.xml.namespace.NO_NAMESPACE = '~'
```

Special string constant used to represent no namespace
pyslet.xml.namespace.match_expanded_names(xname, xmatch, ns_aliases=None)

Compares two expanded names

xname, xmatch  Expanded names, i.e., 2-tuples of character strings containing (namespace URI, element name).

ns_aliases Used to match multiple names based on namespace aliases. It is a list of namespaces that should be treated as equivalent to the namespace of xname. For example:

```python
match_expanded_names(
    ('http://www.example.com/schema-v3', 'data'),
    ('http://www.example.com/schema-v1', 'data'),
    ['http://www.example.com/schema-v2',
     'http://www.example.com/schema-v1'])
```

returns True as xmatch uses an allowed alias for the namespace of xname.

Parsing

pyslet.xml.namespace.is_valid_ncname(name)

Checks a string against NCName

class pyslet.xml.namespace.XMLNSParser(entity=None)

    Bases: pyslet.xml.parser.XMLParser

    A special parser for parsing documents that may use namespaces.

    classmethod register_nsdoc_class(doc_class, xname)

        Registers a document class

        Internally XMLNSParser maintains a single table of document classes which can be used to identify the correct class to use to represent a document based on the expanded name of the root element.

        doc_class  the class object being registered, it must be derived from NSDocument

        xname  A tuple of (namespace, name) representing the name of the root element. If either (or both) components are None a wildcard is registered that will match any corresponding value.

    get_nsdoc_class(xname)

        Returns a doc class object suitable for this root element

        xname  An expanded name.

        Returns a class object derived from NSDocument suitable for representing a document with a root element with the given expanded name.

        This default implementation uses xname to locate a class registered with register_nsdoc_class(). If an exact match is not found then wildcard matches are tried matching only the namespace and root element name in turn.

        A wildcard match is stored in the mapping table either as an expanded name of the form (<uri string>, None) or (None, <element name>). The former is preferred as it enables a document class to be defined that is capable of representing a document with any root element from the given namespace (a common use case) and is thus always tried first.

        If no document class can be found, NSDocument is returned.

    expand_qname(qname, ns_defs, use_default=True)

        Expands a QName, returning a (namespace, name) tuple.

        qname  The qualified name
**ns_defs** A mapping of prefix to namespace URI used to expand the name

If *ns_defs* does not contain a suitable namespace definition then the context’s existing prefix mapping is used, then its parent’s mapping is used, and so on.

**use_default (defaults to True)** Whether or not to return the default namespace for an unqualified name.

If *use_default* is False an unqualified name is returned with *NO_NAMESPACE* as the namespace (this is used when expanding attribute names).

**match_xml_name** (*element, qname*)

Tests if *qname* is a possible name for this element.

This method is used by the parser to determine if an end tag is the end tag of this element.

**parse_nsattrs** (*attrs*)

Manages namespace prefix mappings

Takes a dictionary of attributes as returned by *parse_stag()* and finds any namespace prefix mappings returning them as a dictionary of prefix:namespace suitable for passing to *expand_qname()* It also removes the namespace declarations from *attrs* and expands the attribute names into (ns, name) pairs.

Implementation note: a special attribute called ‘.ns’ (in no namespace) is set to the parsed prefix mapping dictionary enabling the prefix mapping to be passed transparently to *NSElement.set_attribute()* by py:class:*XMLParser*.

**get_stag_class** (*qname, attrs=None*)

[40] STag

Overridden to allow for namespace handling.

### Exceptions

**class** *pyslet.xml.namespace.XMLNSError*

Bases: *pyslet.xml.parser.XMLFatalError*

Raised when an illegal QName is found.

### 6.2.4 XML: Parsing XML Documents

This module exposes a number of internal functions typically defined privately in XML parser implementations which make it easier to reuse concepts from XML in other modules. For example, the IsNameStartChar() tells you if a character matches the production for NameStartChar in the XML standard.

**class** *pyslet.xml.parser.XMLParser* (*entity*)

Bases: *pyslet.pep8.PEP8Compatibility*

An XMLParser object

**entity** The *XMLEntity* to parse.

XMLParser objects are used to parse entities for the constructs defined by the numbered productions in the XML specification.

XMLParser has a number of optional attributes, all of which default to False. Attributes with names started ‘check’ increase the strictness of the parser. All other parser flags, if set to True, will not result in a conforming XML processor.

**classmethod register_doc_class** (*doc_class, root_name, public_id=None, system_id=None*)

Registers a document class
Internally XMLParser maintains a single table of document classes which can be used to identify the correct class to use to represent a document based on the information obtained from the DTD.

doc_class  the class object being registered, it must be derived from Document

root_name  the name of the root element or None if this class can be used with any root element.

public_id  the optional public ID of the doctype, if None or omitted any doctype can be used with this document class.

system_id  the optional system ID of the doctype, if None or omitted (the usual case) the document class can match any system ID.

RefModeNone = 0
Default constant used for setting refMode

RefModeInContent = 1
Treat references as per “in Content” rules

RefModeInAttributeValue = 2
Treat references as per “in Attribute Value” rules

RefModeAsAttributeValue = 3
Treat references as per “as Attribute Value” rules

RefModeInEntityValue = 4
Treat references as per “in EntityValue” rules

RefModeInDTD = 5
Treat references as per “in DTD” rules

PredefinedEntities = {'amp': '&amp;', 'lt': '&lt;', 'gt': '&gt;', 'apos': '&#039;', 'quot': '&#34;'}
A mapping from the names of the predefined entities (lt, gt, amp, apos, quot) to their replacement characters.

check_validity = None
Checks XML validity constraints

If check_validity is True, and all other options are left at their default (False) setting then the parser will behave as a validating XML parser.

close_external_entities = None
whether or not to open external entities

close_remote_entities = None
whether or not to open remote entities (i.e., via http(s)) requires close_external_entities to be True

valid = None
Flag indicating if the document is valid, only set if check_validity is True

nonFatalErrors = None
A list of non-fatal errors discovered during parsing, only populated if check_validity is True

checkCompatibility = None
checks XML compatibility constraints; will cause check_validity to be set to True when parsing

checkAllErrors = None
checks all constraints; will cause check_validity and checkCompatibility to be set to True when parsing.

raiseValidityErrors = None
 treats validity errors as fatal errors
dont_check_wellformedness = None
provides a loose parser for XML-like documents

unicodeCompatibility = None
See http://www.w3.org/TR/unicode-xml/

sgml_namecase_general = None
Option that simulates SGML’s NAMECASE GENERAL YES
Defaults to False for XML behaviour. When True, literals within the document are treated as case insen- sitive. Although the SGML specification refers to names being folded to uppercase, we actually fold to lower-case internally in keeping with XML common practice.

Therefore, an attribute called ‘NAME’ will be treated as if it had been called ‘name’ in the document.

sgml_omittag = None
Option that simulates SGML’s OMITTAG YES
With this option the parser will call structures.Element.get_child_class() to determine if an element indicates a missing start or end tag.

sgml_shorttag = None
This option simulates the special attribute handling of the SGML shorttag feature. If an attribute is declared without a value:

```xml
<section title="Notes to Editor" hidden>
```
then the tag is treated as if it had been written:

```xml
<section title="Notes to Editor" hidden="hidden">
```
In most cases this enables simple attribute mappings to be used, even if there are multiple possible tokens permissible, for example:

```python
class Book(Element):
    XMLATTR_hidden = 'visible'
    XMLATTR_shown = 'visible'
```

Will result in the instance attribute visible being set to either ‘hidden’ or ‘shown’ even though the attribute name is minimized away with use of the shorttag feature. This technique is used extensively in HTML where many attributes are declared using single-token #IMPLIED form, such as the disabled attribute of INPUT:

```xml
<INPUT disabled (disabled) #IMPLIED
```

sgml_content = None
This option simulates some aspects of SGML content handling based on class attributes of the element being parsed.

Element classes with XMLCONTENT=:py:dat XMLEmpty are treated as elements declared EMPTY, these elements are treated as if they were introduced with an empty element tag even if they weren’t, as per SGML’s rules. Note that this SGML feature “has nothing to do with markup minimization” (i.e., sgml_omittag.)

refMode = None
The current parser mode for interpreting references.

XML documents can contain five different types of reference: parameter entity, internal general entity, external parsed entity, (external) unparsed entity and character entity.
The rules for interpreting these references vary depending on the current mode of the parser, for example, in content a reference to an internal entity is replaced, but in the definition of an entity value it is not. This means that the behaviour of the `parse_reference()` method will differ depending on the mode.

The parser takes care of setting the mode automatically but if you wish to use some of the parsing methods in isolation to parse fragments of XML documents, then you will need to set the `refMode` directly using one of the RefMode* family of constants defined above.

```python
entity = None
    The current entity being parsed
the_char = None
    the current character; None indicates end of stream
declaration = None
    The declaration being parsed or None
dtd = None
    The document type declaration of the document being parsed. This member is initialised to None as well-formed XML documents are not required to have an associated dtd.
doc = None
    The document being parsed
docEntity = None
    The document entity
element = None
    The current element being parsed
elementType = None
    The element type of the current element
get_context()
    Returns the parser’s context
    This is either the current element or the document if no element is being parsed.
next_char()
    Moves to the next character in the stream.
    The current character can always be read from `the_char`. If there are no characters left in the current entity then entities are popped from an internal entity stack automatically.
buff_text(unused_chars)
    Buffers characters that have already been parsed.
    `unused_chars` A string of characters to be pushed back to the parser in the order in which they are to be parsed.
    This method enables characters to be pushed back into the parser forcing them to be parsed next. The current character is saved and will be parsed (again) once the buffer is exhausted.
push_entity(entity)
    Starts parsing an entity
    `entity` An `XMLEntity` instance which is to be parsed.
    `the_char` is set to the current character in the entity’s stream. The current entity is pushed onto an internal stack and will be resumed when this entity has been parsed completely.
    Note that in the degenerate case where the entity being pushed is empty (or is already positioned at the end of the file) then push_entity does nothing.
```
check_encoding(entity, declared_encoding)
Checks the entity against the declared encoding

entity An XMLEntity instance which is being parsed.

declared_encoding A string containing the declared encoding in any declaration or None if there was no
declared encoding in the entity.

get_external_entity()
Returns the external entity currently being parsed.

If no external entity is being parsed then None is returned.

standalone()
True if the document should be treated as standalone.

A document may be declared standalone or it may effectively be standalone due to the absence of a DTD,
or the absence of an external DTD subset and parameter entity references.

declared_standalone()
True if the current document was declared standalone.

well_formedness_error(msg='well-formedness error', error_class=<class
'pyslet.xml.parser.XMLWellFormedError'>)
 Raises an XMLWellFormedError error.

msg An optional message string

error_class an optional error class which must be a class object derived from
py:class:XMLWellFormednessError.
Called by the parsing methods whenever a well-formedness constraint is violated.

The method raises an instance of error_class and does not return. This method can be overridden by
derived parsers to implement more sophisticated error logging.

validity_error(msg='validity error', error=<class 'pyslet.xml.structures.XMLValidityError'>)
Called when the parser encounters a validity error.

msg An optional message string

error An optional error class or instance which must be a (class) object derived from
py:class:XMLValidityError.

The behaviour varies depending on the setting of the check_validity and
raiseValidityErrors options. The default (both False) causes validity errors to be ignored.
When checking validity an error message is logged to nonFatalErrors and valid is set to False.
Furthermore, if raiseValidityErrors is True error is raised (or a new instance of error is raised)
and parsing terminates.

This method can be overridden by derived parsers to implement more sophisticated error logging.

compatibility_error(msg='compatibility error')
Called when the parser encounters a compatibility error.

msg An optional message string

The behaviour varies depending on the setting of the checkCompatibility flag. The default (False)
causes compatibility errors to be ignored. When checking compatibility an error message is logged to
nonFatalErrors.

This method can be overridden by derived parsers to implement more sophisticated error logging.

processing_error(msg='Processing error')
Called when the parser encounters a general processing error.
msg  An optional message string

The behaviour varies depending on the setting of the checkAllErrors flag. The default (False) causes processing errors to be ignored. When checking all errors an error message is logged to nonFatalErrors.

This method can be overridden by derived parsers to implement more sophisticated error logging.

**parse_literal** *(match)*

Parses an optional literal string.

  **match**  The literal string to match

  Returns True if match is successfully parsed and False otherwise. There is no partial matching, if match is not found then the parser is left in its original position.

**parse_required_literal** *(match, production='Literal String')*

Parses a required literal string.

  **match**  The literal string to match

  **production**  An optional string describing the context in which the literal was expected.

  There is no return value. If the literal is not matched a wellformed error is generated.

**parse_decimal_digits** ()

Parses a, possibly empty, string of decimal digits.

  Decimal digits match [0-9]. Returns the parsed digits as a string or an empty string if no digits were matched.

**parse_required_decimal_digits** *(production='Digits')*

Parses a required string of decimal digits.

  **production**  An optional string describing the context in which the decimal digits were expected.

  Decimal digits match [0-9]. Returns the parsed digits as a string.

**parse_hex_digits** ()

Parses a, possibly empty, string of hexadecimal digits

  Hex digits match [0-9a-fA-F]. Returns the parsed digits as a string or an empty string if no digits were matched.

**parse_required_hex_digits** *(production='Hex Digits')*

Parses a required string of hexadecimal digits.

  **production**  An optional string describing the context in which the hexadecimal digits were expected.

  Hex digits match [0-9a-fA-F]. Returns the parsed digits as a string.

**parse_quote** *(q=None)*

Parses the quote character

  **q**  An optional character to parse as if it were a quote. By default either one of ‘”’ or ‘’’ is accepted.

  Returns the character parsed or raises a well formed error.

**parse_document** *(doc=None)*


  **doc**  The Document instance that will be parsed. The declaration, dtd and elements are added to this document. If doc is None then a new instance is created using get_document_class() to identify the correct class to use to represent the document based on information in the prolog or, if the prolog lacks a declaration, the root element.
This method returns the document that was parsed, an instance of Document.

get_document_class (dtd)
Returns a class object suitable for this dtd

dtd A XMLDTD instance

Returns a class object derived from Document suitable for representing a document with the given doc-
ument type declaration.

In cases where no doctype declaration is made a dummy declaration is created based on the name of the
root element. For example, if the root element is called “database” then the dtd is treated as if it was
declared as follows:

```xml
<!DOCTYPE database>
```

This default implementation uses the following three pieces of information to locate a class registered with
register_doc_class(). The PublicID, SystemID and the name of the root element. If an exact
match is not found then wildcard matches are attempted, ignoring the SystemID, PublicID and finally the
root element in turn. If a document class still cannot be found then wildcard matches are tried matching
only the PublicID, SystemID and root element in turn.

If no document class cab be found, Document is returned.

is_s ()
Tests if the current character matches S

Returns a boolean value, True if S is matched.

By default calls is_s ()

In Unicode compatibility mode the function maps the unicode white space characters at code points 2028
and 2029 to line feed and space respectively.

parse_s ()

Parses white space returning it as a string. If there is no white space at the current position then an empty
string is returned.

The productions in the specification do not make explicit mention of parameter entity references, they are
covered by the general statement that “Parameter entity references are recognized anwhere in the DTD…”
In practice, this means that while parsing the DTD, anywhere that an S is permitted a parameter entity
reference may also be recognized. This method implements this behaviour, recognizing parameter entity
references within S when refMode is RefModeInDTD.

parse_required_s (production='[3] S')

[3] S: Parses required white space

production An optional string describing the production being parsed. This allows more useful errors
than simply ‘expected [3] S’ to be logged.

If there is no white space then a well-formedness error is raised.

parse_name ()

[5] Name

Parses an optional name. The name is returned as a unicode string. If no Name can be parsed then None is
returned.

parse_required_name (production='Name')

[5] Name
production  An optional string describing the production being parsed. This allows more useful errors than simply ‘expected [5] Name’ to be logged.

Parses a required Name, returning it as a string. If no name can be parsed then a well-formed error is raised.

parse_names ()
[6] Names
This method returns a tuple of unicode strings. If no names can be parsed then None is returned.

parse_nmtoken ()
[7] Nmtoken
Returns a Nmtoken as a string or, if no Nmtoken can be parsed then None is returned.

parse_nmtokens ()
[8] Nmtokens
This method returns a tuple of unicode strings. If no tokens can be parsed then None is returned.

parse_entity_value ()
[9] EntityValue
Parses an EntityValue, returning it as a unicode string.
This method automatically expands other parameter entity references but does not expand general or character references.

parse_att_value ()
[10] AttValue
The value is returned without the surrounding quotes and with any references expanded.
The behaviour of this method is affected significantly by the setting of the dont_check_wellformedness flag. When set, attribute values can be parsed without surrounding quotes. For compatibility with SGML these values should match one of the formal value types (e.g., Name) but this is not enforced so values like width=100% can be parsed without error.

parse_system_literal ()
The value of the literal is returned as a string without the enclosing quotes.

parse_pubid_literal ()
[12] PubidLiteral
The value of the literal is returned as a string without the enclosing quotes.

parse_char_data ()
[14] CharData
Parses a run of character data. The method adds the parsed data to the current element. In the default parsing mode it returns None.
When the parser option sgml_omittag is selected the method returns any parsed character data that could not be added to the current element due to a model violation. Note that in this SGML-like mode any S is treated as being in the current element as the violation doesn’t occur until the first non-S character (so any implied start tag is treated as being immediately prior to the first non-S).

parse_comment (got_literal=False)
[15] Comment

got_literal  If True then the method assumes that the ‘<!–’ literal has already been parsed.
Returns the comment as a string.

\textbf{parse_pi} \texttt{(\textit{got\_literal}=False)}

[16] PI: parses a processing instruction.

\textit{got\_literal} If True the method assumes the ‘<?’ literal has already been parsed.

This method calls the \texttt{Node.processing\_instruction()} of the current element or of the docu-
mament if no element has been parsed yet.

\textbf{parse_pi\_target}()

[17] PITarget

Parses a processing instruction target name, the name is returned.

\textbf{parse_cdsect} \texttt{((\textit{got\_literal}=False, \textit{cdend}=u’}]>’)}

[18] CDSect

\textit{got\_literal} If True then the method assumes the initial literal has already been parsed. (By default, CD-

\textit{cdend} Optional string. The literal used to signify the end of the CDATA section can be overridden by

passing an alternative literal in \textit{cdend}. Defaults to ‘}]>’

This method adds any parsed data to the current element, there is no return value.

\textbf{parse_cdstart}()

[19] CDStart

Parses the literal that starts a CDATA section.

\textbf{parse_cdata} \texttt{(\textit{cdend}=u’}]>’)}

[20] CData

Parses a run of CDATA up to but not including \textit{cdend}.

This method adds any parsed data to the current element, there is no return value.

\textbf{parse_cdend}()

[21] CDEnd

Parses the end of a CDATA section.

\textbf{parse_prolog}()

[22] prolog

Parses the document prolog, including the XML declaration and dtd.

\textbf{parse_xml\_decl} \texttt{(\textit{got\_literal}=False)}

[23] XMLDecl

\textit{got\_literal} If True the initial literal ‘<?xml’ is assumed to have already been parsed.

Returns an \texttt{XMLDeclaration} instance. Also, if an encoding is given in the declaration then the method
changes the encoding of the current entity to match. For more information see \texttt{change\_encoding()}.

\textbf{parse_version\_info} \texttt{(\textit{got\_literal}=False)}

[24] VersionInfo

\textit{got\_literal} If True, the method assumes the initial white space and ‘version’ literal has been parsed al-
ready.

The version number is returned as a string.

\textbf{parse\_eq} \texttt{([25] Eq’)}

[25] Eq
**production** An optional string describing the production being parsed. This allows more useful errors than simply ‘expected [25] Eq’ to be logged.

Parses an equal sign, optionally surrounded by white space.

```
parse_version_num()  # [26] VersionNum
```

Parses the XML version number, returning it as a string, e.g., “1.0”.

```
parse_misc()  # [27] Misc
```

This method parses everything that matches the production Misc*.

```
parse_doctypeddecl(got_literal=False)  # [28] doctypeddecl
```

**got_literal** If True, the method assumes the initial ‘<!DOCTYPE’ literal has been parsed already.

This method creates a new instance of `XMLDTD` and assigns it to `py:attr.dtd`, it also returns this instance as the result.

```
parse_decl_sep()  # [28a] DeclSep
```

Parses a declaration separator.

```
parse_int_subset()  # [28b] intSubset
```

Parses an internal subset.

```
parse_markup_decl(got_literal=False)  # [29] markupDecl
```

**got_literal** If True, the method assumes the initial ‘<’ literal has been parsed already.

Returns True if a markupDecl was found, False otherwise.

```
parse_ext_subset()  # [30] extSubset
```

Parses an external subset.

```
parse_ext_subset_decl()  # [31] extSubsetDecl
```

Parses declarations in the external subset.

```
check_pe_between_declarations(check_entity)  # [31] extSubsetDecl
```

**check_entity** A `XMLEntity` object, the entity we should still be parsing.

Checks the well-formedness constraint on use of PEs between declarations.

```
parse_sd_decl(got_literal=False)  # [32] SDDecl
```

**got_literal** If True, the method assumes the initial ‘standalone’ literal has been parsed already.

Returns True if the document should be treated as standalone; False otherwise.

```
parse_element()  # [39] element
```

---

Chapter 6. Other Supporting Standards
The class used to represent the element is determined by calling the `get_element_class()` method of the current document. If there is no document yet then a new document is created automatically (see `parse_document()` for more information).

The element is added as a child of the current element using `Node.add_child()`.

The method returns a boolean value:
- **True** the element was parsed normally
- **False** the element is not allowed in this context

The second case only occurs when the `sgml_omittag` option is in use and it indicates that the content of the enclosing element has ended. The Tag is buffered so that it can be reparsed when the stack of nested `parse_content()` and `parse_element()` calls is unwound to the point where it is allowed by the context.

### `check_attributes(name, attrs)`

Checks `attrs` against the declarations for an element.

- **name** The name of the element
- **attrs** A dictionary of attributes

Adds any omitted defaults to the attribute list. Also, checks the validity of the attributes which may result in values being further normalized as per the rules for collapsing spaces in tokenized values.

### `match_xml_name(element, name)`

Tests if `name` is a possible name for `element`.

- **element** A `Element` instance.
- **name** The name of an end tag, as a string.

This method is used by the parser to determine if an end tag is the end tag of this element. It is provided as a separate method to allow it to be overridden by derived parsers.

The default implementation simply compares `name` with `GetXMLName()`.

### `check_expected_particle(name)`

Checks the validity of element name in the current context.

- **name** The name of the element encountered. An empty string for `name` indicates the enclosing end tag was found.

This method also maintains the position of a pointer into the element’s content model.

### `get_stag_class(name, attrs=None)`

Returns information suitable for starting the element in the current context.

- **name** The name of the element being started
- **attrs** A dictionary of attributes of the element being started

If there is no `Document` instance yet this method assumes that it is being called for the root element and selects an appropriate class based on the contents of the prolog and/or `name`.

When using the `sgml_omittag` option `name` may be None indicating that the method should return information about the element implied by PCDATA in the current context (only called when an attempt to add data to the current context has already failed).

The result is a triple of:
**element_class** the element class that this STag must introduce or None if this STag does not belong (directly or indirectly) in the current context

**element_name** the name of the element (to pass to add_child) or None to use the default

**buff_flag** True indicates an omitted tag and that the triggering STag (i.e., the STag with name name) should be buffered.

**parse_stag()**

STag, EmptyElemTag

This method returns a tuple of (name, attrs, emptyFlag) where:

- **name** the name of the element parsed
- **attrs** a dictionary of attribute values keyed by attribute name
- **emptyFlag** a boolean; True indicates that the tag was an empty element tag.

**parse_attribute()**

Attribute

Returns a tuple of (name, value) where:

- **name** is the name of the attribute or None if sgml_shorttag is True and a short form attribute value was supplied.
- **value** the attribute value.

If **dont_check_wellformedness** is set the parser uses a very generous form of parsing attribute values to accommodate common syntax errors.

**parse_etag(got_literal=False)**

ETag

**got_literal** If True, the method assumes the initial ‘</’ literal has been parsed already.

The method returns the name of the end element parsed.

**parse_content()**

content

The method returns:

- **True** indicates that the content was parsed normally
- **False** indicates that the content contained data or markup not allowed in this context

The second case only occurs when the sgml_ommittag option is in use and it indicates that the enclosing element has ended (i.e., the element’s ETag has been omitted). See py:meth:parse_element for more information.

**handle_data(data, cdata=False)**

content

data A string of data to be handled

cdata If True data is treated as character data (even if it matches the production for S).

Data is handled by calling add_data() even if the data is optional white space.

**unhandled_data(data)**

content

data A string of unhandled data
This method is only called when the `sgml.omittag` option is in use. It processes data that occurs in a context where data is not allowed.

It returns a boolean result:

- **True** the data was consumed by a sub-element (with an omitted start tag)
- **False** the data has been buffered and indicates the end of the current content (an omitted end tag).

```python
parse_empty_elem_tag()  
```

There is no method for parsing empty element tags alone.

This method raises `NotImplementedError`. Instead, you should call `parse_stag()` and examine the result. If it returns False then an empty element was parsed.

```python
parse_element_decl(got_literal=False)  
```

**got_literal** If True, the method assumes that the `<!ELEMENT` literal has already been parsed.

Declares the element type in the `dtd`, (if present). There is no return result.

```python
parse_content_spec(etype)  
```

**etype** An `ElementType` instance.

Sets the `content_type` and `content_model` attributes of `etype`, there is no return value.

```python
parse_children(got_literal=False, group_entity=None)  
```

**got_literal** If True, the method assumes that the initial `(` literal has already been parsed, including any following white space.

**group_entity** An optional `XMLEntity` object. If `got_literal` is True then `group_entity` must be the entity in which the opening `(` was parsed which started the choice group.

The method returns an instance of `XMLContentParticle`.

```python
parse_cp()  
```

Returns an `XMLContentParticle` instance.

```python
parse_choice(first_child=None, group_entity=None)  
```

**first_child** An optional `XMLContentParticle` instance. If present the method assumes that the first particle and any following white space has already been parsed.

**group_entity** An optional `XMLEntity` object. If `first_child` is given then `group_entity` must be the entity in which the opening `(` was parsed which started the choice group.

Returns an `XMLChoiceList` instance.

```python
parse_seq(first_child=None, group_entity=None)  
```

**first_child** An optional `XMLContentParticle` instance. If present the method assumes that the first particle and any following white space has already been parsed. In this case, `group_entity` must be set to the entity which contained the opening `(` literal.

**group_entity** An optional `XMLEntity` object, see above.
parse_mixed (got_literal=False, group_entity=None)

Mixed
got_literal If True, the method assumes that the #PCDATA literal has already been parsed. In this case,
group_entity must be set to the entity which contained the opening ‘(‘ literal.

group_entity An optional XMLEntity object, see above.

Returns an instance of XMLChoiceList with occurrence ZeroOrMore representing the list of elements
that may appear in the mixed content model. If the mixed model contains #PCDATA only the choice list
will be empty.

parse_attlist_decl (got_literal=False)

AttlistDecl
got_literal If True, assumes that the leading ‘<!ATTLIST’ literal has already been parsed.

Declares the attributes in the dtd, (if present). There is no return result.

parse_att_def (got_s=False)

AttDef
got_s If True, the method assumes that the leading S has already been parsed.

Returns an instance of XMLElement.

parse_att_type (a)

AttType
a A required XMLElement instance.

This method sets the TYPE and VALUES fields of a.

Note that, to avoid unnecessary look ahead, this method does not call parse_string_type() or
parse_enumerated_type().

parse_string_type (a)

StringType
a A required XMLElement instance.

This method sets the TYPE and VALUES fields of a.

This method is provided for completeness. It is not called during normal parsing operations.

parse_tokenized_type (a)

TokenizedType
a A required XMLElement instance.

This method sets the TYPE and VALUES fields of a.

parse_enumerated_type (a)

EnumeratedType
a A required XMLElement instance.

This method sets the TYPE and VALUES fields of a.

This method is provided for completeness. It is not called during normal parsing operations.

parse_notation_type (got_literal=False)

NotationType
got_literal If True, assumes that the leading ‘NOTATION’ literal has already been parsed.
Returns a list of strings representing the names of the declared notations being referred to.

**parse_enumeration()**

[59] Enumeration

Returns a dictionary of strings representing the tokens in the enumeration.

**parse_default_decl(a)**

[60] DefaultDecl: parses an attribute’s default declaration.

a A required XMLAttributeDefinition instance.

This method sets the PRESENCE and DEFAULTVALUE fields of a.

**parse_conditional_sect(got_literal_entity=None)**

[61] conditionalSect

got_literal_entity An optional XMLEntity object. If given, the method assumes that the initial literal ‘<![' has already been parsed from that entity.

**parse_include_sect(got_literal_entity=None)**

[62] includeSect:

got_literal_entity An optional XMLEntity object. If given, the method assumes that the production, up to and including the keyword ‘INCLUDE’ has already been parsed and that the opening ‘<![' literal was parsed from that entity.

There is no return value.

**parse_ignore_sect(got_literal_entity=None)**

[63] ignoreSect

got_literal_entity An optional XMLEntity object. If given, the method assumes that the production, up to and including the keyword ‘IGNORE’ has already been parsed and that the opening ‘<![' literal was parsed from this entity.

There is no return value.

**parse_ignore_sect_contents()**

[64] ignoreSectContents

Parses the contents of an ignored section. The method returns no data.

**parse_ignore()**

[65] Ignore

Parses a run of characters in an ignored section. This method returns no data.

**parse_char_ref(got_literal=False)**

[66] CharRef

got_literal If True, assumes that the leading ‘&’ literal has already been parsed.

The method returns a unicode string containing the character referred to.

**parse_reference()**

[67] Reference

This method returns any data parsed as a result of the reference. For a character reference this will be the character referred to. For a general entity the data returned will depend on the parsing context. For more information see parse_entity_ref().

**parse_entity_ref(got_literal=False)**

[68] EntityRef

got_literal If True, assumes that the leading ‘&’ literal has already been parsed.
This method returns any data parsed as a result of the reference. For example, if this method is called in a context where entity references are bypassed then the string returned will be the literal characters parsed, e.g., “&ref;”.

If the entity reference is parsed successfully in a context where Entity references are recognized, the reference is looked up according to the rules for validating and non-validating parsers and, if required by the parsing mode, the entity is opened and pushed onto the parser so that parsing continues with the first character of the entity’s replacement text.

A special case is made for the predefined entities. When parsed in a context where entity references are recognized these entities are expanded immediately and the resulting character returned. For example, the entity &amp; returns the ‘&’ character instead of pushing an entity with replacement text ‘&#38;’.

Inclusion of an unescaped & is common so when we are not checking well-formedness we treat ‘&’ not followed by a name as if it were ‘&amp;’. Similarly we are generous about the missing ‘;’.

`lookup_predefined_entity(name)`

Looks up pre-defined entities, e.g., “lt”

This method can be overridden by variant parsers to implement other pre-defined entity tables.

`parse_pe_reference(got_literal=False)`

This method returns any data parsed as a result of the reference. Normally this will be an empty string because the method is typically called in contexts where PEReferences are recognized. However, if this method is called in a context where PEReferences are not recognized the returned string will be the literal characters parsed, e.g., “%ref;”.

If the parameter entity reference is parsed successfully in a context where PEReferences are recognized, the reference is looked up according to the rules for validating and non-validating parsers and, if required by the parsing mode, the entity is opened and pushed onto the parser so that parsing continues with the first character of the entity’s replacement text.

`parse_entity_decl(got_literal=False)`

Returns an instance of either `XMLGeneralEntity` or `XMLParameterEntity` depending on the type of entity parsed.

`parse_ge_decl(got_literal=False)`

Returns an instance of `XMLGeneralEntity`.

`parse_pe_decl(got_literal=False)`

Returns an instance of `XMLParameterEntity`.

`parse_entity_def(ge)`

The general entity being parsed, an `XMLGeneralEntity` instance.

This method sets the `definition` and `notation` fields from the parsed entity definition.
parse_pe_def(pe)
[74] PEDef
pe The parameter entity being parsed, an XMLParameterEntity instance.

This method sets the definition field from the parsed parameter entity definition. There is no return value.

parse_external_id(allow_public_only=False)
[75] ExternalID
allow_public_only An external ID must have a SYSTEM literal, and may have a PUBLIC identifier.
If allow_public_only is True then the method will also allow an external identifier with a PUBLIC
identifier but no SYSTEM literal. In this mode the parser behaves as it would when parsing the
production:
(ExternalID | PublicID) S?

Returns an XMLExternalID instance.

resolve_external_id(external_id, entity=None)
[75] ExternalID: resolves an external ID, returning a URI.
external_id A XMLExternalID instance.
entity An optional XMLEntity instance. Can be used to force the resolution of relative URIs to be
relative to the base of the given entity. If it is None then the currently open external entity (where
available) is used instead.

Returns an instance of pyslet.rfc2396.URI or None if the external ID cannot be resolved.
The default implementation simply calls get_location() with the entity’s base URL and ignores the
public ID. Derived parsers may recognize public identifiers and resolve accordingly.

parse_ndata_decl(got_literal=False)
[76] NDDataDecl
got_literal If True, assumes that the literal ‘NDATA’ has already been parsed.

Returns the name of the notation used by the unparsed entity as a string without the preceding ‘NDATA’
literal.

parse_text_decl(got_literal=False)
[77] TextDecl
got_literal If True, assumes that the literal ‘<?xml’ has already been parsed.

Returns an XMLTextDeclaration instance.

parse_encoding_decl(got_literal=False)
[80] EncodingDecl
got_literal If True, assumes that the literal ‘encoding’ has already been parsed.

Returns the declaration name without the enclosing quotes.

parse_enc_name()
[81] EncName

Returns the encoding name as a string or None if no valid encoding name start character was found.

parse_notation_decl(got_literal=False)
[82] NotationDecl
got_literal If True, assumes that the literal ‘<!NOTATION’ has already been parsed.
Declares the notation in the `dtd`, (if present). There is no return result.

```python
parse_public_id()
```

```plaintext
[83] PublicID
The literal string is returned without the PUBLIC prefix or the enclosing quotes.
```

```python
class pyslet.xml.parser.ContentParticleCursor(element_type)
Bases: object
Used to traverse an element’s content model.

The cursor records its position within the content model by recording the list of particles that may represent the current child element. When the next start tag is found the particles’ maps are used to change the position of the cursor. The end of the content model is represented by a special entry that maps the empty string to None.

If a start tag is found that doesn’t have an entry in any of the particles’ maps then the document is not valid.

Note that this cursor is tolerant of non-deterministic models as it keeps track of all possible matching particles within the model.

```plaintext
START_STATE = 0
State constant representing the start state
PARTICLE_STATE = 1
State constant representing a particle
END_STATE = 2
State constant representing the end state
```

```python
next(name=“"
Called when a child element with name is encountered.

Returns True if name is a valid element and advances the model. If name is not valid then it returns False and the cursor is unchanged.

```plaintext
expected()
Sorted list of valid element names in the current state.

If the closing tag is valid it appends a representation of the closing tag too, e.g., </element>. If the cursor is in the end state an empty list is returned.
```

**Character Classes**

The standard defines a number of character classes (see `pyslet.unicode5.CharClass`) to assist with the parsing of XML documents.

The bound test method of each class is exposed for convenience (you don’t need to pass an instance). These pseudo-functions therefore all take a single character as an argument and return `True` if the character matches the class. They will also accept `None` and return `False` in that case.

```python
pyslet.xml.parser.is_char(c)
Tests production for [2] Char
```

This test will be limited on systems with narrow python builds.

```python
pyslet.xml.parser.is_discouraged(c)
Tests if a character is discouraged in the specification.
```

Note that this test will be limited by the range of unicode characters in narrow python builds.

```python
pyslet.xml.parser.is_pubid_char(c)
```
pyset.xml.parser.\texttt{is\_enc\_name}(c)
Tests the second part of production [81] EncName

pyset.xml.parser.\texttt{is\_enc\_name\_start}(c)
Tests the first character of production [81] EncName

pyset.xml.parser.\texttt{is\_letter}(c)
Tests production [84] Letter.

pyset.xml.parser.\texttt{is\_base\_char}(c)
Tests production [85] BaseChar.

pyset.xml.parser.\texttt{is\_ideographic}(c)
Tests production [86] Ideographic.

pyset.xml.parser.\texttt{is\_combining\_char}(c)
Tests production [87] CombiningChar.

pyset.xml.parser.\texttt{is\_digit}(c)
Tests production [88] Digit.

pyset.xml.parser.\texttt{is\_extender}(c)
Tests production [89] Extender.

\textbf{Misc Functions}

pyset.xml.parser.\texttt{is\_white\_space}(data)
Tests if every character in data matches S

pyset.xml.parser.\texttt{contains\_s}(data)
Tests if data contains any S characters

pyset.xml.parser.\texttt{strip\_leading\_s}(data)
Returns data with all leading S removed.

pyset.xml.parser.\texttt{normalize\_space}(data)
Implements attribute value normalization

Returns data normalized according to the further processing rules for attribute-value normalization:

“...by discarding any leading and trailing space (#x20) characters, and by replacing sequences of space (#x20) characters by a single space (#x20) character”

pyset.xml.parser.\texttt{is\_valid\_nmtoken}(nm\_token)
Tests if nm_token is a string matching production [5] Nmtoken

\textbf{Exceptions}

class \texttt{pyslet.xml.parser.XMLFatalError}
Bases: \texttt{pyslet.xml.structures.XMLError}
Raised by a fatal error in the parser.

class \texttt{pyslet.xml.parser.XMLWellFormedError}
Bases: \texttt{pyslet.xml.parser.XMLFatalError}
Raised by when a well-formedness error is encountered.

class \texttt{pyslet.xml.parser.XMLForbiddenEntityReference}
Bases: \texttt{pyslet.xml.parser.XMLFatalError}
Raised when a forbidden entity reference is encountered.
Misc

pyslet.xml.parser.parse_xml_class(class_def_str)

Creates a CharClass from a XML-style definition

The purpose of this function is to provide a convenience for creating character class definitions from the XML specification documents. The format of those declarations is along these lines (this is the definition for Char):

```
#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] |
    [#x10000-#x10FFFF]
```

We parse strings in this format into a `pyslet.unicode5.CharClass` instance returning it as the result:

```python
>>> from pyslet.xml import structures as xml
>>> xml.parse_xml_class("#x9 | #xA | #xD | [#x20-#xD7FF] |
    [#xE000-#xFFFD] | [#x10000-#x10FFFF]")
WARNING:root:Warning: character range outside narrow python build
(10000-10FFFF)
CharClass((	', '
'), '', (' ', '퟿'),
    ('\ue000', '\ufffd'))
```

The built-in function `repr` can be used to print a representation suitable for copy-pasting into Python code.

6.2.5 XML: Schema Datatypes

This module implements some useful concepts drawn from http://www.w3.org/TR/xmlschema-2/

One of the main purposes of this module is to provide classes and functions for converting data between python-native representations of the value-spaces defined by this specification and the lexical representations defined in the specification.

The result is typically a pair of `x_from_str`/`x_to_str` functions that are used to define custom attribute handling in classes that are derived from `Element`. For example:

```python
import xml.xsdatatypes as xsi

class MyElement(XMLElement):
    XMLNAME = "MyElement"
    XMLATTR_flag=('flag', xsi.boolean_from_str, xsi.boolean_to_str)
```

In this example, an element like this:

```xml
<MyElement flag="1">...</MyElement>
```

Would cause the instance of `MyElement` representing this element to have it’s flag attribute set to the Python constant `True` instead of a string value. Also, when serializing the element instance the flag attribute’s value would be converted to the canonical representation, which in this case would be the string “true”. Finally, these functions raise `ValueError` when conversion fails, an error which the XML parser will escalate to an XML validation error (allowing the document to be rejected in strict parsing modes).

Namespace

The XML schema namespace is typically used with the prefix xsi.

```python
pyslet.xml.xsdatatypes.XMLSCHEMA_NAMESPACE = 'http://www.w3.org/2001/XMLSchema-instance'
```

The namespace to use XML schema elements
**Primitive Datatypes**

XML schema’s boolean trivially maps to Python’s True/False

```python
pyslet.xml.xsdatatypes.boolean_from_str(src)
```
Decodes a boolean value from src.

- Returns python constants True or False. As a convenience, if src is None then None is returned.

```python
pyslet.xml.xsdatatypes.boolean_to_str(src)
```
Encodes a boolean using the canonical lexical representation.

- `src` Anything that can be resolved to a boolean except None, which raises ValueError.

The decimal, float and double types are represented by Python’s native float type but the function used to encode and decode them from strings differ from native conversion to adhere more closely to the schema specification and to ensure that, by default, canonical lexical representations are used.

```python
pyslet.xml.xsdatatypes.decimal_from_str(src)
```
Decodes a decimal from a string returning a python float value.

- If string is not a valid lexical representation of a decimal value then ValueError is raised.

```python
pyslet.xml.xsdatatypes.decimal_to_str(value, digits=None, strip_zeros=True, **kws)
```
Encodes a decimal value into a string.

- `digits` You can control the maximum number of digits after the decimal point using `digits` which must be greater than 0 - None indicates no maximum.

- `strip_zeros (aka stripZeros)` This function always returns the canonical representation which means that it will strip trailing zeros in the fractional part. To override this behaviour and return exactly `digits` decimal places set `stripZeros` to False.

```python
pyslet.xml.xsdatatypes.float_from_str(src)
```
Decodes a float value from a string returning a python float.

- The precision of the python float varies depending on the implementation. It typically exceeds the precision of the XML schema `float`. We make no attempt to reduce the precision to that of schema’s float except that we return 0.0 or -0.0 for any value that is smaller than the smallest possible float defined in the specification. (Note that XML schema’s float canonicalizes the representation of zero to remove this subtle distinction but it can be useful to preserve it for numerical operations. Likewise, if we are given a representation that is larger than any valid float we return one of the special float values INF or -INF as appropriate.

```python
pyslet.xml.xsdatatypes.float_to_str(value)
```
Encodes a python float value as a string.

- To reduce the chances of our output being rejected by external applications that are strictly bound to a 32-bit float representation we ensure that we don’t output values that exceed the bounds of float defined by XML schema.

- Therefore, we convert values that are too large to INF and values that are too small to 0.0E0.

```python
pyslet.xml.xsdatatypes.double_from_str(src)
```
Decodes a double value from a string returning a python float.

- The precision of the python float varies depending on the implementation. It may even exceed the precision of the XML schema `double`. The current implementation ignores this distinction.

