XIST Documentation

Release 5.75.1+2 (114b1a42)

Walter Dörwald

Apr 22, 2024

CONTENTS

CHAPTER

ONE

OVERVIEW

XIST provides an extensible HTML and XML generator. XIST is also a XML parser with a very simple and pythonesque tree API. Every XML element type corresponds to a Python class and these Python classes provide a conversion method to transform the XML tree (e.g. into HTML). XIST can be considered "object oriented XSLT".

XIST also includes the following modules and packages:

- *ll.ul4c* is compiler for a cross-platform templating language with similar capabilities to Django's templating language. UL4 templates are compiled to an internal format, which makes it possible to implement template renderers in other languages and makes the template code "secure" (i.e. template code can't open or delete files).

  There are implementations for Python, Java and Javascript.

- *ll.ul4on* provides functions for encoding and decoding a lightweight machine-readable text-based format for serializing the object types supported by UL4. It is extensible to allow encoding/decoding arbitrary instances (i.e. it is basically a reimplementation of `pickle`, but with string input/output instead of bytes and with an eye towards cross-plattform support).

  There are implementations for Python, Java and Javascript.

- *ll.orasql* provides utilities for working with cx_Oracle:
  - It allows calling functions and procedures with keyword arguments.
  - Query results will be put into *Record* objects, where database fields are accessible as object attributes.
  - The *Connection* class provides methods for iterating through the database metadata.
  - Importing the modules adds support for URLs with the scheme `oracle` to *ll.url*.

- *ll.make* is an object oriented make replacement. Like make it allows you to specify dependencies between files and actions to be executed when files don't exist or are out of date with respect to one of their sources. But unlike make you can do this in a object oriented way and targets are not only limited to files.

- *ll.color* provides classes and functions for handling RGB color values. This includes the ability to convert between different color models (RGB, HSV, HLS) as well as to and from CSS format, and several functions for modifying and mixing colors.

- *ll.sisyphus* provides classes for running Python scripts as cron jobs.

- *ll.url* provides classes for parsing and constructing RFC 2396 compliant URLs.

- *ll.nightshade* can be used to serve the output of PL/SQL functions/procedures with CherryPy.

- *ll.misc* provides several small utility functions and classes.

- `ll.astyle` can be used for colored terminal output (via ANSI escape sequences).

- *ll.daemon* can be used on UNIX to fork a daemon process.

- `ll.xml_codec` contains a complete codec for encoding and decoding XML.

CHAPTER

TWO

DOCUMENTATION

For documentation read the files in the `docs/` directory or the web pages generated from those.

For installation instructions read `docs/INSTALL.rst`.

For a history of XIST and a list of new features in this version, read `docs/NEWS.rst`.

For a list of old features and bugfixes read `docs/OLDNEWS.rst`.

For the license read `xist/__init__.py`.

CHAPTER

THREE

DOWNLOAD

XIST is available for download from the download directory or from the Cheeseshop.

CHAPTER

FOUR

SOURCE

Sourcecode is available on GitHub.

CHAPTER

FIVE

AUTHOR

Walter Dörwald <walter@livinglogic.de>

CHAPTER

SIX

CONTENT

6.1 XIST – An extensible HTML/XML generator

XIST is an extensible HTML and XML generator. XIST is also a XML parser with a very simple and pythonesque tree API. Every XML element type corresponds to a Python class and these Python classes provide a conversion method to transform the XML tree (e.g. into HTML). XIST can be considered 'object oriented XSLT'.

XIST was written as a replacement for the HTML preprocessor HSC, and borrows some features and ideas from it.

It also borrows the basic idea (XML/HTML elements as Python objects) from the ancient Python modules `HTMLgen` and `HyperText`.

6.1.1 XIST examples

Creating HTML

You can create and output HTML like this:

```
from ll.xist import xsc
from ll.xist.ns import html, xml, meta

node = xsc.Frag(
   xml.XML(),
   html.DocTypeXHTML10transitional(),
   html.html(
      html.head(
         meta.contenttype(),
         html.title("Example page")
      ),
      html.body(
         html.h1("Welcome to the example page"),
         html.p(
            "This example page has a link to the ",
            html.a("Python home page", href="http://www.python.org/"),
            "."
         )
      )
   )
)
```

You can also use `with` blocks (and the unary + operator) to generate the same HTML:

```
from ll.xist import xsc
from ll.xist.ns import html, xml, meta
```

(continues on next page)

(continued from previous page)

```
with xsc.build():
   with xsc.Frag() as node:
      +xml.XML()
      +html.DocTypeXHTML10transitional()
      with html.html():
         with html.head():
            +meta.contenttype()
            +html.title("Example page")
         with html.body():
            +html.h1("Welcome to the example page")
            with html.p():
               +xsc.Text("This example page has a link to the ")
               with html.a():
                  with xsc.addattr("href"):
                     +xsc.Text("http://www.python.org/")
                  +xsc.Text("Python home page")
               +xsc.Text(".")
```

Printing HTML

When you have an XIST tree you can print it with the `string()` method like this:

```
from ll.xist import xsc
from ll.xist.ns import html, xml, meta

node = xsc.Frag(
   xml.XML(),
   html.DocTypeXHTML10transitional(),
   html.html(
      html.head(
         meta.contenttype(),
         html.title("Example page")
      ),
      html.body(
         html.h1("Welcome to the example page"),
         html.p(
            "This example page has a link to the ",
            html.a("Python home page", href="http://www.python.org/"),
            "."
         )
      )
   )
)

print(node.string(encoding="us-ascii"))
```

When you want to save this into a file, use the `bytes()` method instead of `string()`:

```
with open("example.xml", "wb") as f:
   f.write(node.bytes(encoding="us-ascii"))
```

Defining new elements

You can define new elements and how they should be converted to HTML (or other XML vocabularies) like this:

```
from ll.xist import xsc
from ll.xist.ns import html, xml, meta

class cheeseshoplink(xsc.Element):
   class Attrs(xsc.Element.Attrs):
      class name(xsc.TextAttr): pass

   def convert(self, converter):
      e = html.a(
         self.attrs.name,
         href=("http://cheeseshop.python.org/pypi/", self.attrs.name)
      )
      return e.convert(converter)

names = ["ll-xist", "cx_Oracle", "PIL"]

node = xsc.Frag(
   xml.XML(),
   html.DocTypeXHTML10transitional(),
   html.html(
      html.head(
         meta.contenttype(),
         html.title("Cheeseshop links")
      ),
      html.body(
         html.h1("Cheeseshop links"),
         html.ul(html.li(cheeseshoplink(name=name)) for name in names)
      )
   )
)

print(node.conv().string(encoding="us-ascii"))
```

Parsing HTML

Parsing HTML is done like this:

```
from ll.xist import parse
from ll.xist.ns import html

node = parse.tree(
   parse.URL("http://www.python.org/"),
   parse.Tidy(),
   parse.NS(html),
   parse.Node()
)
```

Finding and counting nodes

The following example shows you how to output the URLs of all images inside links on Python's homepage:

```
>>> from ll.xist import parse
>>> from ll.xist.ns import html
>>> node = parse.tree(
...   parse.URL("http://www.python.org/"),
...   parse.Expat(ns=True),
...   parse.Node()
... )
>>> for img in node.walknodes(html.a/html.img):
...    print(img.attrs.src)
...
http://www.python.org/images/python-logo.gif
http://www.python.org/images/trans.gif
http://www.python.org/images/trans.gif
http://www.python.org/images/success/nasa.jpg
```

If you want to output both the links and the image URLs, do the following:

```
>>> from ll.xist import parse, xfind
>>> from ll.xist.ns import html
>>> node = parse.tree(
...   parse.URL("http://www.python.org/"),
...   parse.Expat(ns=True),
...   parse.Node()
... )
>>> for path in node.walkpaths(html.a/html.img):
...    print(path[-2].attrs.href, path[-1].attrs.src)
http://www.python.org/ http://www.python.org/images/python-logo.gif
http://www.python.org/#left%2dhand%2dnavigation http://www.python.org/images/trans.gif
http://www.python.org/#content%2dbody http://www.python.org/images/trans.gif
http://www.python.org/about/success/usa http://www.python.org/images/success/nasa.jpg
```

If you want to count the number of links on the page you can do the following:

```
>>> from ll import misc
>>> from ll.xist import parse
>>> from ll.xist.ns import html
>>> node = parse.tree(
...   parse.URL("http://www.python.org/"),
...   parse.Expat(ns=True),
...   parse.Node()
... )
>>> misc.count(node.walk(html.a))
83
```

Replacing text

This example demonstrates how to make a copy of an XML tree with some text replacements:

```
from ll.xist import xsc, parse

def p2p(node, converter):
   if isinstance(node, xsc.Text):
      node = node.replace("Python", "Parrot")
      node = node.replace("python", "parrot")
   return node

node = parse.tree(
   parse.URL("http://www.python.org/"),
   parse.Expat(ns=True),
   parse.Node()
)

node = node.mapped(p2p)
node.write(open("parrot_index.html", "wb"))
```

Converting HTML to XIST code

The class *ll.xist.present.CodePresenter* makes it possible to output an XIST tree as usable Python source code:

```
>>> from ll.xist import parse, present
>>> node = parse.tree(
...   parse.URL("http://www.python.org/"),
...   parse.Expat(ns=True),
...   parse.Node()
... )
>>> print(present.CodePresenter(node))
ll.xist.xsc.Frag(
   ll.xist.ns.html.html(
      ll.xist.ns.html.head(
         ll.xist.ns.html.meta(
            http_equiv='content-type',
            content='text/html; charset=utf-8'
         ),
         ll.xist.ns.html.title(
            'Python Programming Language -- Official Website'
         ),
         ll.xist.ns.html.meta(
            name='keywords',
            content='python programming language object oriented web free source'
         ),
         [... Many lines deleted ...]
                  u'\n\tCopyright \xa9 1990-2007, ',
                  ll.xist.ns.html.a(
                     'Python Software Foundation',
                     href='http://www.python.org/psf'
                  ),
                  ll.xist.ns.html.br(),
                  ll.xist.ns.html.a(
                     'Legal Statements',
                     href='http://www.python.org/about/legal'
```

(continues on next page)

(continued from previous page)

```
                ),
                '\n      ',
                id='footer'
            ),
            '\n\n\n    ',
            id='body-main'
        ),
        '\n  ',
        id='content-body'
    ),
    '\n'
),
lang='en'
)
)
```

Using converter contexts to pass information between elements

Converter contexts can be used to pass information between elements. The following example will generate HTML <h1>, ..., <h6> elements according to the nesting depth of a <section> element.

```
from ll.xist import xsc

class section(xsc.Element):
   class Attrs(xsc.Element.Attrs):
      class title(xsc.TextAttr): pass

   class Context(xsc.Element.Context):
      def __init__(self):
         xsc.Element.Context.__init__(self)
         self.level = 1

   def convert(self, converter):
      context = converter[self]
      elementname = f"h{min(context.level, 6)}"
      node = xsc.Frag(
         getattr(converter.target, elementname)(self.attrs.title),
         self.content
      )
      context.level += 1
      node = node.convert(converter)
      context.level -= 1
      return node

with xsc.build():
   with section(title="Python Tutorial") as document:
      with section(title="Using the Python Interpreter"):
         with section(title="Invoking the Interpreter"):
            +section(title="Argument Passing")
            +section(title="Interactive Mode")
         with section(title="The Interpreter and Its Environment"):
            +section(title="Error Handling")
            +section(title="Executable Python Scripts")
            +section(title="Source Code Encoding")
            +section(title="The Interactive Startup File")
```

(continues on next page)

(continued from previous page)

```
print(document.conv().string())
```

The output of this script will be:

```
<h1>Python Tutorial</h1>
<h2>Using the Python Interpreter</h2>
<h3>Invoking the Interpreter</h3>
<h4>Argument Passing</h4>
<h4>Interactive Mode</h4>
<h3>The Interpreter and Its Environment</h3>
<h4>Error Handling</h4>
<h4>Executable Python Scripts</h4>
<h4>Source Code Encoding</h4>
<h4>The Interactive Startup File</h4>
```

Formatting HTML as plain text

The function *ll.xist.ns.html.astext()* can to used to format HTML into plain text:

```
from ll.xist.ns import html

e = html.div(
   html.h1("The Zen of Python, by Tim Peters"),
   html.ul(
      html.li("Beautiful is better than ugly."),
      html.li("Explicit is better than implicit."),
      html.li("Simple is better than complex."),
      html.li("Complex is better than complicated."),
      html.li("Flat is better than nested."),
      html.li("Sparse is better than dense."),
      html.li("Readability counts."),
      html.li("Special cases aren't special enough to break the rules."),
      html.li("Although practicality beats purity."),
      html.li("Errors should never pass silently."),
      html.li("Unless explicitly silenced."),
      html.li("In the face of ambiguity, refuse the temptation to guess."),
      html.li("There should be one-- and preferably only one --obvious way to do it.
↪"),
      html.li("Although that way may not be obvious at first unless you're Dutch."),
      html.li("Now is better than never."),
      html.li("Although never is often better than *right* now."),
      html.li("If the implementation is hard to explain, it's a bad idea."),
      html.li("If the implementation is easy to explain, it may be a good idea."),
      html.li("Namespaces are one honking great idea -- let's do more of those!"),
   )
)

print(html.astext(e, width=40))
```

This will output:

```
The Zen of Python, by Tim Peters
================================

*  Beautiful is better than ugly.
```

(continues on next page)

(continued from previous page)

```
*  Explicit is better than implicit.

*  Simple is better than complex.

*  Complex is better than complicated.

*  Flat is better than nested.

*  Sparse is better than dense.

*  Readability counts.

*  Special cases aren't special enough
   to break the rules.

*  Although practicality beats purity.

*  Errors should never pass silently.

*  Unless explicitly silenced.

*  In the face of ambiguity, refuse the
   temptation to guess.

*  There should be one-- and preferably
   only one --obvious way to do it.

*  Although that way may not be obvious
   at first unless you're Dutch.

*  Now is better than never.

*  Although never is often better than
   *right* now.

*  If the implementation is hard to
   explain, it's a bad idea.

*  If the implementation is easy to
   explain, it may be a good idea.

*  Namespaces are one honking great idea
   -- let's do more of those!
```

6.1.2 Basic XIST concepts

This document explains parsing/generating XML files, XML transformations via XIST classes and other basic concepts.

XIST is an extensible HTML/XML generator written in Python. It was developed as a replacement for an HTML preprocessor named HSC and borrows some features and ideas from it. It also borrows the basic ideas (XML/HTML elements as Python objects) from HTMLgen or HyperText.

(If you're impatient, there's also a *list of examples* that shows what can be done with XIST.)

Overview

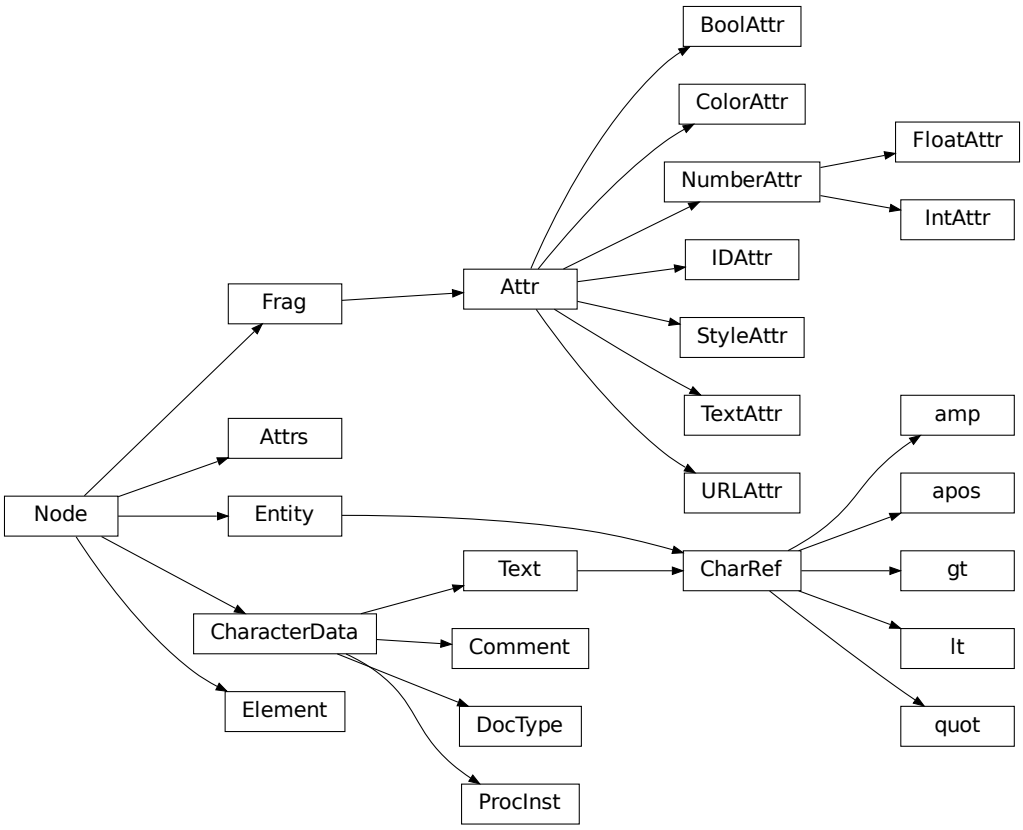
XIST can be used as a compiler that reads an input XML file and generates a transformed output file, or it could be used for generating XML dynamically inside a web server (but note that handling object trees *is* slower than simply sending string fragments). In either case generating the final HTML or XML output requires the following three steps:

- Generating a source XML tree: This can be done either by parsing an XML file, or by directly constructing the tree — as HTMLgen and HyperText do — as a tree of Python objects. XIST provides a very natural and pythonic API for that.
- Converting the source tree into a target tree: This target tree can be a HTML tree or a SVG tree or XSL-FO tree or any other XML tree you like. Every node class provides a `convert()` method for performing this conversion. For your own XML element types you have to define your own element classes and implement an appropriate `convert()` method. This is possible for processing instructions and entity references too.
- Publishing the target tree: For generating the final output a `Publisher` object is used that generates the encoded byte string fragments that can be written to an output stream (or yielded from a WSGI application, etc.).

Constructing XML trees

Like any other XML tree API, XIST provides the usual classes:

- `ll.xist.xsc.Element` for XML elements;
- `ll.xist.xsc.Attr` for attributes;
- `ll.xist.xsc.Attrs` for attribute mappings;
- `ll.xist.xsc.Text` for text data;
- `ll.xist.xsc.Frag` for document fragments, (a `Frag` object is simply a list of nodes);
- `ll.xist.xsc.Comment` for XML comments (e.g. `<!-- the comment -->`);
- `ll.xist.xsc.ProcInst` for processing instructions (e.g. `<?php echo $spam;?>`);
- `ll.xist.xsc.Entity` for entity references (e.g. `&parrot;`) and
- `ll.xist.xsc.DocType` for document type declarations (e.g. `<!DOCTYPE html PUBLIC ...>`).



Creating plain elements, processing instructions and entities

Creating elements

Creating an element can be done with the function *ll.xist.xsc.element()*. Its signature looks like this:

```
xsc.element(xmlns, xmlname, *content, **attrs)
```

`xmlns` is the namespace name (e.g. `"http://www.w3.org/1999/xhtml"` for HTML), and `xmlname` is the name of the element. Additional positional arguments (i.e. items in `content`) will be the child nodes of the element node. Keyword arguments will be attributes. You can pass most of Python's builtin types to *element()*. Strings and integers will be automatically converted to *Text* objects. Constructing an HTML element works like this:

Listing 1: The first example

```
from ll.xist import xsc

html_xmlns = "http://www.w3.org/1999/xhtml"

node = xsc.element(
   html_xmlns,
   "div",
   "Hello ",
   xsc.element(
```

(continues on next page)

(continued from previous page)

```
        html_xmlns,
        "a",
        "Python",
        href="http://www.python.org/"
    ),
    " world!"
)
```

To output this element again, the method *bytes()* can be used:

Listing 2: Output of the first example

```
>>> node.bytes()
b'<div>Hello <a href="http://www.python.org/">Python</a> world!</div>'
```

If you want a namespace declaration you can use the `prefixdefault` argument:

Listing 3: The first example with an `xmlns` declaration

```
>>> node.bytes(prefixdefault=None)
b'<div xmlns="http://www.w3.org/1999/xhtml">Hello <a href="http://www.python.org/">
↪Python</a> world!</div>'
```

For attribute names that collide with Python keywords or are not legal identifiers (most notably `class` in HTML) you can pass the attributes as a dictionary to *element()*:

Listing 4: Passing attributes as dictionaries

```
node = xsc.element(
    html_xmlns,
    "div",
    "Hello world!",
    {"class": "greeting", "id": 42, "title": "Greet the world"},
)
```

Creating processsing instructions

Processing instructions can be created with the function *ll.xist.xsc.procinst()*. Its signature looks like this:

```
xsc.procinst(xmlname, *content)
```

So to create and print a processsing instruction named `code` with the content `x = 42`, you can do the following (the method *string()* is similar to *bytes()*, but returns a `str` object instead of a `bytes` object):

Listing 5: Creating and printing a processsing instruction

```
from ll.xist import xsc

node = xsc.procinst("code", "x = 42")
print(node.string())
```

This will output:

```
<?code x = 42?>
```

Creating entity references

Finally entity references can be created with the function *ll.xist.xsc.entity()*:

Listing 6: Creating and printing an entity reference

```
from ll.xist import xsc

node = xsc.entity("html")
print(node.string())
```

This will output:

```
&html;
```

Creating XML trees with `with` blocks

Furthermore it's possible to use `with` blocks to construct XIST trees. Inside a `with` block the unary + operator or the *add()* function can be used to add nodes or attributes to the current level of the tree:

Listing 7: Using `with` blocks

```
from ll.xist import xsc

html_xmlns = "http://www.w3.org/1999/xhtml"

with xsc.build():
   with xsc.element(html_xmlns, "div", {"class": "quote"}) as node:
      with xsc.element(html_xmlns, "h1", "Confucius (551-479 BC)"):
         xsc.add({"class": "author"})
      with xsc.element(html_xmlns, "ol"):
         +xsc.element(html_xmlns, "li", "I hear and I forget.")
         +xsc.element(html_xmlns, "li", "I see and I believe.")
         +xsc.element(html_xmlns, "li", "I do and I understand.")
```

ll.xist.xsc.build must be used as the top level with block, so that XIST knows what to do with the nodes inside the block.

Creating XML trees from XML files

XML trees can also be generated by parsing XML files. For this the module *ll.xist.parse* provides several tools.

For example, parsing a string can be done like this:

Listing 8: Parsing a string

```
from ll.xist import parse

node = parse.tree(
   b"<p xmlns='http://www.w3.org/1999/xhtml'>Hello <a href='http://www.python.org/'>
↪Python</a> world!</p>",
   parse.Expat(ns=True),
   parse.Node()
)
```

For further info about the arguments to the parsing functions, see the documentation for *parse*.

XML trees as Python objects

XIST works somewhat different from a normal DOM API. Instead of only one element class, XIST has one class for every element type. All the elements from different XML vocabularies known to XIST are defined in modules in the *ll.xist.ns* subpackage. (Of course it's possible to define additional element classes for your own XML vocabulary). The definition of HTML can be found in *ll.xist.ns.html* for example.