```python
pyslet.xml.xsdatatypes.double_to_str(value, digits=None, strip_zeros=True, **kws)
```
Encodes a double value returning a character string.

- `digits` Controls the number of digits after the decimal point in the mantissa, None indicates no maximum and the precision of python’s float is used to determine the appropriate number. You may pass the value 0 in
which case no digits are given after the point and the point itself is omitted, but such values are not in their canonical form.

**strip_zeros (aka stripZeros)** determines whether or not trailing zeros are removed, if False then exactly digits digits will be displayed after the point. By default zeros are stripped (except there is always one zero left after the decimal point).

class pyslet.xml.xsdatatypes.Duration(value=None)
    Bases: pyslet.iso8601.Duration

Represents duration values.

Extends the basic iso duration class to include negative durations.

    sign = None
        an integer with the sign of the duration
dateTime values are represented by pyslet.iso8601.TimePoint instances. These functions are provided for convenience in custom attribute mappings.

class pyslet.xml.xsdatatypes.datetime_from_str(src)
    Returns a pyslet.iso8601.TimePoint instance.

class pyslet.xml.xsdatatypes.datetime_to_str(value)
    Returns the canonical lexical representation for dateTime value:
        An instance of pyslet.iso8601.TimePoint

**Derived Datatypes**

dateTime values are represented by pyslet.iso8601.TimePoint instances. These functions are provided for convenience in custom attribute mappings.

Name represents XML Names, the native Python character string is used.

    pyslet.xml.xsdatatypes.name_from_str(src)
        Decodes a name from a string.
        Returns the same string or raises ValueError if src does not match the XML production Name.

    pyslet.xml.xsdatatypes.name_to_str(src)
        Encodes a name
        A convenience function (equivalent to pyslet.py2.to_text()).

Integer is represented by the native Python integer.

    pyslet.xml.xsdatatypes.integer_from_str(src)
        Decodes an integer
        If string is not a valid lexical representation of an integer then ValueError is raised. This uses XML Schema’s lexical rules which are slightly different from Python’s native conversion.

    pyslet.xml.xsdatatypes.integer_to_str(value)
        Encodes an integer value using the canonical lexical representation.
Constraining Facets

Enumeration

class pyslet.xml.xsdatatypes Enumeration
   Bases: pyslet.xml.xsdatatypes EnumBase

Abstract class for defining enumerations

The class is not designed to be instantiated but to act as a method of defining constants to represent the values of an enumeration and for converting between those constants and the appropriate string representations.

The basic usage of this class is to derive a class from it with a single class member called ‘decode’ which is a mapping from canonical strings to simple integers.

Once defined, the class will be automatically populated with a reverse mapping dictionary (called encode) and the enumeration strings will be added as attributes of the class itself. For example:

```python
class Fruit (Enumeration):
   decode = {
      'Apple': 1,
      'Pear': 2,
      'Orange': 3
   }

Fruit.Apple == 1  # True thanks to metaclass
```

You can add define additional mappings by providing a second dictionary called aliases that maps additional names onto existing values. The aliases dictionary is a mapping from strings onto the equivalent canonical string:

```python
class Vegetables (Enumeration):
   decode = {
      'Tomato': 1,
      'Potato': 2,
      'Courgette': 3
   }

   aliases = {
      'Zucchini': 'Courgette'
   }

Vegetables.Zucchini == 3  # True thanks to metaclass
```

You may also add the special key None to the aliases dictionary to define a default value for the enumeration. This is mapped to an attribute called DEFAULT:

```python
class Staples (Enumeration):
   decode = {
      'Bread': 1,
      'Pasta': 2,
      'Rice': 3
   }

   aliases = {
      None: 'Bread'
   }

Staples.DEFAULT == 1  # True thanks to metaclass

DEFAULT = None
   The DEFAULT value of the enumeration defaults to None
classmethod from_str(src)
Decodes a string returning a value in this enumeration.

If no legal value can be decoded then ValueError is raised.

classmethod from_str_lower(src)
Decodes a string, converting it to lower case first.

Returns a value in this enumeration. If no legal value can be decoded then ValueError is raised.

classmethod from_str_upper(src)
Decodes a string, converting it to upper case first.

Returns a value in this enumeration. If no legal value can be decoded then ValueError is raised.

classmethod from_str_title(src)
Decodes a string, converting it to title case first.

Title case is defined as an initial upper case letter with all other letters lower case.

Returns a value in this enumeration. If no legal value can be decoded then ValueError is raised.

classmethod list_from_str(decoder, src)
Decodes a list of values

decoder One of the from_str methods.
src A space-separated string of values

The result is an ordered list of values (possibly containing duplicates).

Example usage:

```
Fruit.list_from_str(Fruit.from_str_title, 
    "apple orange pear")
# returns [ Fruit.Apple, Fruit.Orange, Fruit.Pear ]
```

classmethod dict_from_str(decoder, src)
Decodes a dictionary of values

decoder One of the from_str methods
src A space-separated string of values.

The result is a dictionary mapping the values found as keys onto the strings used to represent them. Duplicates are mapped to the first occurrence of the encoded value.

Example usage:

```
Fruit.dict_from_str(Fruit.from_str_title, 
    "Apple orange PEARS apple")
# returns {Fruit.Apple: 'Apple', Fruit.Orange: 'orange', 
# Fruit.Pear: 'PEARS' }
```

classmethod to_str(value)
Encodes one of the enumeration constants returning a string.

If value is None then the encoded default value is returned (if defined) or None.

classmethod list_to_str(value_list)
Encodes a list of enumeration constants

value_list A list or iterable of integer values corresponding to enumeration constants.

Returns a space-separated string. If valueList is empty then an empty string is returned.
classmethod dict_to_str(value_dict, sort_keys=True, **kws)
Encodes a dictionary of enumeration constants

value_dict  A dictionary with integer keys corresponding to enumeration constant values.
sort_keys  Boolean indicating that the result should be sorted by constant value. (Defaults to True.)

Returns a space-separated string. If value_dict is empty then an empty string is returned. The values in the
dictionary are ignored, the keys are used to obtain the canonical representation of each value. Extending
the example given in dict_from_str():

Fruit.dict_to_str(
     Fruit.Pear: 'PEARS'}
)
# returns: "Apple Pear Orange"

The order of the values in the string is determined by the sort order of the enumeration constants (not their
string representation). This ensures that equivalent dictionaries are always encoded to the same string. In
the above example:

Fruit.Apple < Fruit.Pear < Fruit.Orange

If you have large lists then you can skip the sorting step by passing False for sort_keys to improve perfor-
mance at the expense of an unpredictable encoding.

classmethod DecodeLowerValue(*args, **kwargs)
Deprecated equivalent to from_str_lower()

classmethod DecodeTitleValue(*args, **kwargs)
Deprecated equivalent to from_str_title()

classmethod DecodeUpperValue(*args, **kwargs)
Deprecated equivalent to from_str_upper()

classmethod DecodeValue(*args, **kwargs)
Deprecated equivalent to from_str()

classmethod DecodeValueDict(*args, **kwargs)
Deprecated equivalent to dict_from_str()

classmethod DecodeValueList(*args, **kwargs)
Deprecated equivalent to list_from_str()

classmethod EncodeValue(*args, **kwargs)
Deprecated equivalent to to_str()

classmethod EncodeValueDict(*args, **kwargs)
Deprecated equivalent to dict_to_str()

classmethod EncodeValueList(*args, **kwargs)
Deprecated equivalent to list_to_str()

class pyslet.xml.xsdatatypes EnumerationNoCase
Bases: pyslet.xml.xsdatatypes Enumeration

Convenience class that automatically adds lower-case aliases

On creation, the enumeration ensures that aliases equivalent to the lower-cased canonical strings are defined. De-
dsigned to be used in conjunction with from_str_lower() for case insensitive matching of enumeration
strings.
**WhiteSpace**

pyslet.xml.xsdatatypes.white_space_replace(value)
Replaces tab, line feed and carriage return with space.

pyslet.xml.xsdatatypes.white_space_collapse(value)
Replaces all runs of white space with a single space. Also removes leading and trailing white space.

**Regular Expressions**

Appendix F of the XML Schema datatypes specification defines a regular expression language. This language differs from the native Python regular expression language but it is close enough to enable us to define a wrapper class which parses schema regular expressions and converts them to equivalent python regular expressions.

```python
class pyslet.xml.xsdatatypes.RegularExpression(src):
    Bases: pyslet.py2.UnicodeMixin
    A regular expression as defined by XML schema.
    Regular expressions are constructed from character strings. Internally they are parsed and converted to Python regular expressions to speed up matching.
    Warning: because the XML schema expression language contains concepts not supported by Python the python regular expression may not be very readable.
    
    src = None
    the original source string

    p = None
    the compiled python regular expression

    match(target)
    Returns True if the expression matches target.
```

For completeness we also document the parser we use to do the conversion, it draws heavily on the `pyslet.unicode5.CharClass` concept.

```python
class pyslet.xml.xsdatatypesRegularExpressionParser(source):
    Bases: pyslet.unicode5.BasicParser
    A custom parser for XML schema regular expressions.
    The parser is initialised from a character string and always operates in text mode.

    require_reg_exp()
    Parses a regExp
    Returns a unicode string representing the regular expression.

    require_branch()
    Parses branch
    Returns a character string representing these pieces as a python regular expression.

    require_piece()
    Parses piece
    Returns a character string representing this piece in python regular expression format.

    require_quantifier()
    Parses quantifier
    Returns a tuple of (n, m).
```
Symbolic values are expanded to the appropriate pair. The second value may be None indicating unbounded.

**require_quantity()**

Parses quantity

Returns a tuple of (n, m) even if an exact quantity is given.

In other words, the exact quantity ‘n’ returns (n, n). The second value may be None indicating unbounded.

**require_quant_exact()**

Parses QuantExact

Returns the integer value parsed.

**require_atom()**

Parses atom

Returns a unicode string representing this atom as a python regular expression.

**is_char(c=None)**

Parses Char

Returns either True or False depending on whether the_char satisfies the production Char.

The definition of this function is designed to be conservative with respect to the specification, which is clearly in error around production [10] as the prose and the BNF do not match. It appears that | was intended to be excluded in the prose but has been omitted, the reverse being true for the curly-brackets.

**require_char_class()**

Parses a charClass.

**require_char_class_expr()**

Parses charClassExpr

**require_char_group()**

Parses charGroup.

This method also handles the case of a class subtraction directly to reduce the need for look-ahead. If you specifically want to parse a subtraction you can do this with require_char_class_sub().

**require_pos_char_group()**

Parses posCharGroup

**require_neg_char_group()**

Parses negCharGroup.

**require_char_class_sub()**

Parses charClassSub

This method is not normally used by the parser as in present for completeness. See require_char_group().

**require_char_range()**

Parses a charRange.

**require_se_range()**

Parses seRange.

**require_char_or_esc()**

Parses charOrEsc.

**require_char_class_esc()**

Parses charClassEsc.
Returns a CharClass instance.

```
require_single_char_esc()
```
Parses SingleCharEsc

Returns a single character.

```
require_cat_esc()
```
Parses catEsc.

```
require_compl_esc()
```
Parses complEsc.

```
require_char_prop()
```
Parses a charProp.

```
require_is_category()
```
Parses IsCategory.

```
require_is_block()
```
Parses IsBlock.

```
require_multi_char_esc()
```
Parses a MultiCharEsc.

```
requireWildcard_esc()
```
Parses '\', the wildcard CharClass

---

**Backwards Compatibility**

pylet.xml.xsd.datatypes.DecodeBoolean(*args, **kwargs)

Deprecation note: equivalent to `boolean_from_str()`

pylet.xml.xsd.datatypes.EncodeBoolean(*args, **kwargs)

Deprecation note: equivalent to `boolean_to_str()`

pylet.xml.xsd.datatypes.DecodeDecimal(*args, **kwargs)

Deprecation note: equivalent to `decimal_from_str()`

pylet.xml.xsd.datatypes.EncodeDecimal(*args, **kwargs)

Deprecation note: equivalent to `decimal_to_str()`

pylet.xml.xsd.datatypes.DecodeFloat(*args, **kwargs)

Deprecation note: equivalent to `float_from_str()`

pylet.xml.xsd.datatypes.EncodeFloat(*args, **kwargs)

Deprecation note: equivalent to `float_to_str()`

pylet.xml.xsd.datatypes.DecodeDouble(*args, **kwargs)

Deprecation note: equivalent to `double_from_str()`

pylet.xml.xsd.datatypes.EncodeDouble(*args, **kwargs)

Deprecation note: equivalent to `double_to_str()`

pylet.xml.xsd.datatypes.DecodeDateTime(*args, **kwargs)

Deprecation note: equivalent to `datetime_from_str()`

pylet.xml.xsd.datatypes.EncodeDateTime(*args, **kwargs)

Deprecation note: equivalent to `datetime_to_str()`

pylet.xml.xsd.datatypes.DecodeName(*args, **kwargs)

Deprecation note: equivalent to `name_from_str()`
pyslet.xml.xsdatatypes.EncodeName(*args, **kwargs)
Deprecated equivalent to name_to_str()
pyslet.xml.xsdatatypes.DecodeInteger(*args, **kwargs)
Deprecated equivalent to integer_from_str()
pyslet.xml.xsdatatypes.EncodeInteger(*args, **kwargs)
Deprecated equivalent to integer_to_str()
pyslet.xml.xsdatatypes.make_enum(cls, default_value=None, **kws)
Deprecated function
   This function is no longer required and does nothing unless default_value is passed in which case it adds the
   DEFAULT attribute to the Enumeration cls as if an alias had been declared for None (see Enumeration above
   for details).
pyslet.xml.xsdatatypes.MakeEnumeration(*args, **kwargs)
Deprecated equivalent to make_enum()
pyslet.xml.xsdatatypes.make_enum_aliases(cls, aliases)
Deprecated function
   Supported for backwards compatibility, modify enum class definitions to include aliases as an attribute directly:

   class MyEnum(Enumeration):
       decode = {
           # strings to ints mapping
        }
       aliases = {
           # aliases to strings mapping
        }

   The new metaclass takes care of processing the aliases dictionary when the class is created.
pyslet.xml.xsdatatypes.MakeEnumerationAliases(*args, **kwargs)
Deprecated equivalent to make_enum_aliases()
pyslet.xml.xsdatatypes.make_lower_aliases(cls)
Deprecated function
   Supported for backwards compatibility. Use new class EnumerationNoCase instead.
   Warning, the new class will only add lower-case aliases for the canonical strings, any additional aliases (defined
   in the aliases dictionary attribute) must already be lower-case or be defined with both case variants.
pyslet.xml.xsdatatypes.MakeLowerAliases(*args, **kwargs)
Deprecated equivalent to make_lower_aliases()

6.3 HTML

This module defines functions and classes for working with HTML documents. The version of the standard imple-
mented is, loosely speaking, the HTML 4.0.1 Specification: http://www.w3.org/TR/html401/

This module contains code that can help parse HTML documents into classes based on the basic xml sub-package,
acting as a gateway to XHTML. The module is designed to provide just enough HTML parsing to support the use of
HTML within other standards (such as Atom and QTI).
6.3.1 (X)HTML Documents

The namespace to use in the declaration of an XHTML document:

```python
pyslet.html401.XHTML_NAMESPACE
```

class pyslet.html401.XHTMLDocument(**args)

    Bases: pyslet.xml.namespace.NSDocument

    Represents an HTML document.

    Although HTML documents are not always represented using XML they can be, and therefore we base our
    implementation on the pyslet.xml.namespace.NSDocument class, the namespace-aware variant of the
    basic pyslet.xml.Document class.

    class_map = {('http://www.w3.org/1999/xhtml', 'dl'): <class 'pyslet.html401.DL'>, ('http://www.w3.org/1999/xhtml', ...
        'h1'): <class 'pyslet.html401.H1'>, ('http://www.w3.org/1999/xhtml', 'b'): <class 'pyslet.html401.B'>}

    Data member used to store a mapping from element names to the classes used to represent them. This
    mapping is initialized when the module is loaded.

    default_ns = 'http://www.w3.org/1999/xhtml'

    the default namespace for HTML elements

XMLParser(entity)

    Create a parser suitable for parsing HTML

    We override the basic XML parser to use a custom parser that is intelligent about the use of omitted tags,
    elements defined to have CDATA content and other SGML-based variations. If the document starts with
    an XML declaration then the normal XML parser is used instead.

    You won’t normally need to call this method as it is invoked automatically when you call pyslet.xml.
    Document.read().

    The result is always a proper element hierarchy rooted in an HTML node, even if no tags are present at all
    the parser will construct an HTML document containing a single Div element to hold the parsed text.

Because HTML 4 is an application of SGML, rather than XML, we need to modify the basic XML parser to support
the parsing of HTML documents. This class is used automatically when reading an XHTMLDocument instance from
an entity with declared type that is anything other than text/xml.

class pyslet.html401.HTMLParser(entity=None, **kws)

    Bases: pyslet.xml.namespace.XMLNSParser

    Custom HTML parser

    This variation on the base pyslet.xml.namespace.XMLNSParser does not have to be customised
    much. Most of the hard work is done by the existing mechanisms for inferring missing tags.

lookup_predefined_entity(name)

    Supports HTML entity references

    XML includes only a small number of basic entity references to allow the most basic encoding of docu-
    ments, for example &lt;, &amp;, and so on.

    HTML supports a much larger set of character entity references: https://www.w3.org/TR/html401/sgml/
    entities.html

parse_prolog()

    Custom prolog parsing.

    We override this method to enable us to dynamically set the parser options based on the presence of an
    XML declaration or DOCTYPE.
Strict DTD

The strict DTD describes the subset of HTML that is more compatible with future versions and generally does not include handling of styling directly but encourages the use of external style sheets.

The public ID to use in the declaration of an HTML document:
pyslet.html401.HTML40_PUBLICID

The system ID to use in the declaration of an HTML document:
pyslet.html401.HTML40_TRANSITIONAL_SYSTEMID

Transitional DTD

The transitional DTD, often referred to as the loose DTD contains additional elements and attribute to support backward compatibility. Although this module has been designed to support the the full set of HTML elements attributes that are deprecated and only appear in the loose DTD are not generally mapped to instance attributes in the corresponding classes and where content models differ the classes may enforce (of infer) the stricter model. In particular, there are a number of element that may support both inline and block elements in the loose DTD but which are restricted to block elements in the strict DTD. On reading such a document an implied Div will be used to wrap the inline elements automatically.

The public ID for transitional documents:
pyslet.html401.HTML40_TRANSITIONAL_PUBLICID

The system ID to use in the declaration of transitional documents:
pyslet.html401.HTML40_TRANSITIONAL_SYSTEMID

It should be noted that it is customary to use the strict DTD in the prolog of most HTML documents as this signals to the rendering agent that the document adheres closely to the specification and generally improves the appearance on web pages. However, IFRAMEs are a popular feature of HTML that require use of the transitional DTD. In practice, most content authors ignore this distinction.

Frameset DTD

Although rarely used, there is a third form of HTML in which the body of the document is replaced by a frameset.

The public ID for frameset documents:
pyslet.html401.HTML40_FRAMESET_PUBLICID

The system ID to use in the declaration of frameset documents:
pyslet.html401.HTML40_FRAMESET_SYSTEMID

6.3.2 (X)HTML Elements

All HTML elements are based on the XHTMLElement class. In general, elements have their HTML-declared attributes mapped to similarly names attributes of the instance. A number of special purposes types are defined to assist with attribute value validation making it easier to reuse these concepts in other modules. See Basic Types for more information.

class pyslet.html401.XHTMLMixin
    Bases: object

    An abstract class representing all HTML-like elements.
This class is used to determine if an element should be treated as if it is HTML-like or if it is simply a foreign element from some unknown schema.

HTML-like elements are subject to appropriate HTML content constraints, for example, block elements are not allowed to appear where inline elements are required. Non-HTML-like elements are permitted more freely.

class pyslet.html401.XHTMLElement (parent, name=None)
Bases: pyslet.html401.XHTMLMixin, pyslet.xml.namespace.NSElement
A base class for XHTML elements.

class check_model (child_class)
Checks the validity of adding a child element

class child_class The class of an element to be tested

If an instance of child_class would cause a model violation then XHTMLValidityError is raised. This logic is factored into its own method to allow it to be used by add_child() and get_child_class(), both of which may need to make a determination of the legality of adding a child (in the latter case to determine if an element’s end tag has been omitted).

The default implementation checks the rules for the inclusion of the Ins and Del elements to prevent nesting and to ensure that they only appear within a Body instance.

It checks that a form does not appear within another form.

It also checks that the NOFRAMES element in a frameset document is not being nested.

Generally speaking, derived classes add to this implementation with element-specific rules based on the element’s content model and do not need to override the add_child().

add_child (child_class, name=None)
Overridden to call check_model()

add_to_cpresource (cp, resource, been_there)
See pyslet.imsqtiv2p1.QTIElement.add_to_cpresource()

render_html (parent, profile, arg)
Renders this HTML element to an external document

parent The parent node to attach a copy of this data too.

profile A dictionary mapping the names of allowed HTML elements to a list of allowed attributes. This allows the caller to filter out unwanted elements and attributes on a whitelist basis. Warning: this argument is deprecated.

arg Allows an additional positional argument to be passed through the HTML tree to any non-HTML nodes contained by it. Warning: this argument is deprecated.

The default implementation creates a node under parent if our name is in the profile.

RenderHTML (*args, **kwargs)
Deprecated equivalent to render_html()

RenderText (*args, **kwargs)
Deprecated equivalent to plain_text()

### 6.3.3 Basic Types

The HTML DTD defines parameter entities to make the intention of each attribute declaration clearer. Theses definitions are often translated into the similarly named classes enabling improved validation of attribute values. The special purpose types also make it easier to parse and format information from and to attribute values.
A special note is required for attributes defined using a form like this:

| option  | (option) | #IMPLIED |

These are mapped to the boolean value True or the value None (or False) indicating an absence of the option. The HTML parser allows the SGML markup minimisation feature so these values can be parsed from attribute definitions such as:

```html
<INPUT disabled name="fred" value="stone">
```

All attributes of this form are based on the following abstract class.

```python
class pyslet.html401.NamedBoolean
    Bases: pyslet.pep8.MigratedClass

    An abstract class for named booleans

    This class is designed to make generating SGML-like single-value enumeration types easier, for example, attributes such as “checked” on `<input>`.

    The class is not designed to be instantiated but to act as a method of defining functions for decoding and encoding attribute values.

    The basic usage of this class is to derive a class from it with a single class member called ‘name’ which is the canonical representation of the name. You can then use it to call any of the following class methods to convert values between python Booleans and the appropriate string representations (None for False and the defined name for True).

    **classmethod from_str**(src)
        Decodes a string
        Returning True if it matches the name attribute and raises ValueError otherwise. If src is None then False is returned.

    **classmethod from_str_lower**(src)
        Decodes a string, converting it to lower case first.

    **classmethod from_str_upper**(src)
        Decodes a string, converting it to upper case first.

    **classmethod to_str**(value)
        Encodes a named boolean value
        Returns either the defined name or None.

    **classmethod DecodeLowerValue(**args, **kwargs)**
        Deprecated equivalent to `from_str_lower()`

    **classmethod DecodeUpperValue(**args, **kwargs)**
        Deprecated equivalent to `from_str_upper()`

    **classmethod DecodeValue(**args, **kwargs)**
        Deprecated equivalent to `from_str()`

    **classmethod EncodeValue(**args, **kwargs)**
        Deprecated equivalent to `to_str()`
```

XML attributes generally use space separation for multiple values. In HTML there are a number of attributes that use comma-separation. For convenience that attributes are represented using a special purpose tuple-like class.

```python
class pyslet.html401.CommaList(src)
    Bases: pyslet.py2.UnicodeMixin
```

6.3. HTML
A tuple-like list of strings
Values can be compared with each other for equality, and with strings though the order of items is important.
They can be indexed, iterated and supports the in operator for value testing.

**Align**

Attributes defined:

```xml
<!ENTITY % align "align (left|center|right|justify) #IMPLIED"
   -- default is left for ltr paragraphs, right for rtl --""
```

may be represented using the following class. These attributes are limited to the loose DTD and this class is not used by any of the element classes defined here. It is provided for convenience only.

```python
class pyslet.html401.Align
    Bases: pyslet.xml.xsdatatypes Enumeration
```

**Button type**

The `Button` element defines a type attribute:

```python
type (button|submit|reset) submit -- for use as form button --
```

```python
class pyslet.html401.ButtonType
    Bases: pyslet.xml.xsdatatypes Enumeration
```

Enumeration used for the types allowed for `Button`

```python
ButtonType.DEFAULT == ButtonType.submit
```

**CDATA**

Attributes defined to have CDATA are represented as character strings or, where space separate values are indicated, lists of character strings.

**Character**

Attributes defined to have type %Character:

```xml
<!ENTITY % Character "CDATA" -- a single character from [ISO10646] -->
```

are parsed using the following function:

```python
.. autofunc:: character_from_str
```

**Charset**

Attributes defined to have type %Charset or %Charsets:

```xml
<!ENTITY % Charset "CDATA" -- a character encoding, as per [RFC2045] -->
<!ENTITY % Charsets "CDATA"
   -- a space-separated list of character encodings, as per [RFC2045] -->
```
are left as character strings or lists of character strings respectively.

**Checked**

The *Input* element defines:

```
checked   (checked)  #IMPLIED -- for radio buttons and check boxes --
```

```python
class pyslet.html401.Checked
    Bases: pyslet.html401.NamedBoolean
```

**Clear**

In the loose DTD the *Br* element defines:

```
clear     (left|all|right|none) none  -- control of text flow --
```

The following class is provided as a convenience and is not used in the implementation of the class.

```python
class pyslet.html401.Clear
    Bases: pyslet.xml.xsdatatypes Enumeration
```

**Color**

```python
class pyslet.html401.Color(src)
    Bases: pyslet.py2.UnicodeMixin
```

Class to represent a color value

```xml
<!ENTITY % Color "CDATA"
    -- a color using sRGB: #RRGGBB as Hex values -->
```

Instances can be created using either a string or a 3-tuple of sRGB values. The string is either in the #xxxxxx format for hex sRGB values or it one of the “16 widely known color names” which are matched case insenstively. The canonical representation used when converting back to a character string is the #xxxxxx form.

Color instances can be compared for equality with each other and with character string and are hashable but are not sortable.

For convenience, the standard colors are provided as module-level constants.

As a convenience, pre-instantiated color constants are defined that resolve to pre-initialised instances.

```python
pyslet.html401.BLACK
pyslet.html401.GREEN
pyslet.html401.SILVER
pyslet.html401.LIME
pyslet.html401.GRAY
pyslet.html401.LIME
pyslet.html401.Olive
pyslet.html401.WHITE
pyslet.html401.YELLOW
pyslet.html401.MAROON
```
**ContentType**

Attributes defined to have type `%ContentType` or `%ContentTypes`:

```xml
<!ENTITY % ContentType "CDATA" -- media type, as per [RFC2045] --
<!ENTITY % ContentTypes "CDATA" -- comma-separated list of media types, as per [RFC2045] -->
```

are represented using instances of `pyslet.http.params.MediaType`. The HTTP header convention of comma-separation is used for multiple values (space being a valid character inside a content type with parameters). These are represented using the tuple-like class:

```python
class pyslet.html401.ContentTypes(src)
    Bases: pyslet.py2.UnicodeMixin

    A tuple-like list of `pyslet.http.params.MediaType`

    Values can be compared with each other for equality, and with strings though the order of items is important. They can be indexed, iterated and supports the `in` operator for value testing.
```

**Coordinate Values**

Coordinate values are simple lists of Lengths. In most cases Pyslet doesn’t define special types for lists of basic types but coordinates are represented in attribute values using comma separation, not space-separation. As a result they require special processing in order to be decoded/encoded correctly from/to XML streams.

```python
class pyslet.html401.Cords(values=())
    Bases: pyslet.py2.UnicodeMixin

    Represents HTML Cords values

    <!ENTITY % Cords "CDATA" -- comma-separated list of lengths -->
```

Instances can be initialized from an iterable of `Length` instances or any object that can be used to construct a Length.

The resulting object behaves like a tuple of Length instances, for example:

```python
x=Coords("10, 50, 60%,75%")
len(x) == 4
str(x[3]) == "75%"
```

It supports conversion to string and can be compared with a string directly or with a list or tuple of Length values.
values = None
   a list of Length values

classmethod from_str(src)
   Returns a new instance parsed from a string.
   The string must be formatted as per the HTML attribute definition, using comma-separation of values.

test_rect (x, y, width, height)
   Tests an x,y point against a rect with these coordinates.
   HTML defines the rect co-ordinates as: left-x, top-y, right-x, bottom-y

test_circle (x, y, width, height)
   Tests an x,y point against a circle with these coordinates.
   HTML defines a circle as: center-x, center-y, radius.
   The specification adds the following note:
   When the radius value is a percentage value, user agents should calculate the final radius value
   based on the associated object’s width and height. The radius should be the smaller value of the
   two.

test_poly (x, y, width, height)
   Tests an x,y point against a poly with these coordinates.
   HTML defines a poly as: x1, y1, x2, y2, . . . , xN, yN.
   The specification adds the following note:
   The first x and y coordinate pair and the last should be the same to close the polygon. When these
   coordinate values are not the same, user agents should infer an additional coordinate pair to close
   the polygon.

   The algorithm used is the “Ray Casting” algorithm described here: http://en.wikipedia.org/wiki/Point_in_polygon

The Coords class strays slightly into the territory of a rendering agent by providing co-ordinate testing methods.
There is no intention to extend this module to support HTML rendering, this functionality is provided to support the
server-side evaluation functions in the IMS QTI response processing model.

Declare

class pyslet.html401.Declare
   Bases: pyslet.html401.NamedBoolean
   Used for the declare attribute of Object.

Defer

class pyslet.html401.Defer
   Bases: pyslet.html401.NamedBoolean
   Used for the defer attribute of Script.

Direction

Attributes defined to have type (ltr|rtl) are represented using integer constants from the following Enumeration.
class pyslet.html401.Direction
   Bases: pyslet.xml.xsdatatypes Enumeration

   Enumeration for weak/neutral text values.

Disabled

class pyslet.html401.Disabled
   Bases: pyslet.html401.NamedBoolean

   Used for the disabled attribute of form controls.

Horizontal Cell Alignment

Attributes defined:

align (left|center|right|justify|char) #IMPLIED

are used in table structures for cell alignment.

class pyslet.html401.HAlign
   Bases: pyslet.xml.xsdatatypes Enumeration

   Values horizontal table cell alignment

Image Alignment

The loose DTD supports alignment of images through IAlign:

<!ENTITY % IAlign "(top|middle|bottom|left|right)" -- center? --
   * * *

This class is provided as a convenience and is not used by the classes in this module.

class pyslet.html401.IAlign
   Bases: pyslet.xml.xsdatatypes Enumeration

InputType

The Input class defines a type attribute based on the following definition
(though we prefer lower-case for better XML compatibility):

<!ENTITY % InputType "(TEXT | PASSWORD | CHECKBOX | RADIO |
   SUBMIT | RESET | FILE | HIDDEN | IMAGE | BUTTON)"

class pyslet.html401.InputType
   Bases: pyslet.xml.xsdatatypes Enumeration

   The type of widget needed for an input element

   InputType.DEFAULT == InputType.text
Disabled

class pyslet.html401.IsMap
    Bases: pyslet.html401.NamedBoolean

    Used for the ismap attribute.

LanguageCodes

Attributes defined to have type %LanguageCode:

```xml
<!ENTITY % LanguageCode "NAME" -- a language code, as per [RFC1766] -->
```

are represented as character strings using the functions `name_from_str()` and `name_to_str()`.

Length Values

Attributes defined to have type %Length:

```xml
<!ENTITY % Length "CDATA"
    -- nn for pixels or nn% for percentage length -->
```

are represented using instances of the following class.

class pyslet.html401.Length(value, value_type=None, **kws)
    Bases: pyslet.py2.UnicodeMixin, pyslet.pep8.MigratedClass

    Represents the HTML Length in pixels or as a percentage

    value Can be either an integer value or another Length instance.

    value_type (defaults to None) if value is an integer then value_type may be used to select a PIXEL or PERCENTAGE using the data constants defined below. If value is a string then value_type argument is ignored as this information is determined by the format defined in the specification (a trailing % indicating a PERCENTAGE).

    Instances can be compared for equality but not ordered (as pixels and percentages are on different scales). They do support non-zero test though, with 0% and 0 pixels both evaluating to False.

    PIXEL = 0

    data constant used to indicate pixel co-ordinates (also available as Pixel for backwards compatibility).

    PERCENTAGE = 1

    data constant used to indicate relative (percentage) co-ordinates (also available as Percentage for backwards compatibility).

    type = None

    type is one of the the Length constants: PIXEL or PERCENTAGE

    value = None

    value is the integer value of the length

    classmethod from_str(src)

    Returns a new instance parsed from a string

    resolve_value(dim=None)

    Returns the absolute value of the Length

    dim The size of the dimension used for interpreting percentage values. For example, if dim=640 and the value represents 50% the value 320 will be returned.
**LinkTypes**

Attributes defined to have type `%LinkTypes`:

```xml
<!ENTITY % LinkTypes "CDATA"
   -- space-separated list of link types handled directly -->
```

are represented as a list of lower-cased strings.

**MediaDesc**

Attributes defined to have type `%MediaDesc`:

```xml
<!ENTITY % MediaDesc "CDATA"
   -- single or comma-separated list of media descriptors -->
```

are represented by instances of the following class.

```python
class pyslet.html401.MediaDesc(value=None)
    Bases: pyslet.py2.UnicodeMixin

    A set-like list of media for which a linked resource is tailored

    value  An iterable (yielding strings)

    Values are reduced according to the algorithm described in the specification, so that “print and resolution >
    90dpi” becomes “print”. Descriptors are further reduced by making them lower case in keeping with their
    behaviour in CSS.

    Instances support the `in` operator, equality testing (the order of individual descriptors is ignored) and the boolean
    & and | operators for intersection and union operations always returning new instances. As a convenience, these
    binary operators will also work with a string argument which is converted to an instance using `from_str()`.

    Instances are canonicalized when converting to string by ASCII sorting.
```

**Method**

```python
class pyslet.html401.Method
    Bases: pyslet.xml.xsdatatypes Enumeration

    HTTP method used to submit a form

    Method.DEFAULT == Method.GET
```
MultiLength(s)

class pyslet.html401.MultiLength(value, value_type=None, **kws)

Bases: pyslet.html401.Length

MultiLength type from HTML.

“A relative length has the form “i*”, where “i” is an integer. . . . . . The value “*” is equivalent to “1*”:

```
<!ENTITY % MultiLength "CDATA"
    -- pixel, percentage, or relative -->
```

Extends the base class Length.

RELATIVE = 2

data constant used to indicate relative (multilength) co-ordinates

classmethod from_str(src)
    Returns a new instance parsed from a string

resolve_value(dim=None, multi_total=0)
    Extended to handle relative dimension calculations.

    dim: For relative lengths dim must be the remaining space to be shared after PIXEL and PERCENTAGE
    lengths have been deducted.

    multi_total: The sum of all MultiLength values in the current scope. If omitted (defaults to 0) or if the
    value passed is less than or equal to the relative value then dim is returned (allocating all remaining
    space to this multilength value).

    The behaviour for PIXEL and PERCENTAGE lengths is unchanged.

classmethod allocate_pixels(dim, lengths)
    Allocates pixels amongst multilength values

        dim: The total number of pixels available.

        lengths: A sequence of MultiLength values.

    Returns a list of integer pixel values corresponding to the values in lengths.

class pyslet.html401.MultiLengths(values)

Bases: pyslet.py2.UnicodeMixin

Behaves like a tuple of MultiLengths

```
<!ENTITY % MultiLengths "CDATA"
    -- comma-separated list of MultiLength -->
```

Constructed from an iterable of values that can be passed to MultiLength's constructor.

Multiple

class pyslet.html401.Multiple

Bases: pyslet.html401.NamedBoolean

For setting the multiple attribute of <select>.
NoHRef

class pyslet.html401.NoHRef
   Bases: pyslet.html401.NamedBoolean
       For setting the nohref attribute.

NoResize

class pyslet.html401.NoResize
   Bases: pyslet.html401.NamedBoolean
       For setting the noresize attribute.

Param Value Types

class pyslet.html401.ParamValueType
   Bases: pyslet.xml.xsdatatypes.Enumeration
       Enumeration for the valuetype of object parameters.

ReadOnly

class pyslet.html401.ReadOnly
   Bases: pyslet.html401.NamedBoolean
       Used for the readonly attribute.

Scope

Attributes defined to have type %Scope:

<!ENTITY % Scope "(row|col|rowgroup|colgroup)">

are represented as integer constants from the following enumeration.

class pyslet.html401.Scope
   Bases: pyslet.xml.xsdatatypes.Enumeration
       Enumeration for the scope of table cells.

Script

Attributes defined to have type %Script:

<!ENTITY % Script "CDATA" -- script expression -->

are not mapped. You can of course obtain their character string values using get_attribute() and set them using set_attribute().
Scrolling

Attributes defined to have type:

| scrolling | (yes|no|auto) auto -- scrollbar or none -- |

are represented with:

```python
class pyslet.html401.Scrolling
    Bases: pyslet.xml.xsdatatypes.Enumeration
```

Enumeration for the scrolling of iframes.

Selected

```python
class pyslet.html401.Selected
    Bases: pyslet.html401.NamedBoolean
```

Used for the selected attribute of `<option>`.

Shape

```python
class pyslet.html401.Shape
    Bases: pyslet.xml.xsdatatypes.Enumeration
```

Enumeration for the shape of clickable areas

```xml
<!ENTITY % Shape "(rect|circle|poly|default)">  
```

StyleSheet

Attributes defined to have type `%StyleSheet`:

```xml
<!ENTITY % StyleSheet "CDATA" -- style sheet data -->
```

are left as uninterpreted character strings.

Text

Attributes defined to have type `%Text`:

```xml
<!ENTITY % Text "CDATA">
```

are left as interpreted character strings.

TFrame

The definition:

```xml
<!ENTITY % TFrame "(void|above|below|hsides|lhs|rhs|vsides|box|border)">
```

is modelled by:
class pyslet.html401.TFrame
   Bases: pyslet.xml.xsdatatypes Enumeration

   Enumeration for the framing rules of a table.

TRules

The definition:

```
<!ENTITY % TRules "(none | groups | rows | cols | all)">
```

is modelled by:

class pyslet.html401.TRules
   Bases: pyslet.xml.xsdatatypes Enumeration

   Enumeration for the framing rules of a table.

URI

Attributes defined to have type %URI:

```
<!ENTITY % URI "CDATA" -- a Uniform Resource Identifier -->
```

are represented using pyslet.rfc2396.URI. This class automatically handles non-ASCII characters using the algorithm recommended in Appendix B of the specification, which involves replacing them with percent-encoded UTF-sequences.

When serialising these attributes we use the classes native string conversion which results in ASCII characters only so we don’t adhere to the principal of using the ASCII encoding only at the latest possible time.

Vertical Cell Alignment

The definition:

```
<!--ENTITY % cellvalign "valign (top|middle|bottom|baseline) #IMPLIED" -->
```

is modelled by:

class pyslet.html401.VAlign
   Bases: pyslet.xml.xsdatatypes Enumeration

   Enumeration for the vertical alignment of table cells

6.3.4 Attribute Mixin Classes

The DTD uses parameter entities to group related attribute definitions for re-use by multiple elements. We define mixin classes for each to group together the corresponding custom attribute mappings.

class pyslet.html401.AlignMixin
   Bases: object

   Mixin class for (loose) align attributes
These attributes are only defined by the loose DTD and are therefore not mapped to attributes of the instance.

```python
class pyslet.html401.BodyColorsMixin
Bases: object

Mixin class for (loose) body color attributes
```

These attributes are only defined by the loose DTD and are deprecated, they are therefore not mapped to attributes of the instance.

```python
class pyslet.html401.CellAlignMixin
Bases: object

Mixin class for table cell alignment attributes
```

```python
class pyslet.html401.CoreAttrsMixin
Bases: object

Mixin class for core attributes
```

The id attribute is declared in the DTD to be of type ID so is mapped to the special unique ID attribute for special handling by `Element`.

The class attribute is mapped to the python attribute `style_class` to avoid the python reserved name.
Pyslet is not an HTML rendering engine and so no attribute mappings are provided for these script hooks. Their values can of course be obtained using the generic get_attribute.

```xml
<!ENTITY % events
  "onclick %Script; #IMPLIED
    -- a pointer button was clicked --
ondblclick %Script; #IMPLIED
    -- a pointer button was double clicked--
onmousedown %Script; #IMPLIED
    -- a pointer button was pressed down --
onmouseup %Script; #IMPLIED
    -- a pointer button was released --
onmouseover %Script; #IMPLIED
    -- a pointer was moved onto --
onmousemove %Script; #IMPLIED
    -- a pointer was moved within --
onmouseout %Script; #IMPLIED
    -- a pointer was moved away --
onkeypress %Script; #IMPLIED
    -- a key was pressed and released --
onkeydown %Script; #IMPLIED
    -- a key was pressed down --
onkeyup %Script; #IMPLIED
    -- a key was released --" >
```

```python
class pyslet.html401.I18nMixin
    Bases: object

    Mixin class for i18n attributes
```

```xml
<!ENTITY % i18n
  "lang %LanguageCode; #IMPLIED -- language code --
dir (ltr|rtl) #IMPLIED
    -- direction for weak/neutral text --">
```

```python
get_lang()
    Replaces access to xml:lang

    If an element has set the HTML lang attribute we return this, otherwise we check if the element has set
xml:lang and return that instead.

    set_lang() is not modified, it always sets the xml:lang attribute.
```

```python
class pyslet.html401.ReservedMixin
    Bases: object

    Attributes reserved for future use
```

```xml
<!ENTITY % reserved
  "datasrc %URI; #IMPLIED
    -- a single or tabular Data Source --
datafld CDATA #IMPLIED
    -- the property or column name --
dataformatas (plaintext|html) plaintext
    -- text or html --" >
```

As these attributes are reserved for future no mappings are provided.
The following classes extend the above to form established groups.

```python
class pyslet.html401.AttrsMixin

    Mixin class for common attributes
```

```xml
<!ENTITY % attrs "%coreattrs; %i18n; %events;">
```

```python
class pyslet.html401.TableCellMixin

    Attributes shared by TD and TH
```

```xml
<!ATTLIST (TH|TD) -- header or data cell --
%attrs; -- %coreattrs, %i18n, %events --
abbr %Text; #IMPLIED
-- abbreviation for header cell --
axis CDATA #IMPLIED
-- comma-separated list of related headers--
headers IDREFS #IMPLIED
-- list of id's for header cells --
scope %Scope; #IMPLIED
-- scope covered by header cells --
rowspan NUMBER 1
-- number of rows spanned by cell --
colspan NUMBER 1
-- number of cols spanned by cell --
%cellhalign; -- horizontal alignment in cells --
%cellvalign; -- vertical alignment in cells --
nowrap (nowrap) #IMPLIED
-- suppress word wrap --
bgccolor %Color; #IMPLIED
-- cell background color --
width %Length; #IMPLIED
-- width for cell --
height %Length; #IMPLIED
-- height for cell --
>
```

The nowrap, bgcolor, width and height attributes are only defined in the loose DTD and are not mapped.

### 6.3.5 Content Mixin Classes

The DTD uses parameter entities to group elements into major roles. The most important roles are block, inline and flow. A block element is something like a paragraph, a list or table. An inline element is something that represents a span of text (including data itself) and a flow refers to either a block or inline. We exploit these definitions to create mixin classes that have no implementation but enable us to use Python issubclass or isinstance to test and enforce the content model. It also enables this model to be extended by external classes that also inherit from these basic classes to declare their role when inserted into HTML documents. This technique is used extensively by IMS QTI where non-HTML markup is intermingled with HTML markup.

```python
class pyslet.html401.FlowMixin
    Bases: pyslet.html401.XHTMLMixin

    Mixin class for flow elements
```
With these basic three classes we can go on to define a number of derived mixin classes representing the remaining specialised element groupings.

class pyslet.html401.FormCtrlMixin
Bases: pyslet.html401.InlineMixin
Form controls are just another type of inline element

class pyslet.html401.HeadContentMixin
Bases: object
Mixin class for HEAD content elements

class pyslet.html401.HeadMiscMixin
Bases: object
Mixin class for head.misc elements

class pyslet.html401.OptItemMixin
Bases: object
Mixin class for (OPTGROUP|OPTION)

class pyslet.html401.PreExclusionMixin
Bases: object
Mixin class for elements excluded from PRE

class pyslet.html401.SpecialMixin
Bases: pyslet.html401.InlineMixin
Specials are just another type of inline element.
6.3.6 Abstract Element Classes

6.3. HTML
class pyslet.html401.Heading (parent, name=None)

Abstract class for representing headings

```xml
<!ELEMENT (%heading;) - - (%inline;)* -- heading -->
<!ATTLIST (%heading;)
    %attrs; -- %coreattrs, %i18n, %events --
    %align; -- align, text alignment --
>
```

The align attribute is unmapped as it is not available in the strict DTD.

class pyslet.html401.InsDelInclusion (parent, name=None)

Represents inserted or deleted content

```xml
<!ELEMENT (INS|DEL) - - (%flow;)* -- inserted text, deleted text -->
<!ATTLIST (INS|DEL)
    %attrs; -- %coreattrs, %il8n, %events --
    cite %URI; #IMPLIED -- info on reason for change --
    datetime %Datetime; #IMPLIED -- date and time of change --
>
```

According to the DTD these elements can be inserted at will anywhere inside the document body (except that they do not appear within their own content models so do not nest). However, the specification suggests that they are actually to be treated as satisfying either block or inline which suggests that the intention is not to allow them to be inserted randomly in elements with more complex structures such as lists and tables. Indeed, there is the additional constraint that an inclusion appearing in an inline context may not contain block-level elements.