Every element class has a constructor of the form:

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

Positional arguments (i.e. items in `content`) will be the child nodes of the element node. Keyword arguments will be attributes. You can pass most of Python's builtin types to such a constructor. Strings and integers will be automatically converted to *Text* objects. Constructing an HTML element works like this:

Listing 9: The first example

```
from ll.xist.ns import html

node = html.div(
   "Hello ",
   html.a("Python", href="http://www.python.org/"),
   " world!"
)
```

For attribute names that collide with Python keywords or are not legal identifiers (most notably `class` in HTML) the attribute name must be slightly modified, so that it's a legal Python identifier (for `class` an underscore is appended):

Listing 10: Illegal attribute names

```
node = html.div(
   "Hello world!",
   class_="greeting"
)
```

(Don't worry: This modified attribute name will be mapped to the real official attribute name once the output is generated.)

You can pass attributes as a dictionary too:

Listing 11: Passing attributes as dictionaries

```
node = html.div(
   "Hello world!",
   dict(class_="greeting", id=42, title="Greet the world")
)
```

Furthermore it's possible to use `with` blocks to construct XIST trees. Inside a `with` block the unary + operator or the *ll.xist.xsc.add()* function can be used to add nodes or attributes to the current level of the tree:

Listing 12: Using `with` blocks

```
with xsc.build():
   with html.div(class_="quote") as node:
      with html.h1("Confucius (551-479 BC)"):
         xsc.add(class_="author")
      with html.ol():
         +html.li("I hear and I forget.")
```

(continues on next page)

(continued from previous page)

```
        +html.li("I see and I believe.")
        +html.li("I do and I understand.")
```

ll.xist.xsc.build must be used as the top level with block, so that XIST knows what to do with the nodes inside the block.

Generating XML trees from XML files

XML trees can also be generated by parsing XML files. For this the module *ll.xist.parse* provides several tools.

For example, parsing a string can be done like this:

Listing 13: Parsing a string

```
from ll.xist import parse
from ll.xist.ns import html

node = parse.tree(
   b"<p>Hello <a href='http://www.python.org/'>Python</a> world!</p>",
   parse.Expat(),
   parse.NS(html),
   parse.Node()
)
```

For further info about the arguments to the parsing functions, see the documentation for *parse*.

Defining new elements and converting XML trees

To be able to parse an XML file, you have to provide an element class for every element type that appears in the file. These classes either come from modules provided by XIST or you can define your own. Defining your own element class for an element named `cool` works like this:

Listing 14: Defining a new element

```
class cool(xsc.Element):
   def convert(self, converter):
      node = html.b(self.content, " is cool!")
      return node.convert(converter)
```

You have to derive your new class from *ll.xist.xsc.Element*. The name of the class will be the element name. For element type names that are invalid Python identifiers, you can use the class attribute `xmlname` in the element class to overwrite the element name.

To be able to convert an element of this type to a new XML tree (probably HTML in most cases), you have to implement the *convert()* method. In this method you can build a new XML tree from the content and attributes of the object.

Using this new element is simple:

Listing 15: Using the new element

```
>>> node = cool("Python")
>>> print(node.conv().bytes())
b'<b>Python is cool!</b>'
```

conv() simply calls *convert()* with a default `converter` argument. We'll come to converters in a minute. *bytes()* is a method that converts the node to a byte string. This method will be explained when we discuss the publishing interface.

Note that it is vital for your own *convert()* methods that you recursively call *convert()* on your own content, because otherwise some unconverted nodes might remain in the tree. Let's define a new element:

```
class python(xsc.Element):
   def convert(self, converter):
      return html.a("Python", href="http://www.python.org/")
```

Now we can do the following:

```
>>> node = cool(python())
>>> print(node.conv().bytes())
b'<b><a href="http://www.python.org/">Python</a> is cool!</b>'
```

But if we forget to call *convert()* for our own content, i.e. if the element `cool` was written like this:

```
class cool(xsc.Element):
   def convert(self, converter):
      return html.b(self.content, " is cool!")
```

we would get:

```
>>> node = cool(python())
>>> print(node.conv().bytes())
b'<b><python></python> is cool!</b>'
```

Furthermore *convert()* should never modify `self`, because *convert()* might be called multiple times for the same node.

Converters

conv() is a convenience method that creates a default converter for you and calls *convert()*. This converter is created once and is passed to all *convert()* calls. It is used to store parameters for the conversion process and it allows *convert()* methods to store additional information, so that it is available elsewhere during the conversion process. You can also call *convert()* yourself, which would look like this:

```
from ll.xist import xsc
from ll.xist.ns import html

node = cool(python())
node = node.convert(xsc.Converter())
```

You can pass the following arguments to the *Converter* constructor:

root

`root` (which defaults to `None`) is the root URL for the conversion process. When you want to resolve a link in some of your own *convert()* methods, the URL must be interpreted relative to this root URL (You can use *ll.xist.xsc.URLAttr.forInput()* for that).

mode

`mode` (which defaults to `None`) works the same way as modes in XSLT. You can use this for implementing different conversion modes.

stage

`stage` (which defaults to `"deliver"`) allows you to implement multi stage conversion: Suppose that you want to deliver a dynamically constructed web page with XIST that contains results from a database query and the current time. The data in the database changes infrequently, so it doesn't make sense to do the query on every request. The query is done every few minutes and the resulting HTML tree is stored in the servlet (using any of the available Python servlet technologies). For this conversion the `stage` would be `"cache"` and your database XML element would do the query when `stage == "cache"`. Your time display element would do the conversion when `stage == "deliver"` and simply returns itself when `stage == "cache"`, so it would still be part of the cached XML tree and would be converted to HTML on every request.

target

`target` (which defaults to *`html`*) specifies what the output should be. Values must be namespace modules (see below for an explanation of namespaces).

lang

`lang` (which defaults to `None`) is the language in which the result tree should be. This can be used in the *`convert()`* method to implement different conversions for different languages, e.g.:

```
class note(xsc.Element):
   def convert(self, converter):
      if converter.lang == "de":
         title = "Anmerkung"
      elif converter.lang == "en":
         title = "Note"
      else:
         title = "???"
      node = xsc.Frag(
         html.h1(title),
         html.div(self.content)
      )
      return node.convert(converter)
```

Additional arguments are passed when a converter is created in the context of a *`ll.make`* script.

Attributes

Setting and accessing the attributes of an element works either via a dictionary interface or by accessing the XML attributes as Python attributes of the elements `attrs` attribute:

```
>>> node = html.a("Python", href="http://www.python.org/")
>>> print(node.bytes())
b'<a href="http://www.python.org/">Python</a>'
>>> del node.attrs.href
>>> print(node.bytes())
b'<a>Python</a>'
>>> node.attrs["href"] = "http://www.python.org"
>>> print(node.bytes())
b'<a href="http://www.python.org/">Python</a>'
```

All attribute values are instances of subclasses of the class *`ll.xist.xsc.Attr`*. Available subclasses are:

- *`ll.xist.xsc.TextAttr`*, for normal text attributes;
- *`ll.xist.xsc.URLAttr`*, for attributes that are URLs;
- *`ll.xist.xsc.BoolAttr`*, for boolean attributes (for such an attribute only its presence is important, it's value will always be the same as the attribute name when publishing);

- *ll.xist.xsc.IntAttr*, for integer attributes;
- *ll.xist.xsc.ColorAttr*, for color attributes (e.g. `#fff`).

IntAttr and *ColorAttr* mostly serve as documentation of the attributes purpose. Both classes have no added functionality.

Attr itself is derived from *Frag* so it is possible to use all the sequence methods on an attribute.

Unset attributes will be treated like empty ones so the following is possible:

```
del node.attrs["spam"]
node.attrs["spam"].append("ham")
```

This also means that after the:

```
del node.attrs["spam"][:]
```

the attribute `spam` will be empty again and will be considered to be unset. Such attributes will be skipped when publishing.

The main purpose of this is to allow you to construct values conditionally and then use those values as attribute values:

```
import random

if random.random() < 0.5:
   class_ = None
else:
   class_ = "foo"

node = html.div("foo", class_=class_)
```

In 50% of the cases the generated *div* element will not have a `class` attribute.

Defining attributes

When you define a new element you have to specify the attributes allowed for this element. For this use the class attribute `Attrs` (which must be a class derived from *ll.xist.xsc.Element.Attrs*) and define the attributes by deriving them from one of the existing attribute classes. We could extend our example element in the following way:

Listing 16: Using attributes

```
class cool(xsc.Element):
   class Attrs(xsc.Element.Attrs):
      class adj(xsc.TextAttr): pass

   def convert(self, converter):
      node = xsc.Frag(self.content, " is")
      if "adj" in self.attrs:
         node.append(" ", html.em(self.attrs.adj))
      node.append(" cool!")
      return node.convert(converter)
```

and use it like this:

```
>>> node = cool(python(), adj="totally")
>>> node.conv().bytes()
<a href="http://www.python.org/">Python</a> is <em>totally</em> cool!
```

Default attributes

It is possible to define default values for attributes via the class attribute `default`:

Listing 17: Defining default attribute values

```
class cool(xsc.Element):
   class Attrs(xsc.Element.Attrs):
      class adj(xsc.TextAttr):
         default = "absolutely"

   def convert(self, converter):
      node = xsc.Frag(self.content, " is")
      if "adj" in self.attrs:
         node.append(" ", html.em(self.attrs.adj))
      node.append(" cool!")
      return node.convert(converter)
```

Now if we instantiate the class without specifying `adj` we'll get the default:

Listing 18: Using default attributes

```
>>> node = cool(python())
>>> print(node.conv().bytes())
b'<a href="http://www.python.org/">Python</a> is <em>absolutely</em> cool!'
```

If we want a `cool` instance without an `adj` attribute, we can pass `None` as the attribute value:

Listing 19: Removing default attributes

```
>>> node = cool(python(), adj=None)
>>> print(node.conv().bytes())
b'<a href="http://www.python.org/">Python</a> is cool!'
```

Allowed attribute values

It's possible to specify that an attribute has a fixed set of allowed values. This can be done with the class attribute `values`. We could extend our example to look like this:

Listing 20: Defining allowed attribute values

```
class cool(xsc.Element):
   class Attrs(xsc.Element.Attrs):
      class adj(xsc.TextAttr):
         default = "absolutely"
         values = ("absolutely", "totally", "very")

   def convert(self, converter):
      node = xsc.Frag(self.content, " is")
      if "adj" in self.attrs:
         node.append(" ", html.em(self.attrs.adj))
      node.append(" cool!")
      return node.convert(converter)
```

These values won't be checked when we create our `cool` instance. Only when this node is parsed from a file will the warning be issued. The warning will also be issued if we publish such a node, but note that for warnings Python's warning framework is used, so the warning will be printed only once (but of course you can change that with `warnings.filterwarnings()`):

```
>>> node = cool(python(), adj="pretty")
>>> print(node.bytes())
/Users/walter/checkouts/LivingLogic.Python.xist/src/ll/xist/xsc.py:2368: \
IllegalAttrValueWarning: Attribute value 'pretty' not allowed for __main__:cool.Attrs.
↪adj
  warnings.warn(IllegalAttrValueWarning(self))
b'<cool adj="very"><python /></cool>'
```

Required attributes

Finally it's possible to specify that an attribute is required. This again will only be checked when parsing or publishing. To specify that an attribute is required simply add the class attribute `required` with the value `True`. The attribute `alt` of the class *`ll.xist.ns.html.img`* is such an attribute, so we'll get:

Listing 21: Missing required attributes

```
>>> from ll.xist.ns import html
>>> node = html.img(src="eggs.png")
>>> print(node.bytes())
/Users/walter/checkouts/LivingLogic.Python.xist/src/ll/xist/xsc.py:2770: \
RequiredAttrMissingWarning: Required attribute 'alt' missing in ll.xist.ns.html:img.
↪Attrs.
  warnings.warn(errors.RequiredAttrMissingWarning(self, attrs.keys()))
<img src="eggs.png" />
```

Namespaces and pools

Now that you've defined your own elements, you have to tell the parser about them, so they can be instantiated when a file is parsed. First you have to assign an XML namespace to these classes. This is done by setting the class attribute `xmlns` to the namespace name:

Listing 22: Assigning a namespace to elements

```
from ll.xist import xsc, parse
from ll.xist.ns import html

xmlns = "http://xmlns.example.org/foo"

class python(xsc.Element):
   xmlns = xmlns

   def convert(self, converter):
      return html.a("Python", href="http://www.python.org/")

class cool(xsc.Element):
   xmlns = xmlns

   def convert(self, converter):
      node = html.b(self.content, " is cool!")
      return node.convert(converter)
```

When parsing the parser fetches the classes it uses from a *`ll.xist.xsc.Pool`* object. We can put our two classes into a pool like this:

Listing 23: Putting elements in a pool

```
pool = xsc.Pool(python, cool)
```

It's also possible to register the element classes in a pool directly at class construction time via a `with` block like this:

Listing 24: Populating a pool with a `with` block

```
from ll.xist import xsc, parse
from ll.xist.ns import html

with xsc.Pool() as pool:
   xmlns = "http://xmlns.example.org/foo"

   class python(xsc.Element):
      xmlns = xmlns

      def convert(self, converter):
         return html.a("Python", href="http://www.python.org/")

   class cool(xsc.Element):
      xmlns = xmlns

      def convert(self, converter):
         node = html.b(self.content, " is cool!")
         return node.convert(converter)
```

Now you can use this pool for parsing:

Listing 25: Parsing XML

```
s = b'<cool xmlns="http://xmlns.example.org/foo"><python/></cool>'

node = parse.tree(s, parse.Expat(ns=True), pool)
```

It's also possible to call the parsing function with a predefined mapping between namespace names and namespace prefixes:

Listing 26: Parsing XML with predefined prefix mapping

```
s = b'<cool><python/></cool>'

node = parse.tree(
   s,
   parse.Expat(),
   parse.NS("http://xmlns.example.org/foo"),
   pool
)
```

If you have many elements, registering them in a pool becomes cumbersome. In this case you can put your element classes into a module and then register all elements in the module:

Listing 27: Registering modules in a pool

```
import foo_xmlns # This is the module containing the element classes

pool = xsc.Pool(foo_xmlns)
```

Global attributes

You can define global attributes belonging to a certain namespace by defining a global `Attrs` class and giving each attribute a namespace name via `xmlns`:

```
class Attrs(xsc.Attrs):
   class foo(xsc.TextAttr):
      xmlns = "http://www.example.com/foo"
```

To make this global attribute know to the parsing, you simply can put the `Attrs` in the pool used for parsing.

Setting and accessing such an attribute can be done by using the attribute class instead of the attribute name like this:

```
>>> from ll.xist.ns import html
>>> node = html.div("foo", {Attrs.foo: "bar"})
>>> str(node[Attrs.foo])
'bar'
```

An alternate way of specifying a global attribute in a constructor looks like this:

```
>>> from ll.xist.ns import html
>>> node = html.div("foo", Attrs(foo="baz"))
>>> str(node[Attrs.foo])
'baz'
```

Entities

In the same way as defining new element types, you can define new entities. The following example is from the module *ll.xist.ns.abbr*:

Listing 28: Defining new entities

```
from ll.xist import xsc
from ll.xist.ns import html

class html(xsc.Entity):
   def convert(self, converter):
      return html.abbr(
         "HTML",
         title="Hypertext Markup Language",
         lang="en"
      )
```

You can use this entity in your XML files like this:

Listing 29: Using the newly defined entity

```
<cool adj="very">&html;</cool>
```

Processing instructions

Defining processing instructions works just like elements and entities. Derive a new class from *ll.xist.xsc.ProcInst* and implement *convert()*. The following example implements a processing instruction that returns an uppercase version of its content as a text node.

Listing 30: Defining new processing instructions

```
class upper(xsc.ProcInst):
   def convert(self, converter):
      return xsc.Text(self.content.upper())
```

It can be used in an XML file like this:

Listing 31: Using the newly defined processing instruction

```
<cool><?upper Python?></cool>
```

There are namespaces containing processing instruction classes that don't provide a *convert()* method. These processing instruction objects will then be published as XML processing instructions. One example is the namespace *ll.xist.ns.php*.

Other namespaces (like *ll.xist.ns.jsp*) contain processing instruction classes, but they will be published in a different (not XML compatible) format. For example `ll.xist.ns.jsp.expression("foo")` will be published as `<%= foo>`.

Publishing XML trees

After creating the XML tree and converting the tree into its final output form, you have to write the resulting tree to a file. This can be done with the publishing API. Three methods that use the publishing API are *ll.xist.xsc.Node.iterbytes()*, *ll.xist.xsc.Node.bytes()* and *ll.xist.xsc.Node.write()*. *ll.xist.xsc.Node.iterbytes()* is a generator that will yield the complete 8-bit XML string in fragments. *ll.xist.xsc.Node.bytes()* returns the complete 8-bit XML string.

Writing a node to a file can be done with the method *ll.xist.xsc.Node.write()*:

```
>>> from ll.xist.ns import html
>>> node = html.div("äöü", html.br(), "ÄÖÜ")
>>> with open("foo.html", "wb") as f:
...   node.write(f, encoding="ascii")
...
```

All these methods use the method *ll.xist.xsc.Node.publish()* internally. *publish()* gets passed an instance of *ll.xist.xsc.Publisher*.

Specifying an encoding

You can specify the encoding with the parameter `encoding` (with the encoding specified in an XML declaration being the default, if there is no such declaration `"utf-8"` is used). Unencodable characters will be escaped with character references when possible (i.e. inside text nodes, for comments or processing instructions you'll get an exception):

```
>>> from ll.xist import xsc
>>> from ll.xist.ns import html
>>> s = "A\xe4\u03a9\u8a9e"
>>> node = html.div(s)
>>> node.bytes(encoding="ascii")
```

(continues on next page)

(continued from previous page)

```
b'<div>;A&#228;&#937;&#35486;</div>;'
>>> node.bytes(encoding="iso-8859-1")
b'<div>;A\xe4&#937;&#35486;</div>;'
>>> xsc.Comment(s).bytes(encoding="ascii")
Traceback (most recent call last):
...
  File "/Users/walter/.local/lib/python3.3/encodings/ascii.py", line 22, in encode
    return codecs.ascii_encode(input, self.errors)[0]
UnicodeEncodeError: 'ascii' codec can't encode characters in position 1-3: ordinal␣
↪not in range(128)
```

When you include an *XML* header or an XML *contenttype*, XIST will automatically insert the correct encoding when publishing:

```
>>> from ll.xist import xsc
>>> from ll.xist.ns import xml, meta
>>> e = xsc.Frag(xml.XML(), "\n", meta.contenttype())
>>> e.conv().bytes(encoding="iso-8859-15")
b'<?xml version="1.0" encoding="iso-8859-15"?>\n<meta http-equiv="Content-Type"␣
↪content="text/html; charset=iso-8859-15" />'
```

HTML compatibility

Another useful parameter is `xhtml`, it specifies whether you want pure HTML or XHTML as output:

xhtml==0
: This will give you pure HTML, i.e. no final `/` for elements with an empty content model, so you'll get e.g. `<br>` in the output. Elements that don't have an empty content model, but are empty will be published with a start and end tag (i.e. `<div></div>`).

xhtml==1
: This gives HTML compatible XHTML. Elements with an empty content model will be published like this: `<br />` (This is the default).

xhtml==2
: This gives full XML output. Every empty element will be published with an empty tag (without an additional space): `<br/>` or `<div/>`.

Namespaces

By default XIST doesn't output any namespace declarations. The simplest way to change that, is to pass `True` for the `prefixdefault` argument when publishing:

Listing 32: Publishing namespace info

```
from ll.xist.ns import html

e = html.html(
   html.head(
      html.title("The page")
   ),
   html.body(
      html.h1("The header"),
      html.p("The content")
   )
)
```

(continues on next page)

(continued from previous page)

```
print(e.bytes(prefixdefault=True))
```

Using `True` allows XIST to choose its own prefixes. The code above will output (rewrapped for clarity):

```
<ns:html xmlns:ns="http://www.w3.org/1999/xhtml">
   <ns:head>
      <ns:title>The page</ns:title>
   </ns:head>
   <ns:body>
      <ns:h1>The header</ns:h1>
      <ns:p>The content</ns:p>
   </ns:body>
</ns:html>
```

You can also use a fixed prefix:

```
print(e.bytes(prefixdefault="h"))
```

This will output (again rewrapped):

```
<h:html xmlns:h="http://www.w3.org/1999/xhtml">
   <h:head>
      <h:title>The page</h:title>
   </h:head>
   <h:body>
      <h:h1>The header</h:h1>
      <h:p>The content</h:p>
   </h:body>
</h:html>
```

If you want the empty prefix you can use `None`:

```
print(e.bytes(prefixdefault=None))
```

This will output (again rewrapped):

```
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <title>The page</title>
   </head>
   <body>
      <h1>The header</h1>
      <p>The content</p>
   </body>
</html>
```

When elements from more than one namespace are present in the tree, `prefixdefault` is unreliable. The first namespace encountered will get the prefix specified by `prefixdefault`, all others will get a different prefix. XIST will never use the same prefix for different namespaces. XIST will also refuse to use an empty prefix for global attributes:

Listing 33: Publishing global attributes

```
from ll.xist import xsc
from ll.xist.ns import html, xlink

with xsc.build():
```

(continues on next page)

(continued from previous page)

```
   with html.html() as e:
      with html.head():
         +html.title("The page")
      with html.body():
         +html.h1("The header"),
         with html.p():
            +xsc.Text("The "),
            +html.a(
               "Python",
               xlink.Attrs(
                  href="http://www.python.org/",
                  title="Python",
                  type="simple"
               ),
               href="http://www.python.org/"
            )
            +xsc.Text(" homepage")

print(e.bytes(prefixdefault=None))
```

This will output:

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:ns="http://www.w3.org/1999/xlink">
   <head>
      <title>The page</title>
   </head>
   <body>
      <h1>The header</h1>
      <p>The <a ns:href="http://www.python.org/" ns:type="simple" ns:title="Python"␣
↪href="http://www.python.org/">Python</a> homepage</p>
   </body>
</html>
```

In the case of multiple namespaces you can use the `prefixes` argument to specify an explicit prefix for each namespace. So we could change the publishing statement from our example above to:

```
print(e.bytes(prefixes={"http://www.w3.org/1999/xhtml": None, "http://www.w3.org/1999/
↪xlink": "xl"}))
```

which would give us the output:

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:xl="http://www.w3.org/1999/xlink">
   <head>
      <title>The page</title>
   </head>
   <body>
      <h1>The header</h1>
      <p>The <a xl:href="http://www.python.org/" xl:type="simple" xl:title="Python"␣
↪href="http://www.python.org/">Python</a> homepage</p>
   </body>
</html>
```

Note that we can shorten the publishing call from above to:

```
print(e.bytes(prefixes={html.xmlns: None, xlink.xmlns: "xl"}))
```

or even to:

```
print(e.bytes(prefixes={html: None, xlink: "xl"}))
```

Furthermore it's possible to suppress output of namespace declarations for certain namespaces by using the `hidexmlns` argument:

```
print(e.bytes(prefixes={html: None, xlink: "xl"}, hidexmlns=(html, xlink)))
```

This will output:

```
<html>
   <head>
      <title>The page</title>
   </head>
   <body>
      <h1>The header</h1>
      <p>The <a xl:href="http://www.python.org/" xl:type="simple" xl:title="Python"␣
↪href="http://www.python.org/">Python</a> homepage</p>
   </body>
</html>
```

Finally it's possible to force the output of namespace declarations for certain namespaces (even if elements from those namespaces are not in the tree) by using the `showxmlns` argument:

```
print(html.div().bytes(prefixes={html: None, xlink: "xl"}, showxmlns=(xlink,)))
```

This will output:

```
<div xmlns="http://www.w3.org/1999/xhtml" xmlns:xl="http://www.w3.org/1999/xlink"></
↪div>
```

6.1.3 Iterating through XIST trees

There are three related methods available for iterating through an XML tree and finding nodes in the tree: The methods *walk()*, *walknodes()* and *walkpaths()*.

The `walk()` method

The method *walk()* is a generator. When called without any arguments it visits each node in the tree once. Furthermore without arguments parent nodes are yielded before their children, and no attribute nodes are yielded. (This can however be changed by passing certain arguments to *walk()*.)

What *walk()* outputs is a *Cursor* object (in fact *walk()* always yields the same cursor object, but the attributes will be updated during the traversal). A *Cursor* object has the following attributes:

root
: The node where traversal has been started (i.e. the object for which the *walk()* method has been called).

node
: The current node being traversed.

path
: A list of nodes that contains the path through the tree from the root to the current node (i.e. `path[0]` is `root` and `path[-1]` is `node`).

index
: A path of indices (e.g. `[0, 1]` if the current node is the second child of the first child of the root). Inside attributes the index path will contain the name of the attribute (or a (attribute name, namespace name) tuple inside a global attribute).

event

A string that specifies which event is currently being handled. Possible values are: "enterelementnode", "leaveelementnode", "enterattrnode", "leaveattrnode", "textnode", "commentnode", "doctypenode", "procinstnode", "entitynode" and "nullnode".

The following example shows the basic usage of the *walk()* method:

Listing 34: Using the walk() method

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i) for i in range(3))
>>> for cursor in e.walk():
...   print(f"{cursor.event} {cursor.node!r}")
...
enterelementnode <ll.xist.ns.html.ul element object (3 children/no attrs) at 0x43fbb0>
enterelementnode <ll.xist.ns.html.li element object (1 child/no attrs) at 0x452750>
textnode <ll.xist.xsc.Text content='0' at 0x5b1670>
enterelementnode <ll.xist.ns.html.li element object (1 child/no attrs) at 0x452830>
textnode <ll.xist.xsc.Text content='1' at 0x5b16e8>
enterelementnode <ll.xist.ns.html.li element object (1 child/no attrs) at 0x5b30d0>
textnode <ll.xist.xsc.Text content='2' at 0x5b1760>
```

The path attribute can be used like this:

Listing 35: Using the path attribute

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i) for i in range(3))
>>> for cursor in e.walk():
...   print([f"{n.__class__.__module__}.{n.__class__.__qualname__}" for n in cursor.
↪path])
...
['ll.xist.ns.html.ul']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li', 'll.xist.xsc.Text']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li', 'll.xist.xsc.Text']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li', 'll.xist.xsc.Text']
```

The following example shows how the index attribute works:

Listing 36: Using the `index` attribute

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i) for i in range(3))
>>> for cursor in e.walk():
...   print(f"{cursor.index} {cursor.node!r}")
...
[] <ll.xist.ns.html.ul element object (5 children/no attrs) at 0x4b7bb0>
[0] <ll.xist.ns.html.li element object (1 child/no attrs) at 0x4ca750>
[0, 0] <ll.xist.xsc.Text content='0' at 0x629670>
[1] <ll.xist.ns.html.li element object (1 child/no attrs) at 0x4ca830>
[1, 0] <ll.xist.xsc.Text content='1' at 0x6296e8>
[2] <ll.xist.ns.html.li element object (1 child/no attrs) at 0x62b0d0>
[2, 0] <ll.xist.xsc.Text content='2' at 0x629760>
```

Changing which parts of the tree are traversed

The *walk()* method has a few additional parameters that specify which part of the tree should be traversed and in which order:

`entercontent` (default `True`)
Should the content of an element be entered? Note that when you call *walk()* with `entercontent` being false, *walk()* will only yield the root node itself.

`enterattrs` (default `False`)
Should the attributes of an element be entered? The following example shows the usage of `enterattrs`:

Listing 37: Using the `enterattrs` paameter

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i, class_=f"li-{i}") for i in range(3))
>>> for cursor in e.walk(enterattrs=True):
...   indent = "\t"*(len(cursor.path)-1)
...   print(f"{indent}{cursor.node!r}")
...
<ll.xist.ns.html.ul element object (3 children/no attrs) at 0x51e790>
   <ll.xist.ns.html.li element object (1 child/1 attr) at 0x51e8b0>
      <ll.xist.ns.html.coreattrs.class_ attr object (1 child) at 0x532f30>
      <ll.xist.xsc.Text content='0' at 0x67e6c0>
   <ll.xist.ns.html.li element object (1 child/1 attr) at 0x67f8b0>
      <ll.xist.ns.html.coreattrs.class_ attr object (1 child) at 0x671720>
      <ll.xist.xsc.Text content='1' at 0x67e7b0>
   <ll.xist.ns.html.li element object (1 child/1 attr) at 0x67f930>
      <ll.xist.ns.html.coreattrs.class_ attr object (1 child) at 0x671630>
      <ll.xist.xsc.Text content='2' at 0x67e990>
```

When both `entercontent` and `enterattrs` are true, the attributes will always be entered before the content. Setting `enterattrs` to true will only visit the attribute nodes themselves, but not their content.

`enterattr` (default `False`)
Should the content of the attributes of an element be entered? (This is only relevant if `enterattrs` is true.) The following example shows the usage of the `enterattr` parameter:

Listing 38: Using the `enterattr` paameter

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i, class_=f"li-{i}") for i in range(3))
>>> for cursor in e.walk(enterattrs=True, enterattr=True):
```

(continues on next page)

(continued from previous page)

```
...    indent = "\t"*(len(cursor.path)-1)
...    print(f"{indent}{cursor.node!r}")
...
<ll.xist.ns.html.ul element object (3 children/no attrs) at 0x4c1790>
   <ll.xist.ns.html.li element object (1 child/1 attr) at 0x4c18b0>
      <ll.xist.ns.html.coreattrs.class_ attr object (1 child) at 0x4d5f30>
         <ll.xist.xsc.Text content='li-0' at 0x621788>
      <ll.xist.xsc.Text content='0' at 0x621710>
   <ll.xist.ns.html.li element object (1 child/1 attr) at 0x6228b0>
      <ll.xist.ns.html.coreattrs.class_ attr object (1 child) at 0x614720>
         <ll.xist.xsc.Text content='li-1' at 0x621968>
      <ll.xist.xsc.Text content='1' at 0x621800>
   <ll.xist.ns.html.li element object (1 child/1 attr) at 0x622930>
      <ll.xist.ns.html.coreattrs.class_ attr object (1 child) at 0x614630>
         <ll.xist.xsc.Text content='li-2' at 0x621ad0>
      <ll.xist.xsc.Text content='2' at 0x6219e0>
```

Changing traversal order

The default traversal order is "top down". The following *walk()* parameters can be used to change that into "bottom up" order or into visiting each element or attribute both on the way down *and* up:

enterelementnode (default True)
Should the generator yield the cursor before it enters an element (i.e. before it visits the attributes and content of the element)? The cursor attribute `event` will have the value `"enterelementnode"` in this case.

leaveelementnode (default False)
Should the generator yield the cursor after it has visited an element? The cursor attribute `event` will have the value `"leaveelementnode"` in this case. Passing `enterelementnode=False`, `leaveelementnode=True` to *walk()* will change "top down" traversal into "bottom up".

enterattrnode (default True)
Should the generator yield the cursor before it enters an attribute? The cursor attribute `event` will have the value `"enterattrnode"` in this case. Note that the attribute will only be entered when `enterattr` is true and it will only be visited if `enterattrs` is true.

leaveattrnode (default False)
Should the generator yield the cursor after it has visited an attribute? The cursor attribute `event` will have the value `"leaveattrnode"` in this case. Note that the attribute will only be entered when `enterattr` is true and it will only be visited if `enterattrs` is true.

Passing `True` for all these parameters gives us the following output:

Listing 39: Full tree traversal

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i, class_=f"li-{i}") for i in range(3))
>>> for cursor in e.walk(entercontent=True, enterattrs=True, enterattr=True,
...    enterelementnode=True, leaveelementnode=True,
...    enterattrnode=True, leaveattrnode=True):
...   indent = "\t"*(len(cursor.path)-1)
...   print(f"{indent}{cursor.event} {cursor.index} {cursor.node!r}")
...
enterelementnode [] <ll.xist.ns.html.ul element object (3 children/no attrs) at␣
↪0x4cbe50>
   enterelementnode [0] <ll.xist.ns.html.li element object (1 child/1 attr) at␣
↪0x4de850>
      enterattrnode [0, 'class'] <ll.xist.ns.html.coreattrs.class_ attr object (1␣
```

(continues on next page)

(continued from previous page)

```
↪child) at 0x4f2f90>
         textnode [0, 'class', 0] <ll.xist.xsc.Text content='li-0' at 0x63f800>
      leaveattrnode [0, 'class'] <ll.xist.ns.html.coreattrs.class_ attr object (1␣
↪child) at 0x4f2f90>
      textnode [0, 0] <ll.xist.xsc.Text content='0' at 0x63f788>
   leaveelementnode [0] <ll.xist.ns.html.li element object (1 child/1 attr) at␣
↪0x4de850>
   enterelementnode [1] <ll.xist.ns.html.li element object (1 child/1 attr) at␣
↪0x63e870>
      enterattrnode [1, 'class'] <ll.xist.ns.html.coreattrs.class_ attr object (1␣
↪child) at 0x631780>
         textnode [1, 'class', 0] <ll.xist.xsc.Text content='li-1' at 0x63f9e0>
      leaveattrnode [1, 'class'] <ll.xist.ns.html.coreattrs.class_ attr object (1␣
↪child) at 0x631780>
      textnode [1, 0] <ll.xist.xsc.Text content='1' at 0x63f878>
   leaveelementnode [1] <ll.xist.ns.html.li element object (1 child/1 attr) at␣
↪0x63e870>
   enterelementnode [2] <ll.xist.ns.html.li element object (1 child/1 attr) at␣
↪0x63e8f0>
      enterattrnode [2, 'class'] <ll.xist.ns.html.coreattrs.class_ attr object (1␣
↪child) at 0x631690>
         textnode [2, 'class', 0] <ll.xist.xsc.Text content='li-2' at 0x63fb48>
      leaveattrnode [2, 'class'] <ll.xist.ns.html.coreattrs.class_ attr object (1␣
↪child) at 0x631690>
      textnode [2, 0] <ll.xist.xsc.Text content='2' at 0x63fa58>
   leaveelementnode [2] <ll.xist.ns.html.li element object (1 child/1 attr) at␣
↪0x63e8f0>
leaveelementnode [] <ll.xist.ns.html.ul element object (3 children/no attrs) at␣
↪0x4cbe50>
```

Skipping parts of the tree

It is possible to change the cursor attributes that specify the traversal order during the traversal to skip certain parts of the tree. In the following example the content of *li* elements is skipped if they have a `class` attribute:

Listing 40: Skipping parts of the tree

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i, class_=None if i%2 else f"li-{i}") for i in range(3))
>>> for cursor in e.walk():
...   if isinstance(cursor.node, html.li) and "class_" in cursor.node.attrs:
...      cursor.entercontent = False
...   indent = "\t"*(len(cursor.path)-1)
...   print(f"{indent}{cursor.event} {cursor.node!r}")
...
enterelementnode <ll.xist.ns.html.ul element object (3 children/no attrs) at 0x495790>
   enterelementnode <ll.xist.ns.html.li element object (1 child/1 attr) at 0x4958d0>
   enterelementnode <ll.xist.ns.html.li element object (1 child/no attrs) at 0x5f6130>
      textnode <ll.xist.xsc.Text content='1' at 0x5f4760>
   enterelementnode <ll.xist.ns.html.li element object (1 child/1 attr) at 0x5f6570>
```

This works for the following attributes:

- `entercontent`
- `enterattrs`
- `enterattr`

- `enterelementnode`
- `leaveelementnode`
- `enterattrnode`
- `leaveattrnode`

After the *walk()* generator has been reentered and the modified attribute has been taken into account all those attributes wil be reset to their initial value (i.e. the value that has been passed to *walk()*).

The methods `walknodes()` and `walkpaths()`

In addition to *walk()* two other methods are available: *walknodes()* and *walkpaths()*.

These generators don't produce a cursor object like *walk()* does. *walknodes()* produces the node itself as the following example demonstrates:

Listing 41: Using *walknodes()*

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i) for i in range(3))
>>> for node in e.walknodes():
...   print(repr(node))
...
<ll.xist.ns.html.ul element object (3 children/no attrs) at 0x43fbb0>
<ll.xist.ns.html.li element object (1 child/no attrs) at 0x452750>
<ll.xist.xsc.Text content='0' at 0x5b1670>
<ll.xist.ns.html.li element object (1 child/no attrs) at 0x452830>
<ll.xist.xsc.Text content='1' at 0x5b16e8>
<ll.xist.ns.html.li element object (1 child/no attrs) at 0x5b30d0>
<ll.xist.xsc.Text content='2' at 0x5b1760>
```

walkpaths() produces the path. This is a copy of the path, so it won't be changed once *walkpaths()* is reentered:

Listing 42: Using *walkpaths()*

```
>>> from ll.xist.ns import html
>>> e = html.ul(html.li(i) for i in range(3))
>>> for path in e.walkpaths():
...   print([f"{n.__class__.__module__}.{n.__class__.__qualname__}" for n in path])
...
['ll.xist.ns.html.ul']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li', 'll.xist.xsc.Text']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li', 'll.xist.xsc.Text']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li']
['ll.xist.ns.html.ul', 'll.xist.ns.html.li', 'll.xist.xsc.Text']
```

Filtering the output of the tree traversal

All three tree traversal methods provide an additional argument (`*selectors`) that can be used to filter which nodes/paths are produced. This argument can be specified multiple times (which also means that all other arguments must be passed as keyword arguments).

Passing a node class

In the simplest case you can pass a *Node* subclass to get only instances of that class. The following example prints all the links on the Python home page:

Listing 43: Finding all links on the Python home page

```
from ll.xist import xsc, parse
from ll.xist.ns import xml, html

doc = parse.tree(
   parse.URL("http://www.python.org"),
   parse.Expat(ns=True),
   parse.Node(pool=xsc.Pool(xml, html, chars))
)

for node in doc.walknodes(html.a):
   print(node.attrs.href)
```

This gives the output:

```
http://www.python.org/
http://www.python.org/#left%2Dhand%2Dnavigation
http://www.python.org/#content%2Dbody
http://www.python.org/search
http://www.python.org/about/
http://www.python.org/news/
http://www.python.org/doc/
http://www.python.org/download/
http://www.python.org/getit/
http://www.python.org/community/
...
```

Passing multiple selector arguments

You can also pass multiple classes to search for nodes that are an instance of any of the classes.

The following example will print all header element on the Python home page:

Listing 44: Finding all headers on the Python home page

```
from ll.xist import xsc, parse
from ll.xist.ns import xml, html, chars

doc = parse.tree(
   parse.URL("http://www.python.org"),
   parse.Expat(ns=True),
   parse.Node(pool=xsc.Pool(xml, html, chars))
)

for node in doc.walknodes(html.h1, html.h2, html.h3, html.h4, html.h5, html.h6):
   print(node.string())
```

This will output:

```
<h1 id="logoheader">
  <a accesskey="1" href="http://www.python.org/" id="logolink">
    <img alt="homepage" border="0" id="logo" src="http://www.python.org/images/python-
↪logo.gif" />
  </a>
</h1>
<h4><a href="http://www.python.org/about/help/">Help</a></h4>
<h4><a href="http://pypi.python.org/pypi" title="Repository of Python Software">
↪Package Index</a></h4>
<h4><a href="http://www.python.org/download/releases/2.7.3/">Quick Links (2.7.3)</a></
↪h4>
<h4><a href="http://www.python.org/download/releases/3.3.0/">Quick Links (3.3.0)</a></
↪h4>
<h4><a href="http://www.python.org/community/jobs/" title="Employers and Job Openings
↪">Python Jobs</a></h4>
<h4><a href="http://www.python.org/community/merchandise/" title="T-shirts &amp; more;
↪ a portion goes to the PSF">Python Merchandise</a></h4>
<h4><a href="http://wiki.python.org/moin/" style="margin-top: 1.5em">Python Wiki</a></
↪h4>
<h4><a href="http://blog.python.org/" style="margin-top: 1.5em">Python Insider Blog</
↪a></h4>
<h4><a href="http://wiki.python.org/moin/Python2orPython3" style="margin-top: 1.5em">
↪Python 2 or 3?</a></h4>
<h4><a href="http://www.python.org/psf/donations/" style="color: #D58228; margin-top:␣
↪1.5em">Help Fund Python</a></h4>
<h4><a href="http://wiki.python.org/moin/Languages">Non-English Resources</a></h4>
<h1 class="pageheading">Python Programming Language - Official Website</h1>
<h4>Support the Python Community</h4>
<h4><a href="http://wiki.python.org/moin/Python2orPython3">Python 3</a> Poll</h4>
<h4>NASA uses Python...</h4>
<h4>What they are saying...</h4>
<h4>Using Python For...</h4>
<h2 class="news">Python 3.3.0 released</h2>
<h2 class="news">Third rc for Python 3.3.0 released</h2>
<h2 class="news">Python Software Foundation announces Distinguished Service Award</h2>
<h2 class="news">ConFoo conference in Canada, February 25th - March 13th</h2>
```

(continues on next page)

(continued from previous page)

```
<h2 class="news">Second rc for Python 3.3.0 released</h2>
<h2 class="news">First rc for Python 3.3.0 released</h2>
<h2 class="news">Fifth annual pyArkansas conference to be held</h2>
```

Passing a callable

It is also possible to pass a function to *walk()*. This function will be called for each visited node and gets passed the path to the visited node. If the function returns true, the node will be output.

The following example will find all external links on the Python home page:

Listing 45: Finding external links on the Python home page

```
from ll.xist import xsc, parse
from ll.xist.ns import xml, html, chars

doc = parse.tree(
   parse.URL("http://www.python.org"),
   parse.Expat(ns=True),
   parse.Node(pool=xsc.Pool(xml, html, chars))
)

def isextlink(path):
   return isinstance(path[-1], html.a) and not str(path[-1].attrs.href).startswith(
↪"http://www.python.org")

for node in doc.walknodes(isextlink):
   print(node.attrs.href)
```

This gives the output:

```
http://docs.python.org/devguide/
http://pypi.python.org/pypi
http://docs.python.org/2/
http://docs.python.org/3/
http://wiki.python.org/moin/
http://blog.python.org/
http://wiki.python.org/moin/Python2orPython3
http://wiki.python.org/moin/Languages
http://wiki.python.org/moin/Languages
...
```

`xfind` selectors

The selector arguments for the walk methods get converted into a so called *xfind* selector. *xfind* selectors look somewhat like XPath expressions, but are implemented as pure Python expressions (overloading various Python operators).

Every subclass of *Node* can be used as an xfind selector and combined with other *xfind* selector to create more complex ones. For example searching for links that contain images works as follows:

Listing 46: Searching for *img* inside *a* with an *xfind* expression

```
for path in doc.walkpaths(html.a/html.img):
   print(path[-2].attrs.href, path[-1].attrs.src)
```

The output looks like this:

```
http://www.python.org/ http://www.python.org/images/python-logo.gif
http://www.python.org/#left%2Dhand%2Dnavigation http://www.python.org/images/trans.gif
http://www.python.org/#content%2Dbody http://www.python.org/images/trans.gif
http://www.python.org/psf/donations/ http://www.python.org/images/donate.png
http://wiki.python.org/moin/Languages http://www.python.org/images/worldmap.jpg
http://www.python.org/about/success/usa/ http://www.python.org/images/success/nasa.jpg
```

If the *img* elements are not immediate children of the *a* elements, the *xfind* selector above won't output them. In this case you can use a "decendant selector" instead of a "child selector". To do this simply replace `html.a/html.img` with `html.a//html.img`.

Apart from the `/` and `//` operators you can also use the `|` and `&` operators to combine *xfind* selector:

```
from ll.xist import xsc, parse, xfind
from ll.xist.ns import xml, html

doc = parse.tree(
   parse.URL("http://www.python.org"),
   parse.Expat(ns=True),
   parse.Node(pool=xsc.Pool(xml, html, chars))
)

for node in doc.walknodes((html.a | html.area) & xfind.hasattr("href")):
   print(node.attrs.href)
```

Here's another example that finds all elements that have an `id` attribute:

```
from ll.xist import xsc, parse, xfind
from ll.xist.ns import xml, html, chars

doc = parse.tree(
   parse.URL("http://www.python.org"),
   parse.Expat(ns=True),
   parse.Node(pool=xsc.Pool(xml, html, chars))
)

for node in doc.walknodes(xfind.hasattr("id")):
   print(node.attrs.id)
```

The output looks like this:

```
screen-switcher-stylesheet
logoheader
logolink
logo
skiptonav
skiptocontent
utility-menu
searchbox
searchform
...
```

For more examples refer to the documentation of the *xfind* module.

CSS selectors

It's also possible to use CSS selectors as selectors for the *walk()* method. The module *ll.xist.css* provides a function *selector()* that turns a CSS selector expression into an *xfind* selector:

Listing 47: Using CSS selectors as *xfind* selectors

```
from ll.xist import xsc, parse, css
from ll.xist.ns import xml, html, chars

doc = parse.tree(
   parse.URL("http://www.python.org"),
   parse.Expat(ns=True),
   parse.Node(pool=xsc.Pool(xml, html, chars))
)

for cursor in doc.walk(css.selector("div#menu ul.level-one li > a")):
   print(cursor.node.attrs.href)
```

This outputs all the first level links in the navigation:

```
http://www.python.org/about/
http://www.python.org/news/
http://www.python.org/doc/
http://www.python.org/download/
http://www.python.org/getit/
http://www.python.org/community/
http://www.python.org/psf/
http://docs.python.org/devguide/
```

Most of the CSS 3 selectors are supported.

For more examples see the documentation of the *css* module.

6.1.4 Transforming XIST trees

Apart from the *convert()* method, XIST provides several tools for manipulating an XML tree.

The `withsep()` method

The method `withsep()` can be used to put a separator node between the child nodes of an *Element* or or a *Frag*:

```
>>> from ll.xist import xsc
>>> from ll.xist.ns import html
>>> node = html.div(range(10))
>>> print(node.withsep(", ").string())
<div>0, 1, 2, 3, 4, 5, 6, 7, 8, 9</div>
```

The shuffled() method

The method *shuffled()* returns a shuffled version of the *Element* or *Frag*:

```
>>> from ll.xist import xsc
>>> from ll.xist.ns import html
>>> node = html.div(range(10))
>>> print(node.shuffled().withsep(", ").string())
<div>8, 1, 3, 6, 7, 5, 2, 9, 4, 0</div>
```

The reversed() method

The method *reversed()* returns a reversed version of an element or fragment:

```
>>> from ll.xist import xsc
>>> from ll.xist.ns import html
>>> node = html.div(range(10))
>>> print(node.reversed().withsep(",").string())
<div>9,8,7,6,5,4,3,2,1,0</div>
```

The mapped() method

The method *mapped()* recursively walks the tree and generates a new tree, where all the nodes are mapped through a function. An example: To replace `Python` with `Parrot` in every text node on the Python home page, do the following:

```
from ll.xist import xsc, parse

def p2p(node, converter):
   if isinstance(node, xsc.Text):
      node = node.replace("Python", "Parrot")
      node = node.replace("python", "parrot")
   return node

node = parse.tree(
   parse.URL("http://www.python.org"),
   parse.Tidy(),
   parse.NS(html),
   parse.Node(pool=xsc.Pool(xml, html)),
)
node = node.mapped(p2p)
node.write(open("parrot_index.html", "wb"))
```

The function must either return a new node, in which case this new node will be used instead of the old one, or return the old node to tell *mapped()* that it should recursively continue with the content of the node.