We don’t allow omitted end tags (as that seems dangerous) so incorrectly nested instances or block/inline misuse will cause validity exceptions. E.g.:

```xml
<body>
    <p>Welcome
    <ins><p>This document is about...</ins>
</body>
```

triggers an exception because `<ins>` is permitted in `<p>` but takes on an inline role. `<p>` is therefore not allowed in `<ins>` but the end tag of `<ins>` is required, triggering an error. This seems harsh given (a) that the markup is compatible with the DTD and (b) the meaning seems clear but I can only reiterate Goldfarb’s words from the SGML handbook where he says of exceptions:

> Like many good power tools, however, if used improperly they can cause significant damage


class pyslet.html401.Phrase (parent, name=None)

Abstract class for phrase elements

```xml
<!ENTITY % phrase "EM | STRONG | DFN | CODE | SAMP | KBD | VAR |
        CITE | ABBR | ACRONYM" >
<!ELEMENT (%fontstyle;|%phrase;) - - (%inline;)*>
<!ATTLIST (%fontstyle;|%phrase;)
    %attrs; -- %coreattrs, %il8n, %events --
>
```
Lists

class pyslet.html401.List (parent, name=None)


Abstract class for representing list elements

```xml
<!ENTITY % list "UL | OL">
```

Although a list item start tag is compulsory we are generous and will imply a list item if data is found. The end tag of the list is required.

Tables

class pyslet.html401.TRContainer (parent, name=None)

Bases: pyslet.html401.XHTMLElement

def get_child_class (stag_class):
    PCDATA or TH|TD trigger TR, end tags may be omitted

Frames

The content model of FRAMESET allows either (nested) FRAMESET or FRAME elements. The following class acts as both base class and as a way to identify the members of this group.

class pyslet.html401.FrameElement (parent, name=None)

Bases: pyslet.html401.CoreAttrsMixin, pyslet.html401.XHTMLElement

6.3.7 Element Reference

The classes that model each HTML element are documented here in alphabetical order for completeness.

class pyslet.html401.A (parent, name=None)


The HTML anchor element

```xml
<!ELEMENT A - - (%inline;)+ -(A) -- anchor -->
<!ATTLIST A
    %attrs; -- %coreattrs, %i18n, %events --
    charset %Charset; #IMPLIED
    -- char encoding of linked resource --
    type %ContentType; #IMPLIED
    -- advisory content type --
    name %CDATA #IMPLIED
    -- named link end --
    href %URI; #IMPLIED
    -- URI for linked resource --
    hreflang %LanguageCode; #IMPLIED
    -- language code --
    target %FrameTarget; #IMPLIED
    -- render in this frame --
    rel %LinkTypes; #IMPLIED
    -- forward link types --
```
The event handler attributes are not mapped but the target is, even though it is only defined in the loose DTD. Note that, despite the default value given in the DTD the shape attribute is not set and it will only have a non-None value in an instance if a value was provided explicitly.

```python
class pyslet.html401.Abb(\textit{parent}, \textit{name=\text{None}})
Bases: pyslet.html401.Phrase

class pyslet.html401.Address(\textit{parent}, \textit{name=\text{None}})

Address (of author)
```

```xml
<!-- ADDRESS - - ((%inline;)|P)* -- information on author -->
<!ATTLIST ADDRESS
 %attrs; -- %coreattrs, %i18n, %events --
 >
```

```python
class pyslet.html401.Area(\textit{parent})
Bases: pyslet.html401.AttrsMixin, pyslet.html401.XHTMLElement

Client-side image map area
```

```xml
<!-- AREA - O EMPTY -- client-side image map area -->
<!ATTLIST AREA
 %attrs; -- %coreattrs, %i18n, %events --
 shape %Shape; rect
 -- controls interpretation of coords --
 coords %Coords; #IMPLIED
 -- comma-separated list of lengths --
 href %URI; #IMPLIED
 -- URI for linked resource --
 target %FrameTarget; #IMPLIED
 -- render in this frame --
 nohref (nohref) #IMPLIED
 -- this region has no action --
 alt %Text; #REQUIRED -- short description --
 tabindex NUMBER #IMPLIED
 -- position in tabbing order --
 accesskey %Character; #IMPLIED
 -- accessibility key character --
 onfocus %Script; #IMPLIED
 -- the element got the focus --
 onblur %Script; #IMPLIED
 -->
```
The event attributes are not mapped however the target attribute is, even though it relates on to frames and the loose DTD.

```python
class pyslet.html401.B(parent, name=None)
    Bases: pyslet.html401.FontStyle
```

```python
class pyslet.html401.Base (parent, name=None)
    Bases: pyslet.html401.HeadContentMixin, pyslet.html401.XHTMLElement
```

Represents the base element

```xml
<!ELEMENT BASE - O EMPTY -- document base URI -->
<!ATTLIST BASE
    href %URI; #REQUIRED -- URI that acts as base URI --
    %coreattrs; -- id, class, style, title --
    lang %LanguageCode; #IMPLIED -- language code --
    dir (ltr|rtl) #REQUIRED -- directionality --
>
```

```python
class pyslet.html401.BaseFont (parent)
    Bases: pyslet.html401.PreExclusionMixin, pyslet.html401.SpecialMixin, pyslet.html401.XHTMLElement
```

Deprecated base font specification

```xml
<!ELEMENT BASEFONT - O EMPTY -- base font size -->
<!ATTLIST BASEFONT
    id ID #IMPLIED -- document-wide unique id --
    size CDATA #REQUIRED -- base font size for FONT elements --
    color %Color; #IMPLIED -- text color --
    face CDATA #IMPLIED -- comma-separated list of font names --
>
```

```python
class pyslet.html401.BDO (parent)
```

BiDi over-ride element

```xml
<!ELEMENT BDO - - (%inline;)* -- I18N BiDi over-ride -->
<!ATTLIST BDO
    %coreattrs; -- id, class, style, title --
    lang %LanguageCode; #IMPLIED -- language code --
    dir (ltr|rtl) #REQUIRED -- directionality --
>
```

The dir attribute is initialised to the `Direction` constant `ltr`.

```python
class pyslet.html401.Big (parent, name=None)
    Bases: pyslet.html401.PreExclusionMixin, pyslet.html401(FontStyle
```

```python
class pyslet.html401.Blockquote (parent, name=None)
```

Blocked quote.

Strict DTD:
Loose DTD:

This implementation enforces the strict DTD by wrapping data and inline content in DIV. The Attributes are common to both forms of the DTD:

```xml
<!ATTLIST BLOCKQUOTE
  %attrs; -- %coreattrs, %i18n, %events --
  cite %URI; #IMPLIED
  -- URI for source document or msg -->
```

```python
class pyslet.html401.Body (parent, name=None)

Represents the HTML BODY element
```

Note that the event handlers are not mapped to instance attributes.

```python
class pyslet.html401.Br (parent, name=None)
Bases: pyslet.html401.CoreAttrsMixin, pyslet.html401.SpecialMixin, pyslet.html401.XHTMLElement

Represents a line break
```

The clear attribute is only in the loose DTD and is not mapped.

```python
class pyslet.html401.Button (parent)

Alternative form of button (with content)
```

390 Chapter 6. Other Supporting Standards
value     CDATA            #IMPLIED
       -- sent to server when submitted --
type     (button|submit|reset)     submit
       -- for use as form button --
disabled  (disabled)             #IMPLIED
       -- unavailable in this context --
tabindex  NUMBER                #IMPLIED
       -- position in tabbing order --
accesskey %Character;         #IMPLIED
       -- accessibility key character --
onfocus   %Script;             #IMPLIED
       -- the element got the focus --
onblur    %Script;             #IMPLIED
       -- the element lost the focus --
%reserved; -- reserved for possible future use -->

The event handlers are not mapped.

class pyslet.html401.Caption (parent, name=None)
Bases: pyslet.html401.AttrsMixin, pyslet.html401.InlineContainer

Represents a table caption

<!ELEMENT CAPTION - - (%inline;)* -- table caption -->
<!ATTLIST CAPTION
 %attrs; -- %coreattrs, %i18n, %events --
align     %CAAlign;         #IMPLIED -- relative to table --
>

The align attribute is along defiend in the loose DTD and is not mapped.

class pyslet.html401.Center (parent, name=None)

Equivalent to <div align="center">, only applies to loose DTD

<!ELEMENT CENTER - - (%flow;)* -- shorthand for DIV align=center -->
<!ATTLIST CENTER
 %attrs; -- %coreattrs, %i18n, %events --
>

class pyslet.html401.Cite (parent, name=None)
Bases: pyslet.html401.Phrase

class pyslet.html401.Code (parent, name=None)
Bases: pyslet.html401.Phrase

class pyslet.html401.Col (parent)

Represents a table column

<!ELEMENT COL - O EMPTY -- table column -->
<!ATTLIST COL
 %attrs; -- %coreattrs, %i18n, %events --
span     NUMBER 1
       -- COL attributes affect N columns --
class pyslet.html401.ColGroup(parent)

Represents a group of columns

```
<!ELEMENT COLGROUP - O (COL)* -- table column group -->
<!ATTLIST COLGROUP
  %attrs;          -- %coreattrs, %i18n, %events --
  span NUMBER 1    -- default number of columns in group --
  width %MultiLength; #IMPLIED
  %cellhalign;     -- horizontal alignment in cells --
  %cellvalign;     -- vertical alignment in cells --
>
```

class pyslet.html401.DD(parent, name=None)

Represents the definition of a defined term

```
<!ELEMENT DD - O (%flow;)* -- definition description -->
<!ATTLIST (DT|DD)
  %attrs;          -- %coreattrs, %i18n, %events --
>
```

class pyslet.html401.Del(parent, name=None)
Bases: pyslet.html401.InsDelInclusion

class pyslet.html401.Dfn(parent, name=None)
Bases: pyslet.html401.Phrase

class pyslet.html401.Div(parent, name=None)

A generic flow container

```
<!ELEMENT DIV - - (%flow;)* -- -->
<!ATTLIST DIV
  %attrs;          -- %coreattrs, %i18n, %events --
  %align;          -- align, text alignment --
  %reserved;       -- reserved for possible future use --
>
```

class pyslet.html401.DL(parent, name=None)

Represents definition lists
The compact attribute is not mapped as it is only defined in the loose DTD.

**class** **pyslet.html401.DT** *(parent, name=None)*

**Bases**: **pyslet.html401.AttrsMixin**, **pyslet.html401.InlineContainer**

Represents a defined term

**class** **pyslet.html401.Em** *(parent, name=None)*

**Bases**: **pyslet.html401.Phrase**

**class** **pyslet.html401.FieldSet** *(parent)*


Represents a group of controls in a form

**class** **pyslet.html401.Font** *(parent, name=None)*


Represents font style information (loose DTD only)

Although defined only in the loose DTD we provide custom mappings for all of the attributes.

**class** **pyslet.html401.Form** *(parent)*


Represents the form element.

**Strict DTD:**

```xml
<!ELEMENT FORM - - (%block;|SCRIPT)+ -(FORM) -- interactive form -->
```
Loose DTD:

```xml
<!ELEMENT FORM - - (%flow;)* -(FORM) -- interactive form -->

Attributes (target is mapped even though it is only in the loose DTD) as it is for use in frame-based documents:

```xml
<!ATTLIST FORM
  %attrs; -- %coreattrs, %i18n, %events --
  action %URI; #REQUIRED
    -- server-side form handler --
  method (GET|POST) GET
    -- HTTP method used to submit the form--
  enctype %ContentType; "application/x-www-form-urlencoded"
  accept %ContentTypes; #IMPLIED
    -- list of MIME types for file upload --
  name CDATA #IMPLIED
    -- name of form for scripting --
  onsubmit %Script; #IMPLIED
    -- the form was submitted --
  onreset %Script; #IMPLIED -- the form was reset --
  target %FrameTarget; #IMPLIED -- render in this frame --
  accept-charset %Charsets; #IMPLIED
    -- list of supported charsets --
>
```

```python
class pyslet.html401.Frame(parent)

Bases: pyslet.html401.FrameElement

Represents a Frame within a frameset document

```xml
<!ELEMENT FRAME - O EMPTY -- subwindow -->
<!ATTLIST FRAME
  %coreattrs; -- id, class, style, title --
  longdesc %URI; #IMPLIED
    -- link to long description (complements title) --
  name CDATA #IMPLIED
    -- name of frame for targetting --
  src %URI; #IMPLIED
    -- source of frame content --
  frameborder (1|0) 1
    -- request frame borders? --
  marginwidth %Pixels; #IMPLIED
    -- margin widths in pixels --
  marginheight %Pixels; #IMPLIED
    -- margin height in pixels --
  noresize (noresize) #IMPLIED
    -- allow users to resize frames? --
  scrolling (yes|no/auto) auto
    -- scrollbar or none --
>
```

The frameborder, marginwidth and marginheight attributes are not mapped.

```python
class pyslet.html401.Frameset(parent)

Bases: pyslet.html401.FrameElement

Represents a frameset (within a Frameset document)

```xml
<!ELEMENT FRAMESET - - ((FRAMESET|FRAME)+ & NOFRAMES?)
  -- window subdivision-->
The event handlers are not mapped to custom attributes.

```python
class pyslet.html401.H1 (parent, name=None)
Bases: pyslet.html401.Heading
```

```python
class pyslet.html401.H2 (parent, name=None)
Bases: pyslet.html401.Heading
```

```python
class pyslet.html401.H3 (parent, name=None)
Bases: pyslet.html401.Heading
```

```python
class pyslet.html401.H4 (parent, name=None)
Bases: pyslet.html401.Heading
```

```python
class pyslet.html401.H5 (parent, name=None)
Bases: pyslet.html401.Heading
```

```python
class pyslet.html401.H6 (parent, name=None)
Bases: pyslet.html401.Heading
```

```python
class pyslet.html401.Head (parent)
Bases: pyslet.html401.I18nMixin, pyslet.html401.XHTMLElement
```

Represents the HTML head structure

```html
<!ELEMENT HEAD O O (%head.content;) +(%head.misc;)
-- document head -->
<!ATTLIST HEAD
 %i18n; -- lang, dir --
 profile %URI; #IMPLIED
 -- named dictionary of meta info --
>
</!ATTLIST HEAD

```python
class pyslet.html401.HR (parent, name=None)
```

Represents a horizontal rule

```html
<!ELEMENT HR - O EMPTY -- horizontal rule -->
<!ATTLIST HR
 %attrs; -- %coreattrs, %i18n, %events --
 align (left|center|right) #IMPLIED
 noshade (noshade) #IMPLIED
 size %Pixels; #IMPLIED
 width %Length; #IMPLIED
>
```
The align, noshade, size and width attributes are not defined in the strict DTD and are not mapped.

```python
class pyslet.html401.HTML (parent)
Bases: pyslet.html401.I18nMixin, pyslet.html401.XHTMLElement

Represents the HTML document structure
```

```xml
<!ENTITY % html.content "HEAD, BODY">
<!ELEMENT HTML O O (%html.content;) -- document root element -->
<!ATTLIST HTML
 %i18n; -- lang, dir -->
>
```

```python
class pyslet.html401.HTMLFrameset (parent)
Bases: pyslet.html401.I18nMixin, pyslet.html401.XHTMLElement

Represents the HTML frameset document element
```

See HTML for a complete declaration.

We omit the default name declaration XMLNAME to ensure uniqueness in the document mapping adding. When creating orphan instances of this element you must use set_xmlname() to set a name for the element before serialization.

```python
class pyslet.html401.I (parent, name=None)
Bases: pyslet.html401.FontStyle
```

```python
class pyslet.html401.IFrame (parent)

Represents the iframe element
```

```xml
<!ELEMENT IFRAME -- (%flow;)* -- inline subwindow -->
<!ATTLIST IFRAME
 %coreattrs; -- id, class, style, title --
 longdesc %URI; #IMPLIED
   -- link to long description (complements title) --
 name %URI; #IMPLIED
   -- name of frame for targeting --
 src %URI; #IMPLIED
   -- source of frame content --
 frameborder (1|0) 1
   -- request frame borders? --
 marginwidth %Pixels; #IMPLIED
   -- margin widths in pixels --
 marginheight %Pixels; #IMPLIED
   -- margin height in pixels --
 scrolling (yes|no|auto) auto
   -- scrollbar or none --
 align %IAAlign; #IMPLIED
   -- vertical or horizontal alignment --
 height %Length; #IMPLIED
   -- frame height --
 width %Length; #IMPLIED
   -- frame width --
>
```

IFrames are not part of the strict DTD, perhaps surprisingly given their widespread adoption. For consistency with other elements we leave the frameborder, marginwidth, marginheight and align attributes unmapped. As a
result, we rely on the default frameborder value provided in the DTD rather than setting an attribute explicitly on construction. In contrast, the scrolling attribute is mapped and is initialised to Scrolling.auto.

```python
class pyslet.html401.Img

Represents the <img> element
```

```xml
<!ELEMENT IMG - O EMPTY -->
<!ATTLIST IMG
   %attrs; -- %coreattrs, %i18n, %events --
   src %URI; #REQUIRED -- URI of image to embed --
   alt %Text; #REQUIRED -- short description --
   longdesc %URI; #IMPLIED -- link to long description (complements alt) --
   name CDATA #IMPLIED -- name of image for scripting --
   height %Length; #IMPLIED -- override height --
   width %Length; #IMPLIED -- override width --
   usemap %URI; #IMPLIED -- use client-side image map --
   ismap (ismap) #IMPLIED -- use server-side image map --
   align %IAlign; #IMPLIED -- vertical or horizontal alignment --
   border %Pixels; #IMPLIED -- link border width --
   hspace %Pixels; #IMPLIED -- horizontal gutter --
   vspace %Pixels; #IMPLIED -- vertical gutter -->
```

The align, border, hspace and vspace attributes are only defined by the loose DTD and are not mapped.

```python
class pyslet.html401.Input
    Bases: pyslet.html401.FormCtrlMixin, pyslet.html401.AttrsMixin, pyslet.html401.XHTMLElement

Represents the input element
```

```xml
<!-- attribute name required for all but submit and reset -->
<!ELEMENT INPUT - O EMPTY -->
<!ATTLIST INPUT
   %attrs; -- %coreattrs, %i18n, %events --
   type %InputType; TEXT -- what kind of widget is needed --
   name CDATA #IMPLIED -- submit as part of form --
   value CDATA #IMPLIED -- Specify for radio buttons and checkboxes --
   checked (checked) #IMPLIED -- for radio buttons and check boxes --
   disabled (disabled) #IMPLIED -- unavailable in this context --
   readonly (readonly) #IMPLIED -- for text and passwd --
   size CDATA #IMPLIED -- specific to each type of field --
   maxlength NUMBER #IMPLIED -- max chars for text fields --
   src %URI; #IMPLIED
```

6.3. HTML 397
-- for fields with images --
alt CDATA #IMPLIED
-- short description --
usemap %URI; #IMPLIED
-- use client-side image map --
ismap (ismap) #IMPLIED
-- use server-side image map --
tabindex NUMBER #IMPLIED
-- position in tabbing order --
accesskey %Character; #IMPLIED
-- accessibility key character --
onfocus %Script; #IMPLIED
-- the element got the focus --
onblur %Script; #IMPLIED
-- the element lost the focus --
onselect %Script; #IMPLIED
-- some text was selected --
onchange %Script; #IMPLIED
-- the element value was changed --
accept %ContentTypes; #IMPLIED
-- list of MIME types for file upload --
align %IAAlign; #IMPLIED
-- vertical or horizontal alignment --
%reserved; -- reserved for possible future use --
>

The event handlers are unmapped. The align attribute is defined only in the loose DTD and is also unmapped.

class pyslet.html401.Ins (parent, name=None)
Bases: pyslet.html401.InsDelInclusion

class pyslet.html401.IsIndex (parent, name=None)

Deprecated one-element form control

<!ELEMENT ISINDEX - O EMPTY -- single line prompt -->
<!ATTLIST ISINDEX
%coreattrs; -- id, class, style, title --
%i8n; -- lang, dir --
prompt %Text; #IMPLIED -- prompt message -->

class pyslet.html401.Kbd (parent, name=None)
Bases: pyslet.html401.Phrase

class pyslet.html401.Label (parent, name=None)

Label element

<!ELEMENT LABEL - - (%inline;)* -(LABEL) -- form field label text -->
<!ATTLIST LABEL
%attrs; -- %coreattrs, %i8n, %events --
for IDREF #IMPLIED
-- matches field ID value --
accesskey %Character; #IMPLIED
To avoid the use of the reserved word ‘for’ this attribute is mapped to the attribute name `for_field`. The event attributes are not mapped.

```python
class pyslet.html401.Legend (parent, name=None)
Bases: pyslet.html401.AttrsMixin, pyslet.html401.InlineContainer

legend element
```

```html
<!ELEMENT LEGEND - - (%inline;)* -- fieldset legend -->
<!ATTLIST LEGEND
%attrs; -- %coreattrs, %i18n, %events --
accesskey %Character; #IMPLIED -- accessibility key character --
>
```

```python
class pyslet.html401.LI (parent, name=None)

Represent list items
```

```html
<!ELEMENT LI - O (%flow;)* -- list item -->
<!ATTLIST LI
%attrs; -- %coreattrs, %i18n, %events --
type %LIStyle; #IMPLIED -- list item style --
value NUMBER #IMPLIED -- reset sequence number --
>
```

The type and value attributes are only defined by the loose DTD and are not mapped.

```python
class pyslet.html401.Link (parent, name=None)

Media-independent link
```

```html
<!ELEMENT LINK - O EMPTY -- a media-independent link -->
<!ATTLIST LINK
%attrs; -- %coreattrs, %i18n, %events --
charset %Charset; #IMPLIED -- char encoding of linked resource --
href %URI; #IMPLIED -- URI for linked resource --
hreflang %LanguageCode; #IMPLIED -- language code --
type %ContentType; #IMPLIED -- advisory content type --
rel %LinkTypes; #IMPLIED -- forward link types --
rev %LinkTypes; #IMPLIED -- reverse link types --
media %MediaDesc; #IMPLIED -- for rendering on these media --
>
```

```python
class pyslet.html401.Map (parent)

Represents a client-side image map
```
class pyslet.html401.Meta (parent)
Bases: pyslet.html401.I18nMixin, pyslet.html401.HeadMiscMixin, pyslet.html401.XHTMLElement

Represents the meta element

The http-equiv attribute cannot be mapped

class pyslet.html401.NoFrames (parent, name=None)

Represents the NOFRAMES element.

This element is deprecated, it is not part of the strict DTD or HTML5. This element is used to represent instances encountered in documents using the loose DTD:

class pyslet.html401.NoFramesFrameset (parent)
Bases: pyslet.html401.AttrsMixin, pyslet.html401.XHTMLElement

Represents the NOFRAMES element in a FRAMESET document.

This element is deprecated, it is not part of the strict DTD or HTML5. This element is used to represent instances encountered in documents using the frameset DTD:

We omit the XMLNAME attribute (the default element name) to prevent a name clash when declaring the elements in the name space. Instead we'll use a special catch to ensure that <noframes> maps to this element in a frameset context.

class pyslet.html401.NoScript (parent)

400 Chapter 6. Other Supporting Standards
Represents the NOSCRIPT element

Loose DTD:

```xml
<!ELEMENT NOSCRIPT - - (%flow;)*
   -- alternate content container for non script-based rendering -->
```

Strict DTD:

```xml
<!ELEMENT NOSCRIPT - - (%block;)+
   -- alternate content container for non script-based rendering -->
```

Common:

```xml
<!ATTLIST NOSCRIPT
   %attrs; -- %coreattrs, %i18n, %events -->
```

We take the liberty of enforcing the stricter DTD which has the effect of starting an implicit `<div>` if inline elements are encountered in `<noscript>` elements.

We also bring forward an element of HTML5 compatibility by allowing NoScript within the document `<head>` with a content model equivalent to:

```xml
<!ELEMENT NOSCRIPT - - (LINK|STYLE|META)* -->
```

class `pyslet.html401.Object` *(parent, name=None)*


Represents the object element

```xml
<!ELEMENT OBJECT - - (PARAM | %flow;)*

<!ATTLIST OBJECT
   %attrs; -- %coreattrs, %i18n, %events --
   declare (declare) #IMPLIED
      -- declare but don't instantiate flag --
   classid %URI; #IMPLIED
      -- identifies an implementation --
   codebase %URI; #IMPLIED
      -- base URI for classid, data, archive--
   data %URI; #IMPLIED
      -- reference to object's data --
   type %ContentType; #IMPLIED
      -- content type for data --
   codetype %ContentType; #IMPLIED
      -- content type for code --
   archive CDATA #IMPLIED
      -- space-separated list of URIs --
   standby %Text; #IMPLIED
      -- message to show while loading --
   height %Length; #IMPLIED
      -- override height --
   width %Length; #IMPLIED
      -- override width --
   usemap %URI; #IMPLIED
      -- use client-side image map --
   name CDATA #IMPLIED
      -- submit as part of form --
   tabindex NUMBER #IMPLIED
```
class pyslet.html401.OL(parent, name=None)
Bases: pyslet.html401.List

Represents ordered lists

<!ELEMENT OL - - (LI)+ -- ordered list -->
<!ATTLIST OL
  %attrs; -- %coreattrs, %i18n, %events --
  type %OLStyle; #IMPLIED -- numbering style --
  compact (compact) #IMPLIED -- reduced interitem spacing --
  start NUMBER #IMPLIED -- starting sequence number --
>

The type, compact and start attributes are only defined in the loose DTD and so are not mapped.

class pyslet.html401.OptGroup(parent)

OptGroup element

<!ELEMENT OPTGROUP - - (OPTION)+ -- option group -->
<!ATTLIST OPTGROUP
  %attrs; -- %coreattrs, %i18n, %events --
  disabled (disabled) #IMPLIED -- unavailable in this context --
  label %Text; #REQUIRED -- for use in hierarchical menus --
>

class pyslet.html401.Option(parent, name=None)

Option element

<!ELEMENT OPTION - O (#PCDATA) -- selectable choice -->
<!ATTLIST OPTION
  %attrs; -- %coreattrs, %i18n, %events --
  selected (selected) #IMPLIED
  disabled (disabled) #IMPLIED -- unavailable in this context --
  label %Text; #IMPLIED -- for use in hierarchical menus --
  value CDATA #IMPLIED -- defaults to element content --
>

class pyslet.html401.P(parent, name=None)

Represents a paragraph
class pyslet.html401.Param(parent)
    Bases: pyslet.html401.XHTMLElement

Represents an object parameter

```xml
<!ELEMENT PARAM - O EMPTY -- named property value -->
<!ATTLIST PARAM
    id ID #IMPLIED -- document-wide unique id --
    name CDATA #REQUIRED -- property name --
    value CDATA #IMPLIED -- property value --
    valuetype (DATA|REF|OBJECT) DATA -- How to interpret value --
    type %ContentType; #IMPLIED -- content type for value when valuetype=ref --
>
```

The name attribute is required and is initialised to "_". The valuetype attribute is not populated automatically so applications processing this element should treat a value of None as equivalent to the integer constant ParamValueType.data. The value of `value` is always a string, even if valuetype is ref, indicating that it should be interpreted as a URI.

class pyslet.html401.Pre(parent, name=None)

Represents pre-formatted text

```xml
<!ELEMENT PRE - - (%inline;)* -- preformatted text -->
<!ATTLIST PRE
    width NUMBER #IMPLIED
>
```

The width attribute is only defined in the loose DTD and is not mapped.

class pyslet.html401.Q(parent, name=None)

Represents an inline quotation

```xml
<!ELEMENT Q - - (%inline;)* -- short inline quotation -->
<!ATTLIST Q
    cite %URI; #IMPLIED -- URI for source document or msg --
>
```

class pyslet.html401.S(parent, name=None)
    Bases: pyslet.html401.FontStyle

6.3. HTML

403
class pyslet.html401.Samp(parent, name=None)
Bases: pyslet.html401.Phrase

class pyslet.html401.Script(parent)
Bases: pyslet.html401.SpecialMixin, pyslet.html401.HeadMiscMixin, pyslet.html401.XHTMLElement

Represents the script element

```xml
<!ELEMENT SCRIPT - - %Script; -- script statements -->
<!ATTLIST SCRIPT
  charset %Charset; #IMPLIED
  -- char encoding of linked resource --
  type %ContentType; #REQUIRED
  -- content type of script language --
  src %URI; #IMPLIED
  -- URI for an external script --
  defer (defer) #IMPLIED
  -- UA may defer execution of script --
  event CDATA #IMPLIED
  -- reserved for possible future use --
  for %URI; #IMPLIED
  -- reserved for possible future use -->
```

As the type is required instances are initialised with text/javascript.

class pyslet.html401.Select(parent, name=None)
Bases: pyslet.html401.AttrsMixin, pyslet.html401.FormCtrlMixin, pyslet.html401.XHTMLElement

Select element

```xml
<!ELEMENT SELECT - - (OPTGROUP|OPTION)+ -- option selector -->
<!ATTLIST SELECT
  %attrs; -- %coreattrs, %i18n, %events --
  name CDATA #IMPLIED -- field name --
  size NUMBER #IMPLIED -- rows visible --
  multiple (multiple) #IMPLIED -- default is single selection --
  disabled (disabled) #IMPLIED -- unavailable in this context --
  tabindex NUMBER #IMPLIED -- position in tabbing order --
  onfocus %Script; #IMPLIED -- the element got the focus --
  onblur %Script; #IMPLIED -- the element lost the focus --
  onchange %Script; #IMPLIED -- the element value was changed --
  %reserved; -- reserved for possible future use --
>
```

No custom mapping is provided for the event handlers.

class pyslet.html401.Small(parent, name=None)
Bases: pyslet.html401.PreExclusionMixin, pyslet.html401.FontStyle

class pyslet.html401.Span(parent, name=None)

Represents a span of text

```xml
<!ELEMENT SPAN - - (%inline;)*
  -- generic language/style container -->
<!ATTLIST SPAN
  %attrs; -- %coreattrs, %i18n, %events --
>
```
class pyslet.html401.Strike (parent, name=None)  
    Bases: pyslet.html401.FontStyle

class pyslet.html401.Strong (parent, name=None)  
    Bases: pyslet.html401Phrase

class pyslet.html401.Style (parent)  
    Bases: pyslet.html401.I18nMixin, pyslet.html401.HeadMiscMixin, pyslet.html401.XHTMLElement

    Represents the style element

    <!ELEMENT STYLE - - %StyleSheet -- style info -->
    <!ATTLIST STYLE
        %i18n; -- lang, dir, for use with title --
        type %ContentType; #REQUIRED
        -- content type of style language --
        media %MediaDesc; #IMPLIED
        -- designed for use with these media --
        title %Text; #IMPLIED
        -- advisory title --
    >

As the content type is required instances are initialised with text/css.

class pyslet.html401.Sub (parent, name=None)  

    Represents a subscript

    <!ELEMENT (SUB|SUP) - - (%inline;)* -- subscript, superscript -->
    <!ATTLIST (SUB|SUP) %attrs; -- %coreattrs, %i18n, %events -->

class pyslet.html401.Sup (parent, name=None)  

    Represents a superscript

    <!ELEMENT (SUB|SUP) - - (%inline;)* -- subscript, superscript -->
    <!ATTLIST (SUB|SUP) %attrs; -- %coreattrs, %i18n, %events -->

class pyslet.html401.Table (parent)  

    Represents a table

    <!ELEMENT TABLE - - {CAPTION?, {COL*|COLGROUP*}, THEAD?, TFOOT?, TBODY*}> 
    <!ATTLIST TABLE
        %attrs; -- %coreattrs, %i18n, %events --
        summary %Text; #IMPLIED
        -- purpose/structure for speech output--
The align and bgcolor attributes are only defined in the loose DTD and are not mapped. The datapagesize is also not mapped.

When parsing we are generous in allowing data to automatically start the corresponding TBody (and hence TR+TD).

class pyslet.html401.TBody (parent, name=None)

Represents a table body

class pyslet.html401.TD (parent, name=None)
Bases: pyslet.html401.TableCellMixin, pyslet.html401.FlowContainer

Represents a table cell

For attribute information see TableCellMixin.

class pyslet.html401.TextArea (parent)

TextArea element
The event handlers are not mapped. As rows and cols are both required the constructor provides initial values of 1 and 80 respectively.

```python
class pyslet.html401.TFoot(parent, name=None)

Represents a table footer
```

```python
class pyslet.html401.TH(parent, name=None)
Bases: pyslet.html401.TableCellMixin, pyslet.html401.FlowContainer

Represents a table header cell
```

For attribute information see TableCellMixin.

```python
class pyslet.html401.THead(parent, name=None)

Represents a table header
```

```python
class pyslet.html401.Title(parent, name=None)
Bases: pyslet.html401.I18nMixin, pyslet.html401.HeadContentMixin, pyslet.html401.XHTMLElement
```
Represents the TITLE element

```xml
<!ELEMENT TITLE - - (#PCDATA) -(%head.misc; -- document title -->
<!ATTLIST TITLE %i18n >
```

```python
class pyslet.html401.TR (parent, name=None)

Represents a table row

```xml
<!ELEMENT TR - O (TH|TD)+ -- table row -->
<!ATTLIST TR
  -- table row --
  %attrs; -- %coreattrs, %i18n, %events --
  %cellhalign; -- horizontal alignment in cells --
  %cellvalign; -- vertical alignment in cells --
  bgcolor %Color; #IMPLIED
    -- background color for row --
  >
```

The bgcolor attribute is only defined by the loose DTD so is left unmapped. We treat data inside `<tr>` as starting an implicit `<td>` element.

```python
class pyslet.html401.TT (parent, name=None)
Bases: pyslet.html401.FontStyle
```

```python
class pyslet.html401.U (parent, name=None)
Bases: pyslet.html401.FontStyle
```

```python
class pyslet.html401.UL (parent, name=None)
Bases: pyslet.html401.List

Represents the unordered list element

```xml
<!ELEMENT UL - - (LI)+ -- ordered list -->
<!ATTLIST UL
  -- %coreattrs, %i18n, %events --
  %attrs; -- %coreattrs, %i18n, %events --
  type %ULStyle; #IMPLIED -- bullet style --
  compact (compact) #IMPLIED
    -- reduced interitem spacing --
  >
```

The type and compact attributes are only defined by the loose DTD and are left unmapped.

```python
class pyslet.html401.Var (parent, name=None)
Bases: pyslet.html401.Phrase
```

### 6.3.8 Exceptions

```python
class pyslet.html401.XHTMLError
Bases: exceptions.Exception

Abstract base class for errors in this module
```

```python
class pyslet.html401.XHTMLValidityError
Bases: pyslet.html401.XHTMLError

General error raised by HTML model constraints.
```
The parser is very generous in attempting to interpret HTML but there some situations where it would be dangerous to infer the intent and this error is raised in those circumstances.

```python
class pyslet.html401.XHTMLLError
    Bases: exceptions.Exception
    
    Abstract base class for errors in this module
```

## 6.4 Uniform Resource Identifiers (RFC2396)

This module defines functions and classes for working with URI as defined by RFC2396: [http://www.ietf.org/rfc/rfc2396.txt](http://www.ietf.org/rfc/rfc2396.txt)

In keeping with usage in the specification we use URI in both the singular and plural sense.

In addition to parsing and formatting URI from strings, this module also supports computing and resolving relative URI. To do this we define two notional operators.

The resolve operator:

\[ U = B \ [\ast] R \]

calculates a new URI ‘U’ from a base URI ‘B’ and a relative URI ‘R’.

The relative operator:

\[ U \ [/] B = R \]

calculates the relative URI ‘R’ formed by expressing ‘U’ relative to ‘B’.

The Relative operator defines the reverse of the resolve operator, however note that in some cases several different values of R can resolve to the same URL with a common base URI.

### 6.4.1 Creating URI Instances

To create URI use the `URI.from_octets()` class method. This method takes both character and binary strings though in the first case the string must contain only ASCII characters and in the latter only bytes that represent ASCII characters. The following function can help convert general character strings to a suitable format but it is not a full implementation of the IRI specification, in particular it does not encode delimiters (such as space) and it does not deal intelligently with unicode domain names (these must be converted to their ASCII URI forms first).

```python
pyslet.rfc2396.encode_unicode_uri(usrc)
    
    Extracts a URI octet-string from a unicode string.

    usrc  A character string

    Returns a character string with any characters outside the US-ASCII range replaced by URI-escaped UTF-8 sequences. This is not a general escaping method. All other characters are ignored, including non-URI characters like space. It is assumed that any (other) characters requiring escaping are already escaped.

    The encoding algorithm used is the same as the one adopted by HTML. This is not part of the RFC standard which only defines the behaviour for streams of octets but it is in line with the approach adopted by the later IRI spec.
```
6.4.2 URI

class pyslet.rfc2396.URI(octets)
    Bases: pyslet.py2.CmpMixin, pyslet.pep8.PEP8Compatibility

Class to represent URI References

You won’t normally instantiate a URI directly as it represents a generic URI. This class is designed to be
overridden by scheme-specific implementations. Use the class method from_octets() to create instances.

If you are creating your own derived classes call the parent constructor to populate the attributes defined
here from the URI’s string representation passing a character string representing the octets of the URI. (For
backwards compatibility a binary string will be accepted provided it can be decoded as US ASCII char-
acters.) You can override the scheme-specific part of the parsing by defining your own implementation of
parse_scheme_specific_part().

It is an error if the octets string contains characters that are not allowed in a URI.

Note: The following details have changed significantly following updates in 0.5.20160123 to introduce support
for Python 3. Although the character/byte/octet descriptions have changed the actual affect on running code is
minimal when running under Python 2.

Unless otherwise stated, all attributes are character strings that encode the ‘octets’ in each component of the
URI. These attributes retain the %-escaping. To obtain the actual data use unescape_data() to obtain the
original octets (as a byte string). The specification does not specify any particular encoding for interpreting
these octets, indeed in some types of URI these binary components may have no character-based interpretation.

For example, the URI “%E8%8B%B1%E5%9B%BD.xml” is a character string that represents a UTF-8 and
URL-encoded path segment using the Chinese word for United Kingdom. To obtain the correct unicode path
segment you would first use unescape_data() to obtain the binary string of bytes and then decode with
UTF-8:

```python
>>> src = "%E8%8B%B1%E5%9B%BD.xml"
>>> uri.unescape_data(src).decode('utf-8')
"u'\u82f1\u56fd.xml"
```

URI can be converted to strings but the result is a character string that retains any %-encoding. Therefore, these
character strings always use the restricted character set defined by the specification (a subset of US ASCII) and,
in Python 2, can be freely converted between the str and unicode types.

URI are immutable and can be compared and used as keys in dictionaries. Two URI compare equal if their
canonical forms are identical. See canonicalize() for more information.

classmethod from_octets (octets, strict=False)

Creates an instance of URI from a string

Note: This method was changed in Pyslet 0.5.20160123 to introduce support for Python 3. It now takes
either type of string but a character string is now preferred.

This is the main method you should use for creating instances. It uses the URI’s scheme to determine the
appropriate subclass to create. See register() for more information.

octets A string of characters that represents the URI’s octets. If a binary string is passed it is assumed to
be US ASCII and converted to a character string.
strict (defaults to False) If the character string contains characters outside of the US ASCII character range then \texttt{encode_unicode_uri() \textcolor{red}{\textit{is called}}} before the string is used to create the instance. You can turn off this behaviour (to enable strict URI-parsing) by passing \texttt{strict=True}

Pyslet manages the importing and registering of the following URI schemes using its own classes: http, https, file and urn. Additional modules are loaded and schemes registered 'on demand' when instances of the corresponding URI are first created.

\texttt{scheme_class = \{'urn': <class 'pyslet.urn.URN'>, 'http': <class 'pyslet.http.params.HTTPURL'>, 'https': <class 'pyslet.http.params.HTTPSURL'>, 'file': <class 'pyslet.rfc2396.FileURL'>\}}

A dictionary mapping lower-case URI schemes onto the special classes used to represent them

\begin{verbatim}
classmethod register(scheme, uri_class)
    Registers a class to represent a scheme

    scheme A string representing a URI scheme, e.g., 'http'. The string is converted to lower-case before it is registered.

    uri_class A class derived from URI that is used to represent URI from scheme

    If a class has already been registered for the scheme it is replaced. The mapping is kept in the \texttt{scheme_class} dictionary.

classmethod from_virtual_path(path)
    Converts a virtual file path into a URI instance

    path A \texttt{pyslet.vfs.VirtualFilePath} instance representing a file path in a virtual file system.

    The path is always made absolute before being converted to a \texttt{FileURL}.

    The authority (host name) in the resulting URL is usually left blank except when running under Windows, in which case the URL is constructed according to the recommendations in this blog post. In other words, UNC paths are mapped to both the network location and path components of the resulting file URL.

    For named virtual file systems (i.e., those that don’t map directly to the functions in Python’s built-in os and os.path modules) the file system name is used for the authority. (If path is from a named virtual file system and is a UNC path then \texttt{URIException} is raised.)

classmethod from_path(path)
    Converts a local file path into a URI instance.

    path A file path string.

    Uses \texttt{path} to create an instance of \texttt{pyslet.vfs.OSFilePath}, see \texttt{from_virtual_path()} for more info.
	octets = None
    The character string representing this URI’s octets

fragment = None
    The fragment string that was appended to the URI or None if no fragment was given.

scheme = None
    The URI scheme, if present

authority = None
    The authority (e.g., host name) of a hierarchical URI

abs_path = None
    The absolute path of a hierarchical URI (None if the path is relative)

query = None
    The optional query associated with a hierarchical URI

scheme_specific_part = None
    The scheme specific part of the URI
\end{verbatim}
rel_path = None
The relative path of a hierarchical URI (None if the path is absolute)

opaque_part = None
None if the URI is hierarchical, otherwise the same as scheme_specific_part

parse_scheme_specific_part()
Parses the scheme specific part of the URI

This method should overridden by derived classes if they use a format other than the hierarchical URI format described in RFC2396.

The default implementation implements the generic parsing of hierarchical URI setting the following attribute values: authority, abs_path and query. If the URI is not of a hierarchical type then opaque_part is set instead. Unset attributes have the value None.

canonicalize()
Returns a canonical form of this URI

For unknown schemes we simply convert the scheme to lower case so that, for example, X-scheme:data becomes x-scheme:data.

Derived classes should apply their own transformation rules.

get_canonical_root()
Returns a new URI comprised of the scheme and authority only.

Only valid for absolute URI, returns None otherwise.

The canonical root does not include a trailing slash. The canonical root is used to define the domain of a resource, often for security purposes.

If the URI is non-hierarchical then the just the scheme is returned.

resolve(base, current_doc_ref=None)
Resolves a relative URI against a base URI

base A URI instance representing the base URI against which to resolve this URI. You may also pass a URI string for this parameter.

current_doc_ref The optional current_doc_ref allows you to handle the special case of resolving the empty URI. Strictly speaking, fragments are not part of the URI itself so a relative URI consisting of the empty string, or a relative URI consisting of just a fragment both refer to the current document.

By default, current_doc_ref is assumed to be the same as base but there are cases where the base URI is not the same as the URI used to originally retrieve the document and this optional parameter allows you to cope with those cases.

Returns a new URI instance.

If the base URI is also relative then the result is a relative URI, otherwise the result is an absolute URI. The RFC does not actually go into the procedure for combining relative URI but if B is an absolute URI and R1 and R2 are relative URI then using the resolve operator ([*], see above):

\[
U_1 = B \[*\] R_1 \\
U_2 = U_1 \[*\] R_2 \\
U_2 = ( B \[*\] R_1 ) \[*\] R_2
\]

The last expression prompts the issue of associativity, in other words, is the following expression also valid?
**U2 = B [*] ( R1 [*] R2 )**

For this to work it must be possible to use the resolve operator to combine two relative URI to make a third, which is what we allow here.

**relative (base)**

Calculates a URI expressed relative to base.

**base** A URI instance representing the base URI against which to calculate the relative URI. You may also pass a URI string for this parameter.

Returns a new URI instance.

As we allow the `resolve()` method for two relative paths it makes sense for the Relative operator to also be defined:

\[ R3 = R1 [*] R2 \]
\[ R3 [/] R1 = R2 \]

There are some significant restrictions, URI are classified by how specified they are with:

- absolute URI > authority > absolute path > relative path

If R is absolute, or simply more specified than B on the above scale and:

\[ U = B [*] R \]

then U = R regardless of the value of B and therefore:

\[ U [/] B = U \] if B is less specified than U

Also note that if U is a relative URI then B cannot be absolute. In fact B must always be less than, or equally specified to U because B is the base URI from which U has been derived:

\[ U [/] B = undefined \] if B is more specified than U

Therefore the only interesting cases are when B is equally specified to U. To give a concrete example:

\[ U = /HD/User/setting.txt \]
\[ B = /HD/folder/file.txt \]

\[ /HD/User/setting.txt [/\] /HD/folder/file.txt = ../User/setting.txt \]
\[ /HD/User/setting.txt = /HD/folder/file.txt [*] ../User/setting.txt \]

And for relative paths:

\[ U = User/setting.txt \]
\[ B = User/folder/file.txt \]

\[ User/setting.txt [/\] User/folder/file.txt = ../setting.txt \]
\[ User/setting.txt = User/folder/file.txt [*] ../setting.txt \]

**match (other_uri)**

Compares this URI with another

**other_uri** Another URI instance.

Returns True if the canonical representations of the URIs match.
**is_absolute()**
Returns True if this URI is absolute
An absolute URI is fully specified with a scheme, e.g., ‘http’.

**get_file_name()**
Gets the file name associated with this resource
Returns None if the URI scheme does not have the concept. By default the file name is extracted from
the last component of the path. Note the subtle difference between returning None and returning an empty
string (indicating that the URI represents a directory-like object).
The return result is always a character string.

class pyslet.rfc2396.ServerBasedURL(octets)
Bases: pyslet.rfc2396.URI
Represents server-based URI
A server-based URI is one of the form:

```
<scheme> '://' [userinfo '@'] <host> [':' <port>] <path>
```

**DEFAULT_PORT = None**
the default port for this type of URL

**get_addr()**
Returns a hostname and integer port tuple
The format is suitable for socket operations. The main purpose of this method is to determine if the port is
set on the URL and, if it isn’t, to return the default port for this URL type instead.

**canonicalize()**
Returns a canonical form of this URI
In addition to returning the scheme in lower-case form, this method forces the host to be lower case and
removes the port specifier if it matches the `DEFAULT_PORT` for this type or URI.
No transformation is performed on the path component.

class pyslet.rfc2396.FileURL(octets='file:///')
Bases: pyslet.rfc2396.ServerBasedURL
Represents the file URL scheme defined by RFC1738
Do not create instances directly, instead use (for example):

```
furl = URI.from_octets('file:///...')
```

**get_pathname**(force8bit=False)
Returns the system path name corresponding to this file URL
If the system supports unicode file names (as reported by `os.path.supports_unicode_filenames`) then
get_pathname also returns a unicode string, otherwise it returns an 8-bit string encoded in the underly-
ing file system encoding.

**force8bit** There are some libraries (notably sax) that will fail when passed files opened using unicode
paths. The force8bit flag can be used to force get_pathname to return a byte string encoded using the
native file system encoding.
If the URL does not represent a path in the native file system then URIException is raised.
get_virtual_file_path()
Returns a virtual file path corresponding to this URL.
The result is a `pyslet.vfs.FilePath` instance.
The host component of the URL is used to determine which virtual file system the file belongs to. If there is no virtual file system matching the URL's host and the native file system support UNC paths (i.e., is Windows) the host will be placed in the machine portion of the UNC path.
Path parameters e.g., /dir/file;lang=en in the URL are ignored.

to_local_text()
Returns a locally portable version of the URL.
The result is a character string, not a URI instance.
In Pyslet, all hierarchical URI are treated as using the UTF-8 encoding for characters outside US ASCII. As a result, file URL are expressed using percent-encoded UTF-8 multi-byte sequences. When converting these URLs to file paths the difference is taken into account correctly but if you attempt to output a URL generated by Pyslet and use it in another application you may find that the URL is not recognised. This is particularly a problem on Windows where file URLs are expected to be encoded with the native file system encoding.
The purpose of this method is to return a version of the URL re-encoded in the local file system encoding for portability such as being copy-pasted into a browser address bar.

6.4.3 Canonicalization and Escaping

pyslet.rfc2396.canonicalize_data(source, unreserved_test=is_unreserved, allowed_test=is_allowed)
Returns the canonical form of source string.
The canonical form is the same string but any unreserved characters represented as hex escapes in source are unencoded and any unescaped characters that are neither reserved nor unreserved are escaped.
source A string of characters. Characters must be in the US ASCII range. Use `encode_unicode_uri()` first if necessary. Will raise UnicodeEncodeError if non-ASCII characters are encountered.
unreserved_test A function with the same signature as `is_unreserved()`, which it defaults to. By providing a different function you can control which characters will have their escapes removed. It does not affect which unescaped characters are escaped.
To give an example, by default the '.' is unreserved so the sequence `%2E` will be removed when canonicalizing the source. However, if the specific part of the URL scheme you are dealing with applies some reserved purpose to '.' then `source` may contain both encoded and unencoded versions to disambiguate its usage. In this case you would want to remove '.' from the definition of unreserved to prevent it being unescaped.
If you don’t want any escapes removed, simply pass:

```
lambda x: False
```

allowed_test Defaults to `is_allowed()`
See `parse_uric()` for more information.
All hex escapes are promoted to upper case.
pyslet.rfc2396.escape_data(source, reserved_test=is_reserved, allowed_test=is_allowed)
Performs URI escaping on source
Returns the escaped character string.
source  The input string. This can be a binary or character string. For character strings all characters must be in
the US ASCII range. Use encode_unicode_uri() first if necessary. Will raise UnicodeEncodeError
if non-ASCII characters are encountered. For binary strings there is no constraint on the range of allowable
octets.

Note: In Python 2 the ASCII character constraint is only applied when source is of type unicode.

reserved_test  Default is_reserved(), the function to test if a character should be escaped. This function
should take a single character as an argument and return True if the character must be escaped. Characters
for which this function returns False will still be escaped if they are not allowed to appear unescaped in
URI (see allowed_test below).

Quoting from RFC2396:

Characters in the “reserved” set are not reserved in all contexts. The set of characters actually
reserved within any given URI component is defined by that component. In general, a character
is reserved if the semantics of the URI changes if the character is replaced with its escaped US-
ASCII encoding.

Therefore, you may want to reduce the set of characters that are escaped based on the target component
for the data. Different rules apply to a path component compared with, for example, the query string. A
number of alternative test functions are provided to assist with escaping an alternative set of characters.

For example, suppose you want to ensure that your data is escaped to the rules of the earlier RFC1738.
In that specification, a fore-runner of RFC2396, the “~” was not classified as a valid URL character and
required escaping. It was later added to the mark category enabling it to appear unescaped. To ensure that
this character is escaped for compatibility with older systems you might do this when escaping data with a
path component (where ‘~’ is often used):

```python
path_component = uri.escape_data(
    dir_name, reserved_test=uri.is_reserved_1738)
```

In addition to escaping “~”, the above will also leave “$”, “+” and “,” unescaped as they were classified as
‘extra’ characters in RFC1738 and were not reserved.

allowed_test  Defaults to is_allowed()

See parse_uric() for more information.

By default there is no difference between RFC2396 and RFC2732 in operation as in RFC2732 “[” and
“]” are legal URI characters but they are also in the default reserved set so will be escaped anyway. In
RFC2396 they were escaped on the basis of not being allowed.

The difference comes if you are using a reduced set of reserved characters. For example:

```python
>>> print uri.escape_data("[file].txt")
%5Bfile%5D.txt
>>> print uri.escape_data(
    "[file].txt", reserved_test=uri.is_path_segment_reserved)
[file].txt
>>> print uri.escape_data(
    "[file].txt", reserved_test=uri.is_path_segment_reserved,
    allowed_test=uri.is_allowed_2396)
%5Bfile%5D.txt
```

pyslet.rfc2396.unescape_data(source)

Performs URI unescaping

source  The URI-encoded string
Removes escape sequences. The string is returned as a binary string of octets, not a string of characters. Escape sequences such as `%E9 will result in the byte value 233 and not the character é.

The character encoding that applies may depend on the context and it cannot always be assumed to be UTF-8 (though in most cases that will be the correct way to interpret the result).

```python
pyslet.rfc2396.path_sep = u'/'
```

Constant for “/” character.

### 6.4.4 Basic Syntax

RFC2396 defines a number of character classes (see `pyslet.unicode5.CharClass`) to assist with the parsing of URI.

The bound test method of each class is exposed for convenience (you don’t need to pass an instance). These pseudo-functions therefore all take a single character as an argument and return True if the character matches the class. They will also accept None and return False in that case.

```python
pyslet.rfc2396.is_upalpha(c)
```
Tests production: upalpha

```python
pyslet.rfc2396.is_lowalpha(c)
```
Tests production: lowalpha

```python
pyslet.rfc2396.is_alpha(c)
```
Tests production: alpha

```python
pyslet.rfc2396.is_digit(c)
```
Tests production: digit

```python
pyslet.rfc2396.is_alphanum(c)
```
Tests production: alphanum

```python
pyslet.rfc2396.is_reserved(c)
```
Tests production: reserved

The reserved characters are:

```
; | / | ? | : | @ | & | = | + | $ | , | [ | ]
```

This function uses the larger reserved set defined by the update in RFC2732. The additional reserved characters are “[” and “]” which were not originally part of the character set allowed in URI by RFC2396.

```python
pyslet.rfc2396.is_reserved_2396(c)
```
Tests production: reserved

The reserved characters are:

```
; | / | ? | : | @ | & | = | + | $ | ,
```

This function enables strict parsing according to RFC2396, for general use you should use `is_reserved()` which takes into consideration the update in RFC2732 to accommodate IPv6 literals.

```python
pyslet.rfc2396.is_reserved_1738(c)
```
Tests production: reserved

The reserved characters are:

```
; | / | ? | : | @ | & | =
```

This function enables parsing according to the earlier RFC1738.

### 6.4. Uniform Resource Identifiers (RFC2396)
pyslet.rfc2396.is_unreserved(c)
Tests production: unreserved

Despite the name, some characters are neither reserved nor unreserved.

pyslet.rfc2396.is_unreserved_1738(c)
Tests production: unreserved

Tests the definition of unreserved from the earlier RFC1738. The following characters were considered 'safe'
in RFC1738 (and so are unreserved there) but were later classified as reserved in RFC2396:

```
$ | + | ,
```

The “~” is considered unreserved in RFC2396 but is neither reserved nor unreserved in RFC1738 and so therefore must be escaped for compatibility with early URL parsing systems.

pyslet.rfc2396.is_safe_1738(c)
Test production: safe (RFC 1738 only)

The safe characters are:

```
$ | - | _ | . | +
```

pyslet.rfc2396.is_extra_1738(c)
Test production: safe (RFC 1738 only)

The safe characters are:

```
! | * | ' | ( | ) | ,
```

pyslet.rfc2396.is_mark(c)
Tests production: mark

The mark characters are:

```
- | _ | . | ! | ~ | * | ' | ( | ) | ,
```

pyslet.rfc2396.is_allowed(c)
Convenience function for testing allowed characters

Returns True if c is a character allowed in a URI according to the looser definitions of RFC2732, False otherwise. A character is allowed (unescaped) in a URI if it is either reserved or unreserved.

pyslet.rfc2396.is_allowed_2396(c)
Convenience function for testing allowed characters

Returns True if c is a character allowed in a URI according to the stricter definitions in RFC2396, False otherwise. A character is allowed (unescaped) in a URI if it is either reserved or unreserved.

pyslet.rfc2396.is_allowed_1738(c)
Convenience function for testing allowed characters

Returns True if c is a character allowed in a URI according to the older definitions in RFC1738, False otherwise. A character is allowed (unescaped) in a URI if it is either reserved or unreserved.

pyslet.rfc2396.is_hex(c)
Tests production: hex

Accepts upper or lower case forms.

pyslet.rfc2396.is_control(c)
Tests production: control
pyslet.rfc2396.is_space(c)
Tests production: space

pyslet.rfc2396.is_delims(c)
Tests production: delims
The delims characters are:

"<" | ">" | "@" | "#" | ";" | ">"

pyslet.rfc2396.is_unwise(c)
Tests production: unwise
The unwise characters are:

"{" | "}" | ";" | "|" | ";\" | "\" | ";^" | ";`"

This function uses the smaller unwise set defined by the update in RFC2732. The characters “[” and “]” were removed from this set in order to support IPv6 literals.

This function is provided for completeness and is not used internally for parsing URLs.

pyslet.rfc2396.is_unwise_2396(c)
Tests production: unwise
The unwise characters are:

"{" | "}" | ";" | "|" | ";\" | "\" | ";^" | ";`" | "[" | "]" | ";`

This function enables strict parsing according to RFC2396, the definition of unwise characters was updated in RFC2732 to exclude “[” and “]”.

pyslet.rfc2396.is_authority_reserved(c)
Convenience function for parsing production authority

Quoting the specification of production authority:

Within the authority component, the characters “;”, “:”, “@”, “?”, and “/” are reserved

pyslet.rfc2396.is_path_segment_reserved(c)
Convenience function for escaping path segments

From RFC2396:

Within a path segment, the characters “/”, “;”, “=”, and “?” are reserved.

pyslet.rfc2396.is_query_reserved(c)
Convenience function for escaping query strings

From RFC2396:

Within a query component, the characters “;”, “/”, “?”, “:”, “@”, “&”, “=”, “+”, “,”, and “$” are reserved

Some fragments of URI parsing are exposed for reuse by other modules.

pyslet.rfc2396.parse_uric(source, pos=0, allowed_test=is_allowed)

Returns the number of URI characters in a source string

source A source string (of characters)
pos The place at which to start parsing (defaults to 0)
allowed_test Defaults to is_allowed()

Test function indicating if a character is allowed unencoded in a URI. For stricter RFC2396 compliant parsing you may also pass is_allowed_2396() or is_allowed_1738().

For information, RFC2396 added “~” to the range of allowed characters and RFC2732 added “[” and “]” to support IPv6 literals.

This function can be used to scan a string of characters for a URI, for example:

```python
x = "http://www.pyslet.org/ is great"
url = x[:parse_uri(x, 0)]
```

It does not check the validity of the URI against the specification. The purpose is to allow a URI to be extracted from some source text. It assumes that all characters that must be encoded in URI are encoded, so characters outside the ASCII character set automatically terminate the URI as do any unescaped characters outside the allowed set (defined by the allowed_test). See encode_unicode_uri() for details of how to create an appropriate source string in contexts where non-ASCII characters may be present.

pyslet.rfc2396.split_server(authority)
Splits an authority component

authority A character string containing the authority component of a URI.

Returns a triple of:

```python
(userinfo, host, port)
```

There is no parsing of the individual components which may or may not be syntactically valid according to the specification. The userinfo is defined as anything up to the “@” symbol or None if there is no “@”. The port is defined as any digit-string (possibly empty) after the last “:” character or None if there is no “:” or if there is non-empty string containing anything other than a digit after the last “:”.

The return values are always character strings (or None). There is no unescaping or other parsing of the values.

pyslet.rfc2396.split_path(path, abs_path=True)
Splits a URI-encoded path into path segments

path A character string containing the path component of a URI. If path is None we treat as for an empty string.

abs_path A flag (defaults to True) indicating whether or not the path is relative or absolute. This flag only affects the handling of the empty path. An empty absolute path is treated as if it were ‘/’ and returns a list containing a single empty path segment whereas an empty relative path returns a list with no path segments, in other words, an empty list.

The return result is always a list of character strings split from path. It will only end in an empty path segment if the path ends with a slash.

pyslet.rfc2396.split_abs_path(path, abs_path=True)
Provided for backwards compatibility

Equivalent to:

```python
split_path(abs_path, True)
```

pyslet.rfc2396.split_rel_path(rel_path)
Provided for backwards compatibility

Equivalent to:

```python
split_path(abs_path, False)
```
normalize_segments(path_segments)

Normalizes a list of path_segments

path_segments A list of character strings representing path segments, for example, as returned by split_path().

Normalizing follows the rules for resolving relative URI paths, ‘.’ and trailing ‘.’ are removed, ‘seg/../’ and trailing seg/.. are also removed.

6.4.5 Exceptions

class pyslet.rfc2396.URIException
    Bases: exceptions.Exception
    Base class for URI-related exceptions

class pyslet.rfc2396.URIRelativeError
    Bases: pyslet.rfc2396.URIException
    Exceptions raised while resolve relative URI

6.4.6 Legacy

The following definitions are provided for backwards compatibility only.

pyslet.rfc2396.URIFactory
    An instance of URIFactoryClass that can be used for creating URI instances.

class pyslet.rfc2396.URIFactoryClass
    Bases: pyslet.pep8.PEP8Compatibility

6.5 Uniform Resource Names (RFC2141)

This module defines functions and classes for working with URI as defined by RFC2141: http://www.ietf.org/rfc/rfc2141.txt

6.5.1 Creating URN Instances

URN instances are created automatically by the from_octets() method and no special action is required when parsing them from character strings.

If you are in a URN specific context you may perform a looser parse of a URN from a surrounding character stream using parse_urn() but the return result is a character string rather than a URN instance.

Finally, you can construct a URN from a namespace identifier and namespace specific string directly. The resulting object can then be converted directly to a well-formatted URN using string conversion or used in any context where a URI instance is required.

parse_urn(src)

 Parses a run of URN characters from a string

 src A character string containing URN characters. Will accept binary strings encoding ASCII characters (only).

 returns the src up to, but not including, the first character that fails to match the production for URN char (as a character string).
6.5.2 URN

class pyslet.urn.URN(octets=None, nid=None, nss=None)

Bases: pyslet.rfc2396.URI

Represents a URN.

There are two forms of constructor, the first uses a single positional argument and matches the constructor for
the base URI class. This enables URNs to be created automatically from from_octets().

octets A character string containing the URN.

The second form of constructor allows you to construct a URN from a namespace identifier and a namespace-
specific string, both values are required in this form of the constructor.

nid The namespace identifier, a string.

nss The namespace-specific string, encoded appropriately for inclusion in a URN.

ValueError is raised if the arguments are not passed correctly, URIException is raised if there a problem
 parsing or creating the URN itself.

nid = None
the namespace identifier for this URN

nss = None
the namespace specific part of the URN

6.5.3 Translating to and from Text

pyslet.urn.translate_to_urnchar(src, reserved_test=is_reserved)

Translates a source string into URN characters

src A binary or unicode string. In the latter case the string is encoded with utf-8 as part of being translated, in
the former case it must be a valid UTF-8 string of bytes.

reserved_test A function that tests if a character is reserved. It defaults to is_reserved() but can be any
function that takes a single argument and returns a boolean. You can’t prevent a character from being
encoded with this function (even if you pass lambda x:False, but you can add additional characters to the
list of those that should be escaped. For example, to encode the ‘.’ character you could pass:

\[
\text{lambda } x: x=='.'
\]

The result is a URI-encode string suitable for adding to the namespace-specific part of a URN.

pyslet.urn.translate_from_urnchar(src)

Translates a URN string into an unencoded source string

The main purpose of this function is to remove %-encoding but it will also check for the illegal 0-byte and raise
an error if one is encountered.

Returns a character string without %-escapes. As part of the conversion the implicit UTF-8 encoding is removed.

6.5.4 Basic Syntax

The module defines a number of character classes (see pyslet.unicode5.CharClass) to assist with the parsing
of URN.
The bound test method of each class is exposed for convenience (you don’t need to pass an instance). These pseudo-functions therefore all take a single character as an argument and return True if the character matches the class. They will also accept None and return False in that case.

```python
def is_upper(c):
    return c.isupper()
def is_lower(c):
    return c.islower()
def is_number(c):
    return c.isdigit()
def is_letnum(c):
    return c.isalnum()
def is_letnumhyp(c):
    return True if c in 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0-9'
def is_reserved(c):
    return c in '%/%/?/#/

def is_other(c):
    return c in '()+,.-:_=@;$/!

def is_trans(c):
    return c in 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0-9_=-@;$/!

def is_hex(c):
    return c in '0123456789abcdefABCDEF'
```

6.5. Uniform Resource Names (RFC2141)
6.6 The Atom Syndication Format (RFC4287)

This module defines functions and classes for working with the Atom Syndication Format as defined by RFC4287: http://www.ietf.org/rfc/rfc4287.txt

6.6.1 Reference

Elements

```
class pyslet.rfc4287.Feed(parent)
    Bases: pyslet.rfc4287.Source
    Represents an Atom feed.
    This is the document (i.e., top-level) element of an Atom Feed Document, acting as a container for metadata and data associated with the feed
    AtomIdClass
        alias of AtomId
    TitleClass
        alias of Title
    UpdatedClass
        alias of Updated
    Entry = None
        atomEntry

class pyslet.rfc4287.Source(parent)
    Bases: pyslet.rfc4287.Entity
    Metadata from the original source feed of an entry.
    This class is also used a base class for Feed.
    Generator = None
        atomGenerator
    Icon = None
        atomIcon
    Logo = None
        atomLogo
    Subtitle = None
        atomSubtitle

class pyslet.rfc4287.Entry(parent)
    Bases: pyslet.rfc4287.Entity
    Represents an individual entry
    Acts as a container for metadata and data associated with the entry.
    AtomIdClass
        alias of AtomId
    TitleClass
        alias of Title
```
**UpdatedClass**
alias of *Updated*

**LinkClass**
alias of *Link*

class *pyslet.rfc4287.Entity*(parent)
Bases: *pyslet.rfc4287.AtomElement*

Base class for feed, entry and source elements.

**LinkClass**
alias of *Link*

**AtomId** = None
the atomId of the object

Note that we qualify the class name used to represent the id to avoid confusion with the existing ‘id’ attribute in *Element*.

**Author** = None
atomAuthor

**Category** = None
atomCategory

**Contributor** = None
atomContributor

**Link** = None
atomLink

**Rights** = None
atomRights

**Title** = None
atomTitle

**Updated** = None
atomUpdated

class *pyslet.rfc4287.Author*(parent)
Bases: *pyslet.rfc4287.Person*

A Person construct that indicates the author of the entry or feed.

class *pyslet.rfc4287.Category*(parent)
Bases: *pyslet.rfc4287.AtomElement*

Information about a category associated with an entry or feed.

**term** = None
a string that identifies the category to which the entry or feed belongs

**scheme** = None
an IRI that identifies a categorization scheme.

This is not converted to a *pyslet.rfc2396.URI* instance as it is not normally resolved to a resource. Instead it defines a type of namespace.

**label** = None
a human-readable label for display in end-user applications

6.6. The Atom Syndication Format (RFC4287)
class pyslet.rfc4287.Content(parent)
    Bases: pyslet.rfc4287.Text
    Contains or links to the content of the entry.
    Although derived from Text this class overloads the meaning of the Text.type attribute allowing it to be a media type.
    src = None
        link to remote content
    get_value()
        Gets a single string representing the value of the element.
        Overloads the basic get_value(), if type is a media type rather than one of the text types then a ValueError is raised.

class pyslet.rfc4287.Contributor(parent)
    Bases: pyslet.rfc4287.Person
    A Person construct representing a contributor
    Indicates a person or other entity who contributed to the entry or feed.

class pyslet.rfc4287.Generator(parent)
    Bases: pyslet.rfc4287.AtomElement
    Identifies the agent used to generate a feed
    The agent is used for debugging and other purposes.
    uri = None
        the uri of the tool used to generate the feed
    version = None
        the version of the tool used to generate the feed
    set_pyslet_info()
        Sets this generator to a default value
        A representation of this Pyslet module.

class pyslet.rfc4287.Icon(parent)
    Bases: pyslet.rfc4287.AtomElement
    An image that provides iconic visual identification for a feed.
    uri = None
        a URI instance representing the URI of the icon
    get_value()
        Returning a pyslet.rfc2396.URI instance.
    set_value(value)
        Enables the value to be set from a URI instance.
        If value is a string it is used to set the element’s content. content_changed() is then called to update the value of uri. If value is a URI instance then uri is set directory and it is then converted to a string and used to set the element’s content.
    content_changed()
        Sets uri accordingly.

class pyslet.rfc4287.AtomId(parent, name=None)
    Bases: pyslet.rfc4287.AtomElement
A permanent, universally unique identifier for an entry or feed.

```python
class pyslet.rfc4287.Link (parent)
Bases: pyslet.rfc4287.AtomElement

A reference from an entry or feed to a Web resource.

href = None
    a URI instance, the link’s IRI

rel = None
    a string indicating the link relation type

type = None
    an advisory media type

hreflang = None
    the language of the resource pointed to by href

title = None
    human-readable information about the link

length = None
    an advisory length of the linked content in octets
```

```python
class pyslet.rfc4287.Logo (parent)
Bases: pyslet.rfc4287.Icon

An image that provides visual identification for a feed.
```

```python
class pyslet.rfc4287.Published (parent)
Bases: pyslet.rfc4287.Date

A Date construct indicating an instant in time associated with an event early in the life cycle of the entry.
```

```python
class pyslet.rfc4287.Rights (parent)
Bases: pyslet.rfc4287.Text

A Text construct that conveys information about rights held in and over an entry or feed.
```

```python
class pyslet.rfc4287.Subtitle (parent)
Bases: pyslet.rfc4287.Text

A Text construct that conveys a human-readable description or subtitle for a feed.
```

```python
class pyslet.rfc4287.Summary (parent)
Bases: pyslet.rfc4287.Text

A Text construct that conveys a short summary, abstract, or excerpt of an entry.
```

```python
class pyslet.rfc4287.Title (parent)
Bases: pyslet.rfc4287.Text

A Text construct that conveys a human-readable title for an entry or feed.
```

```python
class pyslet.rfc4287.Updated (parent)
Bases: pyslet.rfc4287.Date

A Date construct indicating the most recent instant in time when an entry or feed was modified in a way the publisher considers significant.
```
Base Classes

class pyslet.rfc4287.Person (parent)
    Bases: pyslet.rfc4287.AtomElement

    An element that describes a person, corporation, or similar entity

    NameClass
        alias of Name

    URI
        alias of URI

class pyslet.rfc4287.Name (parent, name=None)
    Bases: pyslet.rfc4287.AtomElement

    A human-readable name for a person.

class pyslet.rfc4287.URI (parent, name=None)
    Bases: pyslet.rfc4287.AtomElement

    An IRI associated with a person

class pyslet.rfc4287.Email (parent, name=None)
    Bases: pyslet.rfc4287.AtomElement

    An e-mail address associated with a person

class pyslet.rfc4287.Text (parent)
    Bases: pyslet.rfc4287.AtomElement

    Base class for atomPlainTextConstruct and atomXHTMLTextConstruct.

    set_value (value, type=1)
        Sets the value of the element. type must be a value from the TextType enumeration

        Overloads the basic SetValue() implementation, adding an additional type attribute to enable the value
        to be set to either a plain TextType.text, TextType.html or TextType.xhtml value. In the case of an xhtml
        type, value is parsed for the required XHTML div element and this becomes the only child of the element.
        Given that the div itself is not considered to be part of the content the value can be given without the
        enclosing div, in which case it is generated automatically.

    get_value ()
        Gets a single unicode string representing the value of the element.

        Overloads the basic get_value() implementation to add support for text of type xhtml.

        When getting the value of TextType.xhtml text the child div element is not returned as it is not considered
        to be part of the content.

class pyslet.rfc4287.TextType
    Bases: pyslet.xml.xsdatatypes Enumeration

    text type enumeration:

        "text" | "html" | "xhtml"

    This enumeration is used for setting the Text.type attribute.

    Usage: TextType.text, TextType.html, TextType.xhtml

class pyslet.rfc4287.Date (parent)
    Bases: pyslet.rfc4287.AtomElement

    An element conforming to the definition of date-time in RFC3339.
This class is modeled using the iso8601 module.

def date = None

    a TimePoint instance representing this date

get_value()

    Overrides get_value(), returning a pyslet.iso8601.TimePoint instance.

set_value(value)

    Overrides SetValue(), enabling the value to be set from a pyslet.iso8601.TimePoint instance.

    If value is a string the behaviour is unchanged, if value is a TimePoint instance then it is formatted using
    the extended format of ISO 8601 in accordance with the requirements of the Atom specification.

collection_changed()

    Re-reads the value of the element and sets date accordingly.

class pyslet.rfc4287.AtomElement (parent, name=None)

    Bases: pyslet.xml.namespace.NSElement

    Base class for all APP elements.

    All atom elements can have xml:base and xml:lang attributes, these are handled by the Element base class.

    See GetLang() and SetLang(), GetBase() and SetBase()

Constants

pyslet.rfc4287.ATOM_NAMESPACE = 'http://www.w3.org/2005/Atom'

    The namespace to use for Atom Document elements

pyslet.rfc4287.ATOM_MIMETYPE = 'application/atom+xml'

    The mime type for Atom Document

6.7 The Atom Publishing Protocol (RFC5023)

This module defines functions and classes for working with the Atom Publishing Protocol as defined by RFC5023:
http://www.ietf.org/rfc/rfc5023.txt

6.7.1 Reference

Elements

class pyslet.rfc5023.Service (parent)

    Bases: pyslet.rfc5023.APPElement

    The container for service information

    Associated with one or more Workspaces.

    Workspace = None

        a list of Workspace instances

class pyslet.rfc5023.Workspace (parent)

    Bases: pyslet.rfc5023.APPElement

    Workspaces are server-defined groups of Collections.
Title = None
the title of this workspace

Collection = None
a list of Collection

class pyslet.rfc5023.Collection(parent)
Bases: pyslet.rfc5023.APPElement
Describes a collection (feed).

Title = None
the URI of the collection (feed)

Accept = None
the human readable title of the collection

Categories = None
list of Accept media ranges that can be posted to the collection

get_feed_url()
Returns a fully resolved URL for the collection (feed).

GetFeedURL(*args, **kwargs)
Deprecated equivalent to get_feed_url()

class pyslet.rfc5023.Categories(parent)
Bases: pyslet.rfc5023.APPElement
The root of a Category Document.

A category document is a document that describes the categories allowed in a collection.

fixed = None
an optional URI to the category

scheme = None
indicates whether the list of categories is a fixed set. By default they’re open.

Category = None
identifies the default scheme for categories defined by this element

Base Classes

class pyslet.rfc5023.Accept(parent, name=None)
Bases: pyslet.rfc5023.APPElement
Represents the accept element.

class pyslet.rfc5023.Document(**args)
Bases: pyslet.rfc4287.AtomDocument
Class for working with APP documents.
This call can represent both APP and Atom documents.

ValidateMimeType(mimetype)
Checks mimetype against the APP or Atom specifications.

classmethod get_element_class(name)
Returns the APP or Atom class used to represent name.

Overrides get_element_class() when the namespace is APP_NAMESPACE.
class pyslet.rfc5023.APPElement (parent, name=None)
Bases: pyslet.xml.namespace.NSElement

Base class for all APP elements.
All APP elements can have xml:base, xml:lang and/or xml:space attributes. These are handled by the base Element base class.

Constants

pyslet.rfc5023.APP_NAMESPACE = 'http://www.w3.org/2007/app'
The namespace to use for Atom Publishing Protocol elements

pyslet.rfc5023.ATOMSVC_MIMETYPE = 'application/atomsvc+xml'
The mime type for service documents

pyslet.rfc5023.ATOMCAT_MIMETYPE = 'application/atomcat+xml'
The mime type for category documents

6.8 ISO 8601 Dates and Times

This module defines special classes for handling ISO 8601 dates and times.

class pyslet.iso8601.Date (src=None, base=None, bce=False, century=None, decade=None,
year=None, month=None, day=None, week=None, weekday=None, ordinal_day=None, absolute_day=None, xdigits=None, **kwargs)

A class for representing ISO dates.
Values can represent dates with reduced precision, for example:

Date(century=20, year=13, month=12)

represents December 2013, no specific day.

There are a number of different forms of the constructor based on named parameters, the simplest is:

Date(century=19, year=69, month=7, day=20)

You can also use weekday format (decade must be provided separately when using this format):

Date(century=19, decade=6, year=9, week=29, weekday=7)

...ordinal format (where day 1 is 1st Jan):
Date(century=19, year=69, ordinal_day=201)

...absolute format (where day 1 is the notional 1st Jan 0001):
Date(absolute_day=718998)

An empty constructor is equivalent to:

Date() == Date(absolute_day=1)  # True
By default the calendar used supports dates in the range 0001-01-01 through to 9999-12-31. ISO 8601 allows this range to be extended by agreement using a fixed number of additional digits in the century specification. These dates are referred to as expanded dates and they include provision for negative, as well as larger positive, years using the astronomical convention of including a year 0.

Given that the use of expanded dates can only be done by agreement the constructor supports an additional parameter to enable you to construct a date using an agreed number of additional digits:

```python
Date(century=19, year=69, month=7, day=20, xdigits=2)
```

The above instance represents the same date as the previous examples but with an expanded century representation consisting of an additional two decimal digits. Using xdigits=2 the range of allowable dates is now -999999-01-01 to +999999-12-31. Notice that ISO 8601 uses the leading +/- to indicate the use of an expanded form. If one instance is used to create another, e.g., using `offset()` or the base parameter described below then value of xdigits is copied from the existing instance to the new one.

It is acceptable for xdigits to be set to 0, this indicates expanded dates with no additional decimal digits but has the effect of extending the default range to -9999-01-01 to +9999-12-31.

When constructing instances for negative years you must set the bce flag on the constructor (indicating that the date is “before common era”). The values passed for century and year (and optionally decade when using weekday form) must always be positive as they would be written in the documented ISO decimal forms.

Expanded dates include the year 0 (as per ISO 8601). As a result, the common meaning of 1 BCE would be year 0, not year -1. To represent the year 753 BCE you would use:

```python
Date(bce=True, century=7, year=52, xdigits=0)
```

For year 0, the bce flag can be set either way (or omitted).

The constructor includes a wildcard form of expansion using the special value -1 for xdigits. Such dates are assumed to have been represented in the minimum number of decimal digits for the century (but not less than 4) and will accept a century of any size.

All constructors, except the absolute form, allow the passing of a base date which allows the most-significant values to be omitted, (truncated forms) for example:

```python
base = Date(century=19, year=69, month=7, day=20)
new_date = Date(day=21, base=base) # 21st July 1969
```

*base* always represents a date on or before the newly constructed date, so:

```python
base = Date(century=19, year=99, month=12, day=31)
new_date = Date(day=5, base=base)
```

constructs a Date representing the 5th January 2000. These truncated forms cannot be used with xdigits as the century is never present so cannot be expanded. However, the value of *base* may be an expanded date and the result is another expanded date with the same xdigits constraint. Caution is required when dealing with negative dates:

```python
base = Date(bce=True, century=7, year=52, month=4, day=21, xdigits=0)
new_date = Date(year=53, month=4, day=21, base=base)
```

results in the date -0653-04-21 and not -0753-04-21 because the year -0753 would have been before the base year -0752.

Given that Date can hold imprecise dates, there is some ambiguity over the comparisons between things such as January 1985 and Week 3 1985. Although at first sight it may be tempting to declare 1st April to be greater
than March, it is harder to determine the relationship between 1st April and April itself. Especially if a complete ordering is required.

The approach taken here is to disallow comparisons between dates with different precisions.

Date objects are immutable and so can be used as the keys in dictionaries provided they all share the same precision.

Some older functions did allow modification but these now raise an error with an appropriate suggested refactoring.

Instances can be converted directly to strings using the default, extended calendar format. Other formats are supported through format specific methods.

```python
xdigits = None
  the number of expanded digits in the century
bce = None
  BCE flag (before common era)
century = None
  the century
year = None
  the year, 0..99
month = None
  the month, 1..12 (for dates stored in calendar form)
week = None
  the week (for dates stored in week form)

  Fully specified dates are always stored in calendar form but instances can represent reduced precision dates in week format, e.g., 2016-W01. In these cases, day and month will be None and the week will be recorded instead.

day = None
  the day, 1..31
get_absolute_day()
  Return a notional day number
  The number 1 being the 0001-01-01 which is the base day of our calendar.
get_calendar_day()
  Returns a tuple of: (century, year, month, day)
get_xcalendar_day()
  Returns a tuple of: (bce, century, year, month, day)
get_ordinal_day()
  Returns a tuple of (century, year, ordinal_day)
get_xordinal_day()
  Returns a tuple of (century,year,ordinal_day)
get_week_day()
  Returns a tuple of (century, decade, year, week, weekday), note that Monday is 1 and Sunday is 7
get_xweek_day()
  Returns a tuple of (bce, century, decade, year, week, weekday), note that Monday is 1 and Sunday is 7
expand(xdigits)
  Constructs a new expanded instance
```

6.8. ISO 8601 Dates and Times
The purpose of this method is to create a new instance from an existing Date but with a different expansion (value of xdigits).

The resulting value must still satisfy the constraints imposed by the new xdigits value. In particular, if you pass xdigits=None the new instance will not be expanded and must be in the range 0001-01-01 to 9999-12-31.

classmethod from_struct_time(t)

Constructs a Date from a struct_time, such as might be returned from time.gmtime() and related functions.

update_struct_time(t)

update_struct_time changes the year, month, date, wday and ydat fields of t, a struct_time, to match the values in this date.

classmethod from_now()

Constructs a Date from the current local time.

offset (centuries=0, years=0, months=0, weeks=0, days=0)

Adds an offset

Constructs a Date from the given date + a given offset.

There are significant limitations on this method to avoid ambiguous outcomes such as adding 1 year to a leap day such as 2016-02-29.

A fully specified date can be offset by days or weeks, in the latter case all weeks have 7 days so this is always unambiguous.

Dates known only to week precision can only be offset by weeks.

Dates with month precision can be offset by months, years or centuries because every year has exactly the same number of months. The concept of February next year is always meaningful (unlike the meaning of 29th Feb next year or the similarly problematic week 53 next year).

Dates with year precision can be offset by years or centuries and, for completeness, dates with century precision can only be offset by centuries.

Creating an offset date from an expanded date always results in another expanded date (with the same xdigits value).

classmethod from_str(src, base=None, xdigits=None)

Parses a Date instance from a src string.

classmethod from_string_format(src, base=None, xdigits=None)

Similar to from_str() except that a tuple is returned, the first item is the resulting Date instance, the second is a string describing the format parsed. For example:

d,f=Date.from_string_format("1969-07-20")
# f is set to "YYYY-MM-DD".

get_calendar_string (basic=False, truncation=0)

Formats this date using calendar form, for example 1969-07-20

basic True/False, selects basic form, e.g., 19690720. Default is False. Expanded dates that use the non-conformant xdigits=-1 mode are not compatible with basic formatting.

truncation One of the Truncation constants used to select truncated forms of the date. For example, if you specify Truncation.Year you'll get -07-20 or -0720. Default is NoTruncation.

Calendar format only supports Century, Year and Month truncated forms.

get_ordinal_string (basic=False, truncation=0)

Formats this date using ordinal form, for example 1969-201
**basic**  True/False, selects basic form, e.g., 1969201. Default is False

**truncation**  One of the Truncation constants used to select truncated forms of the date. For example, if you specify Truncation.Year you’ll get -201. Default is NoTruncation.

Note that ordinal format only supports century and year truncated forms.

**get_week_string**  (basic=False, truncation=0)
Formats this date using week form, for example 1969-W29-7

**basic**  True/False, selects basic form, e.g., 1969W297. Default is False

**truncation**  One of the Truncation constants used to select truncated forms of the date. For example, if you specify Truncation.Year you’ll get -W297. Default is NoTruncation.

Note that week format only supports century, decade, year and week truncated forms.

**classmethod from_julian**  (year, month, day, xdigits=None)
Constructs a Date from a year, month and day expressed in the Julian calendar.

If the year is 0 or negative you must provide a value for xdigits in order to construct an expanded date.

**get_julian_day**()
Returns a tuple of: (year,month,day) representing the equivalent date in the Julian calendar.

**static split_year**  (year)
Static method that splits an integer year into a 3-tuple

Returns:

```
(bce, century, year)
```

Can be used as a convenience when constructing new instances:

```
int_year = -752
bce, century, year = Date.split_year(int_year)
d = Date(bce=bce, century=century, year=year, month=4, day=21,
xdigits=2)
str(d) == "-000752-04-21"  # True
```

**leap_year**()

leap_year returns True if this date is (in) a leap year and False otherwise.

Note that leap years fall on all years that divide by 4 except those that divide by 100 but including those that divide by 400.

**complete**()
Returns True if this date has a complete representation, i.e., does not use one of the reduced precision forms.

**get_precision**()
Returns one of the Precision constants representing the precision of this date.

---

### 6.8. ISO 8601 Dates and Times

```python
class Date:
    # Date class definition

    @classmethod
    def from_julian(cls, year, month, day, xdigits=None):
        # Constructor method

    def get_julian_day(self):
        # Getter method

    @staticmethod
    def split_year(year):
        # Static method to split year
```