6.1.5 Advanced topics

Converter contexts

Converter contexts can be used to pass around information in recursive calls to the `convert()` and `mapped()` methods. A *Converter* object will be passed in all calls, so this object is the place to store information. However if each element, procinst and entity class decided on its own which attributes names to use, name collisions would be inevitable. To avoid this, the following system is used.

When a class wants to store information in a converter, it has to define a `Context` class (normally derived from the `Context` class of its base class). The constructor must initialize the context object to a initial state. You can get the context object for a certain class by treating the converter as a dictionary with the class (or an instance) as the key like this:

Listing 48: Defining and using a converter context

```
from ll.xist import xsc

class counter(xsc.Element):
   class Context(xsc.Element.Context):
      def __init__(self):
         xsc.Element.Context.__init__(self)
         self.count = 0

   def convert(self, converter):
      context = converter[self]
      node = xsc.Text(context.count)
      context.count += 1
      return node
```

Chaining pools and extending namespaces

When using *ll.xist.xsc.Pool* objects it's possible to do some sort of "namespace subclassing".

Registering a module in a pool not only registers the element, procinst and entity classes in the pool for parsing, but each attribute of the module (as long as it's weak referencable) is available as an attribute of the pool itself:

Listing 49: Pool attributes

```
from ll.xist import xsc
from ll.xist.ns import html

pool = xsc.Pool(html)
print(pool.img)
```

This outputs `<element class ll.xist.ns.html:img at 0x3eed00>`.

It's possible to chain pools together. When an attribute isn't found in the first pool, it will be looked up in a second pool (the so called base pool):

Listing 50: Pool chaining

```
from ll.xist import xsc
from ll.xist.ns import html, svg

hpool = xsc.Pool(html)
spool = xsc.Pool(svg, hpool)
print(spool.img)
```

Here the `hpool` (containing the *html* namespace) will be used when the attribute can't be found in `spool`. So this will again give the output `<element class ll.xist.ns.html:img at 0x3eed00>`.

It's possible to get automatic pool chaining. If a module has an attribute `__bases__` (which must be a sequence of modules), they will be wrapped in a pool automatically and used as the base pools for the pool created for the first module. This makes it possible to "overwrite" element classes in existing namespaces. For example to replace the *a* class in *ll.xist.ns.html*, put the following into a module `html2`:

Listing 51: Automatic pool chaining (`html2.py`)

```
from ll.xist.ns import html

__bases__ = [html]

class a(html.a):
   xmlns = html.xmlns

   def convert(self, converter):
      node = html.a(self.content, self.attrs, target="_top")
      return node.convert(converter)
```

Now you can use the module in a pool:

Listing 52: Using a pool chain

```
from ll.xist import xsc
import html2

pool = xsc.Pool(html2)
print(pool.a, pool.b)
```

This outputs:

```
<element class html2:a at 0x113ec40> <element class ll.xist.ns.html:b at 0x1101fe0>
```

Note that such a chained pool can of course be used when parsing XML. The parser will recursively search for the first class that has the appropriate name when instantiating the tree nodes.

Conversion targets

The `converter` argument passed to the `convert()` method has an attribute `target` which is a module or pool and specifies the target namespace to which `self` should be converted.

You can check which conversion is wanted by checking e.g. the `xmlns` attribute. Once this is determined you can use element classes from the target to create the required XML object tree. This makes it possible to customize the conversion by passing a chained pool to the `convert()` method that extends an existing namespace.

The following example shows how an element be converted to two different targets:

Listing 53: Using conversion targets

```
from ll.xist import xsc
from ll.xist.ns import html, fo

class bold(xsc.Element):
   def convert(self, converter):
      if converter.target.xmlns == html.xmlns:
         node = converter.target.b(self.content)
      elif converter.target.xmlns == fo.xmlns:
         node = converter.target.inline(self.content, font_weight="bold")
```

(continues on next page)

(continued from previous page)

```
    else:
        raise TypeError(f"unsupported conversion target {converter.target!r}")
    return node.convert(converter)
```

The default target for conversion is *ll.xist.ns.html*. Other targets can be specified via the `target` argument in the `Converter` constructor or the `conv()` method:

```
>>> from ll.xist.ns import html, fo
>>> import foo # This is the code from above
>>> print(foo.bold("foo").conv().string())
<b>foo</b>
>>> print(foo.bold("foo").conv(target=html).string())
<b>foo</b>
>>> print(foo.bold("foo").conv(target=fo).string())
<inline font-weight="bold">foo</inline>
```

Validation and content models

When generating HTML you might want to make sure that your generated code doesn't contain any illegal element nesting (i.e. something bad like foo in HTML). The module *ll.xist.ns.html* does this automatically:

```
>>> from ll.xist.ns import html
>>> node = html.p(html.p(u"foo"))
>>> print(node.string())
/Users/walter/checkouts/LivingLogic.Python.xist/src/ll/xist/sims.py:222: \
WrongElementWarning: element <ll.xist.ns.html.p element object (1 child/no attrs) at␣
↪0x270b30> \
may not contain element <ll.xist.ns.html.p element object (1 child/no attrs) at␣
↪0x69850>
  warnings.warn(WrongElementWarning(node, child, self.elements))
<p><p>foo</p></p>
```

For your own elements you can specify the content model too. This is done by setting the class attribute `model` inside the element class. `model` must be an object that provides a `checkvalid()` method. This method will be called during parsing or publishing with the element as an argument. When invalid content is detected, the Python warning framework should be used to issue a warning.

The module *ll.xist.sims* contains several classes that provide simple validation methods:

- *ll.xist.sims.Empty* can be used to ensure that the element doesn't have any content (like `br` and `img` in HTML).
- *ll.xist.sims.Any* does allow any content.
- *ll.xist.sims.NoElements* will warn about elements from the same namespace (elements from other namespaces will be OK).
- *ll.xist.sims.NoElementsOrText* will warn about elements from the same namespace and non-whitespace text content.
- *ll.xist.sims.Elements* will only allow the elements specified in the constructor.
- *ll.xist.sims.ElementsOrText* will only allow the elements specified in the constructor and text.

None of these classes will check the number of child elements or their order.

For more info see the *ll.xist.sims* module.

6.1.6 Miscellaneous features

URLs

For URL handling XIST uses the module *ll.url*. Refer to its documentation for the basic functionality (especially regarding the methods `__div__()` and and *ll.url.URL.relative()*.

When XIST parses an XML resource it uses a so called "base" URL. This base URL can be passed to all parsing functions. If it isn't specified it defaults to the URL of the resource being parsed. This base URL will be prepended to all URLs that are read during parsing:

```
>>> from ll.xist import parse
>>> from ll.xist.ns import html
>>> node = parse.parsestring('<img src="eggs.png"/>', base="root:spam/index.html")
>>> print node.string()
<img src="root:spam/eggs.png" />
```

For publishing a base URL can be specified too. URLs will be published relative to this base URL with the exception of relative URLs in the tree. This means:

- When you have a relative URL (e.g. `#top`) generated by a `convert()` call, this URL will stay the same when publishing.
- Base URLs for parsing should never be relative: Relative base URLs will be prepended to all relative URLs in the file, but this will not be reverted for publishing. In most cases the base URL should be a `root` URL when you parse local files.
- When you parse remote web pages you can either omit the `base` argument, so it will default to the URL being parsing, so that links, images, etc. on the page will still point back to their original location, or you might want to use the empty URL `URL()` as the base, so you'll get all URLs in the page as they are.
- When XIST is used as a compiler for static pages, you're going to read source XML files, do a conversion and write the result to a new target file. In this case you should probably use the URL of the target file for both parsing and publishing. Let's assume we have an URL `#top` in the source file. When we use the "real" file names for parsing and publishing like this:

  ```
  node = parse.parsefile("spam.htmlxsc", base="root:spam.htmlxsc")
  node = node.conv()
  node.write(open("spam.html", "wb"), base="root:spam.html")
  ```

  the following will happen: The URL `#top` will be parsed as `root:spam.htmlxsc#top`. After conversion this will be written to `spam.html` relative to the URL `root:spam.html`, which results in `spam.html#top`, which works, but is not what you want.

  When you use `root:spam.html` both for parsing and publishing, `#top` will be written to the target file as expected.

Pretty printing XML

The method `pretty()` can be used for pretty printing XML. It returns a new version of the node, with additional white space between the elements:

```
from ll.xist.ns import html
node = html.html(
   html.head(
      html.title("foo"),
   ),
   html.body(
      html.div(
         html.h1("The ", html.em("foo"), " page!"),
```

(continues on next page)

(continued from previous page)

```
            html.p("Welcome to the ", html.em("foo"), " page."),
        ),
    ),
)

print node.pretty().bytes()
```

This will print:

```
<html>
   <head>
      <title>foo</title>
   </head>
   <body>
      <div>
         <h1>The <em>foo</em> page!</h1>
         <p>Welcome to the <em>foo</em> page.</p>
      </div>
   </body>
</html>
```

Element content will only be modified if it doesn't contain `Text` nodes, so mixed content will not be touched.

Automatic generation of image size attributes

The module `ll.xist.ns.htmlspecials` contains an element `autoimg` that extends `img`. When converted to HTML via the `convert()` method the size of the image will be determined and the `height` and `width` attributes will be set accordingly (if those attributes are not set already).

Embedding Python code

It's possible to embed Python code into XIST XML files. For this XIST supports two new processing instructions: `pyexec` and :class:~`ll.xist.ns.code.pyeval` (in the module `ll.xist.ns.code`). The content of `pyexec` will be executed when the processing instruction node is converted.

The result of a call to `convert()` for a :class:~`ll.xist.ns.code.pyeval` processing instruction is whatever the Python code in the content returns. The processing instruction content is treated as the body of a function, so you can put multiple return statements there. The converter is available as the parameter `converter` inside the processing instruction. For example, consider the following XML file:

```
<?pyexec
   # sum
   def gauss(top=100):
      sum = 0
      for i in range(top+1):
         sum += i
      return sum
?>
<b><?pyeval return gauss()?></b>
```

Parsing this file and calling `convert()` results in the following:

```
<b>5050</b>
```

6.1.7 xsc – XIST core classes

This module contains all the central XML tree classes, exception and warning classes and a few helper classes and functions.

`ll.xist.xsc.tonode(value)`

Convert `value` to an XIST *Node*.

If `value` is a tuple or list, it will be (recursively) converted to a *Frag*. Integers, strings, etc. will be converted to a *Text*. If `value` is a *Node* already, it will be returned unchanged. In the case of `None` the XIST Null (`ll.xist.xsc.Null`) will be returned. If `value` is iterable, a *Frag* will be generated from the items. Anything else will raise an *IllegalObjectError* exception.

class `ll.xist.xsc.build`

Bases: `object`

A *build* object can be used as a context handler to create a new XIST tree:

```
with xsc.build():
   with html.ul() as e:
      +html.li("gurk")
      +html.li("hurz")
```

class `ll.xist.xsc.addattr`

Bases: `object`

An *addattr* object can be used as a context handler to modify an attribute of an element:

```
with xsc.build():
   with html.div() as e:
      with xsc.addattr("align"):
         +xsc.Text("right")
```

__init__(*attrname*)

Create an *addattr* object for adding to the attribute named `attrname` (which can be the Python name of an attribute or an attribute class).

`ll.xist.xsc.add(*args, **kwargs)`

add() appends items in `args` and sets attributes in `kwargs` in the currently active node in the `with` stack.

class `ll.xist.xsc.Context`

Bases: `object`

This is an empty class that can be used by the `convert()` method to hold element or namespace specific data during the `convert()` call. The method *Converter.__getitem__()* will return a unique instance of this class.

exception `ll.xist.xsc.Error`

Bases: `Exception`

Base class for all XIST exceptions

exception `ll.xist.xsc.Warning`

Bases: `UserWarning`

Base class for all warning exceptions (i.e. those that won't result in a program termination.)

exception `ll.xist.xsc.IllegalAttrValueWarning`

Bases: *Warning*

Warning that is issued when an attribute has an illegal value.

exception ll.xist.xsc.**RequiredAttrMissingWarning**

Bases: *Warning*

Warning that is issued when a required attribute is missing.

exception ll.xist.xsc.**UndeclaredAttrWarning**

Bases: *Warning*

Warning that is issued when a local attribute is not declared.

exception ll.xist.xsc.**UndeclaredNodeWarning**

Bases: *Warning*

Warning that is issued when a node (i.e. element, entity or processing instruction) is not declared.

exception ll.xist.xsc.**IllegalPrefixError**

Bases: *Error*, LookupError

Exception that is raised when a namespace prefix is undefined.

exception ll.xist.xsc.**FileNotFoundWarning**

Bases: *Warning*

Warning that is issued when a file can't be found.

exception ll.xist.xsc.**IllegalObjectError**

Bases: *Error*, TypeError

Exception that is raised when an XIST constructor gets passed an unconvertable object.

exception ll.xist.xsc.**IllegalCommentContentWarning**

Bases: *Warning*

Warning that is issued when there is an illegal comment, i.e. one containing `--` or ending in `-`. (This can only happen when the comment was created by code, not when parsed from an XML file.)

exception ll.xist.xsc.**IllegalProcInstFormatError**

Bases: *Error*

Exception that is raised when there is an illegal processing instruction, i.e. one containing `?>`. (This can only happen when the processing instruction was created by code, not when parsed from an XML file.)

class ll.xist.xsc.**Converter**

Bases: object

An instance of this class is passed around in calls to the *convert()* method. A `Converter` object can be used when some element needs to keep state across a nested *convert()* call. A typical example are nested chapter/subchapter elements with automatic numbering. For an example see the element *ll.xist.ns.doc.section*.

__init__(*node=None*, *root=None*, *mode=None*, *stage=None*, *target=None*, *lang=None*, *makeaction=None*, *makeproject=None*)

Create a *Converter*. Arguments are used to initialize the *Converter* properties of the same name.

__getitem__(*key*)

Return a context object for `key`. Two variants are supported:

- `key` may be a string, in which case it should be a hierarchical dot-separated name similar to Java package names (e.g. `"org.example.project.handler"`). This helps avoid name collisions. Context objects of this type must be explicitly created via `__setitem__()`.
- `key` may be a *ll.xist.xsc.Node* instance or subclass. Each of these classes that defines its own *Context* class gets a unique instance of this class. This instance will be created on the first access and the element can store information there that needs to be available across calls to *convert()*.

class ll.xist.xsc.**Publisher**

Bases: object

A *Publisher* object is used for serializing an XIST tree into a byte sequence.

__init__(*encoding=None*, *xhtml=1*, *validate=False*, *prefixes={}*, *prefixdefault=False*, *hidexmlns=()*, *showxmlns=()*)

Create a publisher. Arguments have the following meaning:

encoding
[string or None]
Specifies the encoding to be used for the byte sequence. If None is used the encoding in the XML declaration will be used. If there is no XML declaration, UTF-8 will be used.

xhtml
[int]
With the parameter xhtml you can specify if you want HTML output:

HTML (xhtml==0)
Elements with a empty content model will be published as <foo>.

HTML browser compatible XML (xhtml==1)
Elements with an empty content model will be published as <foo /> and others that just happen to be empty as <foo></foo>. This is the default.

Pure XML (xhtml==2)
All empty elements will be published as <foo/>.

validate
[bool]
Specifies whether validation should be done before publishing.

prefixes
[mapping]
A dictionary that specifies which namespace prefixes should be used for publishing. Keys in the dictionary are either namespace names or objects that have an xmlns attribute which is the namespace name. Values can be:

False
Treat elements in this namespace as if they are not in any namespace (if global attributes from this namespace are encountered, a non-empty prefix will be used nonetheless).

None
Treat the namespace as the default namespaces (i.e. use unprefixed element names). Global attributes will again result in a non-empty prefix.

True
The publisher uses a unique non-empty prefix for this namespace.

A string
Use this prefix for the namespace.

prefixdefault
[string or None]
If an element or attribute is encountered whose namespace name is not in prefixes prefixdefault is used as the fallback.

hidexmlns
[list or set]
hidexmlns can be a list or set that contains namespace names for which no xmlns attributes should be published. (This can be used to hide the namespace declarations for e.g. Java taglibs.)

showxmlns
[list or set]
showxmlns can be a list or set that contains namespace names for which xmlns attributes *will* be published, even if there are no elements from this namespace in the tree.

encode(*text*)

Encode `text` with the encoding and error handling currently active and return the resulting byte string.

encodetext(*text*)

Encode `test` as text data. `text` must be a `str` object. The publisher will apply the configured encoding, error handling and the current text filter (which escapes characters that can't appear in text data (like < etc.)) and returns the resulting `str` object.

pushtextfilter(*filter*)

Pushes a new text filter function ontp the text filter stack. This function is responsible for escaping characters that can't appear in text data (like <)). This is used to switch on escaping of " inside attribute values.

poptextfilter()

Pops the current text filter function from the stack.

pusherrors(*errors*)

Pushes a new error handling scheme onto the error handling stack.

poperrors()

Pop the current error handling scheme from the error handling stack.

getencoding()

Return the encoding currently in effect.

getnamespaceprefix(*xmlns*)

Return (and register) a namespace prefix for the namespace name `xmlns`. This honors the namespace configuration from `self.prefixes` and `self.prefixdefault`. Furthermore the same prefix will be returned from now on (except when the empty prefix becomes invalid once global attributes are encountered)

getobjectprefix(*obj*)

Get and register a namespace prefix for the namespace `obj` lives in (specified by the `xmlns` attribute of `obj`). Similar to *getnamespaceprefix()* this honors the namespace configuration from `self.prefixes` and `self.prefixdefault` (except when a global attribute requires a non-empty prefix).

iterbytes(*node*, *base=None*, *allowschemerelurls=False*)

Output the node `node`. This method is a generator that will yield the resulting XML byte sequence in fragments.

URLs in `node` will be published relative to the base URL `base`.

Setting `allowschemerelurls` to true allow schema-relative URLs (e.g. `//www.example.org/about.html`).

bytes(*node*, *base=None*, *allowschemerelurls=False*)

Return a *bytes* object in XML format for the XIST node `node`.

iterstring(*node*, *base=None*, *allowschemerelurls=False*)

A generator that will produce a serialized string of `node`.

string(*node*, *base=None*, *allowschemerelurls=False*)

Return a string for `node`.

write(*stream*, *node*, *base=None*, *allowschemerelurls=False*)

Write `node` to the file-like object `stream` (which must provide a `write()` method).

class `ll.xist.xsc.`**Cursor**

Bases: `object`

A *Cursor* object is used by the *walk()* method during tree traversal. It contains information about the state of the traversal and can be used to influence which parts of the tree are traversed and in which order.

Information about the state of the traversal is provided in the following attributes:

root

The node where traversal has been started (i.e. the object for which the *walk()* method has been called).

node

The current node being traversed.

path

A list of nodes that contains the path through the tree from the root to the current node (i.e. `path[0] is root` and `path[-1] is node`).

index

A path of indices (e.g. `[0, 1]` if the current node is the second child of the first child of the root). Inside attributes the index path will contain the name of the attribute (or a (attribute name, namespace name) tuple inside a global attribute).

event

A string that specifies which event is currently handled. Possible values are: `"enterelementnode"`, `"leaveelementnode"`, `"enterattrnode"`, `"leaveattrnode"`, `"textnode"`, `"commentnode"`, `"doctypenode"` `"procinstnode"`, `"entitynode"` and `"nullnode"`

The following attributes specify which part of the tree should be traversed:

entercontent

Should the content of an element be entered?

enterattrs

Should the attributes of an element be entered? (Note that the attributes will always be entered before the content.)

enterattr

Should the content of the attributes of an element be entered? (This is only relevant if *enterattrs* is true.)

enterelementnode

Should the generator yield a `"enterelementnode"` event (i.e. return before entering the content or attributes of an element)?

leaveelementnode

Should the generator yield an `"leaveelementnode"` event (i.e. return after entering the content or attributes of an element)?

enterattrnode

Should the generator yield a `"enterattrnode"` event (i.e. return before entering the content of an attribute)? This is only relevant if *enterattrs* is true.

leaveattrnode

Should the generator yield an `"leaveattrnode"` event (i.e. return after entering the content of an attribute)? This is only relevant if *enterattrs* is true. Furthermore if *enterattr* is false, the behaviour is essentially the same as for *enterattrnode*.

Note that if any of these attributes is changed by the code consuming the generator, this new value will be used for the next traversal step once the generator is resumed and will be reset to its initial value (specified in the constructor) afterwards.

__init__(*node*, *entercontent=True*, *enterattrs=False*, *enterattr=False*, *enterelementnode=True*, *leaveelementnode=False*, *enterattrnode=True*, *leaveattrnode=False*)

Create a new *Cursor* object for a tree traversal rooted at the node `node`.

The arguments `entercontent`, `enterattrs`, `enterattr`, `enterelementnode`, `leaveelementnode`, `enterattrnode` and `leaveattrnode` are used as the initial values for the attributes of the same name. (see the class docstring for info about their use).

restore()

Restore the attributes *entercontent*, *enterattrs*, *enterattr*, *enterelementnode*, *leaveelementnode*, *enterattrnode* and *leaveattrnode* to their initial value.

class ll.xist.xsc.**Node**

Bases: object

Base class for nodes in the document tree. Derived classes may overwrite *convert()* or *publish()*.

class Context

Bases: object

This is an empty class that can be used by the convert() method to hold element or namespace specific data during the convert() call. The method *Converter.__getitem__()* will return a unique instance of this class.

__truediv__(*other*)

Return a *ChildCombinator* with self as the left hand selector.

__floordiv__(*other*)

Return a *DescendantCombinator* with self as the left hand selector.

__mul__(*other*)

If other is an int, return a *Frag* with other times the node as an entry. Note that the node will not be copied, i.e. this is a "shallow __mul__()".

If other is not an int, treat this a CSS combinator that creates an *AdjacentSiblingCombinator* with self as the left hand selector.

__rmul__(*other*)

Return a *Frag* with other times the node as an entry.

__pow__(*other*)

Return a *GeneralSiblingCombinator* with self as the left hand selector.

__and__(*other*)

Return an *AndCombinator* with self as the left hand selector.

__or__(*other*)

Return an *OrCombinator* with self as the left hand selector.

clone()

Return a clone of self. Compared to *deepcopy()* clone() will create multiple instances of objects that can be found in the tree more than once. clone() can't clone trees that contain cycles.

copy()

Return a shallow copy of self.

deepcopy()

Return a deep copy of self.

present(*presenter*)

present() is used as a central dispatch method for the presenter classes. Normally it is not called by the user, but internally by the presenter. The user should use the appropriate presenter class directly.

conv(*converter=None*, *root=None*, *mode=None*, *stage=None*, *target=None*, *lang=None*, *function=None*, *makeaction=None*, *makeproject=None*)

Convenience method for calling *convert()*.

conv() will automatically set converter.node to self to remember the "document root node" for which conv() has been called. This means that you should not call conv() in any of the recursive calls, as you would loose this information. Call *convert()* directly instead.

convert(*converter*)

Implementation of the conversion method. When you define your own element classes you have to overwrite this method and implement the desired conversion.

This method must return an instance of *Node*. It may *not* change `self`.

__str__()

Return the character content of `self` as a string. This means that comments and processing instructions will be filtered out. For elements you'll get the element content.

`__str__()` can be used everywhere where a plain string representation of the node is required.

For example:

```
>>> from ll.xist.ns import html
>>> e = html.html(
...    html.head(
...       html.title("The page")
...    ),
...    html.body(
...       html.h1("The header"),
...       html.p("The content", class_="content")
...    )
... )
>>> print(e)
The pageThe headerThe content
```

__int__()

Convert the character content of `self` to an `int`.

asFloat(*decimal='.', ignore=''*)

Convert the character content of `self` to an `float`. `decimal` specifies which decimal separator is used in the value (e.g. `"."` (the default) or `","`). `ignore` specifies which characters will be ignored.

__float__()

Convert the character content of `self` to an `float`.

__complex__()

Convert the character content of `self` to an `complex`.

parsed(*parser, event*)

This method will be called by the parser `parser` once after `self` is created by the parser (This is used e.g. by *URLAttr* to incorporate the base URL into the attribute).

`event` is the parser event that initiated the call.

validate(*recursive=True, path=None*)

This method will be called when parsing or publishing to check whether `self` is valid.

If `self` is found to be invalid a warning should be issued through the Python warning framework.

publish(*publisher*)

Generate unicode strings for the node. `publisher` must be an instance of *Publisher*.

The encoding and xhtml specification are taken from the `publisher`.

iterbytes(*base=None, allowschemerelurls=False, publisher=None, **publishargs*)

A generator that will produce this node as a serialized byte string. (i.e. it will output what the method *bytes()* outputs, but incremetally).

For the possible parameters see the *Publisher* constructor and its *iterbytes()* method.

bytes(*base=None*, *allowschemerelurls=False*, *publisher=None*, ***publishargs*)

Return `self` as a serialized bytes object.

For the possible parameters see the *Publisher* constructor.

For example:

```
>>> from ll.xist.ns import html
>>> e = html.div(
...    html.h1("The header"),
...    html.p("The content", class_="content")
... )
>>> print(e.bytes())
b'<div><h1>The header</h1><p class="content">The content</p></div>'
```

iterstring(*base=None*, *allowschemerelurls=False*, *publisher=None*, ***publishargs*)

A generator that will produce a serialized string of `self` (i.e. it will output what the method *string()* outputs, but incremetally).

For the possible parameters see the *Publisher* constructor.

string(*base=None*, *allowschemerelurls=False*, *publisher=None*, ***publishargs*)

Return a serialized (unicode) string for `self`.

For the possible parameters see the *Publisher* constructor.

For example:

```
>>> from ll.xist.ns import html
>>> e = html.div(
...    html.h1("The header"),
...    html.p("The content", class_="content")
... )
>>> print(e.string())
<div><h1>The header</h1><p class="content">The content</p></div>
```

write(*stream*, *base=None*, *allowschemerelurls=False*, *publisher=None*, ***publishargs*)

Write `self` to the file-like object `stream` (which must provide a `write()` method).

For the rest of the parameters see the *Publisher* constructor.

walk(**selectors*, *entercontent=True*, *enterattrs=False*, *enterattr=False*, *enterelementnode=True*, *leaveelementnode=False*, *enterattrnode=True*, *leaveattrnode=False*)

Return an iterator for traversing the tree rooted at `self`.

Each item produced by the iterator is a *Cursor* object. It contains information about the state of the traversal and can be used to influence which parts of the tree are traversed and in which order.

`selectors` is used for filtering which nodes to return from the iterator. The arguments `entercontent`, `enterattrs`, `enterattr`, `enterelementnode`, `leaveelementnode`, `enterattrnode` and `leaveattrnode` specify how the tree should be traversed. For more information see the *Cursor* class.

Note that the *Cursor* object is reused by `walk()`, so you can't rely on any attributes remaining the same across calls to `next()`.

The following example shows how to extract the text of an HTML `label` element for an input element with a specified HTML id:

```
from ll import misc
from ll.xist import xsc, xfind
from ll.xist.ns import html
```

(continues on next page)

(continued from previous page)

```
def label(doc, id):
   label = misc.first(doc.walk(xfind.attrhasvalue("for", id)), None)
   if label is None:
      return None
   texts = []
   for c in label.node.walk(html.textarea, xsc.Text):
      if isinstance(c.node, html.textarea):
         c.entercontent = False
      else:
         texts.append(str(c.node))
   return " ".join("".join(texts).split()).strip()

doc = html.div(
   html.p(
      html.label(
         "Input your text here: ",
         html.textarea("Default value", rows=20, cols=80, id="foo"),
         " (just a test)",
         for_="foo",
      )
   )
)

print(repr(label(doc, "foo")))
```

This will output:

```
'Input your text here: (just a test)'
```

walknodes(**selectors*, *entercontent=True*, *enterattrs=False*, *enterattr=False*, *enterelementnode=True*, *leaveelementnode=False*, *enterattrnode=True*, *leaveattrnode=False*)

Return an iterator for traversing the tree. The arguments have the same meaning as those for *walk()*. The items produced by the iterator are the nodes themselves.

walkpaths(**selectors*, *entercontent=True*, *enterattrs=False*, *enterattr=False*, *enterelementnode=True*, *leaveelementnode=False*, *enterattrnode=True*, *leaveattrnode=False*)

Return an iterator for traversing the tree. The arguments have the same meaning as those for *walk()*. The items produced by the iterator are copies of the path.

compacted()

Return a version of `self`, where textnodes or character references that contain only linefeeds are removed, i.e. potentially useless whitespace is removed.

mapped(*function*, *converter=None*, ***converterargs*)

Return the node mapped through the function `function`. This call works recursively (for *Frag* and *Element*).

When you want an unmodified node you simply can return `self`. `mapped()` will make a copy of it and fill the content recursively. Note that element attributes will not be mapped. When you return a different node from `function` this node will be incorporated into the result as-is.

normalized()

Return a normalized version of `self`, which means that consecutive *Text* nodes are merged.

pretty(*level=0*, *indent='\t'*)

Return a prettyfied version of `self`, i.e. one with properly nested and indented tags (as far as possible). If an element has mixed content (i.e. *Text* and non-*Text* nodes) the content will be returned as is.

Note that whitespace will prevent pretty printing too, so you might want to call *normalized()* and *compacted()* before calling *pretty()* to remove whitespace.

class ll.xist.xsc.**CharacterData**

Bases: *Node*

Base class for XML character data (*Text*, *ProcInst*, *Comment* and *DocType*).

(Provides nearly the same functionality as `UserString`, but omits a few methods.)

class ll.xist.xsc.**Text**

Bases: *CharacterData*

A text node. The characters <, >, & (and " inside attributes) will be "escaped" with the appropriate character entities when this node is published.

class ll.xist.xsc.**Frag**

Bases: *Node*, `list`

A fragment contains a list of nodes and can be used for dynamically constructing content. The attribute `content` of an *Element* is a *Frag*.

clear()

Make `self` empty.

__copy__()

helper for the `copy` module.

__deepcopy__(*memo=None*)

helper for the `copy` module.

__getitem__(*index*)

Return the `index`'th node of the content of the fragment. If `index` is a list *__getitem__()* will work recursively. If `index` is an empty list, `self` will be returned. `__getitem__()` also supports selectors (i.e. `xfind.Selector` objects).

__setitem__(*index*, *value*)

Allows you to replace the `index`'th content node of the fragment with the new value `value` (which will be converted to a node). If `index` is a list *__setitem__()* will be applied to the innermost index after traversing the rest of `index` recursively. If `index` is an empty list, an exception will be raised. `__setitem__()` also supports selectors (i.e. `xfind.Selector` objects).

__delitem__(*index*)

Remove the `index`'th content node from the fragment. If `index` is a list, the innermost index will be deleted, after traversing the rest of `index` recursively. If `index` is an empty list, an exception will be raised. Anything except `list`, `int` and `slice` objects will be turned into a selector (i.e. an `xfind.Selector` objects) and any child node matching this selector will be deleted from `self`.

__mul__(*factor*)

Return a *Frag* with `factor` times the content of `self`. Note that no copies of the content will be generated, so this is a "shallow `__mul__()`".

__rmul__(*factor*)

Return a *Frag* with `factor` times the content of `self`. Note that no copies of the content will be generated, so this is a "shallow `__mul__()`".

append(**others*)

Append every item in `others` to `self`.

extend(*items*)

Append all items from the sequence `items` to `self`.

insert(*index*, **others*)

Insert all items in `others` at the position `index`. (this is the same as `self[index:index] = others`)

withsep(*separator*, *clone=False*)

Return a version of `self` with a separator node between the nodes of `self`.

if `clone` is false, one node will be inserted several times, if `clone` is true, clones of this node will be used.

reversed()

Return a reversed version of the `self`.

filtered(*function*)

Return a filtered version of the `self`, i.e. a copy of `self`, where only content nodes for which `function` returns true will be copied.

shuffled()

Return a shuffled version of `self`, i.e. a copy of `self` where the content nodes are randomly reshuffled.

class ll.xist.xsc.**Comment**

Bases: *CharacterData*

An XML comment.

class ll.xist.xsc.**DocType**

Bases: *CharacterData*

An XML document type declaration.

class ll.xist.xsc.**ProcInst**

Bases: *CharacterData*

Base class for processing instructions.

Processing instructions for specific targets must be implemented as subclasses of *ProcInst*.

class ll.xist.xsc.**Attr**

Bases: *Frag*

Base class of all attribute classes.

The content of an attribute may be any other XIST node. This is different from a normal DOM, where only text and character references are allowed. The reason for this is to allow dynamic content (implemented as elements or processing instructions) to be put into attributes.

Of course, this dynamic content when finally converted to HTML should normally result in a fragment consisting only of text and character references. But note that it is allowed to have elements and processing instructions inside of attributes even when publishing. Processing instructions will be published as is and for elements their content will be published:

```
>>> from ll.xist.ns import html, php
>>> node = html.img(
...    src=php.php("echo 'eggs.gif'"),
...    alt=html.abbr(
...       "EGGS",
...       title="Extensible Graphics Generation System",
...       lang="en"
...    )
... )
>>> print(node.string())
<img alt="EGGS" src="<?php echo 'eggs.gif'?>" />
```

isfancy()

Return whether `self` contains nodes other than *Text*.

validate(*recursive=True*, *path=None*)

Check whether `self` has an allowed value, i.e. one that is specified in the class attribute `values`. If the value is not allowed a warning will be issued through the Python warning framework.

If `self` is "fancy" (i.e. contains non-*Text* nodes), no check will be done.

class ll.xist.xsc.**TextAttr**

Bases: *Attr*

Attribute class that is used for normal text attributes.

class ll.xist.xsc.**IDAttr**

Bases: *Attr*

Attribute used for ids.

class ll.xist.xsc.**NumberAttr**

Bases: *Attr*

Attribute class that is used for when the attribute value may be any kind of number.

class ll.xist.xsc.**IntAttr**

Bases: *NumberAttr*

Attribute class that is used when the attribute value may be an integer.

class ll.xist.xsc.**FloatAttr**

Bases: *NumberAttr*

Attribute class that is used when the attribute value may be a floating point value.

class ll.xist.xsc.**BoolAttr**

Bases: *Attr*

Attribute class that is used for boolean attributes. When publishing the value will always be the attribute name, regardless of the real value.

class ll.xist.xsc.**ColorAttr**

Bases: *Attr*

Attribute class that is used for a color attributes.

class ll.xist.xsc.**StyleAttr**

Bases: *Attr*

Attribute class that is used for CSS style attributes.

replaceurls(*replacer*)

Replace each URL in the style. Each URL will be passed to the callable `replacer` and replaced with the returned value.

urls(*base=None*)

Return a list of all the URLs (as `URL` objects) found in the style attribute.

class ll.xist.xsc.**URLAttr**

Bases: *Attr*

Attribute class that is used for URLs. See the module *ll.url* for more information about URL handling.

asURL()

Return `self` as a `URL` object (note that non-*Text* content will be filtered out).

forInput(*root=None*)

return a `URL` pointing to the real location of the referenced resource. `root` must be the root URL relative to which `self` will be interpreted and usually comes from the `root` attribute of the `converter` argument in `convert()`.

imagesize(*root=None*)

Return the size of an image as a tuple.

contentlength(*root=None*)

Return the size of a file in bytes.

lastmodified(*root=None*)

returns the timestamp for the last modification to the file

openread(*root=None*)

Return a `Resource` for reading from the URL.

openwrite(*root=None*)

Return a `Resource` for writing to the URL.

class ll.xist.xsc.**Attrs**

Bases: *Node*, `dict`

An attribute map. Predefined attributes can be declared through nested subclasses of *Attr*.

__getitem__(*name*)

Return the attribute with the name `name`. `name` can be one of the following types:

A string

`name` will be treated as the XML name of a local attribute.

A two-item tuple

The first item is treated as the XML attribute name and the second item as the namespace name. If the namespace name is `None` this refers to a local attributes, otherwise to a global attribute.

An *Attr* subclass

__setitem__(*name*, *value*)

Set the attribute with the XML `name` to the value `value`. `name` may be a string or an attribute class or instance. The newly set attribute object will be returned.

__delitem__(*name*)

get(*name*, *default=None*)

Works like the dictionary method `get()`, it returns the attribute with the XML name `name`, or `default` if `self` has no such attribute. `name` may also be an attribute class (either from `self.Attrs` or a global attribute).

setdefault(*name*, *default*)

Works like the dictionary method `setdefault()`, it returns the attribute with the Python name `name`. If `self` has no such attribute, it will be set to `default` and `default` will be returned as the new attribute value.

update(**args*, ***kwargs*)

Copies attributes over from all mappings in `args` and from `kwargs`. Keywords are treated as the Python names of attributes.

classmethod declaredattrs()

Return an iterator over all declared attribute classes.

filtered(*function*)

Return a filtered version of `self`.

withnames(**names*)

Return a copy of `self` where only the attributes with XML names in `names` are kept, all others are removed.

withoutnames(**names*)

Return a copy of `self` where all the attributes with XML names in `names` are removed.

class ll.xist.xsc.**Element**

Bases: *Node*

This class represents XML/XIST elements. All elements implemented by the user must be derived from this class.

Elements support the following class variables:

model

[object with `validate()` method]
This is an object that is used for validating the content of the element. See the module *ll.xist.sims* for more info. If `model` is `None` validation will be skipped, otherwise it will be performed when parsing or publishing.

Attrs

[*Element.Attrs* subclass]
This is a class derived from *Element.Attrs* and must define all attributes as classes nested inside this *Attrs* class.

xmlns

[string]
This is the name of the namespace this element belong to.

register

[bool]
If `register` is false the element will never be registered in a *Pool*. The default is `True`.

xmlname

[string]
If the class name has to be different from the XML name (e.g. because the XML name is not a valid Python identifier) `xmlname` can be used to specify the real XML name. Otherwise the XML name will be the Python name.

class Attrs

Bases: *Node*, `dict`

An attribute map. Predefined attributes can be declared through nested subclasses of *Attr*.

__delitem__(*name*)

__getitem__(*name*)

Return the attribute with the name `name`. `name` can be one of the following types:

A string
`name` will be treated as the XML name of a local attribute.

A two-item tuple
The first item is treated as the XML attribute name and the second item as the namespace name. If the namespace name is `None` this refers to a local attributes, otherwise to a global attribute.

An *Attr* subclass

__setitem__(*name*, *value*)

Set the attribute with the XML `name` to the value `value`. `name` may be a string or an attribute class or instance. The newly set attribute object will be returned.

classmethod declaredattrs()

Return an iterator over all declared attribute classes.

filtered(*function*)

Return a filtered version of `self`.

get(*name*, *default=None*)

Works like the dictionary method `get()`, it returns the attribute with the XML name `name`, or `default` if `self` has no such attribute. `name` may also be an attribute class (either from `self.Attrs` or a global attribute).

setdefault(*name*, *default*)

Works like the dictionary method `setdefault()`, it returns the attribute with the Python name `name`. If `self` has no such attribute, it will be set to `default` and `default` will be returned as the new attribute value.

update(**args*, ***kwargs*)

Copies attributes over from all mappings in `args` and from `kwargs`. Keywords are treated as the Python names of attributes.

withnames(**names*)

Return a copy of `self` where only the attributes with XML names in `names` are kept, all others are removed.

withoutnames(**names*)

Return a copy of `self` where all the attributes with XML names in `names` are removed.

__init__(**content*, ***attrs*)

Create a new *`Element`* instance.

Positional arguments are treated as content nodes. Keyword arguments and dictionaries are treated as attributes.

__enter__()

`Element` nodes can be used in `with` blocks to build XIST trees. Inside a `with` block + and *`add()`* can be used to append node to the currently active element in the `with` block:

```
with xsc.build():
   with html.ul() as node:
      +html.li("I hear and I forget.")
      +html.li("I see and I believe.")
      +html.li("I do and I understand.")
      xsc.add(class_="quote")
print(node.bytes())
```

__call__(**content*, ***attrs*)

Calling an element add items in `content` to the element content and set attributes from `attrs`. The element itself will be returned.

append(**items*)

Append every item in `items` to the elements content.

extend(*items*)

Append all items in `items` to the elements content.

insert(*index*, **items*)

Insert every item in `items` at the position `index`.

__getitem__(*index*)

If `index` is a string, return the attribute with this (Python) name. If `index` is an attribute class, return the attribute that is an instance of this class. If `index` is a number or slice return the appropriate content node. `index` may also be a list, in with case `__getitem__()` will be applied recusively. `__getitem__()` also supports walk filters.

__setitem__(*index*, *value*)

Set an attribute or content node to the value `value`. For possible types for `index` see *`__getitem__()`*.

__delitem__(*index*)

Remove an attribute or content node. For possible types for `index` see *`__getitem__()`*.

__len__()

Return the number of children.

withsep(*separator*, *clone=False*)

Return a version of `self` with a separator node between the child nodes of `self`. For more info see *Frag.withsep()*.

reversed()

Return a reversed version of `self`.

filtered(*function*)

Return a filtered version of the `self`.

shuffled()

Return a shuffled version of the `self`.

class ll.xist.xsc.**AttrElement**

Bases: *Element*

Special subclass of *Element*.

When an *AttrElement* node is the only node in an attribute, it takes over publishing of the attribute (via the methods *publishattr()* and *publishboolattr()*). In all other cases publishing is done in the normal way (and must be overwritten with the *publish()* method).

publish(*publisher*)

Publish `self` to the publisher `publisher` (outside of any attribute)

publishattr(*publisher*, *attr*)

Publish the attribute `attr` to the publisher `publisher`.

publishboolattr(*publisher*, *attr*)

Publish the boolean attribute `attr` to the publisher

class ll.xist.xsc.**Entity**

Bases: *Node*

Class for entities. Derive your own entities from it and overwrite `convert()`.

class ll.xist.xsc.**CharRef**

Bases: *Text*, *Entity*

A simple named character reference, the code point is in the class attribute `codepoint`.

class ll.xist.xsc.**Pool**

Bases: *Pool*

A *Pool* stores a collection of XIST classes and can be passed to a parser. The parser will ask the pool which classes to use when elements, processing instructions etc. have to be instantiated.

__init__(**objects*)

Create a *Pool* object. All items in `objects` will be registered in the pool.

register(*object*)

Register `object` in the pool. `object` can be:

- a *Element*, *ProcInst* or *Entity* class;
- an *Attr* class for a global attribute;
- an *Attrs* class containing global attributes;
- a `dict` (all values will be registered, this makes it possible to e.g. register all local variables by passing `vars()`);
- a module (all attributes in the module will be registered).

clear()

Make `self` empty.

clone()

Return a copy of `self`.

elements()

Return an iterator for all registered element classes.

elementclass(*xmlns*, *name*)

Return the element class for the element with the XML name `name` and the namespace `xmlns`. If the element can't be found an *Element* will be returned.

element(*xmlns*, *name*)

Return an element object for the element type with the XML name `name` and the namespace `xmlns`.

haselement(*xmlns*, *name*)

Is there a registered element class in `self` for the element type with the Python name `name` and the namespace `xmlns`?

procinsts()

Return an iterator for all registered processing instruction classes.

procinstclass(*name*)

Return the processing instruction class for the PI with the target name `name`. If the processing instruction can't be found an return *ProcInst*.

procinst(*name*, *content*)

Return a processing instruction object for the PI type with the target name `name`.

hasprocinst(*name*)

Is there a registered processing instruction class in `self` for the PI with the target name `name`?

entities()

Return an iterator for all registered entity classes.

entityclass(*name*)

Return the entity class for the entity with the XML name `name`. If the entity can't be found return *Entity*.

entity(*name*)

Return an entity object for the entity with the XML name `name`.

hasentity(*name*)

Is there a registered entity class in `self` for the entity with the XML name `name`?

attrkey(*xmlns*, *name*)

Return the key that can be used to set the attribute with the name `name` and the namespace `xmlns`. If `self` (or one of the base pools) has any global attribute registered for that name/namespace, the attribute class will be returned. Otherwise the tuple `(name, xmlns)` (or `name` itself for a local attribute) will be returned. With this key *Attrs.__setitem__()* will create the appropriate attribute class.

text(*content*)

Create a text node with the content `content`.

comment(*content*)

Create a comment node with the content `content`.

`ll.xist.xsc.`**docpool**()

Return a pool suitable for parsing XIST docstrings.

`ll.xist.xsc.`**nsname**(*xmlns*)

If `xmlns` is a module, return `xmlns.xmlns`, else return `xmlns` unchanged.

`ll.xist.xsc.nsclark(obj)`

Return a name in Clark notation. `xmlns` can be `None`, a string or a module to return a namespace name, or a *Node* instance to return a namespace name + node name combination:

```
>>> from ll.xist import xsc
>>> from ll.xist.ns import html
>>> xsc.nsclark(None)
'{}'
>>> xsc.nsclark(html)
'{http://www.w3.org/1999/xhtml}'
>>> xsc.nsclark(html.a)
'{http://www.w3.org/1999/xhtml}a'
>>> xsc.nsclark(html.a())
'{http://www.w3.org/1999/xhtml}a'
```

class `ll.xist.xsc.quot`

Bases: *CharRef*

quotation mark = APL quote, U+0022 ISOnum

class `ll.xist.xsc.amp`

Bases: *CharRef*

ampersand, U+0026 ISOnum

class `ll.xist.xsc.lt`

Bases: *CharRef*

less-than sign, U+003C ISOnum

class `ll.xist.xsc.gt`

Bases: *CharRef*

greater-than sign, U+003E ISOnum

class `ll.xist.xsc.apos`

Bases: *CharRef*

apostrophe mark, U+0027 ISOnum

`ll.xist.xsc.element(xmlns, xmlname, *content, **attrs)`

Create a plain element object with the namespace name `xmlns` and the element name `xmlname`. This object will be an instance of *Element* (not an instance of a subclass). `content` and `attrs` will be used to initialize the content and attributes of the element.