```
int_year = -752
bce, century, year = Date.split_year(int_year)
d = Date(bce=bce, century=century, year=year, month=4, day=21,
xdigits=2)
str(d) == "-000752-04-21"  # True
```

---

A class for representing ISO times

Values can represent times with reduced precision, for example:
represents 8pm without a specific minute/seconds value.

There are a number of different forms of the constructor based on named parameters, the simplest is:

\[
\text{Time}(\text{hour}=20, \text{minute}=17, \text{second}=40)
\]

Indicate UTC (Zulu time) by providing a zone direction of 0:

\[
\text{Time}(\text{hour}=20, \text{minute}=17, \text{second}=40, \text{zdirection}=0)
\]

To indicate a UTC offset provide additional values for hours (and optionally minutes) with 1 or -1 for zdirection to indicate the direction of the shift. 1 indicates a more Easterly timezone, -1 indicates a more Westerly zone:

\[
\text{Time}(\text{hour}=15, \text{minute}=17, \text{second}=40, \text{zdirection}=-1, \text{zhour}=5, \text{zminute}=0)
\]

A UTC offset of 0 hours and minutes results in a value that compares as equal to the corresponding Zulu time but is formatted using an explicit offset by \text{str()} rather than using the canonical “Z” form.

You may also specify a total number of seconds past midnight (no zone):

\[
\text{Time}(\text{total_seconds}=73060)
\]

If total_seconds overflows an error is raised. To create a time from an arbitrary number of seconds and catch overflow use offset instead:

\[
\text{Time}(\text{total_seconds}=159460)
\]

# raises \text{DateTimeError}

\[
\text{t, overflow} = \text{Time()}.\text{offset}(\text{seconds}=159460)
\]

# sets t to 20:40:17 and overflow=1

Time supports two representations of midnight: 00:00:00 and 24:00:00 in keeping with the ISO specification. These are considered equivalent by comparisons!

Truncated forms can be created directly from the base time, see \text{extend()} for more information.

Comparisons are dealt with in a similar way to \text{Date} in that times must have the same precision to be comparable. Although this behaviour is consistent it might seem strange at first as it rules out comparing 09:00:15 with 09:00 but, in effect, 09:00 is actually all times in the range 09:00:00-09:00:59.999….

Zones further complicate this method but the rule is very simple, we only ever compare times from the same zone (or if both have unspecified zones). There is one subtlety to this implementation. Times stored with a redundant +00:00 or -00:00 are treated the same as those with a zero zone direction (Zulu time).

Time objects are immutable and so can be used as the keys in dictionaries provided they all share the same precision.

Some older functions did allow modification but these have been deprecated. Use python -Wd to force warnings from these unsafe methods.

Instances can be converted directly to strings or unicode strings using the default, extended format. Other formats are supported through format specific methods.
second = None
the seconds, 0..60 (to include leap seconds)

gooffset = None
the difference in minutes to UTC

get_total_seconds()
Note that leap seconds are handled as if they were invisible, e.g., 23:00:60 returns the same total seconds
as 23:00:59.

get_time()
Returns a tuple of (hour, minute, second).
Times with reduced precision will return None for second and or minute.

get_zone()
Returns a tuple of:

(zdirection, zoffset)
zdirection is defined as per Time’s constructor, zoffset is a non-negative integer minute offset or None, if
the zone is unspecified for this Time.

get_zone_offset()
Returns a single integer representing the zone offset (in minutes) or None if this time does not have a time
zone offset.

get_zone3()
Returns a tuple of:

(zdirection, zhour, zminute)
These values are defined as per Time’s constructor.

get_canonical_zone()
Returns a tuple of:

(zdirection, zhour, zminute)
These values are defined as per Time’s constructor but zero offsets always return zdirection=0. If present,
the zone is always returned with complete (minute) precision.

get_time_and_zone()
Returns a tuple of (hour, minute, second, zone direction, zone offset) as defined in get_time and get_zone.

extend(hour=None, minute=None, second=None)
Constructs a Time instance from an existing time, extended a (possibly) truncated hour/minute/second
value.
The time zone is always copied if present. The result is a tuple of (<Time instance>,overflow) where
overflow 0 or 1 indicating whether or not the time overflowed. For example:

# set base to 20:17:40Z
base=Time(hour=20, minute=17, second=40, zdirection=0)
t, overflow = base.extend(minute=37)
# t is 20:37:40Z, overflow is 0
t, overflow = base.extend(minute=7)
# t is 21:07:40Z, overflow is 0
t, overflow = base.extend(hour=19, minute=7)
# t is 19:07:40Z, overflow is 1
offset (hours=0, minutes=0, seconds=0)
Constructs a Time instance from an existing time and an offset number of hours, minutes and or seconds.
The time zone is always copied (if present). The result is a tuple of (<Time instance>,overflow) where overflow is the number of days by which the time overflowed. For example:

```python
# set base to 20:17:40Z
base = Time(hour=20, minute=17, second=40, zdirection=0)
t, overflow = base.offset(minutes=37)
# t is 20:54:40Z, overflow is 0
nt, overflow = base.offset(hours=4, minutes=37)
# t is 00:54:40Z, overflow is 1
```

Reduced precision times can still be offset but only by matching arguments. In other words, if the time has minute precision then you may not pass a non-zero value for seconds, etc. A similar constraint applies to the passing of floating point arguments. You may pass a fractional offset for seconds if the time has second precision but the minute and hour offsets must be to integer precision. Similarly, you may pass a fractional offset for minutes if the time has minute precision, etc.

with_zone (zdirection, zhour=None, zminute=None, **kwargs)
Replace time zone information

Constructs a new Time instance from an existing time but with the time zone specified. The time zone of the existing time is ignored. Pass zdirection=None to strip the zone information completely.

shift_zone (zdirection, zhour=None, zminute=None, **kwargs)
Constructs a Time instance from an existing time but shifted so that it is in the time zone specified. The return value is a tuple of:

```python
(<Time instance>, overflow)
```

overflow is one of -1, 0 or 1 indicating if the time over- or under-flowed as a result of the time zone shift.

classmethod from_struct_time (t)
Constructs a zone-less Time from a struct_time, such as might be returned from time.gmtime() and related functions.

update_struct_time (t)
update_struct_time changes the hour, minute, second and isdst fields of t, a struct_time, to match the values in this time.

isdst is always set to -1

classmethod from_now ()
Constructs a Time from the current local time.

classmethod from_str (src, base=None)
Constructs a Time instance from a string representation, truncated forms are returned as the earliest time on or after base and may have overflowed. See from_string_format() for more.

with_zone_string (zone_str)
Constructs a Time instance from an existing time but with the time zone parsed from zone_str. The time zone of the existing time is ignored.

with_zone_string_format (zone_str)
Constructs a Time instance from an existing time but with the time zone parsed from zone_str. The time zone of the existing time is ignored.

Returns a tuple of: (<Time instance>,format)
classmethod from_string_format (src, base=None)

Constructs a Time instance from a string representation, truncated forms are returned as the earliest time on or after base.

Returns a tuple of (<Time instance>,overflow,format) where overflow is 0 or 1 indicating whether or not a truncated form overflowed and format is a string representation of the format parsed, e.g., “hhmms”.

get_string (basic=False, truncation=0, ndp=0, zone_precision=7, dp=',', **kwargs)

Formats this time, including zone, for example 20:17:40

basic True/False, selects basic form, e.g., 201740. Default is False

truncation One of the Truncation constants used to select truncated forms of the time. For example, if you specify Truncation.Hour you’ll get -17:40 or -1740. Default is NoTruncation.

ndp Specifies the number of decimal places to display for the least significant component, the default is 0.

dp The character to use as the decimal point, the default is the comma, as per the ISO standard.

zone_precision One of Precision.Hour or Precision.Complete to control the precision of the zone offset.

Note that time formats only support Minute and Second truncated forms.

get_zone_string (basic=False, zone_precision=7, **kwargs)

Formats this time’s zone, for example -05:00.

basic True/False, selects basic form, e.g., -0500. Default is False

zone_precision One of Precision.Hour or Precision.Complete to control the precision of the zone offset.

Times constructed with a zdirection value of 0 are always rendered using “Z” for Zulu time (the name is taken from the phonetic alphabet). To force use of the offset format you must construct the time with a non-zero value for zdirection.

complete ()

Returns True if this date has a complete representation, i.e., does not use one of the reduced precision forms.

(Whether or not a time is complete refers only to the precision of the time value, it says nothing about the presence or absence of a time zone offset.)

get_precision ()

Returns one of the Precision constants representing the precision of this time.

with_precision (precision, truncate=False)

Constructs a Time instance from an existing time but with the precision specified by precision.

precision is one of the Precision constants, only hour, minute and complete precision are valid.

truncate is True/False indicating whether or not the time value should be truncated so that all values are integers. For example:

```python
t = Time(hour=20, minute=17, second=40)
tm = t.with_precision(Precision.Minute, False)
print tm.get_string(ndp=3)
# 20:17,667
tm=t.with_precision(Precision.Minute, True)
print tm.get_string(ndp=3)
# 20:17,000
```
class pyslet.iso8601.TimePoint(src=None, date=None, time=None)

A class for representing ISO timepoints

TimePoints are constructed from a date and a time (which may or may not have a time zone), for example:

```
TimePoint(date=Date(century=19, year=69, month=7, day=20),
         time=Time(hour=20, minute=17, second=40, zdirection=0))
```

If the date is missing then the date origin is used, Date() or 0001-01-01. Similarly, if the time is missing then the time origin is used, Time() or 00:00:00.

Times may be given with reduced precision but the date must be complete. In other words, there is no such thing as a timepoint with month precision, use Date instead. Expanded dates may be used.

When comparing TimePoint instances we deal with partially specified TimePoints in the same way as Time. However, unlike the comparison of Time instances, we reduce all TimePoints with time-zones to a common zone before doing a comparison. As a result, TimePoints which are equal but are expressed in different time zones will still compare equal.

Instances can be converted directly to strings or unicode strings using the default, extended calendar format. Other formats are supported through format specific methods.

get_calendar_time_point()
Returns a tuple representing the calendar time point

The result is:

```
(century, year, month, day, hour, minute, second)
```

This method cannot be used for expanded dates, use get_xcalendar_time_point() instead when dealing with dates outside of the normal ISO 8601 range.

get_xcalendar_time_point()
Returns a tuple representing an expanded calendar time point

The result is:

```
(bce, century, year, month, day, hour, minute, second)
```

get_ordinal_time_point()
Returns a tuple representing the ordinal time point

The result is:

```
(century, year, ordinal_day, hour, minute, second)
```

This method cannot be used for expanded dates, use get_xordinal_time_point() instead when dealing with dates outside of the normal ISO 8601 range.

get_xordinal_time_point()
Returns a tuple representing an expanded ordinal time point

The result is:

```
(bce, century, year, ordinal_day, hour, minute, second)
```

get_week_day_time_point()
Returns a tuple representing the week-day time point
The result is:

\[(\text{century, decade, year, week, weekday, hour, minute, second})\]

This method cannot be used for expanded dates, use `get_xweek_day_time_point()` instead when dealing with dates outside of the normal ISO 8601 range.

**get_xweek_day_time_point()**

Returns a tuple representing an expanded week-day time point

The result is:

\[(\text{bce, century, decade, year, week, weekday, hour, minute, second})\]

generate a tuple representing the zone

The result is \((\text{zdirection}, \text{zoffset})\)

See `Time.get_zone()` for details.

**expand** *(xdigits)*

Constructs a new expanded instance

The purpose of this method is to create a new instance from an existing TimePoint but with a different date expansion (value of xdigits).

This is equivalent to:

\[\text{TimePoint(date=\text{self.date.expand(xdigits)}, time=\text{self.time})}\]

**with_zone** *(zdirection, zhour=None, zminute=None, **kwargs)*

Constructs a `TimePoint` instance from an existing TimePoint but with the time zone specified. The time zone of the existing TimePoint is ignored.

**shift_zone** *(zdirection, zhour=None, zminute=None, **kwargs)*

Shifts time zone

Constructs a `TimePoint` instance from an existing TimePoint but shifted so that it is in the time zone specified.

**update_struct_time** *(t)*

Outputs the TimePoint in struct_time format

Changes the year, month, date, hour, minute and second fields of t, t must be a mutable list arranged in the same order as struct_time.

**classmethod from_struct_time** *(t)*

Constructs an instance from a struct_time

In other words, constructs an instance from the object returned from `time.gmtime()` and related functions.

**classmethod from_str** *(src, base=None, tdesignators='T', xdigits=None, **kwargs)*

Constructs a TimePoint from a string representation. Truncated forms are parsed with reference to `base`.

**classmethod from_string_format** *(src, base=None, tdesignators='T', xdigits=None, **kwargs)*

Creates an instance from a string

Similar to `from_str()` except that a tuple is returned, the first item is the resulting `TimePoint` instance, the second is a string describing the format parsed. For example:
tp, f = TimePoint.from_string_format("1969-07-20T20:40:17")
# f is set to "YYYY-MM-DDTmm:hh:ss".

get_calendar_string(basic=False, truncation=0, ndp=0, zone_precision=7, dp=' ', tdesignator='T', **kwargs)
Formats this TimePoint using calendar form
For example ‘1969-07-20T20:17:40’

basic True/False, selects basic form, e.g., 19690720T201740. Default is False

truncation One of the Truncation constants used to select truncated forms of the date. For example,
if you specify Truncation.Year you’ll get -07-20T20:17:40 or -0720T201740. Default is NoTruncation. Calendar format only supports Century, Year and Month truncated forms, the time component cannot be truncated.

ndp, dp and zone_precision As specified in Time.get_string()
If the instance is an expanded time point with xdigits=-1 then basic format is not allowed.

get_ordinal_string(basic=0, truncation=0, ndp=0, zone_precision=7, dp=' ', tdesignator='T', **kwargs)
Formats this TimePoint using ordinal form
For example ‘1969-201T20:17:40’

basic True/False, selects basic form, e.g., 1969201T201740. Default is False

truncation One of the Truncation constants used to select truncated forms of the date. For example,
if you specify Truncation.Year you’ll get -201T20-17-40. Default is NoTruncation. Note
that ordinal format only supports century and year truncated forms, the time component cannot be truncated.

ndp, dp and zone_precision As specified in Time.get_string()
If the instance is an expanded time point with xdigits=-1 then basic format is not allowed.

get_week_string(basic=0, truncation=0, ndp=0, zone_precision=7, dp=' ', tdesignator='T', **kwargs)
Formats this TimePoint using week form
For example ‘1969-W29-7T20:17:40’

basic True/False, selects basic form, e.g., 1969W297T201740. Default is False

truncation One of the Truncation constants used to select truncated forms of the date. For example,
if you specify Truncation.Year you’ll get -W297T20-17-40. Default is NoTruncation. Note
that week format only supports century, decade, year and week truncated forms, the time component cannot be truncated.

ndp, dp and zone_precision As specified in Time.get_string()
If the instance is an expanded time point with xdigits=-1 then basic format is not allowed.

classmethod from_unix_time(unix_time)
Constructs a TimePoint from unix_time, the number of seconds since the time origin. The resulting time
is in UTC.
This method uses python’s gmtime(0) to obtain the time origin, it isn’t necessarily the Unix base time of
1970-01-01.

get_unixtime()
Returns a unix time value representing this time point.
classmethod from_now()
Constructs a TimePoint from the current local date and time.

classmethod from_now_utc()
Constructs a TimePoint from the current UTC date and time.

complete()
Test for complete precision

Returns True if this TimePoint has a complete representation, i.e., does not use one of the reduced precision forms

(Whether or not a TimePoint is complete refers only to the precision of the time value, it says nothing about the presence or absence of a time zone offset.)

get_precision()
Returns one of the Precision constants representing the precision of this TimePoint.

with_precision (precision, truncate)
Return new instance with precision

Constructs a TimePoint instance from an existing TimePoint but with the precision specified by precision. For more details see Time.with_precision()

class pyslet.iso8601.Duration (value=None)
Bases: pyslet.py2.UnicodeMixin, pyslet.pep8.PEP8Compatibility

A class for representing ISO durations

6.8.1 Supporting Constants

class pyslet.iso8601.Truncation
 Defines constants to use when formatting to truncated forms.

No = 0
constant for no truncation

Century = 1
constant for truncating to century

Decade = 2
constant for truncating to decade

Year = 3
constant for truncating to year

Month = 4
constant for truncating to month

Week = 5
constant for truncating to week

Hour = 6
constant for truncating to hour

Minute = 7
constant for truncating to minute

pyslet.iso8601.NoTruncation = 0
a synonym for Truncation.No

class pyslet.iso8601.Precision
 Defines constants for representing reduced precision.
Century = 1
calendar for century precision

Year = 2
calendar for year precision

Month = 3
calendar for month precision

Week = 4
calendar for week precision

Hour = 5
calendar for hour precision

Minute = 6
calendar for minute precision

Complete = 7
calendar for complete representations

6.8.2 Utility Functions

pyslet.iso8601.leap_year(year)
leap_year returns True if year is a leap year and False otherwise.

Note that leap years famously fall on all years that divide by 4 except those that divide by 100 but including
those that divide by 400.

pyslet.iso8601.day_of_week(year, month, day)
day_of_week returns the day of week 1-7
1 being Monday for the given year, month and day

pyslet.iso8601.week_count(year)
Week count returns the number of calendar weeks in a year.
Most years have 52 weeks of course, but if the year begins on a Thursday or a leap year begins on a Wednesday
then it has 53.

pyslet.iso8601.get_local_zone()
Returns the number of minutes ahead of UTC we are
This is calculated by comparing the return result of the time module’s gmtime and localtime methods.

6.9 Unicode Characters

6.9.1 Utility Functions

pyslet.unicode5.detect_encoding(magic)
Detects text encoding

magic A string of bytes

Given a byte string this function looks at (up to) four bytes and returns a best guess at the unicode encoding
being used for the data.

It returns a string suitable for passing to Python’s native decode method, e.g., ‘utf_8’. The default is ‘utf_8’, an
encoding which will also work if the data is plain ASCII.
6.9.2 Character Classes

class pyslet.unicode5.CharClass(*args)

    Bases: pyslet.py2.UnicodeMixin

    Represents a class of unicode characters.

    A class of characters is represented internally by a list of character ranges that define the class. This is efficient because most character classes are defined in blocks of characters.

    For the constructor, multiple arguments can be provided.

    String arguments add all characters in the string to the class. For example, CharClass('abcxyz') creates a class comprising two ranges: a-c and x-z.

    Tuple/List arguments can be used to pass pairs of characters that define a range. For example, CharClass(('a', 'z')) creates a class comprising the letters a-z.

    Instances of CharClass can also be used in the constructor to add an existing class.

    Instances support Python’s repr function:

>>> c = CharClass('abcxyz')
>>> print repr(c)
CharClass((u'a', u'c'), (u'x', u'z'))

    The string representation of a CharClass is a python regular expression suitable for matching a single character from the CharClass:

>>> print str(c)
[a-cx-z]

classmethod ucd_category(category)

    Returns the character class representing the Unicode category.

    You must not modify the returned instance, if you want to derive a character class from one of the standard Unicode categories then you should create a copy by passing the result of this class method to the CharClass constructor, e.g. to create a class of all general controls and the space character:

    c = CharClass(CharClass.ucd_category(u"Cc"))
    c.add_char(u" ")


classmethod ucd_block(block_name)

    Returns the character class representing the Unicode block.

    You must not modify the returned instance, if you want to derive a character class from one of the standard Unicode blocks then you should create a copy by passing the result of this class method to the CharClass constructor, e.g. to create a class combining all Basic Latin characters and those in the Latin-1 Supplement:

    c = CharClass(CharClass.ucd_block(u"Basic Latin"))
    c.add_class(CharClass.ucd_block(u"Latin-1 Supplement"))

format_re()

    Create a representation of the class suitable for putting in [] in a python regular expression

add_range(a, z)

    Adds a range of characters from a to z to the class

subtract_range(a, z)

    Subtracts a range of characters from the character class
add_char(c)
   Adds a single character to the character class

subtract_char(c)
   Subtracts a single character from the character class

add_class(c)
   Adds all the characters in c to the character class

   This is effectively a union operation.

subtract_class(c)
   Subtracts all the characters in c from the character class

negate()
   Negates this character class

   As a convenience returns the object as the result enabling this method to be used in construction,
   e.g.:

   c = CharClass('`

   ').negate()

   Results in the class of all characters except line feed and carriage return.

test(c)
   Test a unicode character.

   Returns True if the character is in the class.

   If c is None, False is returned.

   This function uses an internal cache to speed up tests of complex classes. Test results are cached in 256
   character blocks. The cache does not require a lock to make this method thread-safe (a lock would have
   a significant performance penalty) as it uses a simple python list. The worst case race condition would
   result in two separate threads calculating the same block simultaneously and assigning it the same slot in
   the cache but python’s list object is thread-safe under assignment (and the two calculated blocks will be
   identical) so this is not an issue.

   Why does this matter? This function is called a lot, particularly when parsing XML. When parsing a tag
   the parser will repeatedly test each character to determine if it is a valid name character and the definition
   of name character is complex. Here are some illustrative figures calculated using cProfile for a typical
   1MB XML file which calls test 142198 times: with no cache 0.42s spent in test, with the cache 0.11s
   spent.

6.9.3 Parsing Text and Binary Data

class pyslet.unicode5.BasicParser(source)
   Bases: pyslet.unicode5.ParserMixin, pyslet.pep8.PEP8Compatibility

   A base class for parsing character strings or binary data

   source Can be either a string of characters or a string of bytes.

   BasicParser instances can parse either characters or bytes but not both simultaneously, you must choose on
   construction by passing an appropriate str (Python 2: unicode), bytes or bytearray object.

   Binary mode is suitable for parsing data described in terms of OCTETS, such as many IETF and internet
   standards. When passing string literals to parsing methods in binary mode use the binary string literal form:
Methods that return the parsed data in its original form will also return bytes objects in binary mode.

Methods are named according to the type of operation they perform.

**match_*** Returns a boolean True or False depending on whether or not a syntax production is matched at the current location. The state of the parser is unchanged. This type of method is only used for very simple productions, e.g., `match_digit()`.

**parse_*** Attempts to parse a syntax element returning an appropriate object as the result or None if the production is not present. The position of the parser is only changed if the element was parsed successfully. This type of method is intended for fairly simple productions, e.g., `parse_integer()`. More complex productions are implemented using require_* methods but the general `parse_production()` can be used to enable more complex look-ahead scenarios.

**require_*** Parses a syntax production, returning an appropriate object as the result. If the production is not matched a `ParserError` is raised.

On success, the position of the parser points to the first character after the parsed production ready to continue parsing. On failure, the parser is positioned at the point at which the exception was raised.

When deriving your own sub-classes you will normally use the require_* pattern to extend the parser.

Compatibility note: if you are attempting to use the same source for both Python 2 and 3 then you may not be able to rely on the parser mode:

```python
>>> from pyslet.unicode5 import BasicParser
>>> p = BasicParser("hello")
>>> p.raw
```

The above interpreter session will print True in Python 2 and False in Python 3. This is just another manifestation of the changes to string handling between the two releases. If you are dealing with ASCII data you can ignore the issue, otherwise you should consider using one of the various techniques for forcing strings to be interpreted as unicode when running in Python 2. The most important thing is consistency between the type of object you pass to the constructor and those that you pass to the various parsing methods. You may find the `pyslet.py2.ul()` and/or `pyslet.py2.u8()` functions useful for forcing text mode.

```
raw = None
True if parser is working in binary mode.
```

```
src = None
the string being parsed
```

```
pos = None
the position of the current character
```

```
the_char = None
The current character or None if the parser is positioned outside the src string.

In binary mode this will be a byte, which is an integer in Python 3 but a character in Python 2. In text mode it is a (unicode) character.
```

```python
setpos(new_pos)
Sets the position of the parser to new_pos
```

Useful for saving the parser state and returning later:
save_pos = parser.pos
#
# do some look-ahead parsing
#
parser.setpos(save_pos)

next_char()
Points the parser at the next character.
 Updates pos and the_char.

parser_error(production=None)
Raises an error encountered by the parser
See ParserError for details.
If production is None then the previous error is re-raised. If multiple errors have been raised previously the one with the most advanced parser position is used. This is useful in situations where there are multiple alternative productions, none of which can be successfully parsed. It allows parser methods to catch the exception from the last possible choice and raise an error relating to the closest previous match. For example:

def require_abc(self):
    result = p.parse_production(p.require_a)
    if result is None:
        result = p.parse_production(p.require_b)
        if result is None:
            result = p.parse_production(p.require_c)
            if result is None:
                # will raise the most advanced error raised during
                # the three previous methods
                p.parser_error()
        else:
            return result

See parse_production() for more details on this pattern.

The position of the parser is always set to the position of the error raised.

peek(nchars)
Returns the next nchars characters or bytes.
If there are less than nchars remaining then a shorter string is returned.

match_end()
True if all of src has been parsed

match(match_string)
Returns true if match_string is at the current position

parse(match_string)
Parses match_string
Returns match_string or None if it cannot be parsed.

require(match_string, production=None)
Parses and requires match_string
match_string The string to be parsed
production Optional name of production, defaults to match_string itself.
For consistency, returns match_string on success.

**match_insensitive** *(lower_string)*

Returns true if *lower_string* is matched (ignoring case).

*lower_string* must already be a lower-cased string.

**parse_insensitive** *(lower_string)*

Parses *lower_string* ignoring case in the source.

*lower_string* Must be a lower-cased string

Advances the parser to the first character after *lower_string*. Returns the matched string which may differ in case from *lower_string*.

**parse_until** *(match_string)*

Parses up to but not including *match_string*.

Advances the parser to the first character of *match_string*. If *match_string* is not found (or is None) then all the remaining characters in the source are parsed.

Returns the parsed text, even if empty. Never returns None.

**match_one** *(match_chars)*

Returns true if one of *match_chars* is at the current position.

The ‘in’ operator is used to test *match_chars* so this can be a list or tuple of characters (or bytes), it does not have to be string.

**parse_one** *(match_chars)*

Parses one of *match_chars*.

*match_chars* A string (list or tuple) of characters or bytes

Returns the character (or byte) or None if no match is found.

Warning: in binary mode, this method will return a single byte value, the type of which will differ in Python 2. In Python 3, bytes are integers, in Python 2 they are binary strings of length 1. You can use the function `py2.byte()` to help ensure your source works on both platforms, for example:

```python
from .py2 import byte
c = parser.parse_one(b"+-")
if c == byte(b"+"):
    # do plus thing...
elif c is not None:
    # must be minus...
else:
    # do something else...
```

**match_digit** ()

Returns true if the current character is a digit

Only ASCII digits are considered, in binary mode byte values 0x30 to 0x39 are matched.

**parse_digit** ()

Parses a digit character.

Returns the digit character/byte, or None if no digit is found. Like `match_digit()` only ASCII digits are parsed.

**parse_digit_value** ()

Parses a single digit value.

Returns the digit value, or None if no digit is found. Like `match_digit()` only ASCII digits are parsed.
parse_digits \((\text{min, max}=\text{None})\)

Parses a string of digits

- **min** The minimum number of digits to parse. There is a special case where min=0, in this case an empty string may be returned.

- **max** (default None) The maximum number of digits to parse, or None there is no maximum.

Returns the string of digits or None if no digits can be parsed. Like `parse_digit()`, only ASCII digits are considered.

parse_integer \((\text{min}=\text{None}, \text{max}=\text{None}, \text{max_digits}=\text{None})\)

Parses an integer (or long).

- **min** (optional, defaults to None) A lower bound on the acceptable integer value, the result will always be \(\geq\) min on success

- **max** (optional, defaults to None) An upper bound on the acceptable integer value, the result will always be \(\leq\) max on success

- **max_digits** (optional, defaults to None) The limit on the number of digits, i.e., the field width.

If a suitable integer can’t be parsed then None is returned. This method only processes ASCII digits.

Warning: in Python 2 the result may be of type long.

match_hex_digit

Returns true if the current character is a hex-digit

Only ASCII digits are considered, letters can be either upper or lower case. In binary mode byte values 0x30 to 0x39, 0x41-0x46 and 0x61-0x66 are matched.

parse_hex_digit

Parses a hex-digit.

Returns the digit, or None if no digit is found. See `match_hex_digit()` for which characters/bytes are considered hex-digits.

parse_hex_digits \((\text{min, max}=\text{None})\)

Parses a string of hex-digits

- **min** The minimum number of hex-digits to parse. There is a special case where min=0, in this case an empty string may be returned.

- **max** (default None) The maximum number of hex-digits to parse, or None there is no maximum.

Returns the string of hex-digits or None if no digits can be parsed. See `match_hex_digit()` for which characters/bytes are considered hex-digits.

class pyslet.unicode5.ParserError \((\text{production}, \text{parser}=\text{None})\)

Exception raised by BasicParser

- **production** The name of the production being parsed

- **parser** The BasicParser instance raising the error (optional)

ParserError is a subclass of ValueError.

- **pos** = None

  the position of the parser when the error was raised

- **left** = None

  up to 40 characters/bytes to the left of pos
right = None
up to 40 characters/bytes to the right of pos

6.10 Streams

6.10.1 Utility Functions

pyslet.streams.io_blocked(err)
Returns True if IO operation is blocked

err  An IOError exception (or similar object with errno attribute).

Bear in mind that EAGAIN and EWOUNDBLOCK are not necessarily the same value and that when running
under windows WSAEWOULDBLOCK may be raised instead. This function removes this complexity making
it easier to write cross platform non-blocking IO code.

pyslet.streams.io_timedout(err)
Returns True if an IO operation timed out

err  An IOError exception (or similar object with errno attribute).

Tests for ETIMEDOUT and when running under windows WSAETIMEDOUT too.

6.10.2 Stream Classes

class pyslet.streams.Pipe(bsize=8192, rblocking=True, wblocking=True, timeout=None, name=None)
Bases: io.RawIOBase
Buffered pipe for inter-thread communication

The purpose of this class is to provide a thread-safe buffer to use for communicating between two parts of an
application that support non-blocking io while reducing to a minimum the amount of byte-copying that takes
place.

Essentially, write calls with immutable byte strings are simply cached without copying (and always succeed)
ensuring them to be passed directly through to the corresponding read operation in streaming situations. How-
ever, to improve flow control a canwrite method is provided to help writers moderate the amount of data that
has to be held in the buffer:

```
# data producer thread
while busy:
    wmax = p.canwrite()
    if wmax:
        data = get_at_most_max_bytes(wmax)
        p.write(data)
    else:
        # do something else while the pipe is blocked
        spin_the_beach_ball()
```

bsize  The buffer size, this is used as a guide only. When writing immutable bytes objects to the pipe the buffer
size may be exceeded as these can simply be cached and returned directly to the reader more efficiently
than slicing them up just to adhere to the buffer size. However, if the buffer already contains bsize bytes
all calls to write will block or return None. Defaults to io.DEFAULT_BUFFER_SIZE.
rblocking  Controls the blocking behaviour of the read end of this pipe. True indicates reads may block waiting for data, False that they will not and read may return None. Defaults to True.

wblocking  Controls the blocking behaviour of the write end of the this pipe. True indicates writes may block waiting for data, False that they will not and write may return None. Defaults to True.

timeout  The number of seconds before a blocked read or write operation will timeout. Defaults to None, which indicates ‘wait forever’. A value of 0 is not the same as placing both ends of the pipe in non-blocking mode (though the effect may be similar).

name  An optional name to use for this pipe, the name is used when raising errors and when logging

name = None
   the name of the pipe

close()
   closed the Pipe

   This implementation works on a ‘reader closes’ principle. The writer should simply write the EOF marker to the Pipe (see write_eof()).

   If the buffer still contains data when it is closed a warning is logged.

readable()
   Pipe’s are always readable

writable()
   Pipe’s are always writeable

readblocking()
   Returns True if reads may block

set_readblocking(blocking=True)
   Sets the readblocking mode of the Pipe.

   blocking  A boolean, defaults to True indicating that reads may block.

writeblocking()
   Returns True if writes may block

set_writeblocking(blocking=True)
   Sets the writeblocking mode of the Pipe.

   blocking  A boolean, defaults to True indicating that writes may block.

empty()
   Returns True if the buffer is currently empty

buffered()
   Returns the number of buffered bytes in the Pipe

canwrite()
   Returns the number of bytes that can be written.

   This value is the number of bytes that can be written in a single non-blocking call to write. 0 indicates that the pipe’s buffer is full. A call to write may accept more than this but the next call to write will always accept at least this many.

   This class is fully multithreaded so in situations where there are multiple threads writing this call is of limited use.

   If called on a pipe that has had the EOF mark written then IOError is raised.
set_rflag(rflag)
Sets an Event triggered when a reader is detected.

rflag  An Event instance from the threading module.

The event will be set each time the Pipe is read. The flag may be cleared at any time by the caller but as a convenience it will always be cleared when canwrite() returns 0.

The purpose of this flag is to allow a writer to use a custom event to monitor whether or not the Pipe is ready to be written. If the Pipe is full then the writer will want to wait on this flag until a reader appears before attempting to write again. Therefore, when canwrite indicates that the buffer is full it makes sense that the flag is also cleared.

If the pipe is closed then the event is set as a warning that the pipe will never be read. (The next call to write will then fail.)

write_wait(timeout=None)
Waits for the pipe to become writable or raises IOError

timeout  Defaults to None: wait forever. Otherwise the maximum number of seconds to wait for.

flush_wait(timeout=None)
Waits for the pipe to become empty or raises IOError

timeout  Defaults to None: wait forever. Otherwise the maximum number of seconds to wait for.

canread()
Returns True if the next call to read will not block.

False indicates that the pipe’s buffer is empty and that a call to read will block.

Note that if the buffer is empty but the EOF signal has been given with write_eof() then canread returns True! The next call to read will not block but return an empty string indicating the EOF.

read_wait(timeout=None)
Waits for the pipe to become readable or raises IOError

timeout  Defaults to None: wait forever. Otherwise the maximum number of seconds to wait for.

write(b)
writes data to the pipe

The implementation varies depending on the type of b. If b is an immutable bytes object then it is accepted even if this overfills the internal buffer (as it is not actually copied). If b is a bytearray then data is copied, up to the maximum buffer size.

write_eof()
Writes the EOF flag to the Pipe

Any waiting readers are notified and will wake to process the Pipe. After this call the Pipe will not accept any more data.

flush()
flushes the Pipe

The intention of flush to push any written data out to the destination, in this case the thread that is reading the data.

In write-blocking mode this call will wait until the buffer is empty, though if the reader is idle for more than timeout seconds then it will raise IOError.

In non-blocking mode it simple raises IOError with EWOULDBLOCK if the buffer is not empty.

Given that flush is called automatically by close() for classes that inherit from the base io classes our implementation of close discards the buffer rather than risk an exception.
**readall()**

Overridden to take care of non-blocking behaviour.

Warning: readall always blocks until it has read EOF, regardless of the rblocking status of the Pipe.

The problem is that, if the Pipe is set for non-blocking reads then we seem to have the choice of returning a partial read (and failing to signal that some of the data is still in the pipe) or raising an error and losing the partially read data.

Perhaps ideally we’d return None indicating that we are blocked from reading the entire stream but this isn’t listed as a possible return result for io.RawIOBase.readall and it would be tricky to implement anyway as we still need to deal with partially read data.

Ultimately the safe choices are raise an error if called on a non-blocking Pipe or simply block. We do the latter on the basis that anyone calling readall clearly intends to wait.

For a deep discussion of the issues around non-blocking behaviour see [http://bugs.python.org/issue13322](http://bugs.python.org/issue13322)

**readmatch**(match='\r\n')

Read until a byte string is matched

**match** A binary string, defaults to CRLF.

This operation will block if the string is not matched unless the buffer becomes full without a match, in which case IOError is raised with code ENOBUFS.

**read**(nbytes=-1)

read data from the pipe

May return fewer than nbytes if the result can be returned without copying data. Otherwise **readinto()** is used.

**readinto**(b)

Reads data from the Pipe into a bytearray.

Returns the number of bytes read. 0 indicates EOF. None indicates an operation that would block in a Pipe that is non-blocking for read operations. May return fewer bytes than would fit into the bytearray as it returns as soon as it has at least some data.

**class** **pyslet.streams.BufferedStreamWrapper**(src, buffsize=8192)

Bases: io.RawIOBase

A buffered wrapper for file-like objects.

**src** A file-like object, we only require a read method

**buffsize** The maximum size of the internal buffer

On construction the src is read until an end of file condition is encountered or until buffsize bytes have been read. EOF is signaled by an empty string returned by src’s read method. Instances then behave like readable streams transparently reading from the buffer or from the remainder of the src as applicable.

Instances behave differently depending on whether or not the entire src is buffered. If it is they become seekable and set a value for the length attribute. Otherwise they are not seekable and the length attribute is None.

If src is a non-blocking data source and it becomes blocked, indicated by read returning None rather than an empty string, then the instance reverts to non-seekable behaviour.

**peek**(nbytes)

Read up to nbytes without advancing the position

If the stream is not seekable and we have read past the end of the internal buffer then an empty string will be returned.
6.11 File System Abstraction

The purpose of this module is to provide an abstraction over the top the native file system, potentially allowing alternative implementations to be provided in the future. This module was particularly developed with operating environments where access to the file system is limited or not-allowed. Pyslet modules that use these classes to access the file system can be easily repointed at some other implementation.

class pyslet.vfs.VirtualFilePath(*args)
   Bases: pyslet.py2.SortableMixin

   Abstract class representing a virtual file system

   Instances represent paths within a file system. You can’t create an instance of VirtualFilePath directly, instead you must create instances using a class derived from it. (Do not call the __init__ method of VirtualFilePath from your derived classes.)

   All instances are created from one or more strings, either byte strings or unicode strings, or existing instances. In the case of byte strings the encoding is assumed to be the default encoding of the file system. If multiple arguments are given then they are joined to make a single path using join().

   Instances can be converted to either binary or character strings, use to_bytes() for the former. Note that the builtin str function returns a binary string in Python 2, not a character string.

   Instances are immutable, and can be used as keys in dictionaries. Instances must be from the same file system to be comparable, the unicode representation is used.

   An empty path is False, other paths are True. You can also compare a file path with a string (or unicode string) which is first converted to a file path instance.

fs_name = None
   The name of the file system, must be overridden by derived classes.

   The purpose of providing a name for a file system is to enable file systems to be mapped onto the authority (host) component of a file URL.

supports_unicode_filenames = False
   Indicates whether this file system supports unicode file names natively. In general, you don’t need to worry about this as all methods that accept strings will accept either type of string and convert to the native representation.

   When creating derived classes you must also override sep, curdir, pardir, ext, drive_sep (if applicable) and empty with the correct string types.

supports_unc = False
   Indicates whether this file system supports UNC paths.

   UNC paths are of the general form:

   \\ComputerName\SharedFolder\Resource

   This format is used in Microsoft Windows. See is_unc() for details.

supports_drives = False
   Indicates whether this file system supports ‘drives’, i.e., is Windows-like in having drive letters that may prefix paths.

codec = 'utf-8'
   The codec used by this file system

   This codec is used to convert between byte strings and unicode strings. The default is utf-8.
sep = '/'
    The path separator used by this file system

This is either a character or byte string, depending on the setting of supports_unicode_filenames.

curdir = '.'
    The path component that represents the current directory

cpardir = '..'
    The path component that represents the parent directory

ext = '.
    The extension character

drive_sep = ':'
    The drive separator

empty = '
    An empty path string (for use with join)

classmethod getcwd()
    Returns an instance representing the working directory.

classmethod getcroot()
    Returns an instance representing the current root.

    UNIX users will find this odd but in other file systems there are multiple roots. Rather than invent an
    abstract concept of the root of roots we just accept that there can be more than one. (We might struggle to
    perform actions like listdir() on the root of roots.)

    The current root is determined by stripping back the current working directory until it can no longer be
    split.

classmethod mkdtemp(suffix=", prefix=")
    Creates a temporary directory in the file system

    Returns an instance representing the path to the new directory.

    Similar to Python’s tempfile.mkdtemp, like that function the caller is responsible for cleaning up the direc-
    tory, which can be done with rmtree().

classmethod path_str(arg)
    Converts a single argument to the correct string type

    File systems can use either binary or character strings and we convert between them using codec. This
    method takes either type of string or an existing instance and returns a path string of the correct type.

path = None
    the path, either character or binary string

to_bytes()
    Returns the binary string representation of the path.

sortkey()
    Instances are sortable using character strings.

join(*components)
    Returns a new instance by joining path components

    Starting with the current instance, this method appends each component, returning a new instance repre-
    senting the joined path. If components contains an absolute path then previous components, including the
    instance’s path, are discarded.

    For details see Python’s os.path.join function.
For the benefit of derived classes a default implementation is provided.

`split()`  
Splits a path  
Returns a tuple of two instances (head, tail) where tail is the last path component and head is everything leading up to it.  
For details see Python’s os.path.split.

`splitext()`  
Splits an extension from a path  
Returns a tuple of (root, ext) where root is an instance containing just the root file path and ext is a string of characters representing the original path’s extension.  
For details see Python’s os.path.splitext.

`splitdrive()`  
Splits a drive designation  
Returns a tuple of two instances (drive, tail) where drive is either a drive specification or is empty.  
Default implementation uses the `drive_sep` to determine if the first path component is a drive.

`splitunc()`  
Splits a UNC path  
Returns a tuple of two instances (mount, path) where mount is an instance representing the UNC mount point or an instance representing the empty path if this isn’t a UNC path.  
Default implementation checks for a double separator at the start of the path and at least one more separator.

`abspath()`  
Returns an absolute path instance.

`realpath()`  
Returns a real path, with any symbolic links removed.  
The default implementation normalises the path using `normpath()` and `normcase()`.

`normpath()`  
Returns a normalised path instance.

`normcase()`  
Returns a case-normalised path instance.  
The default implementation returns the path unchanged.

`is_unc()`  
Returns True if this path is a UNC path.  
UNC paths contain a host designation, a path cannot contain a drive specification and also be a UNC path.  
Default implementation calls `splitunc()` and returns True if the unc component is non-empty.

`is_single_component()`  
Returns True if this path is a single, non-root, component.  
E.g., tests that the path does not contain a slash (it may be empty)

`is_empty()`  
Returns True if this path is empty
**is_dirlike()**
Returns True if this is a directory-like path.
E.g., test that the path ends in a slash (last component is empty).

**is_root()**
Returns True if this is a root path.
E.g., tests if it consists of just one or more slashes only (not counting any drive specification in file systems that support them).

**isabs()**
Returns True if the path is an absolute path.

**stat()**
Return information about the path.

**exists()**
Returns True if this is existing item in the file system.

**isfile()**
Returns True if this is a regular file in the file system.

**isdir()**
Returns True if this is a directory in the file system.

**open**(mode='r')
Returns an open file-like object from this path.

**copy**(dst)
Copies a file to dst path like Python’s shutil.copy.
Note that you can’t copy between file system implementations.

**move**(dst)
Moves a file to dst path like Python’s os.rename.

**remove()**
Removes a file.

**listdir()**
List directory contents
Returns a list containing path instances of the entries in the directory.

**chdir()**
Changes the current working directory to this path

**mkdir()**
Creates a new directory at this path.
If an item at this path already exists OSError is raised. This method ignores any trailing separator.

**makedirs()**
Recursive directory creation function.
Like mkdir(), but makes all intermediate-level directories needed to contain the leaf directory.
The default implementation repeatedly uses a combination of split and mkdir.

**walk()**
A generator function that walks the file system
Similar to os.walk. For each directory in the tree rooted at this path (including this path itself), it yields a 3-tuple of:
(dirpath, dirnames, filenames)

dirpath is an instance, dirnames and filenames are lists of path instances.

rmmtree (ignore_errors=False)
Removes the tree rooted at this directory

ignore_errors can be used to ignore any errors from the file system.

6.11.1 Accessing the Local File System

class pyslet.vfs.OSFilePath(*path)
Bases: pyslet.vfs.VirtualFilePath

A concrete implementation mapping to Python’s os modules
In most cases the methods map straightforwardly to functions in os and os.path.

fs_name = ''
An empty string.
The file system name affects the way URIs are interpreted, an empty string is consistent with the use of file:// to reference the local file system.
supports_unicode_filenames = False
Copied from os.path
That means you won’t know ahead of time whether paths are expected as binary or unicode strings. In most cases it won’t matter as the methods will convert as appropriate but it does affect the type of the static path constants defined below.
supports_unc = False
Automatically determined from os.path
Tests if os.path has defined splitunc.
supports_drives = False
Automatically determined
The method chosen is straight out of the documentation for os.path. We join the segments “C:” and “foo” and check to see if the result contains the path separator or not.
codec = 'UTF-8'
as returned by sys.getfilesystemencoding()
sep = '/'
copied from os.sep
curdir = '.'
copied from os.curdir
pardir = '..'
copied from os.pardir
ext = '.'
copied from os.extsep
drive_sep = ':'
always set to ':'
Correctly set to either binary or character string depending on the setting of supports_unicode_filenames.
empty = ''
Set to the empty string

Uses either a binary or character string depending on the setting of supports_unicode_filenames.

## 6.11.2 Misc Definitions

class pyslet.vfs.ZipHooks
    Bases: object

    Context manager for compatibility with zipfile

    The zipfile module allows you to write either a string or the contents of a named file to a zip archive. This class monkey-patches the builtin open function and os.stat with versions that support VirtualFilePath objects allowing us to copy the contents of a virtual represented file path directly to a zip archive without having to load it into memory first.

    For more information on this approach see this blog post.

    This implementation uses a lock on the class attributes to ensure thread safety.

    As currently implemented, Pyslet does not contain a full implementation of VirtualFilePath so this class is provided in readiness for a more comprehensive implementation based on pyslet.blockstore.StreamStore.

    • genindex
    • modindex
    • search
Welcome to Pyslet

Note: You are reading the latest development version of the documentation which corresponds to the master branch of the source on GitHub. The last release on PyPi was pyslet-0.7.20170805 and the corresponding documentation is here.

Pyslet is a Python package for Standards in Learning Education and Training (LET). It implements a number of LET-specific standards, including IMS QTI, Content Packaging and Basic LTI. It also includes support for some general standards, including the data access standard OData (see http://www.odata.org).

Pyslet was originally written to be the engine behind the QTI migration tool but it can be used independently as a support module for your own Python applications.

Full documentation is hosted at http://pyslet.readthedocs.org

Pyslet currently supports Python 2.6, 2.7 and 3.3+, see docs for details.

7.1 Distribution

Pyslet is developed on GitHub: https://github.com/swl10/pyslet but it can be downloaded and installed from the popular PyPi package distribution site: https://pypi.python.org/pypi/pyslet using pip.

While Pyslet is being actively developed the version on PyPi may lag a few months behind the master branch on GitHub. The unittests are fairly comprehensive and are automatically run against the master branch using TravisCI: [build passing]

Users of older Python builds (e.g., Python 2.6 installed on older OS X versions) should be aware that pip may well fail to install itself or other modules due to a failure to connect to the PyPi repository. Fixing this is hard and installing from source is recommended instead if you are afflicted by this issue.
7.1.1 Installing from Source

The Pyslet package contains a setup.py script so you can install it by downloading the compressed archive, uncompressing it and then running the following command inside the package:

```bash
python setup.py install
```

Windows users should note that when downloading a zipped archive of the distribution some unitests may fail due to the ambiguity in character encoding file names in zip archives. This is not an issue with Pyslet itself but an issue with some of the test data in the unitests folder. If you use Git (or GitHub desktop) to checkout the master instead then the unitests should work, please report any errors as the continuous build system does not catch Windows-specific bugs.

7.1.2 Current Status & Road Map

Pyslet is going through a transition process at the moment as the QTI migration tool that drives its development is gradually moving towards being distributed as an LTI tool rather than a desktop application.

The OData support is fairly robust, it is used to run the Cambridge Weather OData service which can be found at http://odata.pyslet.org/weather

What’s next?

- OData version 4: this will be a rewrite of the OData modules though they will ultimately behave in a similar way to the existing sub-package.
- MySQL shim for the OData SQL storage model (90% complete and functional)
- Improved support for LTI to take it beyond ‘basic’ (60% complete)

I also write about Pyslet on my blog: http://swl10.blogspot.co.uk/search/label/Pyslet

7.1.3 Feedback

The best way to get something changed is to create an issue or Pull request on GitHub, however, my contact details are available there on my profile page if you just want to drop me an email with a suggestion or question.

7.1.4 License

Pyslet is distributed under the ‘New’ BSD license: http://opensource.org/licenses/BSD-3-Clause, this decision was inherited from the early days of the code. Although Copyright to much of the source is owned by the author personally earlier parts are owned by the University of Cambridge and are marked as such.

Pyslet is written and maintained by the main author on a spare time basis and is not connected to my current employer.

7.1.5 Acknowledgements

Thank you to everyone who has raised issues, questions and pull requests on GitHub!

Some historical information is available on the QTI Migration tool’s Google Code project: https://code.google.com/p/qtimigration/

Some of the code was written in the 1990s and it owes a lot to the University of Cambridge and, in particular, to the team I worked with at UCLES (aka Cambridge Assessment) who were instrumental in getting this project started.
7.2 Format of the Documentation

The documentation has been written using ReStructuredText, a simple format created as part of the docutils package on SourceForge. The documentation files you are most likely reading have been generated using Sphinx. Parts of the documentation are auto-generated from the Python source files to make it easier to automatically discover the documentation using other tools capable of reading Python docstrings. However, this requires that the docstrings be written using ReStructuredText too, which means there is some additional markup for python-cross referencing in the code that may not be interpretable by other system (see below for details).

- ReStructuredText Primer: http://docutils.sourceforge.net/docs/user/rst/quickstart.html
  - Quick Reference: http://docutils.sourceforge.net/docs/user/rst/quickref.html
  - Sphinx: http://sphinx.pocoo.org/
    - Autodoc externsion: http://sphinx.pocoo.org/ext/autodoc.html
    - Python-cross references: http://sphinx.pocoo.org/domains.html#python-roles
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>pyslet.html401</td>
<td>365</td>
</tr>
<tr>
<td>pyslet.http.auth</td>
<td>238</td>
</tr>
<tr>
<td>pyslet.http.client</td>
<td>231</td>
</tr>
<tr>
<td>pyslet.http.cookie</td>
<td>269</td>
</tr>
<tr>
<td>pyslet.http.grammar</td>
<td>263</td>
</tr>
<tr>
<td>pyslet.http.messages</td>
<td>242</td>
</tr>
<tr>
<td>pyslet.http.params</td>
<td>255</td>
</tr>
<tr>
<td>pyslet.imscpv1p0</td>
<td>107</td>
</tr>
<tr>
<td>pyslet.imscpv1p2</td>
<td>21</td>
</tr>
<tr>
<td>pyslet.iso8601</td>
<td>431</td>
</tr>
<tr>
<td>pyslet.odata2.client</td>
<td>200</td>
</tr>
<tr>
<td>pyslet.odata2.core</td>
<td>195</td>
</tr>
<tr>
<td>pyslet.odata2.csdl</td>
<td>160</td>
</tr>
<tr>
<td>pyslet.odata2.memds</td>
<td>204</td>
</tr>
<tr>
<td>pyslet.odata2.metadata</td>
<td>198</td>
</tr>
<tr>
<td>pyslet.odata2.server</td>
<td>230</td>
</tr>
<tr>
<td>pyslet.odata2.sqllds</td>
<td>204</td>
</tr>
<tr>
<td>pyslet.pep8</td>
<td>17</td>
</tr>
<tr>
<td>pyslet.py2</td>
<td>11</td>
</tr>
<tr>
<td>pyslet.py26</td>
<td>11</td>
</tr>
<tr>
<td>pyslet.qtiv1.assessment</td>
<td>51</td>
</tr>
<tr>
<td>pyslet.qtiv1.common</td>
<td>36</td>
</tr>
<tr>
<td>pyslet.qtiv1.core</td>
<td>30</td>
</tr>
<tr>
<td>pyslet.qtiv1.objectbank</td>
<td>50</td>
</tr>
<tr>
<td>pyslet.qtiv1.xml</td>
<td>49</td>
</tr>
<tr>
<td>pyslet.qtiv2.content</td>
<td>58</td>
</tr>
<tr>
<td>pyslet.qtiv2.core</td>
<td>103</td>
</tr>
<tr>
<td>pyslet.qtiv2.expressions</td>
<td>89</td>
</tr>
<tr>
<td>pyslet.qtiv2.interactions</td>
<td>60</td>
</tr>
<tr>
<td>pyslet.qtiv2.items</td>
<td>52</td>
</tr>
<tr>
<td>pyslet.qtiv2.metadata</td>
<td>106</td>
</tr>
<tr>
<td>pyslet.qtiv2.processing</td>
<td>83</td>
</tr>
<tr>
<td>pyslet.qtiv2.tests</td>
<td>53</td>
</tr>
<tr>
<td>pyslet.qtiv2.variables</td>
<td>68</td>
</tr>
<tr>
<td>pyslet.qtiv2.xml</td>
<td>107</td>
</tr>
<tr>
<td>pyslet.rfc2396</td>
<td>409</td>
</tr>
<tr>
<td>pyslet.rfc4287</td>
<td>424</td>
</tr>
<tr>
<td>pyslet.rfc5023</td>
<td>429</td>
</tr>
<tr>
<td>pyslet.streams</td>
<td>451</td>
</tr>
<tr>
<td>pyslet.unicode5</td>
<td>444</td>
</tr>
<tr>
<td>pyslet.urn</td>
<td>421</td>
</tr>
<tr>
<td>pyslet.vfs</td>
<td>455</td>
</tr>
<tr>
<td>pyslet.wsgi</td>
<td>279</td>
</tr>
<tr>
<td>pyslet.xml.namespace</td>
<td>332</td>
</tr>
<tr>
<td>pyslet.xml.parser</td>
<td>337</td>
</tr>
<tr>
<td>pyslet.xml.structures</td>
<td>301</td>
</tr>
<tr>
<td>pyslet.xml.xsdatatypes</td>
<td>356</td>
</tr>
</tbody>
</table>
Symbols

__contains__() (pyslet.odata2.csdl.DictionaryLike method), 192
__delitem__() (pyslet.odata2.csdl.DictionaryLike method), 192
__eq__() (pyslet.odata2.csdl.SimpleValue method), 171
__getitem__() (pyslet.odata2.csdl.DictionaryLike method), 192
__getitem__() (pyslet.odata2.csdl.NameTableMixin method), 190
__getitem__() (pyslet.qtiv2.variables.RecordContainer method), 82
__getitem__() (pyslet.qtiv2.variables.SessionState method), 76
__getitem__() (pyslet.qtiv2.variables.TestSessionState method), 78
__iter__() (pyslet.odata2.csdl.DictionaryLike method), 192
__iter__() (pyslet.odata2.csdl.Entity method), 167
__iter__() (pyslet.odata2.csdl.NameTableMixin method), 191
__len__() (pyslet.odata2.csdl.DictionaryLike method), 192
__len__() (pyslet.odata2.csdl.NameTableMixin method), 191
__len__() (pyslet.qtiv2.variables.TestSessionState method), 78
__setitem__() (pyslet.odata2.csdl.DictionaryLike method), 192
__setitem__() (pyslet.qtiv2.variables.RecordContainer method), 83
__setitem__() (pyslet.qtiv2.variables.SessionState method), 76
__unicode__() (pyslet.odata2.csdl.SimpleValue method), 171
__weakref__ (pyslet.odata2.csdl.DictionaryLike attribute), 193

A
A (class in pyslet.html401), 387
Abbr (class in pyslet.html401), 388
abort_sending() (pyslet.http.messages.Message method), 246
abs_path (pyslet.rfc2396.URI attribute), 411
abspath() (pyslet.vfs.VirtualFilePath method), 457
Accept (class in pyslet.rfc5023), 430
Accept (pyslet.rfc5023.Collection attribute), 430
AcceptCharsetItem (class in pyslet.http.messages), 252
AcceptCharsetList (class in pyslet.http.messages), 252
AcceptEncodingItem (class in pyslet.http.messages), 252
AcceptEncodingList (class in pyslet.http.messages), 252
AcceptItem (class in pyslet.http.messages), 252
AcceptLanguageItem (class in pyslet.http.messages), 252
AcceptLanguageList (class in pyslet.http.messages), 252
AcceptList (class in pyslet.http.messages), 251
AcceptRanges (class in pyslet.http.messages), 253
AcceptToken (class in pyslet.http.messages), 253
AcceptTokenList (class in pyslet.http.messages), 253
access_time (pyslet.http.cookie.Cookie attribute), 271
Action (class in pyslet.qtiv1.core), 30
active_cleanup() (pyslet.http.client.Client method), 235
active_count() (pyslet.http.client.Client method), 234
Add() (pyslet.html401.Length method), 376
add() (pyslet.html401.Length method), 376
add_char() (pyslet.vertx5.CharClass method), 445
add_child() (pyslet.html401.XHTMLElement method), 368
add_child() (pyslet.qtiv1.common.ContentMixin method), 37
add_child() (pyslet.xml.structures.Document method), 304
add_child() (pyslet.xml.structures.Element method), 311
add_child() (pyslet.xml.structures.Node method), 302
add_class() (pyslet.vertx5.CharClass method), 446
add_credentials() (pyslet.http.client.Client method), 235
add_data() (pyslet.xml.structures.Element method), 314
add_header() (pyslet.wsgi.WSGIContext method), 286
add_join() (pyslet.odata2.sqlds.SQLAssociationCollection
method), 223
add_join() (pyslet.odata2.sqlds.SQLCollectionBase
method), 215
add_options() (pyslet.imsbltiv1p0.ToolProviderApp
class method), 110
add_options() (pyslet.wsgi.WSGIApp class method), 290
add_options() (pyslet.wsgi.WSGIDataApp class method), 293
add_param() (pyslet.odata2.sqlds.SQLParams method), 229
add_particles() (pyslet.xml.structures.XMLContentParticle
method), 323
add_private_suffix() (pyslet.http.cookie.CookieStore
method), 273
add_public_suffix() (pyslet.http.cookie.CookieStore
method), 273
add_range() (pyslet.unicode5.CharClass method), 445
add_success_path() (pyslet.http.auth.BasicCredentials
method), 241
add_to_content_package()
(pyslet.qtiv2.items.AssessmentItem method), 53
add_to_content_package()
(pyslet.qtiv2.xml.QTIDocument method), 107
add_to_cresource() (pyslet.html401.XHTMLElement
method), 368
add_to_cresource() (pyslet.qtiv2.core.QTIElement
method), 103
AddData() (pyslet.xmlstructures.Element method), 318
AddProperty() (pyslet.odata2.csdl.TypeInstance method), 179
Address (class in pyslet.html401), 388
AddToContentPackage() (pyslet.qtiv2.items.AssessmentItemAssociationSet
method), 53
AddToContentPackage() (pyslet.qtiv2.xml.QTIDocument
method), 107
AddToZip() (pyslet.imscp1p2.ContentPackage method), 24
AESAppCipher (class in pyslet.wsgi), 300
Align (class in pyslet.html401), 370
AlignMixin (class in pyslet.html401), 380
allocate_pixels() (pyslet.html401.MultiLength class
method), 377
Allow (class in pyslet.http.messages), 254
AltMaterial (class in pyslet.qtiv1.common), 38
ambiguous (pyslet.odata2.csdl.NavigationProperty
attribute), 187
And (class in pyslet.qtiv1.common), 48
And (class in pyslet.qtiv2.expressions), 95
ANY (pyslet.xmlstructures.ElementType attribute), 322
AnyN (class in pyslet.qtiv2.expressions), 95
api (pyslet.odata2.sqlds.SQLTransaction attribute), 228
app_cipher (pyslet.wsgi.WSGIDataApp attribute), 294
APP_NAMESPACE (in module pyslet.rfc5023), 431
AppCipher (class in pyslet.wsgi), 298
APElement (class in pyslet.rfc5023), 430
AQUA (module pyslet.html401), 372
Area (class in pyslet.html401), 388
Area (class in pyslet.qtiv1.core), 31
AreaMapEntry (class in pyslet.qtiv2.variables), 73
AreaMapping (class in pyslet.qtiv2.variables), 72
ascii_sign() (pyslet.wsgi.AppCipher method), 299
AssessFeedback (class in pyslet.qtiv1.assessment), 52
Assessment (class in pyslet.qtiv1.assessment), 51
AssessmentControl (class in pyslet.qtiv1.assessment), 51
AssessmentItem (class in pyslet.qtiv2.items), 52
AssessmentItemRef (class in pyslet.qtiv2.tests), 58
AssessmentSection (class in pyslet.qtiv2.tests), 57
AssessmentTest (class in pyslet.qtiv2.tests), 53
AssessProcExtension (class in pyslet.qtiv1.assessment), 51
AssociableChoice (class in pyslet.qtiv2.interactions), 61
AssociateInteraction (class in pyslet.qtiv2.interactions), 62
Association (class in pyslet.odata2.csdl), 187
association (pyslet.odata2.csdl.AssociationSet attribute), 184
Association (pyslet.odata2.csdl.Schema attribute), 180
AssociationEnd (class in pyslet.odata2.csdl), 188
AssociationEnd (class in pyslet.odata2.csdl.Association
attribute), 187
associationEnd (pyslet.odata2.csdl.AssociationSetEnd
attribute), 185
associationName (pyslet.odata2.csdl.AssociationSet
attribute), 184
AssociationSet (class in pyslet.odata2.csdl), 184
AssociationSet (pyslet.odata2.csdl.EntityContainer
attribute), 180
AssociationSetEnd (class in pyslet.odata2.csdl), 184
ATOM_MIMETYPE (module pyslet.rfc4287), 429
ATOM_NAMESPACE (module pyslet.rfc4287), 429
ATOMCAT_MIMETYPE (module pyslet.rfc4287), 429
ATOMCAT_MIMETYPE (module pyslet.rfc5023), 431
AtomElement (class in pyslet.rfc4287), 429
 AtomId (class in pyslet.rfc4287), 426
 AtomId (pyslet.rfc4287.Entity attribute), 425
 AtomIdClass (pyslet.rfc4287.Entity attribute), 424
 AtomIdClass (pyslet.rfc4287.Feed attribute), 424
ATOMSVC_MIMETYPE (module pyslet.rfc4287), 431
attach_to_doc() (pyslet.xmlstructures.Element method), 313
attach_to_parent() (pyslet.xmlstructures.Element
method), 313
AttachToDocument() (pyslet.xmlstructures.Element
method), 318
AttachToParent() (pyslet.xml.structures.Element method), 318
attribute_lists (pyslet.xml.structures.XMLDTD attribute), 321
AttrsMixin (class in pyslet.html401), 383
Author (class in pyslet.rfc4287), 425
Author (pyslet.rfc4287.Entity attribute), 425
authority (pyslet.rfc2396.URI attribute), 411
AuthorizationRequired (class in pyslet.odata2.client), 204
auto_detect_encoding() (pyslet.xml.structures.XMLEntity method), 327
auto_fields() (pyslet.odata2.sqlds.SQLCollectionBase method), 217
auto_key() (pyslet.odata2.csdl.Entity method), 168
auto_redirect (pyslet.http.client.ClientRequest attribute), 236
aux_table (pyslet.odata2.sqlds.SQLEntityContainer attribute), 206

B
B (class in pyslet.html401), 389
backLink (pyslet.odata2.csdl.NavigationProperty attribute), 187
BadRequest (class in pyslet.wsgi), 300
BadSyntax (class in pyslet.http.grammar), 269
Base (class in pyslet.html401), 389
base (pyslet.http.auth.Credentials attribute), 240
base (pyslet.wsgi.WSGIApp attribute), 289
base_uri (pyslet.xml.structures.Document attribute), 304
BaseFont (class in pyslet.html401), 389
baseType (class in pyslet.html401), 389
baseType (pyslet.odata2.csdl.Type attribute), 185
baseType (pyslet.odata2.csdl.Value attribute), 78
BaseValue (class in pyslet.odata2.csdl, 172
BasicChallenge (class in pyslet.http.auth), 242
BasicCredentials (class in pyslet.http.auth), 241
BasicParser (class in pyslet.unicode5), 446
bce (pyslet.iso8601.Date attribute), 433
BDO (class in pyslet.html401), 389
begin_attempt() (pyslet.odata2.csdl.DictionaryLike method), 193
begin_session() (pyslet.odata2.csdl.DictionaryLike method), 193
Big (class in pyslet.html401), 389
bigclear() (pyslet.odata2.csdl.DictionaryLike method), 193
BinaryValue (class in pyslet.odata2.csdl), 172
bind() (pyslet.odata2.csdl.EntitySet method), 182
bind_entity() (pyslet.odata2.csdl.DeferredValue method), 176
bind_navigation() (pyslet.odata2.csdl.EntitySet method), 183
BindEntity() (pyslet.odata2.csdl.DeferredValue method), 177
bindings (pyslet.odata2.csdl.DeferredValue attribute), 176
BindNavigation() (pyslet.odata2.csdl.EntitySet method), 183
BLACK (in module pyslet.html401), 371
BlockContainer (class in pyslet.html401), 385
BlockInteraction (class in pyslet.html401, 384
BlockMixIn (class in pyslet.html401), 384
Blockquote (class in pyslet.html401), 389
BLITToolProvider (class in pyslet.imsbltiv1p0), 117
BLUE (in module pyslet.html401), 372
Body (class in pyslet.html401), 390
BodyColorsMixin (class in pyslet.html401), 381
BodyElement (class in pyslet.html401), 381
body (pyslet.xml.structures.XMLAttribute attribute), 326
boolean_from_str() (in module pyslet.xml.xsdatatypes), 357
boolean_to_str() (in module pyslet.xml.xsdatatypes), 357
BooleanValue (class in pyslet.odata2.csdl, 172
BooleanValue (class in pyslet.odata2.csdl, 172
Bparameters() (pyslet.http.params.Parameter class method), 257
Br (class in pyslet.html401), 390
BranchRule (class in pyslet.html401), 389
break_connection() (pyslet.odata2.sqlds.SQLEntityContainer method), 208
break_connection() (pyslet.odata2.sqlds.SQLiteEntityContainer method), 227
bstr() (pyslet.http.cookie.Cookie class method), 271
bstr() (pyslet.http.params.Parameter class method), 257
buff_text (pyslet.xml.structures.XMLElement attribute), 326
buff_text() (pyslet.xml.parser.XMLParser method), 340
buffered() (pyslet.streams.Pipe method), 452
BufferedStreamWrapper (class in pyslet.streams), 454
build_model() (pyslet.xml.structures.ElementType method), 323
build_particle_maps() (pyslet.xml.structures.XMLElementParticle method), 323
Button (class in pyslet.html401), 390
ButtonValue (class in pyslet.html401), 390
byte() (in module pyslet.py2), 15
byte_to_bstr() (module pyslet.py2), 16
byte_value() (module pyslet.py2), 15
ByteValue (class in pyslet.odata2.csdl), 172

C
CacheControl (class in pyslet.http.messages), 250
calculate_order_key() (pyslet.odata2.core.EntityCollection method), 196
calculate_order_key() (pyslet.odata2.csdl.EntityCollection method), 163

Index 469
call_wrapper() (pyslet.wsgi.WSGIApp method), 291

can_parse (pyslet.http.auth.Challenge attribute), 241

can_pretty_print() (pyslet.html401.FlowContainer method), 385

can_pretty_print() (pyslet.xmlstructures.Element method), 317

can_retry() (pyslet.http.ClientRequest method), 237

canonicalize() (pyslet.http.params.HTTPURL method), 256

canonicalize() (pyslet.rfc2396.ServerBasedURL method), 414

canonicalize() (pyslet.rfc2396.URI method), 412

canonicalize_data() (in module pyslet.rfc2396), 415

canread() (pyslet.streams.Pipe method), 453

canwrite() (pyslet.streams.Pipe method), 452

Caption (class in pyslet.html401), 391

Cardinality (class in pyslet.qtiv2.variables), 69

Cardinality() (pyslet.qtiv2.variables.Value method), 79

cardinality() (pyslet.qtiv2.variables.Value method), 79

cast() (pyslet.odata2.csdl.SimpleValue method), 171

Categories (class in pyslet.rfc5023), 430

Categories (pyslet.rfc5023.Collection attribute), 430

Category (class in pyslet.rfc4287), 425

Category (pyslet.rfc4287.Entity attribute), 425

Category (pyslet.rfc5023.Categories attribute), 430

CDATA (pyslet.xmlstructures.XMLAttributeDefinition attribute), 324

CDATA_END (in module pyslet.xmlstructures), 331

CDATA_START (in module pyslet.xmlstructures), 331

CellAlignMixinn (class in pyslet.html401), 381

Center (class in pyslet.html401), 391

century (pyslet.iso8601.Date attribute), 433

Century (pyslet.iso8601.Precision attribute), 443

Century (pyslet.iso8601.Truncation attribute), 443

cfail_page() (pyslet.wsgi.SessionApp method), 297

Challenge (class in pyslet.http.auth), 241

change_encoding() (pyslet.xmlstructures.XMLEntity method), 328

change_key() (pyslet.wsgi.AppCipher method), 298

character() (in module pyslet.py2), 14

CharClass (class in pyslet.unicode5), 445

chdir() (pyslet.vfs.VirtualFilePath method), 458

check_attributes() (pyslet.xmlparser.XMLParser method), 347

check_encoding() (pyslet.xmlparser.XMLParser method), 340

check_expected_particle() (pyslet.xmlparser.XMLParser method), 347

check_filter() (pyslet.odata2.core.EntityCollection method), 196

check_filter() (pyslet.odata2.csdl.EntityCollection method), 163

check_model() (pyslet.html401.XHTMLEntity method), 368

check_navigation_constraint() (pyslet.odata2.csdl.DeferredValue method), 177

check_navigation_constraints() (pyslet.odata2.csdl.Entity method), 167

check_pe_between_declarations() (pyslet.xmlparser.XMLParser method), 346

check_pre_conditions() (pyslet.qtiv2.tests.SectionPart method), 57

check_pre_conditions() (pyslet.qtiv2.tests.TestPart method), 56

check_public_suffix() (pyslet.http.cookie.CookieStore method), 274

check_redirect() (pyslet.wsgi.SessionApp method), 297

check_signature() (pyslet.wsgi.AppCipher method), 299

cast() (in module pyslet.http.grammar), 268

check_validity (pyslet.xmlparser.XMLParser attribute), 338

checkAllErrors (pyslet.xmlparser.XMLParser attribute), 338

checkCompatibility (pyslet.xmlparser.XMLParser attribute), 338

CheckNavigationConstraints() (pyslet.odata2.csdl.Entity method), 169

ChoiceElement() (pyslet.xmlstructures.Node method), 303

Choice (class in pyslet.qtiv2.interactions), 61

ChoiceInteraction (class in pyslet.qtiv2.interactions), 61

Chunk (class in pyslet.http.params), 259

chunk_size (pyslet.xmlstructures.XMLAttributeDefinition attribute), 326

cipher (pyslet.imsltiv1p0.ToolConsumer attribute), 113

cipher (pyslet.imsltiv1p0.ToolProvider attribute), 115

Cite (class in pyslet.html401), 391

class_map (pyslet.html401.XHTMLDocument attribute), 366

Clear (class in pyslet.html401), 371

clear() (pyslet.odata2.csdl.DictionaryLike method), 193

clear_bindings() (pyslet.odata2.csdl.DeferredValue method), 177

clear_keep_alive() (pyslet.http.messages.Message method), 245

clear_links() (pyslet.odata2.sqlds.SQLAssociationCollection method), 224

clear_links() (pyslet.odata2.sqlds.SQLEntityCollection method), 214

clear_links() (pyslet.odata2.sqlds.SQLReverseKeyCollection method), 222

clear_links_unbound() (pyslet.odata2.sqlds.SQLAssociationCollection class method), 224

clear_session_cookie() (pyslet.wsgi.SessionApp method), 299

Index
pyslet.imsbltiv1p0, 116
ContextClass (pyslet.imsbltiv1p0.ToolProviderApp attribute), 110
ContextClass (pyslet.wsgi.SessionApp attribute), 295
ContextClass (pyslet.wsgi.WSGIApp attribute), 288
Contributor (class in pyslet.rfc4287), 426
Contributor (pyslet.rfc4287.Entity attribute), 425
Cookie (class in pyslet.http.cookie), 270
CookieError (class in pyslet.http.cookie), 278
CookieParser (class in pyslet.http.cookie), 275
CookieStore (class in pyslet.http.cookie), 272
Coords (class in pyslet.html401), 372
copy() (pyslet.odata2.csdl.DictionaryLike method), 193
copy() (pyslet.odata2.csdl.SimpleValue class method), 171
copy() (pyslet.vfs.VirtualFileMethod method), 458
Copy() (pyslet.xml.structures.Element method), 318
copy_entity() (pyslet.odata2.csdl.EntityCollection method), 164
copy_value() (pyslet.qtiv2.variables.Value class method), 79
CopyEntity() (pyslet.odata2.csdl.EntityCollection method), 165
CopyValue() (pyslet.qtiv2.variables.Value class method), 79
CoreAttrsMixin (class in pyslet.html401), 381
Correct (class in pyslet.qtiv2.expressions), 91
CorrectResponse (class in pyslet.qtiv2.variables), 72
CPElement (class in pyslet.imscpv1p2), 27
Create() (pyslet.xml.structures.Document method), 306
create() (pyslet.xml.structures.Document method), 305
create_all_tables() (pyslet.odata2.sqlds.SQLEntityContainer method), 208
create_table() (pyslet.odata2.sqlds.SQLAssociationCollection class method), 224
create_table() (pyslet.odata2.sqlds.SQLEntityCollection method), 214
create_table_query() (pyslet.odata2.sqlds.SQLAssociationCollection method), 224
create_table_query() (pyslet.odata2.sqlds.SQLEntityCollection method), 214
created (pyslet.odata2.core.StreamInfo attribute), 197
creation_time (pyslet.http.cookie.Cookie attribute), 271
Credentials (class in pyslet.http.auth), 239
CSDLElement (class in pyslet.odata2.csdl), 180
csrf_token (pyslet.wsgi.SessionApp attribute), 295
cstr() (pyslet.http.cookie.Cookie class method), 271
ctest() (pyslet.wsgi.SessionApp method), 296
ctest_page() (pyslet.wsgi.SessionApp method), 297
curdir (pyslet.vfs.OSFilePath attribute), 459
curdir (pyslet.vfs.VirtualFilePath attribute), 456
cursor (pyslet.odata2.sqlds.SQLTransaction attribute), 228
CustomOperator (class in pyslet.qtiv2.expressions), 102

data_items() (pyslet.odata2.csdl.Entity method), 167
data_keys() (pyslet.odata2.csdl.Entity method), 167
data_services_version() (pyslet.odata2.metadata.DataServices method), 200
data_source (pyslet.wsgi.WSGIDataApp attribute), 294
DatabaseBusy (class in pyslet.odata2.sqlds), 230
DataKeys() (pyslet.odata2.csdl.Entity method), 169
DataServices (class in pyslet.odata2.metadata), 200
DataServicesVersion() (pyslet.odata2.metadata.DataServices method), 200
Date (class in pyslet.iso8601), 431
Date (class in pyslet.rfc4287), 428
date (pyslet.rfc4287.Date attribute), 429
datetime_from_str() (in module pyslet.xml.xsdatatypes), 358
datetime_to_str() (in module pyslet.xml.xsdatatypes), 358
DateTimeOffsetValue (class in pyslet.odata2.csdl), 173
DateTimeValue (class in pyslet.odata2.csdl), 173
day (pyslet.iso8601.Date attribute), 433
day_of_week() (in module pyslet.iso8601), 444
dbapi (pyslet.odata2.sqlds.SQLEntityContainer attribute), 206
DD (class in pyslet.html401), 392
Decade (pyslet.iso8601.Truncation attribute), 443
decimal_from_str() (in module pyslet.xml.xsdatatypes), 357
decimal_to_str() (in module pyslet.xml.xsdatatypes), 357
DecimalValue (class in pyslet.odata2.csdl), 173
declaration (pyslet.xml.parser.XMLParser attribute), 340
declaration (pyslet.xml.structures.Document attribute), 304
DeclarationError (class in pyslet.qtiv2.core), 104
declare (class in pyslet.html401), 373
declare() (pyslet.odata2.csdl.NameTableMixin method), 191
declare_attribute() (pyslet.xml.structures.XMLDTD method), 322
declare_element_type() (pyslet.xml.structures.XMLDTD method), 321
declare_entity() (pyslet.xml.structures.XMLDTD method), 321
declare_metadata() (pyslet.qtiv1.core.QTIElement method), 36
declare_notation() (pyslet.xml.structures.XMLDTD method), 321
declared_standalone() (pyslet.xml.parser.XMLParser attribute), 341
decode_quoted_string() (in module pyslet.http.grammar), 268
DecodeBoolean() (in module pyslet.xml.xsdatatypes), 364
DecodeDateTime() (in module pyslet.xml.xsdatatypes), 364

D

data_items() (pyslet.odata2.csdl.Entity method), 167
data_keys() (pyslet.odata2.csdl.Entity method), 167
data_services_version() (pyslet.odata2.metadata.DataServices method), 200
data_source (pyslet.wsgi.WSGIDataApp attribute), 294
DatabaseBusy (class in pyslet.odata2.sqlds), 230
DataKeys() (pyslet.odata2.csdl.Entity method), 169
DataServices (class in pyslet.odata2.metadata), 200
DataServicesVersion() (pyslet.odata2.metadata.DataServices method), 200
Date (class in pyslet.iso8601), 431
Date (class in pyslet.rfc4287), 428
date (pyslet.rfc4287.Date attribute), 429
datetime_from_str() (in module pyslet.xml.xsdatatypes), 358
datetime_to_str() (in module pyslet.xml.xsdatatypes), 358
DateTimeOffsetValue (class in pyslet.odata2.csdl), 173
DateTimeValue (class in pyslet.odata2.csdl), 173
day (pyslet.iso8601.Date attribute), 433
day_of_week() (in module pyslet.iso8601), 444
dbapi (pyslet.odata2.sqlds.SQLEntityContainer attribute), 206
DD (class in pyslet.html401), 392
Decade (pyslet.iso8601.Truncation attribute), 443
decimal_from_str() (in module pyslet.xml.xsdatatypes), 357
decimal_to_str() (in module pyslet.xml.xsdatatypes), 357
DecimalValue (class in pyslet.odata2.csdl), 173
declaration (pyslet.xml.parser.XMLParser attribute), 340
declaration (pyslet.xml.structures.Document attribute), 304
DeclarationError (class in pyslet.qtiv2.core), 104
declare (class in pyslet.html401), 373
declare() (pyslet.odata2.csdl.NameTableMixin method), 191
declare_attribute() (pyslet.xml.structures.XMLDTD method), 322
declare_element_type() (pyslet.xml.structures.XMLDTD method), 321
declare_entity() (pyslet.xml.structures.XMLDTD method), 321
declare_metadata() (pyslet.qtiv1.core.QTIElement method), 36
declare_notation() (pyslet.xml.structures.XMLDTD method), 321
declared_standalone() (pyslet.xml.parser.XMLParser method), 341
decode_quoted_string() (in module pyslet.http.grammar), 268
DecodeBoolean() (in module pyslet.xml.xsdatatypes), 364
DecodeDateTime() (in module pyslet.xml.xsdatatypes), 364
DecodeDecimal() (in module pyslet.xml.xsdatatypes), 364
DecodeDouble() (in module pyslet.xml.xsdatatypes), 364
DecodeFloat() (in module pyslet.xml.xsdatatypes), 364
DecodeInteger() (in module pyslet.xml.xsdatatypes), 365
DecodeLowerValue() (pyslet.html401.NamedBoolean class method), 369
DecodeLowerValue() (pyslet.xml.xsdatatypes.Enumeration class method), 361
DecodeName() (in module pyslet.xml.xsdatatypes), 364
DecodeTitleValue() (pyslet.xml.xsdatatypes.Enumeration class method), 361
DecodeUpperValue() (pyslet.html401.NamedBoolean class method), 369
DecodeUpperValue() (pyslet.xml.xsdatatypes.Enumeration class method), 361
DecodeValue() (pyslet.html401.NamedBoolean class method), 369
DecodeValue() (pyslet.xml.xsdatatypes.Enumeration class method), 361
DecodeValue() (pyslet.xml.xsdatatypes.Enumeration class method), 361
Dict() (pyslet.xml.xsdatatypes.Enumeration class method), 361
dict_from_str() (pyslet.xml.xsdatatypes.Enumeration class method), 360
dict_keys() (in module pyslet.py2), 16
dict_values() (in module pyslet.py2), 16
DiffString() (pyslet.xml.xsdatatypes.Document method), 306
DefVar (class in pyslet.qtiv1.common), 44
deepcopy() (pyslet.xml.structures.Element method), 315
default (pyslet.qtiv2.variables.LookupTable attribute), 74
default Fields() (pyslet.odata2.sqls.SQLCollectionBase method), 217
default_ns (pyslet.html401.XHTMLDocument attribute), 366
default_ns (pyslet.xml.namespace.NSDocument attribute), 333
DEFAULT (pyslet.xml.xsdatatypes.XMLAttributeDefinition attribute), 325
default_value (pyslet.odata2.csdl.Property attribute), 186
default_value (pyslet.xml.xsdatatypes.XMLAttributeDefinition attribute), 325
Defer (class in pyslet.html401), 373
DeferredValue (class in pyslet.odata2.csdl), 175
definition (pyslet.xml.xsdatatypes.XMLDeclaredEntity attribute), 328
Del (class in pyslet.html401), 392
Delete (class in pyslet.qtiv2.expressions), 94
Delete() (pyslet.odata2.csdl.Entity method), 169
delete() (pyslet.odata2.csdl.Entity method), 168
delete_entity() (pyslet.odata2.sqls.SQLEntityCollection method), 214
delete_file() (pyslet.imscpv1p2.ContentPackage method), 26
delete_link() (pyslet.odata2.sqls.SQLAssociationCollection method), 224
delete_link() (pyslet.odata2.sqls.SQLEntityCollection method), 214
delete_link() (pyslet.odata2.sqls.SQLNavigationCollection method), 221
delete_link() (pyslet.odata2.sqls.SQLReverseKeyCollection method), 222
DeleteChild() (pyslet.xml.xsdatatypes.Element method), 318
DeleteDependency() (pyslet.imscpv1p2.Resource method), 29
DeleteFile() (pyslet.imscpv1p2.ContentPackage method), 24
DeleteFile() (pyslet.imscpv1p2.Resource method), 29
Dependency (class in pyslet.imscpv1p2), 30
Dependency (pyslet.imscpv1p2.Resource attribute), 29
DependencyClass (pyslet.imscpv1p2.Resource attribute), 29
detach_from_doc() (pyslet.xml.xsdatatypes.Element method), 313
detach_from_parent() (pyslet.xml.xsdatatypes.Element method), 313
DetachFromDocument() (pyslet.xml.xsdatatypes.Element method), 318
DetachFromParent() (pyslet.xml.xsdatatypes.Element method), 318
detect_encoding() (in module pyslet.unicode5), 444
Dfn (class in pyslet.html401), 392
dict_from_str() (pyslet.xml.xsdatatypes.Enumeration class method), 360
dict_keys() (in module pyslet.py2), 16
dict_values() (in module pyslet.py2), 16
DictionaryLike (class in pyslet.odata2.csdl), 191
diff_string() (pyslet.xml.xsdatatypes.Document method), 306
DiffString() (pyslet.xml.xsdatatypes.Document method), 306
Index 473
find_parent() (pyslet.xml.structures.Element method), 313
find_visit() (pyslet.imsbltiv1p0.ToolProviderApp method), 111
FindChildren() (pyslet.xml.structures.Element method), 318
FindChildrenBreadthFirst() (pyslet.xml.structures.Element method), 318
FindChildrenDepthFirst() (pyslet.xml.structures.Element method), 318
FindParent() (pyslet.xml.structures.Element method), 318
finished() (pyslet.http.client.ClientRequest method), 237
first_byte (pyslet.http.messages.ContentRange attribute), 254
fixed (pyslet.rfc5023.Categories attribute), 430
FIXED (pyslet.xml.structures.XMLAttributeDefinition attribute), 325
fixedLength (pyslet.odata2.csdl.Property attribute), 186
fk_table (pyslet.odata2.sqlds.SQLEntityContainer attribute), 206
float_from_str() (in module pyslet.xml.xsdatatypes), 357
float_or_template_ref() (pyslet.qtiv2.expressions.Expression method), 90
float_to_str() (in module pyslet.xml.xsdatatypes), 357
FloatValue (class in pyslet.odata2.csdl), 172
FloatValue (class in pyslet.qtiv2.variables), 80
FlowContainer (class in pyslet.html401), 385
FlowContainerMixin (class in pyslet.qtiv1.common), 41
FlowMat (class in pyslet.qtiv1.common), 41
FlowMatContainer (class in pyslet.qtiv1.common), 41
FlowMixIn (class in pyslet.html401), 383
FlowMixIn (class in pyslet.qtiv1.common), 41
flush() (pyslet.streams.Pipe method), 453
flush_dns() (pyslet.http.client.Client method), 235
flush_wait() (pyslet.streams.Pipe method), 453
Font (class in pyslet.html401), 393
force_ascii() (in module pyslet.py2), 14
force_bytes() (in module pyslet.py2), 15
force_text() (in module pyslet.py2), 13
Form (class in pyslet.html401), 393
format_parameters() (in module pyslet.http.grammar), 269
format_re() (pyslet.unicode5.CharClass method), 445
FormCtrlMixin (class in pyslet.html401), 384
fragment (pyslet.rfc2396.URI attribute), 411
Frame (class in pyslet.html401), 394
FrameElement (class in pyslet.html401), 387
Frameset (class in pyslet.html401), 387
from_end (pyslet.odata2.csdl.NavigationCollection attribute), 178
from_entity (pyslet.odata2.csdl.DeferredValue attribute), 175
from_entity (pyslet.odata2.csdl.NavigationCollection attribute), 178
from_http_str() (pyslet.http.params.FullDate class method), 258
from_julian() (pyslet.iso8601.Date class method), 435
from_now() (pyslet.iso8601.Date class method), 434
from_now() (pyslet.iso8601.Time class method), 438
from_now() (pyslet.iso8601.TimePoint class method), 442
from_now_utc() (pyslet.iso8601.TimePoint class method), 443
from_octets() (pyslet.rfc2396.URI class method), 410
from_path() (pyslet.rfc2396.URI class method), 411
from_property() (pyslet.odata2.csdl.EDMValue class method), 179
from_str() (pyslet.html401.Cords class method), 373
from_str() (pyslet.html401.Length class method), 375
from_str() (pyslet.html401.MultiLength class method), 377
from_str() (pyslet.html401.NamedBoolean class method), 369
from_str() (pyslet.http.auth.Challenge class method), 242
from_str() (pyslet.http.auth.Credentials class method), 240
from_str() (pyslet.http.cookie.Cookie class method), 272
from_str() (pyslet.http.cookie.Section4Cookie class method), 272
from_str() (pyslet.http.messages.AcceptItem class method), 252
from_str() (pyslet.http.messages.AcceptList class method), 251
from_str() (pyslet.http.messages.AcceptRanges class method), 253
from_str() (pyslet.http.messages.AcceptToken class method), 253
from_str() (pyslet.http.messages.AcceptTokenList class method), 253
from_str() (pyslet.http.messages.Allow class method), 254
from_str() (pyslet.http.messages.CacheControl class method), 250
from_str() (pyslet.http.messages.ContentRange class method), 254
from_str() (pyslet.http.messages.MediaRange class method), 251
from_str() (pyslet.http.params.Chunk class method), 259
from_str() (pyslet.http.params.EntityTag class method), 261
from_str() (pyslet.http.params.HTTPVersion class method), 258
from_str() (pyslet.http.params.LanguageTag class method), 261

Index 477
from_str() (pyslet.http.params.MediaType class method), 260
from_str() (pyslet.http.params.ProductToken class method), 260
from_str() (pyslet.http.params.TransferEncoding class method), 259
from_str() (pyslet.iso8601.Date class method), 434
from_str() (pyslet.iso8601.Time class method), 438
from_str() (pyslet.iso8601.TimePoint class method), 441
from_str() (pyslet.xml.xsdatatypes.Enumeration class method), 359
from_str_lower() (pyslet.html401.NamedBoolean class method), 369
from_str_lower() (pyslet.xml.xsdatatypes.Enumeration class method), 360
from_str_title() (pyslet.xml.xsdatatypes.Enumeration class method), 360
from_str_upper() (pyslet.html401.NamedBoolean class method), 369
from_str_upper() (pyslet.xml.xsdatatypes.Enumeration class method), 360
from_string_format() (pyslet.iso8601.Date class method), 434
from_string_format() (pyslet.iso8601.Time class method), 438
from_string_format() (pyslet.iso8601.TimePoint class method), 441
from_struct_time() (pyslet.iso8601.Date class method), 434
from_struct_time() (pyslet.iso8601.Time class method), 438
from_struct_time() (pyslet.iso8601.TimePoint class method), 441
generate_ctoken() (pyslet.odata2.csdl.Entity method), 169
generate_entity_set_in_json() (pyslet.odata2.core.EntityCollection method), 196
generate_entity_type_in_json() (pyslet.odata2.core.Entity method), 196
generate_key() (pyslet.imscbltiv1p0.BLTIToolProvider method), 117
generate_key() (pyslet.wsgi.WSGIContext method), 286
generate_xml() (pyslet.xml.xmlstructures.Document method), 306
generate_xml() (pyslet.xml.xmlstructures.Element method), 314
generate_xml() (pyslet.xml.xmlstructures.Element method), 317
generate_xml() (pyslet.xml.xmlstructures.Document method), 306
generate_xml() (pyslet.xml.xmlstructures.Document method), 306
generatexml() (pyslet.xml.xmlstructures.Element method), 318
Generator (class in pyslet.rfc4287), 426
Generator (pyslet.rfc4287.Source attribute), 424
get() (pyslet.odata2.csdl.DictionaryLike method), 193
generate_entity_set_in_json() (pyslet.odata2.core.EntityCollection method), 196
generate_entity_type_in_json() (pyslet.odata2.core.Entity method), 196
generate_key() (pyslet.imscbltiv1p0.BLTIToolProvider method), 117
generatexml() (pyslet.xml.xmlstructures.Document method), 306
generatexml() (pyslet.xml.xmlstructures.Document method), 306
get() (pyslet.odata2.csdl.DictionaryLike method), 193
get_absolute_day() (pyslet.iso8601.Date method), 433
get_accept() (pyslet.http.messages.Request method), 243
generate_key() (pyslet.imscbltiv1p0.BLTIToolProvider method), 117
get_accept_charset() (pyslet.http.messages.Request method), 243
generate_value() (pyslet.xml.xmlstructures.Document method), 306
get_accept_encoding() (pyslet.http.messages.Request method), 243
generatexml() (pyslet.xml.xmlstructures.Document method), 306
get_accept_ranges() (pyslet.http.messages.Response method), 244
get_addr() (pyslet.rfc2396.ServerBasedURL method), 411
generatexml() (pyslet.xml.xmlstructures.Document method), 306
get_age() (pyslet.http.messages.Response method), 244
get_allow() (pyslet.http.messages.Message method), 248
get_app_root() (pyslet.wsgi.WSGIContext method), 286
generate_key() (pyslet.imscbltiv1p0.BLTIToolProvider method), 117
get_attribute() (pyslet.xml.xmlstructures.Element method), 318
get_attribute() (pyslet.xml.xmlstructures.Element method), 317
get_attribute() (pyslet.xml.xmlstructures.Element method), 315
get_attribute() (pyslet.xml.xmlstructures.Element method), 315
get_attribute definition() (pyslet.xml.xmlstructures.XMLDTD attribute), 321
GETTER (in module pyslet.rfc4287), 426
get_attribute_list() (pyslet.xml.xmlstructures.XMLDTD method), 322
get_attribute_name() (pyslet.xml.xmlstructures.XMLDTD method), 322
get_attribute_value() (pyslet.xml.xmlstructures.XMLDTD method), 322
get_attribute_value() (pyslet.xml.xmlstructures.XMLDTD method), 322
generate_key() (pyslet.imscbltiv1p0.BLTIToolProvider method), 117
generatexml() (pyslet.xml.xmlstructures.Document method), 306
generatexml() (pyslet.xml.xmlstructures.Document method), 306
generatexml() (pyslet.xml.xmlstructures.Document method), 306
get_age() (pyslet.http.messages.Response method), 244
get_allow() (pyslet.http.messages.Message method), 248
get_app_root() (pyslet.wsgi.WSGIContext method), 286
get_attribute() (pyslet.xml.xmlstructures.Element method), 318
get_attribute() (pyslet.xml.xmlstructures.Element method), 317
get_attribute() (pyslet.xml.xmlstructures.Element method), 315
get_attribute definition() (pyslet.xml.xmlstructures.XMLDTD attribute), 321
G
Gap (class in pyslet.qtiv2.interactions), 64
gap_attribute_name() (pyslet.xml.xmlstructures.XMLDTD attribute), 321
gap_attribute_value() (pyslet.xml.xmlstructures.XMLDTD method), 322
get_attribute() (pyslet.xml.xmlstructures.Element method), 318
get_attribute() (pyslet.xml.xmlstructures.Element method), 317
get_attribute() (pyslet.xml.xmlstructures.Element method), 315
get_attribute definition() (pyslet.xml.xmlstructures.XMLDTD attribute), 321
INDEX
method), 322
get_attributes() (pyslet.xml.structures.Element method), 310
get_authorization() (pyslet.http.messages.Message method), 248
get_base() (pyslet.xml.structures.Document method), 304
get_base() (pyslet.xml.structures.Element method), 315
get_base() (pyslet.xml.structures.Node method), 302
get_branch_target() (pyslet.qtiv2.tests.SectionPart method), 57
get_branch_target() (pyslet.qtiv2.tests.TestPart method), 56
get_cache_control() (pyslet.http.messages.Message method), 248
get_calendar_day() (pyslet.iso8601.Date method), 433
get_calendar_string() (pyslet.iso8601.Date method), 434
get_calendar_string() (pyslet.iso8601.TimePoint method), 442
get_calendar_time_point() (pyslet.iso8601.TimePoint method), 440
get_canonical_children() (pyslet.xml.structures.Element method), 311
get_canonical_root() (pyslet.rfc2396.URI method), 412
get_canonical_zone() (pyslet.iso8601.Date method), 437
get_child_class() (pyslet.html401.TRContainer method), 387
get_child_class() (pyslet.xml.structures.Node method), 302
get_children() (pyslet.xml.structures.Document method), 304
get_children() (pyslet.xml.structures.Element method), 311
get_children() (pyslet.xml.structures.Node method), 301
get_collection_class() (pyslet.odata2.sqlqs.SQLEntityContainer method), 207
get_collection_class() (pyslet.odata2.sqlqs.SQLiteEntityContainer method), 226
get_connection() (pyslet.http.messages.Message method), 248
get_content() (pyslet.wsgi.WSGIContext method), 287
get_content_encoding() (pyslet.http.messages.Message method), 249
get_content_language() (pyslet.http.messages.Message method), 249
get_content_length() (pyslet.http.messages.Message method), 249
get_content_location() (pyslet.http.messages.Message method), 249
get_content_md5() (pyslet.http.messages.Message method), 249
get_content_range() (pyslet.http.messages.Message method), 249
get_content_type() (pyslet.http.messages.Message method), 249
get_context() (pyslet.imscpv1p2.ToolConsumer method), 114
get_context() (pyslet.xml.parser.XMLParser method), 340
generate_cookie() (pyslet.http.messages.Request method), 243
generate_cookies() (pyslet.wsgi.WSGIContext method), 288
generate_correct_value() (pyslet.qtiv2.variables.ResponseDeclaration method), 72
generate_current_question() (pyslet.qtiv2.variables.TestSessionState method), 78
generate_current_test_part() (pyslet.qtiv2.variables.TestSessionState method), 78
generate_date() (pyslet.http.messages.Message method), 249
generate_declaration() (pyslet.qtiv2.variables.SessionState method), 75
generate_default_value() (pyslet.qtiv2.variables.VariableDeclaration method), 69
generate_document() (pyslet.xml.structures.Element method), 310
generate_document_class() (pyslet.xml.parser.XMLParser method), 343
generate_element_by_id() (pyslet.xml.structures.Document method), 305
generate_element_class() (pyslet.imscpv1p2.ManifestDocument method), 26
generate_element_class() (pyslet.odata2.metadata.Document class method), 199
generate_element_class() (pyslet.qtiv1.xml.QTIDocument method), 49
generate_element_class() (pyslet.rfc5023.Document class method), 430
generate_element_class() (pyslet.xml.namespace.NSDocument class method), 333
generate_element_class() (pyslet.xml.structures.Document class method), 304
generate_element_class() (pyslet.xml.structures.Node class method), 301
generate_element_type() (pyslet.xml.structures.XMLelement type method), 301
generate_entity() (pyslet.odata2.csdl.DeferredValue method), 321
generate_entity() (pyslet.xml.structures.XMLDocument method), 176
generate_entry_point() (pyslet.imscpv1p2.Resource method), 29
generate_etag() (pyslet.http.messages.Message method), 244
generate_external_entity() (pyslet.xml.parser.XMLParser method), 341
generate_feed_url() (pyslet.rfc5023.Collection method), 430
generate_file_name() (pyslet.rfc2396.URI method), 414

Index 479
get_fk_class() (pyslet.odata2.sqlds.SQLEntityContainer method), 208
get_fk_class() (pyslet.odata2.sqlds.SQLiteEntityContainer method), 227
get_form() (pyslet.wsgi.WSGIContext method), 288
get_form_long() (pyslet.wsgi.WSGIContext method), 288
get_form_string() (pyslet.wsgi.WSGIContext method), 288
get_fqname() (pyslet.odata2.csdl.Association method), 187
get_fqname() (pyslet.odata2.csdl.EntitySet method), 182
get_fqname() (pyslet.odata2.csdl.Type method), 185
get_header() (pyslet.http.messages.Message method), 247
get_headerlist() (pyslet.http.messages.Message method), 247
get_item() (pyslet.qtiv2.tests.AssessmentItemRef method), 58
get_julian_day() (pyslet.iso8601.Date method), 435
get_key() (pyslet.odata2.csdl.EntitySet method), 182
get_key_dict() (pyslet.odata2.csdl.EntitySet method), 182
get_lang() (pyslet.html401.I18nMixin method), 382
get_lang() (pyslet.xml.structures.Document method), 304
get_lang() (pyslet.xml.structures.Element method), 316
get_lang() (pyslet.xml.structures.Node method), 303
get_last_modified() (pyslet.http.messages.Message method), 250
get_local_zone() (in module pyslet.iso8601), 444
get_location() (pyslet.http.messages.Response method), 244
get_location() (pyslet.odata2.core.NavigationCollection method), 197
get_location() (pyslet.odata2.csdl.EntityCollection method), 162
get_location() (pyslet.odata2.csdl.EntitySet method), 182
get_mimetype() (pyslet.odata2.metadata.Property method), 198
get_mimetype() (pyslet.odata2.metadata.Type method), 198
get_multiplicity() (pyslet.odata2.csdl.EntitySet method), 183
get_name() (pyslet.xml.structures.XMLDeclaredEntity method), 328
get_name() (pyslet.xml.structures.XMLElement method), 326
get_namespace() (pyslet.qtiv2.variables.TestSessionState method), 78
get_notification() (pyslet.xml.structures.XMLDT method), 195
get_notation() (pyslet.xml.structures.XMLOUT method), 321
get ns() (pyslet.xml.namespace.NSNode method), 332
get_nsdoc_class() (pyslet.xml.namespace.XMLNSParser method), 336
get_or_add_child() (pyslet.xml.navigation.Element method), 311
get_ordinal_day() (pyslet.iso8601.Date method), 433
get_ordinal_string() (pyslet.iso8601.Date method), 434
get_ordinal_string() (pyslet.iso8601.TimePoint method), 442
get_ordinal_time_point() (pyslet.iso8601.TimePoint method), 440
get_package_name() (pyslet.imscpv1p2.ContentPackage method), 26
get_parameter_entity() (pyslet.xml.structures.XMLDT method), 321
get_part() (pyslet.qtiv2.tests.AssessmentTest method), 54
get_pathname() (pyslet.rfc2396.FileURL method), 414
get_permissions() (pyslet.imscbltiv1p0.ToolProviderApp class method), 111
get_position_str() (pyslet.xml.structures.XMLEntity method), 327
get_precision() (pyslet.iso8601.Date method), 435
get_precision() (pyslet.iso8601.Time method), 439
get_precision() (pyslet.iso8601.TimePoint method), 443
get_prefix() (pyslet.xml.namespace.NSNode method), 332
get_prefix_map() (pyslet.xml.namespace.NSNode method), 333
get_qname() (pyslet.odata2.csdl.AssociationEnd method), 188
get_qname() (pyslet.odata2.csdl.AssociationSetEnd method), 188
get_query() (pyslet.wsgi.WSGIContext method), 287
get_registered_domain() (pyslet.http.cookie.CookieStore method), 274
get_resource() (pyslet.imscbltiv1p0.ToolConsumer method), 114
get_resource() (pyslet.imscbltiv1p0.ToolProviderApp method), 111
get_resource_title() (pyslet.imscbltiv1p0.ToolProviderApp method), 111
get_response() (pyslet.http.auth.Credentials method), 240
get_rk_class() (pyslet.odata2.sqlds.SQLEntityContainer method), 208
get_rk_class() (pyslet.odata2.sqlds.SQLiteEntityContainer method), 227
get_server_certificate_chain() (pyslet.http.client.Client class method), 234
get_set_cookie() (pyslet.http.messages.Response method), 244
get_source_path() (pyslet.odata2.metadata.EntityType method), 198
get_space() (pyslet.xml.structures.Document method),
Index

481
is_allowed_2396() (in module pyslet.rfc2396), 418
is_alpha() (in module pyslet.http.grammar), 267
is_alpha() (in module pyslet.rfc2396), 417
is_alphanum() (in module pyslet.rfc2396), 417
is_ascii() (in module pyslet.py2), 14
is_authority_reserved() (in module pyslet.rfc2396), 419
is_base_char() (in module pyslet.xml.parser), 355
is_char() (in module pyslet.http.grammar), 267
is_char() (in module pyslet.xml.parser), 354
is_char() (pyslet.xml.xsdatatypes.RegularExpressionParser method), 363
is_combining_char() (in module pyslet.xml.parser), 355
is_control() (in module pyslet.rfc2396), 418
is_cookie_octet() (in module pyslet.http.cookie), 277
is_ctl() (in module pyslet.http.grammar), 277
is_default_entity_container() (pyslet.odata2.metadata.EntityContainer method), 199
is_delimiter() (in module pyslet.http.cookie), 277
is_delims() (in module pyslet.rfc2396), 419
is_deterministic() (pyslet.xml.structures.ElementType method), 323
is_deterministic() (pyslet.xml.structures.XMLContentParticle method), 323
is_digit() (in module pyslet.http.grammar), 268
is_digit() (in module pyslet.rfc2396), 417
is_digits() (in module pyslet.http.grammar), 268
is_dirlke() (pyslet.vfs.VirtualFilePath method), 457
is_discouraged() (in module pyslet.xml.parser), 354
is_empty() (pyslet.vfs.VirtualFilePath method), 457
is_empty() (pyslet.xml.structures.Element method), 311
is_enc_name() (in module pyslet.xml.parser), 354
is_enc_name_start() (in module pyslet.xml.structures), 330
is_entity_collection() (pyslet.odata2.csdl.Entity method), 168
is_entity_collection() (pyslet.odata2.csdl.EntitySet method), 183
is_extender() (in module pyslet.xml.parser), 355
is_external() (pyslet.xml.structures.XMLDeclaredEntity method), 328
is_external() (pyslet.xml.structures.XMLEntity method), 326
is_extra_1738() (in module pyslet.rfc2396), 418
is_hex() (in module pyslet.http.grammar), 268
is_hex() (in module pyslet.rfc2396), 418
is_hex() (in module pyslet.urh.parser), 423
is_hexdigits() (in module pyslet.http.grammar), 268
is_hexinteger() (pyslet.http.grammar.WordParser method), 266
is_hostonly() (pyslet.http.cookie.Cookie method), 272
is_idempotent() (pyslet.http.messages.Request method), 243
is_ideographic() (in module pyslet.xml.parser), 355
is_inline() (pyslet.qtiv1.common.ContentMixin method), 37
is_inline() (pyslet.qtiv1.common.FlowMat method), 41
is_integer() (pyslet.http.grammar.WordParser method), 266
is_ldh_label() (in module pyslet.http.cookie), 278
is_letnum() (in module pyslet.urn), 423
is_letnumhyp() (in module pyslet.urn), 423
is_letter() (in module pyslet.xml.parser), 355
is_loalpha() (in module pyslet.http.grammar), 267
is_lowalpha() (in module pyslet.rfc2396), 417
is_lower() (in module pyslet.urn), 423
is_mark() (in module pyslet.rfc2396), 418
is_medialink_collection() (pyslet.odata2.core.EntityCollection method), 195
is_mixed() (pyslet.xml.structures.Element method), 311
is_name() (in module pyslet.xml.structures), 330
is_name_start_char() (in module pyslet.xml.structures), 330
is_navigation_property() (pyslet.odata2.csdl.Entity method), 168
is_non_delimiter() (in module pyslet.http.cookie), 277
is_non_digit() (in module pyslet.http.cookie), 277
is_null() (pyslet.odata2.csdl.Complex method), 175
is_null() (pyslet.odata2.csdl.EDMValue method), 179
is_null() (pyslet.qtiv2.variables.Value method), 79
is_number() (in module pyslet.urn), 423
is_octet() (in module pyslet.http.grammar), 267
is_open() (pyslet.xml.structures.XMLEntity method), 326
is_other() (in module pyslet.urn), 423
is_outcome() (pyslet.qtiv2.variables.ItemSessionState method), 77
is_outcome() (pyslet.qtiv2.variables.SessionState method), 76
is_path_segment_reserved() (in module pyslet.xml.structures), 330
is_persistent() (pyslet.http.cookie.Cookie method), 272
is_pubid_char() (in module pyslet.xml.parser), 354
is_query_reserved() (in module pyslet.http.cookie), 277
is_reserved() (in module pyslet.rfc2396), 419
is_reserved() (in module pyslet.urh.parser), 423
is_reserved_1738() (in module pyslet.rfc2396), 417
is_reserved_2396() (in module pyslet.rfc2396), 417
is_reserved_1738() (in module pyslet.rfc2396), 417
is_reserved_name() (in module pyslet.xml.structures), 330
is_response() (pyslet.qtiv2.variables.ItemSessionState method), 77
is_response() (pyslet.qtiv2.variables.SessionState method), 75
is_response() (pyslet.qtiv2.variables.TestSessionState method), 78
is_rldh_label() (in module pyslet.http.cookie), 278
is_root() (pyslet.vfs.VirtualFilePath method), 458
is_s() (in module pyslet.xml.xmlstructures), 330
is_s() (pyslet.xml.parser.XMLParser method), 343
is_safe_1738() (in module pyslet.rfc2396), 418
is_selected() (pyslet.odata2.csdl.Entity method), 168
is_separator() (in module pyslet.http.grammar), 268
is_separator() (pyslet.http.grammar.WordParser method), 266
is_single_component() (pyslet.vfs.VirtualFilePath method), 457
is_space() (in module pyslet.rfc2396), 418
is_string() (in module pyslet.py2), 13
is_subrole() (in module pyslet.imsbltiv1p0), 116
is_template() (pyslet.qtiv2.variables.SessionState method), 76
is_text() (in module pyslet.py2), 13
is_token() (pyslet.http.grammar.WordParser method), 265
is_trans() (in module pyslet.urn), 423
is_unc() (pyslet.vfs.VirtualFilePath method), 457
is_unreserved() (in module pyslet.rfc2396), 417
is_unreserved_1738() (in module pyslet.rfc2396), 418
is_unwise() (in module pyslet.rfc2396), 419
is_upalpha() (in module pyslet.rfc2396), 419
is_upalpha() (in module pyslet.http.grammar), 267
is_upper() (in module pyslet.urn), 423
is_valid() (pyslet.http.messages.ContentRange method), 254
is_valid_name() (in module pyslet.xml.namespace), 330
is_valid_nmtoken() (in module pyslet.xml.parser), 355
is_valid_space() (in module pyslet.xml.parser), 355
isabs() (pyslet.vfs.VirtualFilePath method), 458
isCollection (pyslet.odata2.csdl.DeferredValue attribute), 176
IsDefaultEntityContainer() (pyslet.odata2.metadata.EntityContainer method), 199
isdir() (pyslet.vfs.VirtualFilePath method), 458
IsEmpty() (pyslet.xml.xmlstructures.Element method), 319
IsEntityCollection() (pyslet.odata2.csdl.Entity method), 169
IsEntityCollection() (pyslet.odata2.csdl.EntitySet method), 183
isExpanded (pyslet.odata2.csdl.DeferredValue attribute), 176
isfile() (pyslet.vfs.VirtualFilePath method), 458
IsIndex (class in pyslet.html401), 398
IsMap (class in pyslet.html401), 375
IsMixed() (pyslet.xml.xmlstructures.Element method), 319
IsNavigationProperty() (pyslet.odata2.csdl.Entity method), 170
IsNull (class in pyslet.qtiv2.expressions), 93
IsNull() (pyslet.odata2.csdl.Complex method), 175
IsNull() (pyslet.odata2.csdl.SimpleValue method), 171
IsNull() (pyslet.qtiv2.variables.Value method), 79
IsOutcome() (pyslet.qtiv2.variables.SessionState method), 76
isRequired (pyslet.odata2.csdl.DeferredValue attribute), 175
IsResponse() (pyslet.qtiv2.variables.SessionState method), 76
IsTemplate() (pyslet.qtiv2.variables.SessionState method), 76
IsValidName() (pyslet.xml.xmlstructures.Element method), 319
ItemBody (class in pyslet.qtiv2.content), 58
ItemClass (pyslet.http.messages.AcceptCharsetList attribute), 252
ItemClass (pyslet.http.messages.AcceptEncodingList attribute), 252
ItemClass (pyslet.http.messages.AcceptLanguageList attribute), 252
ItemClass (pyslet.http.messages.AcceptTokenList attribute), 253
items() (pyslet.odata2.csdl.DictionaryLike method), 192
ItemSessionState (class in pyslet.qtiv2.variables), 76
iteritems() (pyslet.odata2.csdl.DictionaryLike method), 192
iterkeys() (pyslet.odata2.csdl.DictionaryLike method), 192
itervalues() (pyslet.odata2.csdl.DictionaryLike method), 192
itervalues() (pyslet.odata2.csdl.EntityCollection method), 165
J
join() (pyslet.vfs.VirtualFilePath method), 456
join_bytes() (in module pyslet.py2), 15
join_characters() (in module pyslet.py2), 14
join_clause() (pyslet.odata2.sqls.SQLCollectionBase method), 215
js_origin (pyslet.wsgi.WSGIApp attribute), 290
json_response() (pyslet.wsgi.WSGIApp method), 292
K
Kbd (class in pyslet.html401), 398
keep_alive (pyslet.http.messages.Message attribute), 245
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Class/Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>keep_encoding()</td>
<td>pyslet.xml.structures.XMLEntity method</td>
</tr>
<tr>
<td>keep_in_content()</td>
<td>pyslet.odata2.metadata.FeedCustomisationMixin method</td>
</tr>
<tr>
<td>KeepEncoding()</td>
<td>pyslet.xml.structures.XMLEntity method</td>
</tr>
<tr>
<td>KeepInContent()</td>
<td>pyslet.odata2.metadata.FeedCustomisationMixin method</td>
</tr>
<tr>
<td>Key</td>
<td>pyslet.odata2.csdl</td>
</tr>
<tr>
<td>key</td>
<td>pyslet.imsbltiv1p0.ToolConsumer attribute</td>
</tr>
<tr>
<td>Key</td>
<td>pyslet.odata2.csdl.EntityType attribute</td>
</tr>
<tr>
<td>key</td>
<td>pyslet.qtiv2.variables.TestSessionState attribute</td>
</tr>
<tr>
<td>key60()</td>
<td>module pyslet.wsgi</td>
</tr>
<tr>
<td>key_dict()</td>
<td>pyslet.odata2.csdl.Entity method</td>
</tr>
<tr>
<td>key_dict()</td>
<td>pyslet.odata2.csdl.EntitySet method</td>
</tr>
<tr>
<td>key_fields()</td>
<td>pyslet.odata2.sqlds.SQLCollectionBase method</td>
</tr>
<tr>
<td>KeyDict()</td>
<td>pyslet.odata2.csdl.Entity method</td>
</tr>
<tr>
<td>KeyKeys()</td>
<td>pyslet.odata2.csdl.EntitySet method</td>
</tr>
<tr>
<td>keyMap</td>
<td>pyslet.qtiv2.variables.TestSessionState attribute</td>
</tr>
<tr>
<td>keys</td>
<td>pyslet.odata2.csdl.EntitySet attribute</td>
</tr>
<tr>
<td>keys()</td>
<td>pyslet.odata2.csdl.DictionaryLike method</td>
</tr>
<tr>
<td>Label</td>
<td>class in pyslet.html401</td>
</tr>
<tr>
<td>label</td>
<td>pyslet.rfc4287.Category attribute</td>
</tr>
<tr>
<td>lang</td>
<td>pyslet.xmlstructures.Document attribute</td>
</tr>
<tr>
<td>LanguageTag</td>
<td>class in pyslet.http.params</td>
</tr>
<tr>
<td>last_byte</td>
<td>pyslet.http.messages.ContentRange attribute</td>
</tr>
<tr>
<td>Launch()</td>
<td>pyslet.imsbltiv1p0.ToolProvider method</td>
</tr>
<tr>
<td>launch()</td>
<td>pyslet.imsbltiv1p0.ToolProvider method</td>
</tr>
<tr>
<td>launch_redirec</td>
<td>pyslet.imsbltiv1p0.ToolProviderApp method</td>
</tr>
<tr>
<td>leap_year()</td>
<td>module pyslet.iso8601</td>
</tr>
<tr>
<td>leap_year()</td>
<td>pyslet.iso8601.Date method</td>
</tr>
<tr>
<td>left</td>
<td>pyslet.unicode5.ParserError attribute</td>
</tr>
<tr>
<td>Legend</td>
<td>class in pyslet.html401</td>
</tr>
<tr>
<td>Length</td>
<td>class in pyslet.html401</td>
</tr>
<tr>
<td>length</td>
<td>pyslet.rfc4287.Limit attribute</td>
</tr>
<tr>
<td>LI</td>
<td>class in pyslet.html401</td>
</tr>
<tr>
<td>LIME</td>
<td>module pyslet.html401</td>
</tr>
<tr>
<td>limit_clause()</td>
<td>pyslet.odata2.sqlds.SQLEntityContainer method</td>
</tr>
<tr>
<td>line_num</td>
<td>pyslet.xmlstructures.XMLEntity attribute</td>
</tr>
<tr>
<td>line_pos</td>
<td>pyslet.xmlstructures.XMLEntity attribute</td>
</tr>
<tr>
<td>Link</td>
<td>class in pyslet.html401</td>
</tr>
<tr>
<td>Link</td>
<td>class in pyslet.rfc4287</td>
</tr>
<tr>
<td>Link</td>
<td>pyslet.rfc4287.Entity attribute</td>
</tr>
<tr>
<td>link_json()</td>
<td>pyslet.odata2.core.Entity method</td>
</tr>
<tr>
<td>LinkClass</td>
<td>pyslet.rfc4287.Entity attribute</td>
</tr>
<tr>
<td>linkEnds</td>
<td>pyslet.odata2.csdl.EntitySet attribute</td>
</tr>
<tr>
<td>ListElements</td>
<td>module pyslet.qtiv2.content</td>
</tr>
<tr>
<td>load_from_file()</td>
<td>pyslet.imsbltiv1p0.BLTIToolProvider method</td>
</tr>
<tr>
<td>load_metadata()</td>
<td>module pyslet.imsbltiv1p0</td>
</tr>
<tr>
<td>load_service()</td>
<td>pyslet.odata2.client.Client method</td>
</tr>
<tr>
<td>load_visit()</td>
<td>pyslet.imsbltiv1p0.ToolProviderApp method</td>
</tr>
<tr>
<td>LoadFromFile()</td>
<td>pyslet.imsbltiv1p0.BLTIToolProvider method</td>
</tr>
<tr>
<td>LoadService()</td>
<td>pyslet.odata2.client.Client method</td>
</tr>
<tr>
<td>location</td>
<td>pyslet.xmlstructures.XMLEntity attribute</td>
</tr>
<tr>
<td>lock</td>
<td>pyslet.http.messages.Message attribute</td>
</tr>
<tr>
<td>Logo</td>
<td>class in pyslet.rfc4287</td>
</tr>
<tr>
<td>Logo</td>
<td>pyslet.rfc4287.Source attribute</td>
</tr>
<tr>
<td>long2()</td>
<td>module pyslet.py2</td>
</tr>
<tr>
<td>lookup()</td>
<td>pyslet.qtiv2.variables.InterpolationTable method</td>
</tr>
<tr>
<td>lookup()</td>
<td>pyslet.qtiv2.variables.MatchTable method</td>
</tr>
<tr>
<td>lookup_consumer()</td>
<td>pyslet.imsbltiv1p0.ToolProvider method</td>
</tr>
<tr>
<td>lookup_predefined_entity()</td>
<td>pyslet.html401.HTMLParser method</td>
</tr>
<tr>
<td>lookup_predefined_entity()</td>
<td>pyslet.xml.parser.XMLElement method</td>
</tr>
<tr>
<td>LookupTable</td>
<td>class in pyslet.qtiv2.variables</td>
</tr>
<tr>
<td>LT</td>
<td>class in pyslet.qtiv2.expressions</td>
</tr>
<tr>
<td>LTE</td>
<td>class in pyslet.qtiv2.expressions</td>
</tr>
<tr>
<td>LTI_MESSAGE_TYPE</td>
<td>module pyslet.imsbltiv1p0</td>
</tr>
<tr>
<td>LTI_VERSION</td>
<td>module pyslet.imsbltiv1p0</td>
</tr>
<tr>
<td>LTIAuthenticationError</td>
<td>class in pyslet.imsbltiv1p0</td>
</tr>
<tr>
<td>LTIError</td>
<td>class in pyslet.imsbltiv1p0</td>
</tr>
<tr>
<td>LTIProtocolError</td>
<td>class in pyslet.imsbltiv1p0</td>
</tr>
<tr>
<td>main()</td>
<td>pysgi.WSGIApp class method</td>
</tr>
<tr>
<td>major</td>
<td>pyslet.http.params.HTTPVersion attribute</td>
</tr>
</tbody>
</table>
make_attr_name() (in module pyslet.pep8), 18
make_enum() (in module pyslet.xml.xsd.datatypes), 365
make_enum_aliases() (in module pyslet.xml.xsd.datatypes), 365
make_file_path() (pyslet.imscpv1p2.ContentPackage method), 23
make_lower_aliases() (in module pyslet.xml.xsd.datatypes), 365
make_prefix() (pyslet.xml.namespace.NSNode method), 333
make_valid_name() (in module pyslet.qtiv1.core), 35
makedirs() (pyslet.vfs.VirtualFilePath method), 458
MakeEnumeration() (in module pyslet.xml.xsd.datatypes), 365
MakeEnumerationAliases() (in module pyslet.xml.xsd.datatypes), 365
MakeLowerAliases() (in module pyslet.xml.xsd.datatypes), 365
MakePrefix() (pyslet.xml.namespace.NSNode method), 333
manager (pyslet.http.client.ClientRequest attribute), 236
mangle_aname() (pyslet.xml.namespace.NSElement class method), 334
mangle_aname() (pyslet.xml.structures.Element class method), 310
mangle_name() (pyslet.odata2.sqlds.SQLEntityContainer method), 207
MangleAttributeName() (pyslet.xml.structures.Element class method), 319
mangled_names (pyslet.odata2.sqlds.SQLEntityContainer attribute), 206
Manifest (class in pyslet.imscpv1p2), 27
manifest (pyslet.imscpv1p2.ContentPackage attribute), 23
Manifest (pyslet.imscpv1p2.Manifest attribute), 28
ManifestClass (pyslet.imscpv1p2.Manifest attribute), 27
ManifestDocument (class in pyslet.imscpv1p2), 26
ManifestDocumentClass (pyslet.imscpv1p2.ContentPackage attribute), 23
Map (class in pyslet.html401), 399
map_class_elements() (in module pyslet.xml.namespace), 335
map_class_elements() (in module pyslet.xml.structures), 320
map_value() (pyslet.qtiv2.variables.AreaMapping method), 72
map_value() (pyslet.qtiv2.variables.Mapping method), 71
MapEntry (class in pyslet.qtiv2.variables), 71
mapKey (pyslet.qtiv2.variables.MapEntry attribute), 71
mappedValue (pyslet.qtiv2.variables.AreaMapEntry attribute), 73
mappedValue (pyslet.qtiv2.variables.MapEntry attribute), 71
Mapping (class in pyslet.qtiv2.variables), 70
MapResponse (class in pyslet.qtiv2.expressions), 91
MapResponsePoint (class in pyslet.qtiv2.expressions), 91
MapValue() (pyslet.qtiv2.variables.AreaMapping method), 73
MapValue() (pyslet.qtiv2.variables.Mapping method), 71
MAROON (in module pyslet.html401), 371
MatApplet (class in pyslet.qtiv1.common), 40
MatApplication (class in pyslet.qtiv1.common), 40
MatAudio (class in pyslet.qtiv1.common), 39
MatBreak (class in pyslet.qtiv1.common), 39
Match (class in pyslet.qtiv2.expressions), 96
match() (pyslet.rfc2396.URI method), 413
match() (pyslet.unicode5.BasicParser method), 448
match() (pyslet.xml.xsd.datatypes.RegularExpression method), 362
match_challenge() (pyslet.http.auth.Credentials method), 240
match_digit() (pyslet.unicode5.BasicParser method), 449
match_end() (pyslet.http.grammar.WordParser method), 265
match_end() (pyslet.unicode5.BasicParser method), 448
match_expanded_names() (in module pyslet.xml.namespace), 336
match_hex_digit() (pyslet.unicode5.BasicParser method), 450
match_insensitive() (pyslet.unicode5.BasicParser method), 449
match_media_type() (pyslet.http.messages.MediaRange method), 251
match_one() (pyslet.unicode5.BasicParser method), 449
match_xml_name() (pyslet.xml.namespace.XMLNSParser method), 337
match_xml_name() (pyslet.xml.parser.XMLParser method), 347
MatchInteraction (class in pyslet.qtiv2.interactions), 63
MatchTable (class in pyslet.qtiv2.variables), 74
MatchTableEntry (class in pyslet.qtiv2.variables), 74
MatEmText (class in pyslet.qtiv1.common), 39
Material (class in pyslet.qtiv1.common), 37
MaterialRef (class in pyslet.qtiv1.common), 42
MatExtension (class in pyslet.qtiv1.common), 41
MatImage (class in pyslet.qtiv1.common), 39
MatRef (class in pyslet.qtiv1.common), 41
MatText (class in pyslet.qtiv1.common), 38
MatThingMixin (class in pyslet.qtiv1.common), 38
MatVideo (class in pyslet.qtiv1.common), 40
MAX (in module pyslet.odata2.csdl), 189
Max (pyslet.odata2.csdl.DoubleValue attribute), 174
Max (pyslet.odata2.csdl.SingleValue attribute), 174
max_age (pyslet.http.cookie.Cookie attribute), 271
MAX_AGE (pyslet.wsgi.AppCipher attribute), 298
MAX_CHUNK (pyslet.wsgi.WSGIApp attribute), 290
MAX_CONTENT (pyslet.wsgi.WSGIContext attribute), 286

Index
MAX_READAHEAD (pyslet.http.messages.Message attribute), 245
max_retries (pyslet.http.client.ClientRequest attribute), 236
MaxD (pyslet.odata2.csdl.DoubleValue attribute), 174
MaxD (pyslet.odata2.csdl.SingleValue attribute), 174
maxLength (pyslet.odata2.csdl.Property attribute), 186
maxlength_from_str() (in module pyslet.odata2.csdl), 189
maxlength_to_str() (in module pyslet.odata2.csdl), 189
md5 (pyslet.odata2.core.StreamInfo attribute), 197
MDOperator (class in pyslet.qtiv1.core), 32
MediaDesc (class in pyslet.html401), 376
MediaRange (class in pyslet.http.messages), 251
MediaType (class in pyslet.http.params), 259
Member (class in pyslet.qtiv2.expressions), 94
merge() (pyslet.odata2.csdl.Complex method), 175
merge() (pyslet.odata2.csdl.Entity method), 167
merge_fields() (pyslet.odata2.sqlds.SQLCollectionBase method), 217
merge_session() (pyslet.imsbltiv1p0.ToolProviderApp method), 112
merge_session() (pyslet.wsgi.SessionApp method), 296
Message (class in pyslet.http.messages), 244
Meta (class in pyslet.html401), 400
Metadata (class in pyslet.imscpv1p2), 14
Metadata (pyslet.imscpv1p2.Manifest attribute), 27
Metadata (pyslet.imscpv1p2.Resource attribute), 29
metadata (pyslet.wsgi.WSGIDataApp attribute), 294
MetadataClass (pyslet.imscpv1p2.Manifest attribute), 27
MetadataClass (pyslet.imscpv1p2.Resource attribute), 29
MetadataContainerMixin (class in pyslet.qtiv1.common), 42
Method (class in pyslet.html401), 376
Method (pyslet.http.messages.Request attribute), 242
MethodNotAllowed (class in pyslet.wsgi), 301
migrate_area_to_v2() (in module pyslet.qtiv1.core), 31
migrate_cardinality_to_v2() (in module pyslet.qtiv1.core), 34
migrate_content_to_v2() (pyslet.qtiv1.common.ContentMixin method), 37
migrate_content_to_v2() (pyslet.qtiv1.common.FlowMat method), 42
migrate_inequality_to_v2() (pyslet.qtiv1.common.VarInequality method), 45
migrate_objectives_to_lrm() (pyslet.qtiv1.common.Objectives method), 44
migrate_orientation_to_v2() (in module pyslet.qtiv1.core), 33
migrate_to_v2() (pyslet.qtiv1.xml.QTIDocument method), 50
migrate_to_v2() (pyslet.qtiv1.xml.QuesTestInterop method), 50
migrate_vartype_to_v2() (in module pyslet.qtiv1.core), 34
migrate_view_to_v2() (in module pyslet.qtiv1.core), 35
MigratedClass (class in pyslet.pep8), 17
mimetype (pyslet.xml.structures.XMLEntity attribute), 326
minor (pyslet.http.params.HTTPVersion attribute), 258
Minute (pyslet.iso8601.Precision attribute), 444
Minute (pyslet.iso8601.Time attribute), 436
Minute (pyslet.iso8601.Truncation attribute), 443
MIXED (pyslet.xml.structures.ElementType attribute), 322
mkdir() (pyslet.vfs.VirtualFilePath method), 458
mkdtemp() (pyslet.vfs.VirtualFilePath class method), 456
model (pyslet.odata2.client.Client attribute), 203
ModelConstraintError (class in pyslet.odata2.csdl), 194
ModelIncomplete (class in pyslet.odata2.csdl), 194
modified (pyslet.odata2.core.StreamInfo attribute), 197
module_lock (pyslet.odata2.sqlds.SQLEntityContainer attribute), 206
Month (pyslet.http.params.ParameterParser attribute), 262
Month (pyslet.iso8601.Date attribute), 433
Month (pyslet.iso8601.Precision attribute), 444
Month (pyslet.iso8601.Truncation attribute), 443
move() (pyslet.vfs.VirtualFilePath method), 458
mtype (pyslet.odata2.csdl.SimpleValue attribute), 170
MultiLength (class in pyslet.html401), 377
MultiLengths (class in pyslet.html401), 377
Multiple (class in pyslet.qtiv2.expressions), 92
MultipleContainer (class in pyslet.qtiv2.variables), 82
Multiplicity (class in pyslet.odata2.csdl), 189
multiplicity (pyslet.odata2.csdl.AssociationEnd attribute), 188
multiply_to_str() (in module pyslet.odata2.csdl), 189
multipleity_from_str() (in module pyslet.odata2.csdl), 189

N
Name (class in pyslet.rfc4287), 428
name (pyslet.http.cookie.Cookie attribute), 271
name (pyslet.odata2.csdl.Association attribute), 187
name (pyslet.odata2.csdl.AssociationEnd attribute), 188
name (pyslet.odata2.csdl.AssociationSet attribute), 184
name (pyslet.odata2.csdl.AssociationSetEnd attribute), 184
name (pyslet.odata2.csdl.DeferredValue attribute), 175
name (pyslet.odata2.csdl.EntityContainer attribute), 180
name (pyslet.odata2.csdl.EntitySet attribute), 180
name (pyslet.odata2.csdl.NameTableMixin attribute), 190
NoHRef (class in pyslet.html401), 378
nonce_key() (pyslet.imsbltiv1p0.ToolConsumer method), 114
nonces (pyslet.imsbltiv1p0.ToolProvider attribute), 115
NonExistingEntity (class in pyslet.odata2.csdl), 193
nonFatalErrors (pyslet.xml.parser.XMLParser attribute), 338
NOperator (class in pyslet.qtiv2.expressions), 90
NoResize (class in pyslet.html401), 378
normalize_segments() (in module pyslet.rfc2396), 420
normalize_space() (in module pyslet.xml.parser), 355
normcase() (pyslet.vfs.VirtualFilePath method), 457
normpath() (pyslet.vfs.VirtualFilePath method), 457
NoScript (class in pyslet.html401), 400
Not (class in pyslet.qtiv1.common), 48
Not (class in pyslet.qtiv2.expressions), 95
NOTATION (pyslet.xml.structures.XMLAttributeDefinition attribute), 325
notation (pyslet.xml.structures.XMLGeneralEntity attribute), 329
notations (pyslet.xml.structures.XMLElement attribute), 331
NoTruncation (in module pyslet.iso8601), 443
NSDocument (class in pyslet.xml.namespace), 333
NSElement (class in pyslet.xml.namespace), 334
NSNode (class in pyslet.xml.namespace), 332
nss (pyslet.urn.URN attribute), 422
Null (class in pyslet.qtiv2.expressions), 91
nullable (pyslet.odata2.csdl.Property attribute), 186
NumericParams (class in pyslet.odata2.sqlds), 229
NumericValue (class in pyslet.odata2.csdl), 171
NumType (class in pyslet.qtiv1.core), 33
Object (class in pyslet.html401), 401
ObjectBank (class in pyslet.qtiv1.objectbank), 50
ObjectElements (in module pyslet.qtiv2.content), 60
Objectives (class in pyslet.qtiv1.common), 43
ObjectMixin (class in pyslet.qtiv1.core), 36
occurrence (pyslet.xml.structures.XMLContentParticle attribute), 323
OctetParser (class in pyslet.http.grammar), 263
octets (pyslet.rfc2396.URI attribute), 411
offset() (pyslet.iso8601.Date method), 434
offset() (pyslet.iso8601.Time method), 437
OL (class in pyslet.html401), 402
OLIVE (in module pyslet.html401), 371
OneOrMore (pyslet.xml.structures.XMLContentParticle attribute), 323
opaque_part (pyslet.rfc2396.URI attribute), 412
open() (pyslet.odata2.sqlds.SQLiteEntityContainer method), 227
open() (pyslet.vfs.VirtualFilePath method), 458
open() (pyslet.xml.structures.XMLDeclaredEntity method), 329
Open() (pyslet.xml.structures.XMLEntity method), 328
open() (pyslet.xml.structures.XMLEntity method), 326
open_as_pe() (pyslet.xml.structures.XMLParameterEntity method), 329
open_external_entities (pyslet.xml.parser.XMLParser attribute), 338
open_file() (pyslet.xml.structures.XMLEntity method), 327
open_http_response() (pyslet.xml.structures.XMLEntity method), 327
open_navigation() (pyslet.odata2.csdl.EntitySet method), 183
open_remote_entities (pyslet.xml.parser.XMLParser attribute), 338
open_string() (pyslet.xml.structures.XMLEntity method), 327
open_unicode() (pyslet.xml.structures.XMLEntity method), 326
open_uri() (pyslet.xml.structures.XMLEntity method), 327
OpenCollection() (pyslet.odata2.csdl.DeferredValue method), 177
OpenCollection() (pyslet.odata2.csdl.EntitySet method), 184
OpenNavigation() (pyslet.odata2.csdl.EntitySet method), 184
OptGroup (class in pyslet.html401), 402
Option (class in pyslet.html401), 402
OptItemMixin (class in pyslet.html401), 384
Or (class in pyslet.qtiv1.common), 48
Or (class in pyslet.qtiv2.expressions), 95
order_entities() (pyslet.odata2.csdl.EntityCollection method), 163
orderby (pyslet.odata2.csdl.EntityCollection attribute), 162
OrderBy() (pyslet.odata2.csdl.EntityCollection method), 166
order_clause() (pyslet.odata2.sqlds.SQLCollectionBase method), 216
order_cols() (pyslet.odata2.sqlds.SQLCollectionBase method), 216
Ordered (class in pyslet.qtiv2.expressions), 92
OrderedContainer (class in pyslet.qtiv2.variables), 82
Ordering (class in pyslet.qtiv2.tests), 56
OrderInteraction (class in pyslet.qtiv2.interactions), 62
Organization (class in pyslet.imscpv1p2), 28
Organization (pyslet.imscpv1p2.Organizations attribute), 28
OrganizationClass (pyslet.imscpv1p2.Organizations at-
tribute), 28
Organizations (class in pyslet.imscpv1p2), 28
Organizations (pyslet.imscpv1p2.Manifest attribute), 27
OrganizationsClass (pyslet.imscpv1p2.Manifest attribute), 27
Orientation (class in pyslet.qtiv1.core), 33
Orientation (class in pyslet.qtiv2.core), 105
OSFilePath (class in pyslet.vfs), 459
Other (class in pyslet.qtiv1.common), 48
otherEnd (pyslet.odata2.csdl.AssociationEnd attribute), 188
otherEnd (pyslet.odata2.csdl.AssociationSetEnd attribute), 185
otherkey() (pyslet.http.params.SortableParameter method), 257
otherkey() (pyslet.py2.SortableMixin method), 17
OutcomeDeclaration (class in pyslet.qtiv2.variables), 73
output() (in module pyslet.py2), 16

P
P (class in pyslet.html401), 402
p (pyslet.xml.xsdatatypes.RegularExpression attribute), 362
p_def (pyslet.odata2.csdl.DeferredValue attribute), 175
p_def (pyslet.odata2.csdl.EDMValue attribute), 179
p_def (pyslet.odata2.csdl.NavigationCollection attribute), 178
package_path() (pyslet.imscpv1p2.ContentPackage method), 25
package_path() (pyslet.imscpv1p2.File method), 30
PackagePath() (pyslet.imscpv1p2.ContentPackage method), 25
PackagePath() (pyslet.imscpv1p2.File method), 30
PageNotAuthorized (class in pyslet.wsgi), 300
PageNotFoundError (class in pyslet.wsgi), 300
PairValue (class in pyslet.qtiv2.variables), 81
Param (class in pyslet.html401), 403
Parameter (class in pyslet.http.params), 256
parameter_entities (pyslet.xml.xsdatatypes.XMLDTD attribute), 321
ParameterParser (class in pyslet.http.params), 261
parameters (pyslet.http.params.TransferEncoding attribute), 259
parameters (pyslet.imscblitiv1p0.ToolProviderContext attribute), 112
ParamValueType (class in pyslet.html401), 378
pardir (pyslet.vfs.OSFilePath attribute), 459
pardir (pyslet.vfs.VirtualFilePath attribute), 456