`ll.xist.xsc.entity(xmlname)`

Create a plain entity object with the entity name `xmlname`. This object will be an instance of *Entity* (not an instance of a subclass).

`ll.xist.xsc.procinst(xmlname, *content)`

Create a plain processing instruction object with the target name `xmlname`. This object will be an instance of *ProcInst* (not an instance of a subclass). `content` will be used to initialize the content of the processing instruction.

class `ll.xist.xsc.Location`

Bases: `object`

Represents a location in an XML entity.

`__init__(url=None, line=None, col=None)`

Create a new *Location* object using the arguments passed in. `url` is the URL/filename. `line` is the line number and `col` is the column number (both starting at 0).

offset(*offset*)

Return a location where the line number is incremented by offset (and the column number is reset to 0).

6.1.8 ns – Package containing namespaces

This package contains all the modules that provide namespaces to to XIST. For example the definition of HTML can be found in the module *ll.xist.ns.html*.

Some of these namespaces can be considered target namespaces (e.g. *ll.xist.ns.html*, ll.xist.ns.wml and *ll.xist.ns.docbook*). The element and entity classes in these namespaces don't implement a convert method, i.e. they inherit the convert() method from ll.xist.xsc.Element.convert.

Other namespace modules provide additional functionality through new element classes. Calling *ll.xist.xsc.Node.convert()* on these elements might convert them to one of these target namespaces (depending on the target attribute of the *ll.xist.xsc.Converter* object passed around.) Some of these namespace modules completely ignore the target and convert to one fixed target namespace (*ll.xist.ns.html* in most cases).

html – HTML namespace

An XIST namespace that contains definitions for all the elements in HTML5 as well as some (deprecated) elements that were in use in previous HTML versions.

This namespace also supports the elements and attributes from the microdata specification.

For all deprecated elements and attributes the class attribute deprecated is set to True.

The function *astext()* can be used to convert a HTML XIST tree into plain text.

class ll.xist.ns.html.**DocTypeHTML40transitional**

Bases: *DocType*

document type for HTML 4.0 transitional

class ll.xist.ns.html.**DocTypeHTML401transitional**

Bases: *DocType*

document type for HTML 4.01 transitional

class ll.xist.ns.html.**DocTypeXHTML10strict**

Bases: *DocType*

document type for XHTML 1.0 strict

class ll.xist.ns.html.**DocTypeXHTML10transitional**

Bases: *DocType*

document type for XHTML 1.0 transitional

class ll.xist.ns.html.**DocTypeXHTML11**

Bases: *DocType*

document type for XHTML 1.1

class ll.xist.ns.html.**DocTypeHTML5**

Bases: *DocType*

document type for HTML5

class ll.xist.ns.html.**GlobalAttrs**

Bases: *Attrs*

Attributes that are common to and may be specified on all HTML elements

class accesskey

Bases: *TextAttr*

This attribute's value is used by the user agent as a guide for creating a keyboard shortcut that activates or focuses the element.

If specified, the value must be an ordered set of unique space-separated tokens that are case-sensitive, each of which must be exactly one Unicode code point in length.

class class_

Bases: *TextAttr*

This attribute, if specified, must have a value that is a set of space-separated tokens representing the various classes that the element belongs to.

class contenteditable

Bases: *TextAttr*

Indicates whether the element is editable.

class contextmenu

Bases: *TextAttr*

The element's context menu. The value must be the ID of a menu element in the DOM.

class dir

Bases: *TextAttr*

The element's text directionality.

class draggable

Bases: *TextAttr*

Specifies whether the element is draggable.

class dropzone

Bases: *TextAttr*

Specifies which types of objects are allowed to be dropped on the element and how they are handled.

class hidden

Bases: *BoolAttr*

When specified, indicates that the element is not yet, or is no longer, directly relevant to the page's current state.

User agents should not render elements that have the hidden attribute specified.

class id

Bases: *IDAttr*

Specifies its element's unique identifier.

class lang

Bases: *TextAttr*

Specifies the primary language for the element's contents and for any of the element's attributes that contain text.

class spellcheck

Bases: *TextAttr*

Specifies whether the user agent should indicate spelling and/or grammar errors in content of the element.

class style

Bases: *StyleAttr*

A CSS styling attribute

class tabindex

Bases: *IntAttr*

Specifies whether an element is supposed to be focusable and what is to be the relative order of the element for the purposes of sequential focus navigation.

class title

Bases: *TextAttr*

Advisory information for the element, such as would be appropriate for a tooltip.

class translate

Bases: *TextAttr*

An enumerated attribute that is used to specify whether an element's attribute values and the values of its text node children are to be translated when the page is localized, or whether to leave them unchanged.

class role

Bases: *TextAttr*

If specified, must have a value that is a set of space-separated tokens representing the various WAI-ARIA roles that the element belongs to.

class onabort

Bases: *TextAttr*

Event handler

class onblur

Bases: *TextAttr*

Event handler

class oncancel

Bases: *TextAttr*

Event handler

class oncanplay

Bases: *TextAttr*

Event handler

class oncanplaythrough

Bases: *TextAttr*

Event handler

class onchange

Bases: *TextAttr*

Event handler

class onclick

Bases: *TextAttr*

Event handler

class onclose

Bases: *TextAttr*

Event handler

class oncontextmenu

Bases: *TextAttr*

Event handler

class oncuechange

Bases: *TextAttr*

Event handler

class ondblclick

Bases: *TextAttr*

Event handler

class ondrag

Bases: *TextAttr*

Event handler

class ondragend

Bases: *TextAttr*

Event handler

class ondragenter

Bases: *TextAttr*

Event handler

class ondragleave

Bases: *TextAttr*

Event handler

class ondragover

Bases: *TextAttr*

Event handler

class ondragstart

Bases: *TextAttr*

Event handler

class ondrop

Bases: *TextAttr*

Event handler

class ondurationchange

Bases: *TextAttr*

Event handler

class onemptied

Bases: *TextAttr*

Event handler

class onended

Bases: *TextAttr*

Event handler

class onerror

Bases: *TextAttr*

Event handler

class onfocus

Bases: *TextAttr*

Event handler

class oninput

Bases: *TextAttr*

Event handler

class oninvalid

Bases: *TextAttr*

Event handler

class onkeydown

Bases: *TextAttr*

Event handler

class onkeypress

Bases: *TextAttr*

Event handler

class onkeyup

Bases: *TextAttr*

Event handler

class onload

Bases: *TextAttr*

Event handler

class onloadeddata

Bases: *TextAttr*

Event handler

class onloadedmetadata

Bases: *TextAttr*

Event handler

class onloadstart

Bases: *TextAttr*

Event handler

class onmousedown

Bases: *TextAttr*

Event handler

class onmousemove

Bases: *TextAttr*

Event handler

class onmouseout

Bases: *TextAttr*

Event handler

class onmouseover

Bases: *TextAttr*

Event handler

class onmouseup

Bases: *TextAttr*

Event handler

class onmousewheel

Bases: *TextAttr*

Event handler

class onpause

Bases: *TextAttr*

Event handler

class onplay

Bases: *TextAttr*

Event handler

class onplaying

Bases: *TextAttr*

Event handler

class onprogress

Bases: *TextAttr*

Event handler

class onratechange

Bases: *TextAttr*

Event handler

class onreset

Bases: *TextAttr*

Event handler

class onscroll

Bases: *TextAttr*

Event handler

class onseeked

Bases: *TextAttr*

Event handler

class onseeking

Bases: *TextAttr*

Event handler

class onselect

Bases: *TextAttr*

Event handler

class onshow

Bases: *TextAttr*

Event handler

class onstalled

Bases: *TextAttr*

Event handler

class onsubmit

Bases: *TextAttr*

Event handler

class onsuspend

Bases: *TextAttr*

Event handler

class ontimeupdate

Bases: *TextAttr*

Event handler

class onvolumechange

Bases: *TextAttr*

Event handler

class onwaiting

Bases: *TextAttr*

Event handler

class itemscope

Bases: *BoolAttr*

Microdata attribute: Creates a new item, a group of name-value pairs.

class itemtype

Bases: *TextAttr*

Microdata attribute: Space separated list of absolute URLs specifying the type of the item.

class itemid

Bases: *URLAttr*

Microdata attribute: A global identifier for the item.

class itemprop

Bases: *TextAttr*

Microdata attribute: The name of an item property.

class itemref

Bases: *TextAttr*

Microdata attribute: List of additional element IDs to crawl to find the name-value pairs of the item.

class ll.xist.ns.html.**html**

Bases: *Element*

The root of an HTML document.

class ll.xist.ns.html.**head**

Bases: *Element*

A collection of metadata for the document.

class ll.xist.ns.html.**title**

Bases: *Element*

The document's title or name.

class ll.xist.ns.html.**base**

Bases: *Element*

Allows authors to specify the document base URL for the purposes of resolving relative URLs, and the name of the default browsing context for the purposes of following hyperlinks.

class ll.xist.ns.html.**link**

Bases: *Element*

Allows authors to link their document to other resources.

class ll.xist.ns.html.**meta**

Bases: *Element*

Various kinds of metadata that cannot be expressed using the `title`, `base`, `link`, `style`, and `script` elements.

class ll.xist.ns.html.**style**

Bases: *Element*

Allows authors to embed style information in their documents.

class ll.xist.ns.html.**script**

Bases: *Element*

Allows authors to include dynamic script and data blocks in their documents.

class ll.xist.ns.html.**noscript**

Bases: *Element*

Represents nothing if scripting is enabled, and represents its children if scripting is disabled.

class ll.xist.ns.html.**body**

Bases: *Element*

The main content of the document.

class ll.xist.ns.html.**article**

Bases: *Element*

A self-contained composition in a document, page, application, or site and that is, in principle, independently distributable or reusable, e.g. in syndication.

This could be a forum post, a magazine or newspaper article, a blog entry, a user-submitted comment, an interactive widget or gadget, or any other independent item of content.

class ll.xist.ns.html.**section**

Bases: *Element*

A generic section of a document or application. A section, in this context, is a thematic grouping of content, typically with a heading.

class ll.xist.ns.html.**nav**

Bases: *Element*

A section of a page that links to other pages or to parts within the page: a section with navigation links.

class ll.xist.ns.html.**aside**

Bases: *Element*

A section of a page that consists of content that is tangentially related to the content around the aside element, and which could be considered separate from that content. Such sections are often represented as sidebars in printed typography.

class ll.xist.ns.html.**h**

Bases: *Element*

Base class of h1, h2, h3, h4, h5 and h6, which represent headings for their sections.

class ll.xist.ns.html.**hgroup**

Bases: *Element*

The heading of a section. The element is used to group a set of h1–h6 elements when the heading has multiple levels, such as subheadings, alternative titles, or taglines.

class ll.xist.ns.html.**header**

Bases: *Element*

A group of introductory or navigational aids.

class ll.xist.ns.html.**footer**

Bases: *Element*

A footer for its nearest ancestor sectioning content or sectioning root element. A footer typically contains information about its section such as who wrote it, links to related documents, copyright data, and the like.

class ll.xist.ns.html.**address**

Bases: *Element*

The contact information for its nearest article or body element ancestor.

class ll.xist.ns.html.**p**

Bases: *Element*

A paragraph.

class ll.xist.ns.html.**hr**

Bases: *Element*

A paragraph-level thematic break, e.g. a scene change in a story, or a transition to another topic within a section of a reference book.

class ll.xist.ns.html.**pre**

Bases: *Element*

A block of preformatted text, in which structure is represented by typographic conventions rather than by elements.

class ll.xist.ns.html.**blockquote**

Bases: *Element*

A section that is quoted from another source.

class ll.xist.ns.html.**ol**

Bases: *Element*

A list of items, where the items have been intentionally ordered, such that changing the order would change the meaning of the document.

class ll.xist.ns.html.**ul**

Bases: *Element*

A list of items, where the order of the items is not important — that is, where changing the order would not materially change the meaning of the document.

class ll.xist.ns.html.**li**

Bases: *Element*

A list item.

class ll.xist.ns.html.**dl**

Bases: *Element*

An association list consisting of zero or more name-value groups (a description list). Each group must consist of one or more names (`dt` elements) followed by one or more values (`dd` elements). Within a single dl element, there should not be more than one `dt` element for each name.

Name-value groups may be terms and definitions, metadata topics and values, questions and answers, or any other groups of name-value data.

class ll.xist.ns.html.**dt**

Bases: *Element*

The term, or name, part of a term-description group in a description list (`dl` element).

class ll.xist.ns.html.**dd**

Bases: *Element*

The description, definition, or value, part of a term-description group in a description list (`dl` element).

class ll.xist.ns.html.**figure**

Bases: *Element*

Some flow content, optionally with a caption, that is self-contained and is typically referenced as a single unit from the main flow of the document.

The element can thus be used to annotate illustrations, diagrams, photos, code listings, etc, that are referred to from the main content of the document, but that could, without affecting the flow of the document, be moved away from that primary content, e.g. to the side of the page, to dedicated pages, or to an appendix.

class ll.xist.ns.html.**figcaption**

Bases: *Element*

A caption or legend for the rest of the contents of the `figcaption` element's parent `figure` element, if any.

class ll.xist.ns.html.**div**

Bases: *Element*

The `div` element has no special meaning at all. It represents its children. It can be used with the `class`, `lang`, and `title` attributes to mark up semantics common to a group of consecutive elements.

class ll.xist.ns.html.**a**

Bases: *Element*

If the `a` element has an `href` attribute, then it represents a hyperlink (a hypertext anchor) labeled by its contents.

If the `a` element has no `href` attribute, then the element represents a placeholder for where a link might otherwise have been placed, if it had been relevant, consisting of just the element's contents.

class ll.xist.ns.html.**em**

Bases: *Element*

Stress emphasis.

class ll.xist.ns.html.**strong**

Bases: *Element*

Strong importance.

class ll.xist.ns.html.**small**

Bases: *Element*

Side comments such as small print.

class ll.xist.ns.html.**s**

Bases: *Element*

Contents that are no longer accurate or no longer relevant.

class ll.xist.ns.html.**cite**

Bases: *Element*

The title of a work (e.g. a book, a paper, an essay, a poem, a score, a song, a script, a film, a TV show, a game, a sculpture, a painting, a theatre production, a play, an opera, a musical, an exhibition, a legal case report, etc). This can be a work that is being quoted or referenced in detail (i.e. a citation), or it can just be a work that is mentioned in passing.

class ll.xist.ns.html.**q**

Bases: *Element*

Some phrasing content quoted from another source.

class ll.xist.ns.html.**dfn**

Bases: *Element*

The defining instance of a term. The paragraph, description list group, or section that is the nearest ancestor of the `dfn` element must also contain the definition(s) for the term given by the `dfn` element.

class ll.xist.ns.html.**abbr**

Bases: *Element*

An abbreviation or acronym, optionally with its expansion. The `title` attribute may be used to provide an expansion of the abbreviation. The attribute, if specified, must contain an expansion of the abbreviation, and nothing else.

class ll.xist.ns.html.**time**

Bases: *Element*

Represents its contents, along with a machine-readable form of those contents in the `datetime` attribute.

class ll.xist.ns.html.**code**

Bases: *Element*

A fragment of computer code. This could be an XML element name, a filename, a computer program, or any other string that a computer would recognize.

class ll.xist.ns.html.**var**

Bases: *Element*

A variable. This could be an actual variable in a mathematical expression or programming context, an identifier representing a constant, a symbol identifying a physical quantity, a function parameter, or just be a term used as a placeholder in prose.

class ll.xist.ns.html.**samp**

Bases: *Element*

A (sample) output from a program or computing system.

class ll.xist.ns.html.**kbd**

Bases: *Element*

User input (typically keyboard input, although it may also be used to represent other input, such as voice commands).

class ll.xist.ns.html.**sub**

Bases: *Element*

A subscript.

class ll.xist.ns.html.**sup**

Bases: *Element*

A superscript.

class ll.xist.ns.html.**i**

Bases: *Element*

A span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text, such as a taxonomic designation, a technical term, an idiomatic phrase or short span of transliterated prose from another language, a thought, or a ship name in Western texts.

class ll.xist.ns.html.**b**

Bases: *Element*

A span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a review, actionable words in interactive text-driven software, or an article lede.

class ll.xist.ns.html.**u**

Bases: *Element*

A span of text with an unarticulated, though explicitly rendered, non-textual annotation, such as labeling the text as being a proper name in Chinese text (a Chinese proper name mark), or labeling the text as being misspelt.

class ll.xist.ns.html.**mark**

Bases: *Element*

A run of text in one document marked or highlighted for reference purposes, due to its relevance in another context. When used in a quotation or other block of text referred to from the prose, it indicates a highlight that was not originally present but which has been added to bring the reader's attention to a part of the text that might not have been considered important by the original author when the block was originally written, but which is now under previously unexpected scrutiny. When used in the main prose of a document, it indicates a part of the document that has been highlighted due to its likely relevance to the user's current activity.

class ll.xist.ns.html.**ruby**

Bases: *Element*

Allows one or more spans of phrasing content to be marked with ruby annotations. Ruby annotations are short runs of text presented alongside base text, primarily used in East Asian typography as a guide for pronunciation or to include other annotations. In Japanese, this form of typography is also known as "furigana".

class ll.xist.ns.html.**rt**

Bases: *Element*

The ruby text component of a ruby annotation.

class ll.xist.ns.html.**rp**

Bases: *Element*

Can be used to provide parentheses around a ruby text component of a ruby annotation, to be shown by user agents that don't support ruby annotations.

class ll.xist.ns.html.**bdi**

Bases: *Element*

A span of text that is to be isolated from its surroundings for the purposes of bidirectional text formatting.

class ll.xist.ns.html.**bdo**

Bases: *Element*

Explicit text directionality formatting control for its children. It allows authors to override the Unicode bidirectional algorithm by explicitly specifying a direction override.

class ll.xist.ns.html.**span**

Bases: *Element*

Doesn't mean anything on its own, but can be useful when used together with the global attributes, e.g. `class`, `lang`, or `dir`. It represents its children.

class ll.xist.ns.html.**br**

Bases: *Element*

A line break.

class ll.xist.ns.html.**wbr**

Bases: *Element*

A line break opportunity.

class ll.xist.ns.html.**ins**

Bases: *Element*

An addition to the document.

class ll.xist.ns.html.**del_**

Bases: *Element*

A removal from the document.

class ll.xist.ns.html.**img**

Bases: *Element*

An image.

class ll.xist.ns.html.**iframe**

Bases: *Element*

A nested browsing context.

class ll.xist.ns.html.**embed**

Bases: *Element*

An integration point for an external (typically non-HTML) application or interactive content.

class ll.xist.ns.html.**object**

Bases: *Element*

This element can represent an external resource, which, depending on the type of the resource, will either be treated as an image, as a nested browsing context, or as an external resource to be processed by a plugin.

class ll.xist.ns.html.**param**

Bases: *Element*

Defines parameters for plugins invoked by `object` elements. It does not represent anything on its own.

class ll.xist.ns.html.**video**

Bases: *Element*

Used for playing videos or movies, and audio files with captions.

class ll.xist.ns.html.**audio**

Bases: *Element*

A sound or audio stream.

class ll.xist.ns.html.**source**

Bases: *Element*

Allows authors to specify multiple alternative media resources for media elements. It does not represent anything on its own.

class ll.xist.ns.html.**track**

Bases: *Element*

Allows authors to specify explicit external timed text tracks for media elements. It does not represent anything on its own.

class ll.xist.ns.html.**canvas**

Bases: *Element*

Provides scripts with a resolution-dependent bitmap canvas, which can be used for rendering graphs, game graphics, art, or other visual images on the fly.

class ll.xist.ns.html.**map**

Bases: *Element*

Defines an image map in conjunction with any `area` element descendants. The element represents its children.

class ll.xist.ns.html.**area**

Bases: *Element*

Represents either a hyperlink with some text and a corresponding area on an image map, or a dead area on an image map.

class ll.xist.ns.html.**table**

Bases: *Element*

Data with more than one dimension, in the form of a table.

The `table` element takes part in the table model. Tables have rows, columns, and cells given by their descendants. The rows and columns form a grid; a table's cells must completely cover that grid without overlap.

class ll.xist.ns.html.**caption**

Bases: *Element*

The title of the `table` that is its parent.

class ll.xist.ns.html.**colgroup**

Bases: *Element*

A group of one or more columns in the `table` that is its parent.

class ll.xist.ns.html.**col**

Bases: *Element*

One or more columns in the column group represented by the `colgroup` that is its parent.

class ll.xist.ns.html.**tbody**

Bases: *Element*

A block of rows that consist of a body of data for the parent `table` element.

class ll.xist.ns.html.**thead**

Bases: *Element*

The block of rows that consist of the column labels (headers) for the parent `table` element.

class ll.xist.ns.html.**tfoot**

Bases: *Element*

The block of rows that consist of the column summaries (footers) for the parent `table` element

class ll.xist.ns.html.**tr**

Bases: *Element*

A row of cells in a table.

class ll.xist.ns.html.**td**

Bases: *Element*

A data cell in a table.

class ll.xist.ns.html.**th**

Bases: *Element*

A header cell in a table.

class ll.xist.ns.html.**form**

Bases: *Element*

A collection of form-associated elements, some of which can represent editable values that can be submitted to a server for processing.

class ll.xist.ns.html.**fieldset**

Bases: *Element*

A set of form controls optionally grouped under a common name.

class ll.xist.ns.html.**legend**

Bases: *Element*

A caption for the rest of the contents of the legend element's parent fieldset element, if any.

class ll.xist.ns.html.**label**

Bases: *Element*

A caption in a user interface. The caption can be associated with a specific form control, known as the label element's labeled control, either using `for` attribute, or by putting the form control inside the label element itself.

class ll.xist.ns.html.**input**

Bases: *Element*

A caption in a user interface. The caption can be associated with a specific form control, known as the label element's labeled control, either using `for` attribute, or by putting the form control inside the label element itself.

class ll.xist.ns.html.**button**

Bases: *Element*

A button labeled by its contents. The `type` attribute controls the behavior of the button when it is activated.

class ll.xist.ns.html.**select**

Bases: *Element*

A control for selecting amongst a set of options.

class ll.xist.ns.html.**datalist**

Bases: *Element*

A set of option elements that represent predefined options for other controls. The contents of the element represents fallback content for legacy user agents, intermixed with `option` elements that represent the predefined options. In the rendering, the `datalist` element represents nothing and it, along with its children, should be hidden.

The datalist element is hooked up to an `input` element using the `list` attribute on the `input` element.

class ll.xist.ns.html.**optgroup**

Bases: *Element*

A group of `option` elements with a common label.

class ll.xist.ns.html.**option**

Bases: *Element*

An option in a `select` element or as part of a list of suggestions in a `datalist` element.

class ll.xist.ns.html.**textarea**

Bases: *Element*

A multiline plain text edit control for the element's raw value.

class ll.xist.ns.html.**keygen**

Bases: *Element*

A key pair generator control. When the control's form is submitted, the private key is stored in the local keystore, and the public key is packaged and sent to the server.

class ll.xist.ns.html.**output**

Bases: *Element*

The result of a calculation.

Allows an explicit relationship to be made between the result of a calculation and the elements that represent the values that went into the calculation or that otherwise influenced the calculation.

class ll.xist.ns.html.**progress**

Bases: *Element*

The completion progress of a task.