parse() (pyslet.unicode5.BasicParser method), 448
parse_att_def() (pyslet.xml.parser.XMLParser method), 350
parse_att_type() (pyslet.xml.parser.XMLParser method), 350
parse_attlist_decl() (pyslet.xml.parser.XMLParser method), 350
parse_attribute() (pyslet.xml.parser.XMLParser method), 348
parse_binary_literal() (pyslet.odata2.csdl.Parser method), 189
parse_boolean_literal() (pyslet.odata2.csdl.Parser method), 189
parse_byte_literal() (pyslet.odata2.csdl.Parser method), 189
parse_cdata() (pyslet.xml.parser.XMLParser method), 345
parse_cdend() (pyslet.xml.parser.XMLParser method), 345
parse_cdsect() (pyslet.xml.parser.XMLParser method), 345
parse_cdstart() (pyslet.xml.parser.XMLParser method), 345
parse_char_data() (pyslet.xml.parser.XMLParser method), 344
parse_char_ref() (pyslet.xml.parser.XMLParser method), 351
parse_charset() (pyslet.http.params.ParameterParser method), 262
parse_children() (pyslet.xml.parser.XMLParser method), 349
parse_choice() (pyslet.xml.parser.XMLParser method), 349
parse_comment() (pyslet.http.grammar.OctetParser method), 263
parse_comment() (pyslet.xml.parser.XMLParser method), 344
parse_conditional_sect() (pyslet.xml.parser.XMLParser method), 351
parse_content() (pyslet.xml.parser.XMLParser method), 348
parse_content_coding() (pyslet.http.params.ParameterParser method), 262
parse_content_spec() (pyslet.xml.parser.XMLParser method), 349
parse_cookie_av() (pyslet.http.cookie.CookieParser method), 276
parse_cookie_date_tokens() (pyslet.http.cookie.CookieParser method), 276
parse_cookie_pair() (pyslet.http.cookie.CookieParser method), 275
parse_cp() (pyslet.xml.parser.XMLParser method), 349
parse_ctext() (pyslet.http.grammar.OctetParser method), 263
parse_datetime_literal() (pyslet.odata2.csdl.Parser method), 190

Index

491
parse_decimal_digits() (pyslet.xml.parser.XMLParser method), 342
parse Decl_sep() (pyslet.xml.parser.XMLParser method), 346
parse Default_decl() (pyslet.xml.parser.XMLParser method), 351
parse_delta_seconds() (pyslet.http.params.ParameterParser method), 262
parse_digit() (pyslet.unicode5.BasicParser method), 449
parse_digit_value() (pyslet.unicode5.BasicParser method), 449
parse_digits() (pyslet.unicode5.BasicParser method), 449
parse_doctypedecl() (pyslet.xml.parser.XMLParser method), 346
parse_document() (pyslet.xml.parser.XMLParser method), 342
parse_element() (pyslet.xml.parser.XMLParser method), 346
parse_element_decl() (pyslet.xml.parser.XMLParser method), 349
parse_empty_elem_tag() (pyslet.xml.parser.XMLParser method), 349
parse_enc_name() (pyslet.xml.parser.XMLParser method), 353
parse_encoding_decl() (pyslet.xml.parser.XMLParser method), 353
parse_entity_decl() (pyslet.xml.parser.XMLParser method), 352
parse_entity_def() (pyslet.xml.parser.XMLParser method), 352
parse_entity_ref() (pyslet.xml.parser.XMLParser method), 352
parse_entity_value() (pyslet.xml.parser.XMLParser method), 351
parse_names() (pyslet.xml.parser.XMLParser method), 344
parse_enumerated_type() (pyslet.xml.parser.XMLParser method), 350
parse_enumeration() (pyslet.xml.parser.XMLParser method), 351
parse_eq() (pyslet.xml.parser.XMLParser method), 345
parse_etag() (pyslet.xml.parser.XMLParser method), 348
parse_ext_subset() (pyslet.xml.parser.XMLParser method), 346
parse_ext_subset_decl() (pyslet.xml.parser.XMLParser method), 346
parse_external_id() (pyslet.xml.parser.XMLParser method), 353
parse_ge_decl() (pyslet.xml.parser.XMLParser method), 352
parse_guid_literal() (pyslet.odata2.csdl.Parser method), 190
parse_hex_digit() (pyslet.unicode5.BasicParser method), 450
parse_hex_digits() (pyslet.unicode5.BasicParser method), 450
parse_hex_integer() (pyslet.http.grammar.WordParser method), 266
parse_ignore() (pyslet.xml.parser.XMLParser method), 351
parse_ignore_sect() (pyslet.xml.parser.XMLParser method), 351
parse_ignore_sect_contents() (pyslet.xml.parser.XMLParser method), 351
parse_include_sect() (pyslet.xml.parser.XMLParser method), 351
parse_insensitive() (pyslet.unicode5.BasicParser method), 449
parse_int_subset() (pyslet.xml.parser.XMLParser method), 346
parse_integer() (pyslet.http.grammar.WordParser method), 266
parse_integer() (pyslet.unicode5.BasicParser method), 450
parse_lws() (pyslet.http.grammar.OctetParser method), 263
parse_markup_decl() (pyslet.xml.parser.XMLParser method), 346
parse_mixed() (pyslet.xml.parser.XMLParser method), 350
parse_literal() (pyslet.xml.parser.XMLParser method), 342
parse_likes() (pyslet.http.grammar.OctetParser method), 263
parse_name() (pyslet.xml.parser.XMLParser method), 343
parse_names() (pyslet.xml.parser.XMLParser method), 344
parse_ndata_decl() (pyslet.xml.parser.XMLParser method), 350
parse_nmtoken() (pyslet.xml.parser.XMLParser method), 344
parse_nmtoken() (pyslet.xml.parser.XMLParser method), 344
parse_nmtokens() (pyslet.xml.parser.XMLParser method), 344
parse_notation_decl() (pyslet.xml.parser.XMLParser method), 353
parse_notation_type() (pyslet.xml.parser.XMLParser method), 350
parse_nsattrs() (pyslet.xml.namespace.XMLNSParser method), 337
parse_numeric_literal() (pyslet.odata2.csdl.Parser method), 190
parse_one() (pyslet.unicode5.BasicParser method), 449
parse_ontext() (pyslet.http.grammar.OctetParser method), 263
parse_parameters() (pyslet.http.grammar.WordParser method), 267
parse_pe_decl() (pyslet.xml.parser.XMLParser method),

492 Index
parse_pe_def() (pyslet.xml.parser.XMLParser method), 352
parse_pe_reference() (pyslet.xml.parser.XMLParser method), 352
parse_pi() (pyslet.xml.parser.XMLParser method), 345
parse_pi_target() (pyslet.xml.parser.XMLParser method), 345
parse_prolog() (pyslet.html401.HTMLParser method), 366
parse_pubid_literal() (pyslet.xml.parser.XMLParser method), 344
parse_public_id() (pyslet.xml.parser.XMLParser method), 354
parse_qdtext() (pyslet.http.grammar.OctetParser method), 264
parse_qs() (in module pyslet.py2), 17
parse_quote() (pyslet.xml.parser.XMLParser method), 342
parse_quoted_pair() (pyslet.http.grammar.OctetParser method), 264
parse_quoted_string() (pyslet.http.grammar.OctetParser method), 264
parse_quoted_string() (pyslet.http.grammar.WordParser method), 265
parse_qvalue() (pyslet.http.params.ParameterParser method), 262
parse_reference() (pyslet.xml.parser.XMLParser method), 351
parse_remainder() (pyslet.xml.parser.XMLParser method), 351
parse_required_decimal_digits() (pyslet.xml.parser.XMLParser method), 342
parse_required_hex_digits() (pyslet.xml.parser.XMLParser method), 342
parse_required_literal() (pyslet.xml.parser.XMLParser method), 342
parse_required_name() (pyslet.xml.parser.XMLParser method), 343
parse_required_s() (pyslet.xml.parser.XMLParser method), 343
parse_s() (pyslet.xml.parser.XMLParser method), 343
parse_sane_cookie_date() (pyslet.http.cookie.CookieParser method), 276
parse_scheme_specific_part() (pyslet.rfc2396.URI method), 412
parse_sd_decl() (pyslet.xml.parser.XMLParser method), 346
parse_seq() (pyslet.xml.parser.XMLParser method), 349
parse_sp() (pyslet.http.grammar.WordParser method), 266
parse_stag() (pyslet.xml.parser.XMLParser method), 348
parse_string_type() (pyslet.xml.parser.XMLParser method), 350
parse_system_literal() (pyslet.xml.parser.XMLParser method), 344
parse_text() (pyslet.http.grammar.OctetParser method), 263
parse_text_decl() (pyslet.xml.parser.XMLParser method), 353
parse_time_literal() (pyslet.odata2.csdl.Parser method), 190
parse_token() (pyslet.http.grammar.OctetParser method), 263
parse_token() (pyslet.http.grammar.WordParser method), 265
parse_tokenized_type() (pyslet.xml.parser.XMLParser method), 350
parse_tokenlist() (pyslet.http.grammar.WordParser method), 266
parse_tokenlower() (pyslet.http.grammar.WordParser method), 265
parse_until() (pyslet.unicode5.BasicParser method), 449
parse_uric() (in module pyslet.rfc2396), 419
parse_urn() (in module pyslet.rfc2396), 421
parse_version_info() (pyslet.xml.parser.XMLParser method), 345
parse_version_num() (pyslet.xml.parser.XMLParser method), 346
parse_word() (pyslet.http.grammar.WordParser method), 265
parse_word_as_bstr() (pyslet.http.grammar.WordParser method), 265
parse_xml_class() (in module pyslet.xml.parser), 356
Parser (class in pyslet.odata2.csdl), 189
parser_error() (pyslet.http.grammar.WordParser method), 265
parser_error() (pyslet.unicode5.BasicParser method), 448
ParserError (class in pyslet.unicode5), 450
partial_match() (pyslet.http.params.LanguageTag method), 261
particle_map (pyslet.xml.structures.ElementType attribute), 323
particle_map (pyslet.xml.structures.XMLNameParticle attribute), 324
PARTICLE_STATE (pyslet.xml.parser.ContentParticleCursor attribute), 354
path (pyslet.http.cookie.Cookie attribute), 271
path (pyslet.vfs.VirtualFilePath attribute), 456

Index 493
pyslet.qtiv2.expressions (module), 89
pyslet.qtiv2.interactions (module), 60
pyslet.qtiv2.items (module), 52
pyslet.qtiv2.metadata (module), 106
pyslet.qtiv2.processing (module), 83
pyslet.qtiv2.tests (module), 53
pyslet.qtiv2.variables (module), 68
pyslet.qtiv2.xml (module), 107
pyslet.rfc2396 (module), 409
pyslet.rfc4287 (module), 424
pyslet.rfc5023 (module), 429
pyslet.streams (module), 451
pyslet.unicode5 (module), 444
pyslet.urn (module), 421
pyslet.vfs (module), 455
pyslet.wsgi (module), 279
pyslet.xml.parser (module), 337
pyslet.xml.structures (module), 301
pyslet.xml.xsdatatypes (module), 356
PythonType (pyslet.odata2.csdl.SimpleType attribute), 188

Q
Q (class in pyslet.html401), 403
q (pyslet.http.messages.AcceptItem attribute), 252
QMarkParams (class in pyslet.odata2.sqlqds), 229
QTI_SOURCE (in module pyslet.qtiv1.core), 36
QTIComment (class in pyslet.qtiv1.common), 49
QTICommentContainer (class in pyslet.qtiv1.common), 49
QTICommentDocument (class in pyslet.qtiv1.xml), 49
QTIElement (class in pyslet.qtiv2.core), 103
QTIError (class in pyslet.qtiv2.core), 103
QTIMetadata (class in pyslet.qtiv1.common), 42
QTIMetadataField (class in pyslet.qtiv1.common), 43
query (pyslet.rfc2396.URI attribute), 411
query_count (pyslet.odata2.sqlqds.SQLTransaction attribute), 228
QuesTestInterop (class in pyslet.qtiv1.xml), 50
queue_request() (pyslet.http.client.Client method), 234
quote_identifier() (pyslet.odata2.sqlqds.SQLEntityContainer method), 209
quote_string() (in module pyslet.http.grammar), 268

R
raise ValidityErrors (pyslet.xml.parser.XMLParser attribute), 338
Random (class in pyslet.qtiv2.expressions), 93
RandomFloat (class in pyslet.qtiv2.expressions), 92
RandomInteger (class in pyslet.qtiv2.expressions), 92
range (pyslet.http.messages.AcceptItem attribute), 252
range3() (module pyslet.py2), 16
raw (pyslet.unicode5.BasicParser attribute), 447
RawIOBase (class in pyslet.py26), 11
RCardinality (class in pyslet.qtiv1.core), 34
read() (pyslet.streams.Pipe method), 454
Read() (pyslet.xml.structures.Document method), 307
read_from_entity() (pyslet.xml.structures.Document method), 305
read_from_stream() (pyslet.xml.structures.Document method), 305
RECV_PERMISSION (pyslet.imsbltiv1p0.ToolProviderApp attribute), 111
read_sql_value() (pyslet.odata2.sqlqds.SQLEntityContainer method), 210
read_sql_value() (pyslet.odata2.sqlqds.SQLiteEntityContainer method), 227
read_stream() (pyslet.odata2.core.EntityCollection method), 195
read_stream_close() (pyslet.odata2.core.EntityCollection method), 196
read_wait() (pyslet.streams.Pipe method), 453
readable() (pyslet.streams.Pipe method), 452
readall() (pyslet.streams.Pipe method), 453
readblocking() (pyslet.streams.Pipe method), 452
ReadFromEntity() (pyslet.xml.structures.Document method), 307
ReadFromStream() (pyslet.xml.structures.Document method), 307
readinto() (pyslet.streams.Pipe method), 454
readmatch() (pyslet.streams.Pipe method), 454
ReadOnly (class in pyslet.html401), 378
realm (pyslet.http.auth.Credentials attribute), 240
realpath() (pyslet.vfs.VirtualFilePath method), 457
REASON (pyslet.http.messages.Response attribute), 243
rebuild_file_table() (pyslet.imscpv1p2.ContentPackage method), 24
RebuildFileTable() (pyslet.imscpv1p2.ContentPackage method), 25
RecordContainer (class in pyslet.imscpv1p2.ContentPackage method), 25
RECV_ALL (pyslet.http.messages.Message attribute), 246
RECV_HEADERS (pyslet.http.messages.Message attribute), 246
RECV_LINE (pyslet.http.messages.Message attribute), 246
recv_mode() (pyslet.http.messages.Message method), 246
recv_pipe (pyslet.http.client.ClientRequest attribute), 237
recv_start() (pyslet.http.messages.Message method), 247
require_chunk() (pyslet.http.params.ParameterParser method), 362
require_compl_esc() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 364
require_contentrange() (pyslet.http.messages.HeaderParser method), 255
require_cookie_date() (pyslet.http.cookie.CookieParser method), 276
require_cookie_pair() (pyslet.http.cookie.CookieParser method), 275
require_cookie_string() (pyslet.http.cookie.CookieParser method), 275
require_cookie_value() (pyslet.http.cookie.CookieParser method), 275
require_entity_tag() (pyslet.http.params.ParameterParser method), 262
require_fulldate() (pyslet.http.params.ParameterParser method), 262
require_hexinteger() (pyslet.http.grammar.WordParser method), 266
require_http_version() (pyslet.http.params.ParameterParser method), 262
require_integer() (pyslet.http.grammar.WordParser method), 266
require_is_block() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 364
require_is_category() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 364
require_language_tag() (pyslet.http.params.ParameterParser method), 262
require_media_range() (pyslet.http.messages.HeaderParser method), 255
require_media_type() (pyslet.http.params.ParameterParser method), 262
require_multi_char_esc() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 364
require_name_value_pair() (pyslet.http.cookie.CookieParser method), 275
require_neg_char_group() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 363
require_pos_char_group() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 363
require_product_token() (pyslet.http.params.ParameterParser method), 262
require_product_token_list() (pyslet.http.messages.HeaderParser method), 255
require_quant_exact() (pyslet.xml.xsd.datatypesRegularExpressionParser method), 363
require_quantifier() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 362
require_quantity() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 363
require_reg_exp() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 362
require_se_range() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 363
require_separator() (pyslet.http.grammar.WordParser method), 266
require_set_cookie_string() (pyslet.http.cookie.CookieParser method), 275
require_single_char_esc() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 364
require_token() (pyslet.http.grammar.WordParser method), 266
require_transfer_encoding() (pyslet.http.params.ParameterParser method), 262
require_wildcard_esc() (pyslet.xml.xsd.datatypes.RegularExpressionParser method), 364
resolve() (pyslet.rfc2396.URI method), 412
resolve_base() (pyslet.xml.structures.Element method), 315
reset() (pyslet.http.client.ClientRequest attribute), 236
reset_joins() (pyslet.odata2.sql.SqlAssociationCollection method), 223
reset_joins() (pyslet.odata2.sql.SqlCollectionBase method), 215
reset_joins() (pyslet.odata2.sql.SqlForeignKeyCollection method), 221
resolve() (pyslet.rfc2396.URI method), 412
resolve_base() (pyslet.xml.structures.Element method), 315
resolve_external_id() (pyslet.xml.parser.XMLParser attribute), 325
resolve_setup_path() (pyslet.wsgi.WSGIApp class method), 291
resolve_space() (pyslet.xml.structures.Element method), 317
resolve_uri() (pyslet.xml.structures.Element method), 316
resolve_value() (pyslet.html401.Length method), 375
resolve_value() (pyslet.html401.MultiLength method), 377
ResolveBase() (pyslet.xml.structures.Element method), 353

Index 497
Index 499
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Class/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_expansion()</td>
<td>pyslet.odata2.csdl.DeferredValue Method</td>
<td>176</td>
</tr>
<tr>
<td>set_expansion_values()</td>
<td>pyslet.odata2.csdl.DeferredValue Method</td>
<td>176</td>
</tr>
<tr>
<td>set_filter()</td>
<td>pyslet.odata2.csdl.EntityCollection Method</td>
<td>163</td>
</tr>
<tr>
<td>set_from_json_object()</td>
<td>pyslet.odata2.core.Entity Method</td>
<td>196</td>
</tr>
<tr>
<td>set_from_literal()</td>
<td>pyslet.odata2.csdl.SimpleValue Method</td>
<td>171</td>
</tr>
<tr>
<td>set_from_numeric_literal()</td>
<td>pyslet.odata2.csdl.NumericValue Method</td>
<td>171</td>
</tr>
<tr>
<td>set_from_numeric_literal()</td>
<td>pyslet.odata2.csdl.SingleValue Method</td>
<td>171</td>
</tr>
<tr>
<td>set_from_simple_value()</td>
<td>pyslet.odata2.csdl.SimpleValue Method</td>
<td>171</td>
</tr>
<tr>
<td>set_from_value()</td>
<td>pyslet.odata2.csdl.SimpleValue Method</td>
<td>171</td>
</tr>
<tr>
<td>set_header()</td>
<td>pyslet.http.messages.Message Method</td>
<td>248</td>
</tr>
<tr>
<td>set_id()</td>
<td>pyslet.xml.structures.Element Method</td>
<td>310</td>
</tr>
<tr>
<td>set_ignore_files()</td>
<td>pyslet.imscp1p2.ContentPackage Method</td>
<td>23</td>
</tr>
<tr>
<td>set_inlinecount()</td>
<td>pyslet.odata2.csdl.EntityCollection Method</td>
<td>164</td>
</tr>
<tr>
<td>set_key()</td>
<td>pyslet.odata2.csdl.Entity Method</td>
<td>168</td>
</tr>
<tr>
<td>set_lang()</td>
<td>pyslet.xml.structures.Document Method</td>
<td>304</td>
</tr>
<tr>
<td>set_lang()</td>
<td>pyslet.xml.structures.Element Method</td>
<td>316</td>
</tr>
<tr>
<td>set_lang()</td>
<td>pyslet.xml.structures.Node Method</td>
<td>303</td>
</tr>
<tr>
<td>set_last_modified()</td>
<td>pyslet.http.messages.Message Method</td>
<td>250</td>
</tr>
<tr>
<td>set_launch_group()</td>
<td>pyslet.imsbltiv1p0.ToolProviderApp Method</td>
<td>111</td>
</tr>
<tr>
<td>set_launch_permissions()</td>
<td>pyslet.imsbltiv1p0.ToolProviderApp Method</td>
<td>111</td>
</tr>
<tr>
<td>set_launch_resource()</td>
<td>pyslet.imsbltiv1p0.ToolProviderApp Method</td>
<td>111</td>
</tr>
<tr>
<td>set_launch_user()</td>
<td>pyslet.imsbltiv1p0.ToolProviderApp Method</td>
<td>111</td>
</tr>
<tr>
<td>set_location()</td>
<td>pyslet.http.messages.Response Method</td>
<td>244</td>
</tr>
<tr>
<td>set_location()</td>
<td>pyslet.odata2.csdl.EntitySet Method</td>
<td>182</td>
</tr>
<tr>
<td>set_location()</td>
<td>pyslet.odata2.metadata.EntitySet Method</td>
<td>199</td>
</tr>
<tr>
<td>set_method()</td>
<td>pyslet.wsgi.WSGIApp Method</td>
<td>291</td>
</tr>
<tr>
<td>set_null()</td>
<td>pyslet.odata2.csdl.Complex Method</td>
<td>175</td>
</tr>
<tr>
<td>set_null()</td>
<td>pyslet.odata2.csdl.SimpleValue Method</td>
<td>171</td>
</tr>
<tr>
<td>set_orderby()</td>
<td>pyslet.odata2.csdl.EntityCollection Method</td>
<td>163</td>
</tr>
<tr>
<td>set_orderby()</td>
<td>pyslet.odata2.sqlds.SQLCollectionBase Method</td>
<td>216</td>
</tr>
<tr>
<td>set_page()</td>
<td>pyslet.odata2.csdl.EntityCollection Method</td>
<td>165</td>
</tr>
<tr>
<td>set_page()</td>
<td>pyslet.odata2.sqlds.SQLCollectionBase Method</td>
<td>215</td>
</tr>
<tr>
<td>set_protocol()</td>
<td>pyslet.http.messages.Message Method</td>
<td>245</td>
</tr>
<tr>
<td>set_public_list()</td>
<td>pyslet.http.cookie.CookieStore Method</td>
<td>274</td>
</tr>
<tr>
<td>set_pyslet_info()</td>
<td>pyslet.rfc4287.Generator Method</td>
<td>426</td>
</tr>
<tr>
<td>set_random_value()</td>
<td>pyslet.odata2.csdl.SimpleValue Method</td>
<td>171</td>
</tr>
<tr>
<td>set_readblocking()</td>
<td>pyslet.streams.Pipe Method</td>
<td>452</td>
</tr>
<tr>
<td>set_rflag()</td>
<td>pyslet.streams.Pipe Method</td>
<td>452</td>
</tr>
<tr>
<td>set_session()</td>
<td>pyslet.wsgi.SessionApp Method</td>
<td>295</td>
</tr>
<tr>
<td>set_session_cookie()</td>
<td>pyslet.wsgi.SessionApp Method</td>
<td>296</td>
</tr>
<tr>
<td>set_set_cookie()</td>
<td>pyslet.http.messages.Response Method</td>
<td>244</td>
</tr>
<tr>
<td>set_space()</td>
<td>pyslet.xml.structures.Element Method</td>
<td>317</td>
</tr>
<tr>
<td>set_status()</td>
<td>pyslet.wsgi.WSGIContext Method</td>
<td>286</td>
</tr>
<tr>
<td>set_test_cookie()</td>
<td>pyslet.wsgi.SessionApp Method</td>
<td>296</td>
</tr>
<tr>
<td>set_to_zero()</td>
<td>pyslet.odata2.csdl.NumericValue Method</td>
<td>171</td>
</tr>
<tr>
<td>set_topmax()</td>
<td>pyslet.odata2.csdl.EntityCollection Method</td>
<td>165</td>
</tr>
<tr>
<td>set_transfer_encoding()</td>
<td>pyslet.http.messages.Message Method</td>
<td>250</td>
</tr>
<tr>
<td>set_upgrade()</td>
<td>pyslet.http.messages.Message Method</td>
<td>250</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.BooleanValue Method</td>
<td>80</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.DirectedPairValue Method</td>
<td>80</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.FileValue Method</td>
<td>80</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.FloatValue Method</td>
<td>81</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.IdentifierValue Method</td>
<td>81</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.IntegerValue Method</td>
<td>81</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.MultilineContainer Method</td>
<td>81</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.OrderedContainer Method</td>
<td>82</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.PairValue Method</td>
<td>82</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.RecordContainer Method</td>
<td>82</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.URIValue Method</td>
<td>81</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.qtiv2.variables.Value Method</td>
<td>82</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.rfc4287.Date Method</td>
<td>429</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.rfc4287.Icon Method</td>
<td>426</td>
</tr>
<tr>
<td>set_value()</td>
<td>pyslet.rfc4287.Text Method</td>
<td>428</td>
</tr>
</tbody>
</table>
set_value() (pyslet.xml.structures.Element method), 314
set_writeblocking() (pyslet.streams.Pipe method), 452
set_www_authenticate() (pyslet.http.messages.Response method), 244
set_xmlname() (pyslet.xml.namespace.NSElement method), 334
set_xmlname() (pyslet.xml.structures.Element method), 310
SetAttribute() (pyslet.xml.structures.Element method), 319
SetBase() (pyslet.xml.structures.Node method), 319
SetConcurrencyTokens() (pyslet.odata2.csdl.Entity method), 170
SetCorrectResponse (class in pyslet.qtiv2.csdl.Entity), 87
setdefault() (pyslet.odata2.csdl.DictionaryLike method), 193
SetDefaultValue (class in pyslet.qtiv2.processing), 88
SetEntryPoint() (pyslet.imscpv1p2.Resource method), 29
SetExpansion() (pyslet.odata2.csdl.DeferredValue method), 177
SetFromLiteral() (pyslet.odata2.csdl.DeferredValue method), 177
SetFromNum2eral() (pyslet.odata2.csdl.NumericValue method), 172
SetFromSimpleValue() (pyslet.odata2.csdl.NumericValue method), 172
SetID() (pyslet.xml.structures.Element method), 319
SetIgnoreFiles() (pyslet.imscpv1p2.ContentPackage method), 25
SetInlineCount() (pyslet.odata2.csdl.EntityCollection method), 166
SetLang() (pyslet.xml.structures.Node method), 303
SetLocation() (pyslet.odata2.csdl.EntitySet method), 184
SetOutcomeValue (class in pyslet.qtiv2.processing), 85
setpos() (pyslet.http.grammar.WordParser method), 265
setpos() (pyslet.unicode5.BasicParser method), 447
SetSpace() (pyslet.xml.structures.Element method), 319
SetTemplateDefaults() (pyslet.qtiv2.tests.AssessmentItemRef method), 58
SetTemplateValue (class in pyslet.qtiv2.processing), 85
settings (pyslet.wsgi.WSGIApp attribute), 289
sort_declarations() (pyslet.odata2.csdl.EntityCollection method), 166
Source (class in pyslet.rfc4287), 289
sortkey() (pyslet.py2.SortableMixin method), 16
sortkey() (pyslet.vfs.VirtualFilePath method), 447
Source (class in pyslet.xml.structures.Element static method), 319
SortableMixin (class in pyslet.py2), 16
SortableParameter (class in pyslet.http.params), 289
split() (pyslet.vfs.VirtualFilePath method), 456
split_abs_path() (in module pyslet.rfc2396), 456
split_day_of_month() (in module pyslet.http.cookie), 420
split_domain() (in module pyslet.http.cookie), 420
split_month() (in module pyslet.http.cookie), 420
sgml_namecase_general (pyslet.xml.parser.XMLParser attribute), 339
sgml_omittag (pyslet.xml.parser.XMLParser attribute), 339
sgml_shorttag (pyslet.xml.parser.XMLParser attribute), 339
SGMLCDATA (pyslet.xml.structures.SubElement attribute), 322
Shape (class in pyslet.html401), 379
Shape (class in pyslet.qtiv2.core), 105
shift_zone() (pyslet.iso8601.Time method), 438
shift_zone() (pyslet.iso8601.TimePoint method), 441
ShowHide (class in pyslet.qtiv2.core), 105
sign() (pyslet.xml.xsdatatypes.Duration attribute), 358
sign() (pyslet.wsgi.AppCipher method), 299
SILVER (in module pyslet.html401), 371
simple_cast() (pyslet.odata2.csdl.SimpleValue method), 170
SimpleAssociableChoice (class in pyslet.qtiv2.interactions), 63
SimpleChoice (class in pyslet.qtiv2.interactions), 62
SimpleMatchSet (class in pyslet.qtiv2.interactions), 63
SimpleType (class in pyslet.odata2.csdl), 188
simpleTypeCode (pyslet.odata2.csdl.Property attribute), 186
SimpleValue (class in pyslet.odata2.csdl), 170
SingleValue (class in pyslet.odata2.csdl), 174
SingleValue (class in pyslet.qtiv2.variables), 80
size (pyslet.http.params.Chunk attribute), 259
size (pyslet.odata2.core.StreamInfo attribute), 197
skip (pyslet.odata2.csdl.EntityCollection attribute), 162
Small (class in pyslet.html401), 404
sort_declarations() (pyslet.odata2.csdl.AssessmentItem method), 53
sort_declarations() (pyslet.qtiv2.tests.AssessmentTest method), 54
sort_names() (pyslet.xml.structures.Element static method), 315
SpecialMixin (class in pyslet.html401), 424
source_path_generator() (pyslet.odata2.sqls.SQLEntityContainer method), 207
Span (class in pyslet.html401), 404
SpecialMixin (class in pyslet.html401), 384
split() (pyslet.vfs.VirtualFilePath method), 457
split_abs_path() (in module pyslet.rfc2396), 420
split_day_of_month() (in module pyslet.http.cookie), 276
split_domain() (in module pyslet.http.cookie), 277
split_month() (in module pyslet.http.cookie), 278
sgml_content (pyslet.xml.parser.XMLParser attribute), 339
sgml_namecase_general (pyslet.xml.parser.XMLParser attribute), 339
sgml_omittag (pyslet.xml.parser.XMLParser attribute), 339
sgml_shorttag (pyslet.xml.parser.XMLParser attribute), 339
{}
split_path() (in module pyslet.rfc2396), 420
split_rel_path() (in module pyslet.rfc2396), 420
split_role() (in module pyslet.imsbltiv1p0), 116
split_server() (in module pyslet.rfc2396), 420
split_time() (in module pyslet.http.cookie), 276
split_year() (in module pyslet.http.cookie), 276
split_year() (pyslet.iso8601.Date static method), 435
splitdrive() (pyslet.vfs.VirtualFilePath method), 457
splitext() (pyslet.vfs.VirtualFilePath method), 457
splitunc() (pyslet.vfs.VirtualFilePath method), 457
sql_bracket() (pyslet.odata2.sqlds.SQLCollectionBase method), 218
sql_expression() (pyslet.odata2.sqlds.SQLCollectionBase method), 218
sql_expression_add() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_and() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_cast() (pyslet.odata2.sqlds.SQLCollectionBase method), 218
sql_expression_ceiling() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_concat() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_day() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_day() (pyslet.odata2.sqlds.SQLiteEntityCollectionBase method), 225
sql_expression_div() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_endswith() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_eq() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_floor() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_ge() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_generic_binary() (pyslet.odata2.sqlds.SQLCollectionBase method), 218
sql_expression_gt() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_hour() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_hour() (pyslet.odata2.sqlds.SQLiteEntityCollectionBase method), 225
sql_expression_indexof() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_isof() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_le() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_length() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_length() (pyslet.odata2.sqlds.SQLiteEntityCollectionBase method), 225
sql_expression_lt() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_member() (pyslet.odata2.sqlds.SQLCollectionBase method), 218
sql_expression_minute() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_minute() (pyslet.odata2.sqlds.SQLiteEntityCollectionBase method), 225
sql_expression_mod() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_month() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_month() (pyslet.odata2.sqlds.SQLiteEntityCollectionBase method), 225
sql_expression_mul() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_ne() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_or() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_replace() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_round() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_second() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_second() (pyslet.odata2.sqlds.SQLiteEntityCollectionBase method), 225
sql_expression_sub() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_substring() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_substringof() (pyslet.odata2.sqlds.SQLCollectionBase method), 220
sql_expression_tolower() (pyslet.odata2.sqlds.SQLCollectionBase method), 219
sql_expression_tolower() (pyslet.odata2.sqlds.SQLiteEntityCollectionBase method), 219

SYSROLE_HANDLES (in module pyslet.imsbltiv1p0), 116

T

T (pyslet.qtiv2.variables.TestSessionState attribute), 77
Table (class in pyslet.html401), 405
TableCellMixin (class in pyslet.html401), 383
TableColMixin (class in pyslet.html401), 385
tag (pyslet.http.params.EntityTag attribute), 261
Target() (pyslet.odata2.csdl.DeferredValue method), 177
target() (pyslet.odata2.csdl.DeferredValue method), 176
TBody (class in pyslet.html401), 406
TD (class in pyslet.html401), 406
TEAL (in module pyslet.html401), 372
TemplateCondition (class in pyslet.qtiv2.processing), 86
TemplateDeclaration (class in pyslet.qtiv2.variables), 75
TemplateDefault (class in pyslet.qtiv2.processing), 89
TemplateElse (class in pyslet.qtiv2.processing), 87
TemplateElseIf (class in pyslet.qtiv2.processing), 87
TemplateIf (class in pyslet.qtiv2.processing), 86
TemplateProcessing (class in pyslet.qtiv2.processing), 85
term (pyslet.rfc4287.Category attribute), 425
test (pyslet.qtiv2.variables.TestSessionState attribute), 77
test() (pyslet.unicode5.CharClass method), 446
test_circle() (pyslet.html401.Coords method), 373
test_path() (pyslet.http.auth.BasicCredentials method), 373
test_public_domain() (pyslet.http.cookie.CookieStore method), 274
test_rect() (pyslet.html401.Coords method), 373
test_url() (pyslet.http.auth.BasicCredentials method), 241
test_url() (pyslet.httpcreds.BasicCredentials method), 240
TestOperator (in module pyslet.qtiv1.core), 34
TestPart (class in pyslet.qtiv2.tests), 55
TestPartCondition (class in pyslet.qtiv2.processing), 88
TestSessionState (class in pyslet.qtiv2.variables), 77
Text (class in pyslet.rfc4287), 428
text_response() (pyslet.wsgi.WSGIApp method), 292
TextArea (class in pyslet.html401), 406
TextElements (in module pyslet.html401), 60
TextEditInteraction (class in pyslet.qtiv2.interactions), 66
TextFormat (class in pyslet.qtiv2.interactions), 66
TextType (class in pyslet.rfc4287), 428
TFrame (class in pyslet.html401), 407
TFrame (class in pyslet.html401), 379
TH (class in pyslet.html401), 407
the_char (pyslet.unicode5.BasicParser attribute), 447
the_char (pyslet.xml.parser.XMLParser attribute), 340
the_char (pyslet.xml.structures.XMLEntity attribute), 326
the_word (pyslet.http.grammar.WordParser attribute), 264
THead (class in pyslet.html401), 407
thread_active_count() (pyslet.http.client.Client method), 234
thread_loop() (pyslet.http.client.Client method), 235
thread_task() (pyslet.http.client.Client method), 234
Time (class in pyslet.iso8601), 435
TimePoint (class in pyslet.iso8601), 439
TimeValue (class in pyslet.odata2.csdl), 175
Title (class in pyslet.html401), 407
Title (class in pyslet.rfc4287), 427
Title (pyslet.rfc4287.Entity attribute), 425
title (pyslet.rfc4287.Link attribute), 427
Title (pyslet.rfc5023.Collection attribute), 430
Title (pyslet.rfc5023.Workspace attribute), 429
TitleClass (pyslet.rfc4287.Entry attribute), 424
TitleClass (pyslet.rfc4287.Feed attribute), 424
to_bytes() (in module pyslet.py2), 15
to_bytes() (pyslet.http.params.FullDate method), 258
to_bytes() (pyslet.http.params.Parameter method), 257
to_bytes() (pyslet.vfs.VirtualFilePath method), 456
to_end (pyslet.odata2.csdl.NavigationProperty attribute), 187
to_local_text() (pyslet.rfc2396.FileURL method), 415
to_str() (pyslet.html401.NamedBoolean class method), 369
to_str() (pyslet.xml.xsdatatypes Enumeration class method), 360
to_text() (in module pyslet.py2), 14
token (pyslet.http.messages.AcceptToken attribute), 253
token (pyslet.http.params.ProductToken attribute), 260
token (pyslet.http.params.TransferEncoding attribute), 259
ToleranceMode (class in pyslet.qtiv2.expressions), 97
ToolConsumer (class in pyslet.imsbltiv1p0), 112
ToolProvider (class in pyslet.imsbltiv1p0), 114
ToolProviderApp (class in pyslet.imsbltiv1p0), 110
ToolProviderContext (class in pyslet.imsbltiv1p0), 112
top (pyslet.odata2.csdl.EntityCollection attribute), 162
topMax (pyslet.odata2.csdl.EntityCollection attribute), 162
TopMax() (pyslet.odata2.csdl.EntityCollection method), 166
toRole (pyslet.odata2.csdl.NavigationProperty attribute), 187
touch() (pyslet.http.cookie.Cookie method), 272
TR (class in pyslet.html401), 408
TransferEncoding (class in pyslet.http.params), 259
translate_from_urnchar() (in module pyslet.urn), 429
translate_to_urnchar() (in module pyslet.urn), 422
TRContainer (class in pyslet.html401), 387
TRules (class in pyslet.html401), 380
Truncate (class in pyslet.qtiv2.expressions), 102
write_xml() (pyslet.xml.structures.Element method), 318
write_xml_attributes() (pyslet.xml.structures.Element method), 317
writeblocking() (pyslet.streams.Pipe method), 452
WriteXML() (pyslet.xml.structures.Document method), 307
WriteXML() (pyslet.xml.structures.Element method), 320
WriteXMLAttributes() (pyslet.xml.structures.Element method), 320
WSGIApp (class in pyslet.wsgi), 288
WSGIContext (class in pyslet.wsgi), 286
WSGiDataApp (class in pyslet.wsgi), 293

X
xdigits (pyslet.iso8601.Date attribute), 433
XHTML_NAMESPACE (in module pyslet.html401), 366
XHTMLDocument (class in pyslet.html401), 366
XHTEElement (class in pyslet.html401), 368
XHTEError (class in pyslet.html401), 408, 409
XHTLMixin (class in pyslet.html401), 367
XHTMLValidityError (class in pyslet.html401), 408
XML_NAMESPACE (in module pyslet.xml.namespace), 335
XMLAttributeDefinition (class in pyslet.xml.structures), 324
XMLAttributeSetter (class in pyslet.xml.structures), 332
XMLChoiceList (class in pyslet.xml.structures), 324
XMLCONTENT (pyslet.xml.structures.Element attribute), 309
XMLContentTypeParticle (class in pyslet.xml.structures), 323
XMLDeclaration (class in pyslet.xml.structures), 322
XMLDeclaredEntity (class in pyslet.xml.structures), 328
XMLDTD (class in pyslet.xml.structures), 320
XMLEntity (class in pyslet.xml.structures), 325
XMLError (class in pyslet.xml.structures), 331
XMLExternalID (class in pyslet.xml.parser), 329
XMLFatalError (class in pyslet.xml.parser), 355
XMLForbiddenEntityReference (class in pyslet.xml.parser), 355
XMLGeneralEntity (class in pyslet.xml.structures), 329
XMLIDClashError (class in pyslet.xml.structures), 331
XMLIDValueError (class in pyslet.xml.structures), 331
XMLMissingLocationError (class in pyslet.xml.structures), 320
XMLMissingResourceError (class in pyslet.xml.structures), 320
XMLMixedContentError (class in pyslet.xml.structures), 332
XMLNameParticle (class in pyslet.xml.structures), 324
XMLNotation (class in pyslet.xml.structures), 330
XMLNS_NAMESPACE (in module pyslet.xml.namespace), 335
XMLNSElement (class in pyslet.xml.namespace), 335
XMLNSError (class in pyslet.xml.namespace), 337
XMLNSParser (class in pyslet.xml.namespace), 336
XMLParameterEntity (class in pyslet.xml.structures), 329
XMLParentError (class in pyslet.xml.structures), 332
XMLParser (class in pyslet.xml.parser), 337
XMLParser() (pyslet.html401.XHTMLDocument method), 366
XMLParser() (pyslet.qtiv1.xml.QTIDocument method), 49
XMLParser() (pyslet.xml.namespace.NSDocument method), 333
XMLParser() (pyslet.xml.structures.Document method), 304
XMLSCHEMA_NAMESPACE (in module pyslet.xml.xsdatatypes), 356
XMLSequenceList (class in pyslet.xml.structures), 324
XMLTextDeclaration (class in pyslet.xml.structures), 329
XMLUnexpectedHTTPResponse (class in pyslet.xml.structures), 320
XMLUnknownChild (class in pyslet.xml.structures), 332
XMLUnsupportedSchemeError (class in pyslet.xml.structures), 320
XMLValidityError (class in pyslet.xml.structures), 331
XMLWellFormedError (class in pyslet.xml.parser), 355

Y
year (pyslet.iso8601.Date attribute), 433
Year (pyslet.iso8601.Precision attribute), 444
Year (pyslet.iso8601.Truncation attribute), 443
YELLOW (in module pyslet.html401), 371
yn_from_str() (in module pyslet.qtiv1.core), 35
yn_to_str() (in module pyslet.qtiv1.core), 35

Z
ZeroOrOne (pyslet.xml.structures.XMLContentParticle attribute), 323
ZipHooks (class in pyslet.vfs), 460
zoffset (pyslet.iso8601.Time attribute), 437