The progress is either indeterminate, indicating that progress is being made but that it is not clear how much more work remains to be done before the task is complete (e.g. because the task is waiting for a remote host to respond), or the progress is a number in the range zero to a maximum, giving the fraction of work that has so far been completed.

class ll.xist.ns.html.**meter**

Bases: *Element*

A scalar measurement within a known range, or a fractional value; for example disk usage, the relevance of a query result, or the fraction of a voting population to have selected a particular candidate.

This is also known as a gauge.

class ll.xist.ns.html.**details**

Bases: *Element*

A disclosure widget from which the user can obtain additional information or controls.

class ll.xist.ns.html.**summary**

Bases: *Element*

A summary, caption, or legend for the rest of the contents of the `summary` element's parent `details` element, if any.

class ll.xist.ns.html.**command**

Bases: *Element*

A summary, caption, or legend for the rest of the contents of the `summary` element's parent `details` element, if any.

class ll.xist.ns.html.**menu**

Bases: *Element*

A list of commands.

class ll.xist.ns.html.**dialog**

Bases: *Element*

A part of an application that a user interacts with to perform a task, for example a dialog box, inspector, or window.

class ll.xist.ns.html.**data**

Bases: *Element*

Microdata element: Contains the name and value of an item property.

class ll.xist.ns.html.**noframes**

Bases: *Element*

Alternate content container for non frame-based rendering (deprecated).

class ll.xist.ns.html.**dir**

Bases: *Element*

Multiple column list (deprecated).

class ll.xist.ns.html.**center**

Bases: *Element*

Centered block level element (deprecated).

class ll.xist.ns.html.**acronym**

Bases: *Element*

Indicates an acronym (e.g., WAC, radar, etc.) (deprecated).

class ll.xist.ns.html.**tt**

Bases: *Element*

Teletype or monospaced text style (deprecated).

class ll.xist.ns.html.**big**

Bases: *Element*

Large text style (deprecated).

class ll.xist.ns.html.**strike**

Bases: *Element*

Strike-through text style (deprecated).

class ll.xist.ns.html.**basefont**

Bases: *Element*

Base font size (deprecated).

class ll.xist.ns.html.**font**

Bases: *Element*

Local change to font (deprecated).

class ll.xist.ns.html.**applet**

Bases: *Element*

Java applet (deprecated).

class ll.xist.ns.html.**isindex**

Bases: *Element*

(deprecated).

class ll.xist.ns.html.**frameset**

Bases: *Element*

Window subdivision (deprecated)

class ll.xist.ns.html.**frame**

Bases: *Element*

Subwindow (deprecated).

class ll.xist.ns.html.**nobr**

Bases: *Element*

Prevents line breaks (deprecated).

ll.xist.ns.html.**astext**(*node*, *width=None*, *tabwidth=8*, *default={'display': 'inline'}*, *h1={'bottom': 1, 'display': 'block', 'overline': '=', 'top': 2, 'underline': '=', 'whitespace': 'nowrap'}*, *h2={'bottom': 1, 'display': 'block', 'top': 2, 'underline': '-', 'whitespace': 'nowrap'}*, *h3={'bottom': 1, 'display': 'block', 'top': 2, 'underline': '"', 'whitespace': 'nowrap'}*, *h4={'bottom': 1, 'display': 'block', 'top': 2, 'underline': '"', 'whitespace': 'nowrap'}*, *h5={'bottom': 1, 'display': 'block', 'top': 2, 'underline': '"', 'whitespace': 'nowrap'}*, *h6={'bottom': 1, 'display': 'block', 'top': 2, 'underline': '"', 'whitespace': 'nowrap'}*, *dl={'bottom': 1, 'display': 'block', 'top': 1}*, *dt={'display': 'block', 'top': 1}*, *dd={'bottom': 1, 'display': 'block', 'left': ' '}*, *ol={'bottom': 1, 'display': 'block', 'top': 1}*, *ol_li={'bottom': 1, 'display': 'block', 'left': '{pos:{width}}. \n', 'top': 1}*, *ul={'bottom': 1, 'display': 'block', 'top': 1}*, *ul_li={'bottom': 1, 'display': 'block', 'left': ('* \n', '- \n'), 'top': 1}*, *li={'bottom': 1, 'display': 'block', 'top': 1}*, *pre={'bottom': 1, 'display': 'block', 'left': ' ', 'top': 1, 'whitespace': 'pre'}*, *blockquote={'bottom': 1, 'display': 'block', 'left': ' ', 'top': 1}*, *div={'bottom': 1, 'display': 'block'}*, *p={'bottom': 1, 'display': 'block'}*, *hr={'bottom': 1, 'display': 'block'}*, *address={'bottom': 1, 'display': 'block'}*, *th={'bottom': 1, 'display': 'block'}*, *td={'bottom': 1, 'display': 'block'}*, *b={'display': 'inline', 'prefix': '*', 'suffix': '*'}*, *em={'display': 'inline', 'prefix': '*', 'suffix': '*'}*, *strong={'display': 'inline', 'prefix': '**', 'suffix': '**'}*, *i={'display': 'inline', 'prefix': '*', 'suffix': '*'}*, *u={'display': 'inline', 'prefix': '_', 'suffix': '_'}*, *code={'display': 'inline', 'prefix': '``', 'suffix': '``'}*, ***kwargs*)

Return the node `node` formatted as plain text. `node` must contain an HTML tree.

`width` is the maximum line length. If `width` is `None` line length is unlimited (i.e. no line wrapping will be done).

`tabwidth` specifies the number of spaces for tab expansion (the default is 8).

The rest of the parameters specify the formatting styles for HTML elements. The parameter names are the names of the HTML elements, except for `ol_li` which is used for `li` elements inside `ol` elements and `ul_li` which is used for `li` elements inside `ul` elements. `default` is used if the parameter for the HTML element is not passed.

The parameter value must be a dictionary which might contain any of the following keys (if the key is missing a default value is used):

display

This is either `"block"` for a block level element or `"inline"` for an inline element.

prefix
A string that should be output before any of the content of the block.

suffix
A string that should be output after any of the content of the block.

The following keys will only be used for `display == "block"`:

top
The minimum number of empty lines before the block. (The default is `0`)

bottom
The minimum number of empty lines after the block. (The default is `0`)

left
The left margin for the block. This margin can be different for different nesting levels (e. g. different "bullets" can be used for nested `ul`s). If the value is a string it will be used as the indentation on the left side for all levels, otherwise it must be a list of strings. If the nesting of this element is deeper than the list, the last item in the list will be used.

If a margin contains multiple lines, the first indentation line will be used for the first content line, the second indentation line for the second content line etc. If the content has more lines than the indentation the last indentation line will be repeated. All indentation lines will be padded to the longest line. For example the indentation for a `li` element inside an `ul` element on level 1 is `"* \n"`, i.e. the first line will be indented with `"* "`, all subsequent lines with three spaces.

right
The right margin for the block. This supports the same semantics regarding levels and multiple lines as the `left` argument.

whitespace
Specifies how lines in the block should be wrapped. `"normal"` collapses consecutive whitespace and wraps the lines at the specified width. `"nowrap"` collapses consecutive whitespace but doesn't wrap and `"pre"` uses the lines as given.

overline
A character that is repeated for the width of the content as a rule before the content. If `None` is used, no rule will be output. (Note that this will only work on the innermost block level element.)

underline
A rule after the content of the block. (Note that this will only work on the innermost block level element.)

`xml` – Global attributes from the XML namespace

Contains the global attributes for the XML namespace (like `xml:lang`), and classes for the XML declaration.

class `ll.xist.ns.xml.`**XML**

Bases: *ProcInst*

XML declaration. The encoding will be automatically set when publishing.

class `ll.xist.ns.xml.`**XMLStyleSheet**

Bases: *ProcInst*

XML stylesheet declaration.

class `ll.xist.ns.xml.`**declaration**

Bases: *Element*

The XML declaration as an element. This makes it possible to generate a declaration from within an XML file.

docbook – DocBook 5.0

An XIST namespace module that contains definitions for all the elements in DocBook 5.0.

svg – SVG 1.1

This is a namespace module implementing SVG 1.1.

class ll.xist.ns.svg.**DocTypeSVG11**

Bases: *DocType*

document type for SVG 1.1

abbr – Abbreviation entities

This module contains entities for many abbreviations and acronyms.

class ll.xist.ns.abbr.**base**

Bases: *Entity*

The base of all entity classes in this namespace.

class ll.xist.ns.abbr.**rmi**

Bases: *base*

"Remote Method Invocation"

class ll.xist.ns.abbr.**jini**

Bases: *base*

"Java Intelligent Network Infrastructure"

class ll.xist.ns.abbr.**jfc**

Bases: *base*

"Java Foundation Classes"

class ll.xist.ns.abbr.**awt**

Bases: *base*

"Abstract Window Toolkit"

class ll.xist.ns.abbr.**jdbc**

Bases: *base*

"Java Database Connectivity"

class ll.xist.ns.abbr.**jndi**

Bases: *base*

"Java Naming and Directory Interface"

class ll.xist.ns.abbr.**jpda**

Bases: *base*

"Java Platform Debugger Architecture"

class ll.xist.ns.abbr.**jvmpi**

Bases: *base*

"Java Virtual Machine Profiler Interface"

class ll.xist.ns.abbr.**jni**

Bases: *base*

"Java Native Interface"

class ll.xist.ns.abbr.**ejb**

Bases: *base*

"Enterprice Java Beans"

class ll.xist.ns.abbr.**jnlp**

Bases: *base*

"Java Network Launch Protocol"

class ll.xist.ns.abbr.**jaoe**

Bases: *base*

"Java Acronym Overflow Error"

class ll.xist.ns.abbr.**jgl**

Bases: *base*

"Java Generic Library"

class ll.xist.ns.abbr.**sgml**

Bases: *base*

"Standard Generalized Markup Language"

class ll.xist.ns.abbr.**html**

Bases: *base*

"Hypertext Markup Language"

class ll.xist.ns.abbr.**xml**

Bases: *base*

"Extensible Markup Language"

class ll.xist.ns.abbr.**css**

Bases: *base*

"Cascading Style Sheet"

class ll.xist.ns.abbr.**cgi**

Bases: *base*

"Common Gateway Interface"

class ll.xist.ns.abbr.**www**

Bases: *base*

"World Wide Web"

class ll.xist.ns.abbr.**pdf**

Bases: *base*

"Protable Document Format"

class ll.xist.ns.abbr.**url**

Bases: *base*

"Uniform Resource Locator"

class ll.xist.ns.abbr.**http**

Bases: *base*

"Hypertext Transfer Protocol"

class ll.xist.ns.abbr.**smtp**

Bases: *base*

"Simple Mail Transfer Protocol"

class ll.xist.ns.abbr.**ftp**

Bases: *base*

"File Transfer Protocol"

class ll.xist.ns.abbr.**pop3**

Bases: *base*

"Post Office Protocol 3"

class ll.xist.ns.abbr.**cvs**

Bases: *base*

"Concurrent Versions System"

class ll.xist.ns.abbr.**faq**

Bases: *base*

"Frequently Asked Question"

class ll.xist.ns.abbr.**gnu**

Bases: *base*

"GNU's Not UNIX"

class ll.xist.ns.abbr.**dns**

Bases: *base*

"Domain Name Service"

class ll.xist.ns.abbr.**ppp**

Bases: *base*

"Point To Point Protocol"

class ll.xist.ns.abbr.**isdn**

Bases: *base*

"Integrated Services Digital Network"

class ll.xist.ns.abbr.**corba**

Bases: *base*

"Common Object Request Broker Architecture"

class ll.xist.ns.abbr.**wap**

Bases: *base*

"Wireless Application Protocol"

class ll.xist.ns.abbr.**wml**

Bases: *base*

"Wireless Markup Language"

class ll.xist.ns.abbr.**mac**

Bases: *base*

"Media Access Control"

class ll.xist.ns.abbr.**nat**

Bases: *base*

"Network Address Translation"

class ll.xist.ns.abbr.**sql**

Bases: *base*

"Structured Query Language"

class ll.xist.ns.abbr.**xsl**

Bases: *base*

"Extensible Stylesheet Language"

class ll.xist.ns.abbr.**xslt**

Bases: *base*

"Extensible Stylesheet Language For Transformations"

class ll.xist.ns.abbr.**smil**

Bases: *base*

"Synchronized Multimedia Integration Language"

class ll.xist.ns.abbr.**dtd**

Bases: *base*

"Document Type Definiton"

class ll.xist.ns.abbr.**dom**

Bases: *base*

"Document Object Model"

class ll.xist.ns.abbr.**api**

Bases: *base*

"Application Programming Interface"

class ll.xist.ns.abbr.**sax**

Bases: *base*

"Simple API for XML"

class ll.xist.ns.abbr.**dbms**

Bases: *base*

"Database Management System"

class ll.xist.ns.abbr.**ansi**

Bases: *base*

"American National Standards Institute"

class ll.xist.ns.abbr.**jsp**

Bases: *base*

"Java Server Pages"

class ll.xist.ns.abbr.**ascii**

Bases: *base*

"American Standard Code for Information Interchange"

class ll.xist.ns.abbr.**sms**

Bases: *base*

"Small Message Service"

class ll.xist.ns.abbr.**p2p**

Bases: *base*

"Peer To Peer"

class ll.xist.ns.abbr.**gif**

Bases: *base*

"Graphics Interchange Format"

class ll.xist.ns.abbr.**png**

Bases: *base*

"Portable Network Graphics"

class ll.xist.ns.abbr.**uddi**

Bases: *base*

"Universal Description, Discovery and Integration"

class ll.xist.ns.abbr.**wsdl**

Bases: *base*

"Web Services Description Language"

class ll.xist.ns.abbr.**snmp**

Bases: *base*

"Simple Network Management Protocol"

class ll.xist.ns.abbr.**ssl**

Bases: *base*

"Secure Socket Layer"

class ll.xist.ns.abbr.**vrml**

Bases: *base*

"Virtual Reality Modelling Language"

class ll.xist.ns.abbr.**tco**

Bases: *base*

"Total Cost of Ownership"

class ll.xist.ns.abbr.**crm**

Bases: *base*

"Customer Relationship Management"

class ll.xist.ns.abbr.**cms**

Bases: *base*

"Content Management System"

class ll.xist.ns.abbr.**bnf**

Bases: *base*

"Backus Naur Form"

class ll.xist.ns.abbr.**mime**

Bases: *base*

"Multipurpose Internet Mail Extensions"

class ll.xist.ns.abbr.**wysiwyg**

Bases: *base*

"What You See Is What You Get"

class ll.xist.ns.abbr.**svg**

Bases: *base*

"Scalable Vector Graphics"

class ll.xist.ns.abbr.**utc**

Bases: *base*

"Coordinated Universal Time"

class ll.xist.ns.abbr.**wsgi**

Bases: *base*

"Web Server Gateway Interface"

code – Embedding Python code in XML

An XIST module that allows embedding Python code in XML.

class ll.xist.ns.code.**pyexec**

Bases: _base

When converting a *pyexec* object the content of the processing instruction is executed as Python code. Execution is done when the node is converted. When converted such a node will result in an empty Null node.

These processing instructions will be evaluated and executed in the namespace of the module sandbox (which will be store in the converter context).

class ll.xist.ns.code.**pyeval**

Bases: _base

The content of a *pyeval* processing instruction will be executed when the node is converted as if it was the body of a function, so you can return an expression here. Although the content is used as a function body no indentation is necessary or allowed. The returned value will be converted to a node and this resulting node will be converted.

These processing instructions will be evaluated and executed in the namespace of the module sandbox (which will be store in the converter context).

Note that you should not define the symbol __ in any of your XIST processing instructions, as it is used by XIST for internal purposes.

convert(*converter*)

Evaluates the code as if it was the body of a Python function. The converter argument will be available under the name converter as an argument to the function.

form – Form related elements

This XIST module contains convenience classes for form elements. These are just abbreviations for the various `<input type="...">` elements.

php – PHP processing instructions

A module that allows you to embed PHP processing instructions.

class `ll.xist.ns.php.`**`php`**

Bases: *ProcInst*

PHP processing instruction (must be used with an explicit target php to work with XML)

jsp – JSP processing instructions

A module that allows you to embed JSP content as processing instructions.

class `ll.xist.ns.jsp.`**`scriptlet`**

Bases: *ProcInst*

Will be published as `<% content %>`.

class `ll.xist.ns.jsp.`**`expression`**

Bases: *ProcInst*

Will be published as `<%= content %>`.

class `ll.xist.ns.jsp.`**`declaration`**

Bases: *ProcInst*

Will be published as `<%! content %>`.

class `ll.xist.ns.jsp.`**`block`**

Bases: *Element*

This element embeds its content in `{}` brackets. Note that the content itself will not be turned into a scriptlet automatically but will be used as-is.

`ll.xist.ns.jsp.`**`fromul4`**(*template*, *variables='variables'*, *indent=0*)

Return the UL4 template `template` as JSP source code. `variables` is the variable name of the map object containing the top level variables. `indent` is the initial indentation of the source code.

The code produced requires the UL4 Java package.

meta – HTML meta information

An XIST module that contains elements that simplify handling meta data. All elements in this module will generate a *ll.xist.ns.html.meta* element when converted.

class `ll.xist.ns.meta.`**`contenttype`**

Bases: *meta*

Can be used for a `<meta http-equiv="Content-Type" content="text/html"/>`, where the character set will be automatically inserted on a call to *ll.xist.xsc.Node.publish()*.

Usage is simple: `meta.contenttype()`.

class ll.xist.ns.meta.**contentscripttype**

Bases: *meta*

Can be used for a `<meta http-equiv="Content-Script-Type" content="..."/>`.

Usage is simple: `<markup>meta.contentscripttype(type="text/javascript")`.

class ll.xist.ns.meta.**keywords**

Bases: *meta*

Can be used for a `<meta name="keywords" content="..."/>`.

Usage is simple: `meta.keywords("foo, bar")`.

class ll.xist.ns.meta.**description**

Bases: *meta*

Can be used for a `<meta name="description" content="..."/>`.

Usage is simple: `meta.description("This page describes the ...")`.

class ll.xist.ns.meta.**stylesheet**

Bases: *link*

Can be used for a `<link rel="stylesheet" type="text/css" href="..."/>`.

Usage is simple: `meta.stylesheet(href="root:stylesheets/main.css")`.

class ll.xist.ns.meta.**made**

Bases: *link*

Can be used for a `<link rel="made" href="mailto:..."/>`.

Usage is simple: `meta.made(href="foobert@bar.org")`.

class ll.xist.ns.meta.**author**

Bases: *Element*

Can be used to embed author information in the header. It will generate `<link rel="made"/>` and `<meta name="author"/>` elements.

class ll.xist.ns.meta.**refresh**

Bases: *Element*

A refresh header.

ruby – W3C Ruby annotations

An XIST module that contains definitions for all the elements in Ruby 1.0.

class ll.xist.ns.ruby.**DocTypeRuby10**

Bases: *DocType*

document type for Ruby 1.0

class ll.xist.ns.ruby.**rb**

Bases: *Element*

The *rb* element is the container for the text of the ruby base.

class ll.xist.ns.ruby.**rbc**

Bases: *Element*

The *rbc* (ruby base component) element is the container for *rb* elements.

class ll.xist.ns.ruby.**rp**

Bases: *Element*

The *rp* element is intended to contain parenthesis characters in simple ruby.

class ll.xist.ns.ruby.**rt**

Bases: *Element*

The *rt* element is the container for the ruby text.

class ll.xist.ns.ruby.**rtc**

Bases: *Element*

The *rtc* ("ruby text component") element is the container for *rt* elements.

class ll.xist.ns.ruby.**ruby**

Bases: *Element*

The *ruby* element is an inline (or text-level) element that serves as the container for either the *rb*, *rt* and optional *rp* elements or the *rbc* and *rtc* elements.

specials – Common useful elements

An XIST module that contains a collection of useful elements that can be used for all conversion targets, because they only generate text.

class ll.xist.ns.specials.**filesize**

Bases: *Element*

The size (in bytes) of the file whose URL is the attribute href as a text node.

class ll.xist.ns.specials.**filetime**

Bases: *Element*

The time of the last modification of the file whose URL is in the attribute `href` as a text node. This will always be an UTC timestamp.

class ll.xist.ns.specials.**time**

Bases: *Element*

the current time (i.e. the time when `convert()` is called). You can specify the format of the string in the attribute `format`, which is a `strftime()` compatible string.

class ll.xist.ns.specials.**ignore**

Bases: *Element*

Element that will be ignored when converted.

ignore can be used to comment out stuff. The content of the element must of course still be wellformed.

class ll.xist.ns.specials.**wrap**

Bases: *Element*

A wrapper element that returns its content when converted.

This is e.g. useful if you want to parse a file that starts with *ll.xist.ns.jsp* processing instructions.

class ll.xist.ns.specials.**literal**

Bases: *ProcInst*

literal is a processing instruction that will output its content literally when published.

class ll.xist.ns.specials.**url**

Bases: *ProcInst*

url is a processing instruction containing an URL. On publishing it will be replaced by an URL that is relative to the base URL of the publisher.

class ll.xist.ns.specials.**lf**

Bases: *CharRef*

line feed

class ll.xist.ns.specials.**cr**

Bases: *CharRef*

carriage return

class ll.xist.ns.specials.**tab**

Bases: *CharRef*

horizontal tab

class ll.xist.ns.specials.**esc**

Bases: *CharRef*

escape

htmlspecials – Elements for HTML generation

An XIST module that contains a collection of useful elements for generating HTML.

class ll.xist.ns.htmlspecials.**html**

Bases: *html*

Creates an *ll.xist.ns.html.html* element and automatically sets the `lang` and `xml:lang` attributes to the `converter`s configured language.

class ll.xist.ns.htmlspecials.**plaintable**

Bases: *table*

a HTML table where the values of the attributes `cellpadding`, `cellspacing` and `border` default to `0`.

class ll.xist.ns.htmlspecials.**plainbody**

Bases: *body*

a HTML body where the attributes `leftmargin`, `topmargin`, `marginheight` and `marginwidth` default to `0`.

class ll.xist.ns.htmlspecials.**pixel**

Bases: `_pixelbase`

Element for single transparent pixel image.

You can specify the pixel color via the `color` attribute (which will set the background-color in the style attribute).

In addition to that you can specify width and height attributes (and every other allowed attribute for the `img` element) as usual.

class ll.xist.ns.htmlspecials.**autoimg**

Bases: *img*

An image were width and height attributes are automatically generated.

If the attributes are already there, they won't be modified.

class ll.xist.ns.htmlspecials.**autopixel**

Bases: _pixelbase

A pixel image were width and height attributes are automatically generated.

This works like *pixel* but the size is "inherited" from the image specified via the `src` attribute.

class ll.xist.ns.htmlspecials.**autoinput**

Bases: *input*

Extends *ll.xist.ns.html.input* with the ability to automatically set the size, if this element has `type=="image"`.

class ll.xist.ns.htmlspecials.**javascript**

Bases: *script*

Can be used for javascript.

doc – Automated documentation generation

This namespace module provides classes that can be used for generating documentation (in HTML, DocBook and XSL-FO).

ll.xist.ns.doc.**getdoc**(*thing*, *format*)

Return the docstring for `thing`, as an XIST node using this namespace module.

`format` specifies how to treat the docstring:

"plaintext"
Treat to docstring as text. This returns a single *Text* node.

"restructuredtext"
This interprets the docstring as ReST source and converts it to use this namespace.

"xist"
This treats the docstring as XML, which will be parsed using this namespace as the default namespace.

ll.xist.ns.doc.**explain**(*thing*, *name=None*, *format=None*, *context=[]*)

Return a XML representation of the doc string of `thing`, which can be a function, method, class or module.

If `thing` is not a module, you must pass the context in `context`, i.e. a list of names of objects into which `thing` is nested. This means the first entry will always be the module name, and the other entries will be class names.

For the meaning for `format` see, *getdoc()*.

class ll.xist.ns.doc.**base**

Bases: *Element*

The base of all element classes. Used for dispatching to conversion targets.

class ll.xist.ns.doc.**block**

Bases: *base*

Base class for all block level elements.

class ll.xist.ns.doc.**inline**

Bases: *base*

Base class for all inline elements.

class ll.xist.ns.doc.**abbr**

Bases: *inline*

Abbreviation.

class ll.xist.ns.doc.**tab**

Bases: *Element*

Used for displaying a tab character in the HTML output.

class ll.xist.ns.doc.**litblock**

Bases: *block*

A literal text block (like source code or a shell session).

class ll.xist.ns.doc.**prog**

Bases: *litblock*

A literal listing of all or part of a program.

class ll.xist.ns.doc.**tty**

Bases: *litblock*

A dump of a shell session.

class ll.xist.ns.doc.**prompt**

Bases: *inline*

The prompt in a *tty* dump.

class ll.xist.ns.doc.**input**

Bases: *inline*

Can be used inside a *tty* to mark the parts typed by the user.

class ll.xist.ns.doc.**rep**

Bases: *inline*

Content that may or must be replaced by the user.

class ll.xist.ns.doc.**option**

Bases: code

An option for a software command.

class ll.xist.ns.doc.**lit**

Bases: code

Inline text that is some literal value.

class ll.xist.ns.doc.**func**

Bases: code

The name of a function or subroutine, as in a programming language.

class ll.xist.ns.doc.**meth**

Bases: code

The name of a method or memberfunction in a programming language.

class ll.xist.ns.doc.**attr**

Bases: code

The name of an attribute of a class/object.

class ll.xist.ns.doc.**prop**

Bases: code

The name of a property in a programming language.

class ll.xist.ns.doc.**class_**

Bases: code

The name of a class, in the object-oriented programming sense.

class ll.xist.ns.doc.**exc**

Bases: code

The name of an exception class.

class ll.xist.ns.doc.**markup**

Bases: code

A string of formatting markup in text that is to be represented literally.

class ll.xist.ns.doc.**self**

Bases: code

Use this class when referring to the object for which a method has been called, e.g.:

```
this function fooifies the object <self/>;.
```

ll.xist.ns.doc.**self_**

alias of *self*

class ll.xist.ns.doc.**cls**

Bases: *inline*

Use this class when referring to the object for which a class method has been called, e.g.:

```
this function fooifies the class <cls/>.
```

class ll.xist.ns.doc.**obj**

Bases: code

A object of unspecified type.

class ll.xist.ns.doc.**mod**

Bases: code

The name of a Python module.

class ll.xist.ns.doc.**file**

Bases: code

The name of a file.

class ll.xist.ns.doc.**dir**

Bases: code

The name of a directory.

class ll.xist.ns.doc.**user**

Bases: code

The name of a user account.

class ll.xist.ns.doc.**host**

Bases: code

The name of a computer.

class ll.xist.ns.doc.**const**

Bases: code

The name of a constant.

class ll.xist.ns.doc.**data**

Bases: code

The name of a data object.

class ll.xist.ns.doc.**app**

Bases: *inline*

The name of a software program.

class ll.xist.ns.doc.**h**

Bases: *base*

The text of the title of a *section* or an *example*.

class ll.xist.ns.doc.**section**

Bases: *block*

A recursive section.

class ll.xist.ns.doc.**p**

Bases: *block*

A paragraph.

class ll.xist.ns.doc.**dt**

Bases: *block*

A term inside a *dl*.

class ll.xist.ns.doc.**li**

Bases: *block*

A wrapper for the elements of a list item in *ul* or *ol*.

class ll.xist.ns.doc.**dd**

Bases: *block*

A wrapper for the elements of a list item *dl*.

class ll.xist.ns.doc.**list**

Bases: *block*

Common baseclass for *ul*, *ol* and *dl*.

class ll.xist.ns.doc.**ul**

Bases: *list*

A list in which each entry is marked with a bullet or other dingbat.

class ll.xist.ns.doc.**ol**

Bases: *list*

A list in which each entry is marked with a sequentially incremented label.

class ll.xist.ns.doc.**dl**

Bases: *list*

A list in which each entry is marked with a label.

class ll.xist.ns.doc.**example**

Bases: *block*

A formal example.

class ll.xist.ns.doc.**a**

Bases: *inline*

A hypertext link.

class ll.xist.ns.doc.**xref**

Bases: *inline*

An internal cross reference.

class ll.xist.ns.doc.**email**

Bases: *inline*

An email address.

class ll.xist.ns.doc.**em**

Bases: *inline*

Emphasized text.

class ll.xist.ns.doc.**strong**

Bases: *inline*

Emphasized text.

class ll.xist.ns.doc.**z**

Bases: *inline*

Put the content into double quotes.

class ll.xist.ns.doc.**pyref**

Bases: *inline*

Reference to a Python object: module, class, method, property or function.

detox – Detox templates

This module is an XIST namespace. It provides a simple template language based on processing instructions embedded in XML or plain text.

The following example is a simple "Hello, World" type template:

```
from ll.xist.ns import detox

template = """
<?def helloworld(n=10)?>
   <?for i in range(n)?>
      Hello, World!
   <?end for?>
<?end def?>
"""

module = detox.xml2mod(template)

print "".join(module.helloworld())
```

class ll.xist.ns.detox.**expr**

Bases: *ProcInst*

Embed the value of the expression

class ll.xist.ns.detox.**code**

Bases: *ProcInst*

Embed the PI data literally in the generated code.

For example `<?code foo = 42?>` will put the statement `foo = 42` into the generated Python source.

class ll.xist.ns.detox.**if_**

Bases: *ProcInst*

Starts an if block. An if block can contain zero or more *elif_* blocks, followed by zero or one *else_* block and must be closed with an *end* PI.

For example:

```
<?code import random?>
<?code n = random.choice("123?")?>
<?if n == "1"?>
   One
<?elif n == "2"?>
   Two
<?elif n == "3"?>
   Three
<?else?>
   Something else
<?end if?>
```

class ll.xist.ns.detox.**elif_**

Bases: *ProcInst*

Starts an elif block.

class ll.xist.ns.detox.**else_**

Bases: *ProcInst*

Starts an else block.

class ll.xist.ns.detox.**def_**

Bases: *ProcInst*

Start a function (or method) definition. A function definition must be closed with an *end* PI.

Example:

```
<?def persontable(persons)?>
   <table>
      <tr>
         <th>first name</th>
         <th>last name</th>
      </tr>
      <?for person in persons?>
         <tr>
            <td><?textexpr person.firstname?></td>
            <td><?textexpr person.lastname?></td>
         </tr>
      <?end for?>
   </table>
<?end def?>
```

If the generated function contains output (i.e. if there is text content or *expr*, `textexpr` or `attrexpr` PIs before the matching *end*) the generated function will be a generator function.

Output outside of a function definition will be ignored.

class ll.xist.ns.detox.**class_**

Bases: *ProcInst*

Start a class definition. A class definition must be closed with an *end* PI.

Example:

```
<?class mylist(list)?>
   <?def output(self)?>
      <ul>
         <?for item in self?>
            <li><?textexpr item?></li>
         <?endfor?>
      </ul>
   <?end def?>
<?end class?>
```

class ll.xist.ns.detox.**for_**

Bases: *ProcInst*

Start a `for` loop. A for loop must be closed with an *end* PI.

For example:

```
<ul>
   <?for i in range(10)?>
      <li><?expr str(i)?></li>
   <?end for?>
</ul>
```

class ll.xist.ns.detox.**while_**

Bases: *ProcInst*

Start a `while` loop. A while loop must be closed with an *end* PI.

For example:

```
<?code i = 0?>
<ul>
   <?while True?>
      <li><?expr str(i)?><?code i += 1?></li>
      <?code if i > 10: break?>
   <?end while?>
</ul>
```

class ll.xist.ns.detox.**end**

Bases: *ProcInst*

Ends a *while_* or *for_* loop or a *if_*, *def_* or *class_* block.

`toxic` – Embed PL/SQL in XIST XML

This module is an XIST namespace. It provides the processing instruction classes needed to create TOXIC functions and procedures via XIST. For more info about the TOXIC compiler see the module `ll.toxicc`.

class ll.xist.ns.toxic.**args**

Bases: *ProcInst*

Specifies the arguments to be used by the generated function. For example:

```
<?args
   key in integer,
   lang in varchar2
?>
```

for Oracle, or:

```
<?args
   @key int,
   @lang varchar(10)
?>
```

for SQL Server. If *args* is used multiple times, the contents will be concatenated with a `,` in between.

class `ll.xist.ns.toxic.`**vars**

Bases: *ProcInst*

Specifies the local variables to be used by the function. For example:

```
<?vars
   buffer varchar2(200) := 'foo';
   counter integer;
?>
```

for Oracle, or:

```
<?vars
   declare @buffer varchar(200) := 'foo';
   declare @counter int;
?>
```

If *vars* is used multiple times, the contents will simple be concatenated. (Note that for SQL Server this could be done via a normal *code* PI too.)

class `ll.xist.ns.toxic.`**code**

Bases: *ProcInst*

A SQL code fragment that will be embedded literally in the generated function. For example:

```
<?code select user into v_user from dual;?>
```

for Oracle, or:

```
<?code set @user = user;?>
```

for SQL Server

class `ll.xist.ns.toxic.`**expr**

Bases: *ProcInst*

The data of an *expr* processing instruction must contain a SQL expression whose value will be embedded in the string returned by the generated function. This value will not be escaped in any way, so you can generate XML tags with *expr* PIs but you must make sure to generate the value in the encoding that the caller of the generated function expects.

class `ll.xist.ns.toxic.`**proc**

Bases: *ProcInst*

When this processing instruction is found in the source `compile()` will not generate a function as a result, but a procedure. This procedure must have `c_out` as an "out" variable (of the appropriate type (see *type*) where the output will be written to.

class ll.xist.ns.toxic.**type**

Bases: *ProcInst*

Can be used to specify the return type of the generated function/procedure. The default is `clob` for Oracle and `varchar(max)` for SQL Server.

rng – Relax NG

This module is an XIST namespace for Relax NG files.

class ll.xist.ns.rng.**base**

Bases: *Element*

"Abstract" basis class, providing common attributes.

class ll.xist.ns.rng.**anyName**

Bases: *base*

Matches any name from any namespace.

class ll.xist.ns.rng.**attribute**

Bases: *base*

Specifies an XML attribute.

class ll.xist.ns.rng.**choice**

Bases: *base*

nameclass: a name matches choice if, and only if, it matches at least one of the subname classes. pattern: it matches a node if, and only if, at least one of its subpatterns matches the node

class ll.xist.ns.rng.**data**

Bases: *base*

Specifies data of a certain kind.

class ll.xist.ns.rng.**define**

Bases: *base*

Defines a part of a grammar pattern (also a pattern), recursion possible only inside an element.

class ll.xist.ns.rng.**div**

Bases: *base*

Allows logical divisions, no effect on validation, annotations can be made here

class ll.xist.ns.rng.**element_**

Bases: *base*

Specifies an XML element.

class ll.xist.ns.rng.**empty**

Bases: *base*

Specifies empty content.

class ll.xist.ns.rng.**except_**

Bases: *base*

An *except_* element can remove a name class from another (this class has no attributes) (inside a *name* element) or it is used to remove a set of values from a data pattern.

class ll.xist.ns.rng.**externalRef**

Bases: *base*

Reference to an extern pattern stored in a file.

class ll.xist.ns.rng.**grammar**

Bases: *base*

A *grammar* element has a single *start* child element, and zero or more *define* child elements. The *start* and *define* elements contain patterns. These patterns can contain *ref* elements that refer to patterns defined by any of the *define* elements in that grammar element. A *grammar* pattern is matched by matching the pattern contained in the *start* element.

class ll.xist.ns.rng.**group**

Bases: *base*

Is implied, can be also explicitly specified: the patterns have to appear in the specified order (except for the attributes, they are allowed to appear in any order in the start tag)

class ll.xist.ns.rng.**include**

Bases: *base*

Includes an extern grammar pattern. Can contain define parts to overwrite that part (same name) in the extern pattern. A possible start element inside include overwrites the start element of the extern pattern.

class ll.xist.ns.rng.**interleave**

Bases: *base*

Child elements can appear in any order, if one is a group, the order must be kept, other direct childs can mix between.

class ll.xist.ns.rng.**list**

Bases: *base*

Matches whitespace separated values.

class ll.xist.ns.rng.**mixed**

Bases: *base*

`<mixed> p </mixed>` is short for `<interleave> <text/> p </interleave>`

class ll.xist.ns.rng.**name**

Bases: *base*

Defines a class with a single name.

class ll.xist.ns.rng.**notAllowed**

Bases: *base*

Used to make extension points in patterns.

class ll.xist.ns.rng.**nsName**

Bases: *base*

Allows any name in a specific namespace.

class ll.xist.ns.rng.**oneOrMore**

Bases: *base*

There can be one or more recurrence of the enclosed pattern.

class ll.xist.ns.rng.**optional**

Bases: *base*

The enclosed tags can be left out.

class ll.xist.ns.rng.**param**

Bases: *base*

Specifies parameters passed to the datatype library to determine whether a value is valid per a datatype.

class ll.xist.ns.rng.**parentRef**

Bases: *base*

Escapes out of the current grammar and references a definition from the parent of the current grammar.

class ll.xist.ns.rng.**ref**

Bases: *base*

A *ref* pattern refers to a definition from the nearest grammar ancestor.

class ll.xist.ns.rng.**start**

Bases: *base*

Required start tag inside a *grammar* tag.

class ll.xist.ns.rng.**text**

Bases: *base*

Matches arbitrary text (one or more text nodes), including empty text.

class ll.xist.ns.rng.**value**

Bases: *base*

By default, the *value* pattern will consider the string in the pattern to match the string in the document if the two strings are the same after the whitespace in both strings is normalized. Whitespace normalization strips leading and trailing whitespace characters, and collapses sequences of one or more whitespace characters to a single space character. This corresponds to the behaviour of an XML parser for an attribute that is declared as other than CDATA.

class ll.xist.ns.rng.**zeroOrMore**

Bases: *base*

There can be zero or more recurrence of the enclosed pattern.

rss091 – RSS 0.91

This is a namespace module implementing RSS 0.91.

class ll.xist.ns.rss091.**DocType**

Bases: *DocType*

Document type for RSS 0.91

class ll.xist.ns.rss091.**channel**

Bases: *Element*

Information about a particular channel. Everything pertaining to an individual channel is contained within this tag.

class ll.xist.ns.rss091.**copyright**

Bases: *Element*

Copyright string.

class ll.xist.ns.rss091.**day**

Bases: *Element*

The day of the week, spelled out in English.

class ll.xist.ns.rss091.**description**

Bases: *Element*

A plain text description of an *item*, *channel*, *image*, or *textinput*.

class ll.xist.ns.rss091.**docs**

Bases: *Element*

This tag should contain a URL that references a description of the channel.

class ll.xist.ns.rss091.**height**

Bases: *Element*

Specifies the height of an *image*. Should be an integer value.

class ll.xist.ns.rss091.**hour**

Bases: *Element*

Specifies an hour of the day. Should be an integer value between 0 and 23. See *skipHours*.

class ll.xist.ns.rss091.**image**

Bases: *Element*

Specifies an image associated with a *channel*

class ll.xist.ns.rss091.**item**

Bases: *Element*

An item that is associated with a *channel*. The item should represent a web-page, or subsection within a web page. It should have a unique URL associated with it. Each item must contain a *title* and a *link*. A *description* is optional.

class ll.xist.ns.rss091.**language**

Bases: *Element*

Specifies the language of a channel.

class ll.xist.ns.rss091.**lastBuildDate**

Bases: *Element*

The last time the channel was modified.

class ll.xist.ns.rss091.**link**

Bases: *Element*

This is a url that a user is expected to click on, as opposed to a *url* that is for loading a resource, such as an image.

class ll.xist.ns.rss091.**managingEditor**

Bases: *Element*

The email address of the managing editor of the site, the person to contact for editorial inquiries.

class ll.xist.ns.rss091.**name**

Bases: *Element*

The name of an object, corresponding to the `name` attribute of an HTML `input` element. Currently, this only applies to *textinput*.

class ll.xist.ns.rss091.**pubDate**

Bases: *Element*

Date when channel was published.

class ll.xist.ns.rss091.**rating**

Bases: *Element*

PICS rating of the channel.

class ll.xist.ns.rss091.**rss**

Bases: *Element*

Root element.

class ll.xist.ns.rss091.**skipDays**

Bases: *Element*

A list of *day*s of the week, in English, indicating the days of the week when your channel will not be updated. As with `activeHours`, if you know your channel will never be updated on Saturday or Sunday, for example.

class ll.xist.ns.rss091.**skipHours**

Bases: *Element*

A list of *hour*s indicating the hours in the day, GMT, when the channel is unlikely to be updated. If this sub-item is omitted, the channel is assumed to be updated hourly.

class ll.xist.ns.rss091.**textinput**

Bases: *Element*

An input field for the purpose of allowing users to submit queries back to the publisher's site.

class ll.xist.ns.rss091.**title**

Bases: *Element*

An identifying string for a resource. When used in an *item*, this is the name of the item's link. When used in an *image*, this is the "alt" text for the image. When used in a *channel*, this is the channel's title. When used in a *textinput*, this is the textinput's title.

class ll.xist.ns.rss091.**url**

Bases: *Element*

Location to load a resource from. Note that this is slightly different from the *link* tag, which specifies where a user should be re-directed to if a resource is selected.

class ll.xist.ns.rss091.**webMaster**

Bases: *Element*

The email address of the webmaster for the site, the person to contact if there are technical problems with the channel.

class ll.xist.ns.rss091.**width**

Bases: *Element*

Specifies the width of an *image*. Should be an integer value.

`rss20` – RSS 2.0

This is a namespace module implementing RSS 2.0.

class ll.xist.ns.rss20.**rss**

Bases: *Element*

The root element.

class ll.xist.ns.rss20.**channel**

Bases: *Element*

Information about a particular channel. Everything pertaining to an individual channel is contained within this tag.

class ll.xist.ns.rss20.**title**

Bases: *Element*

The name of the *channel*, *item*, *image* or *textInput*.

class ll.xist.ns.rss20.**link**

Bases: *Element*

The URLto the HTML website corresponding to the *channel*, *item* or *image*. Inside *textInput* element it's the URL of the CGI script that processes text input requests.

class ll.xist.ns.rss20.**description**

Bases: *Element*

Phrase or sentence describing the *channel*, *item* or *textInput*.

class ll.xist.ns.rss20.**language**

Bases: *Element*

The language the channel is written in.

class ll.xist.ns.rss20.**copyright**

Bases: *Element*

Copyright notice for content in the channel.

class ll.xist.ns.rss20.**managingEditor**

Bases: *Element*

Email address for person responsible for editorial content.

class ll.xist.ns.rss20.**webMaster**

Bases: *Element*

Email address for person responsible for technical issues relating to channel.

class ll.xist.ns.rss20.**pubDate**

Bases: *Element*

The publication date for the content in the channel.

class ll.xist.ns.rss20.**lastBuildDate**

Bases: *Element*

The last time the content of the channel changed.

class ll.xist.ns.rss20.**category**

Bases: *Element*

Specify one or more categories that the channel or item belongs to.

class ll.xist.ns.rss20.**generator**

Bases: *Element*

A string indicating the program used to generate the channel.

class ll.xist.ns.rss20.**docs**

Bases: *Element*

A URL that points to the documentation for the format used in the RSS file.

class ll.xist.ns.rss20.**cloud**

Bases: *Element*

Allows processes to register with a cloud to be notified of updates to the channel, implementing a lightweight publish-subscribe protocol for RSS feeds.

class ll.xist.ns.rss20.**ttl**

Bases: *Element*

ttl stands for time to live. It's a number of minutes that indicates how long a channel can be cached before refreshing from the source.

class ll.xist.ns.rss20.**image**

Bases: *Element*

Specifies a GIF, JPEG or PNG image that can be displayed with the channel.

class ll.xist.ns.rss20.**textInput**

Bases: *Element*

Specifies a text input box that can be displayed with the channel.

class ll.xist.ns.rss20.**name**

Bases: *Element*

The name of the text object in the *textInput* area.

class ll.xist.ns.rss20.**skipHours**

Bases: *Element*

A hint for aggregators telling them which hours they can skip.

class ll.xist.ns.rss20.**skipDays**

Bases: *Element*

A hint for aggregators telling them which days they can skip.

class ll.xist.ns.rss20.**day**

Bases: *Element*

The day of the week, spelled out in English.

class ll.xist.ns.rss20.**hour**

Bases: *Element*

Specifies an hour of the day. Should be an integer value between 0 and 23. See *skipHours*.

class ll.xist.ns.rss20.**url**

Bases: *Element*

The URL of a GIF, JPEG or PNG image that represents the channel.

class ll.xist.ns.rss20.**width**

Bases: *Element*

Image width.

class ll.xist.ns.rss20.**height**

Bases: *Element*

Image height.

class ll.xist.ns.rss20.**item**

Bases: *Element*

An item that is associated with a *channel*. The item should represent a web-page, or subsection within a web page. It should have a unique URL associated with it. Each item must contain a *title* or *description*.

class ll.xist.ns.rss20.**author**

Bases: *Element*

Author of an *item*.

class ll.xist.ns.rss20.**comments**

Bases: *Element*

URL of a page for comments relating to the item.

class ll.xist.ns.rss20.**enclosure**

Bases: *Element*

Describes a media object that is attached to the item.

class ll.xist.ns.rss20.**guid**

Bases: *Element*

A string that uniquely identifies the item.

class ll.xist.ns.rss20.**source**

Bases: *Element*

The RSS channel that the item came from.

atom – Atom 1.0

This namespace module implements Atom 1.0 as specified by **RFC 4287**.

class ll.xist.ns.atom.**feed**

Bases: *Element*

The *feed* element 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.

class ll.xist.ns.atom.**entry**

Bases: *Element*

The *entry* element represents an individual entry, acting as a container for metadata and data associated with the entry.

class ll.xist.ns.atom.**content**

Bases: *Element*

The *content* element either contains or links to the content of the *entry*.

class ll.xist.ns.atom.**author**

Bases: *Element*

The *author* element indicates the author of the *entry* or *feed*.

class ll.xist.ns.atom.**category**

Bases: *Element*

The *category* element conveys information about a category associated with an *entry* or *feed*.

class ll.xist.ns.atom.**contributor**

Bases: *Element*

The *contributor* element indicates a person or other entity who contributed *entry* or *feed*.

class ll.xist.ns.atom.**generator**

Bases: *Element*

The *generator* element's content identifies the agent used to generate a feed, for debugging and other purposes.

class ll.xist.ns.atom.**icon**

Bases: *Element*

The *icon* element's content is an IRI reference that identifies an image that provides iconic visual identification for a feed.

class ll.xist.ns.atom.**id**

Bases: *Element*

The *id* element conveys a permanent, universally unique identifier for an *entry* or *feed*.

class ll.xist.ns.atom.**link**

Bases: *Element*

The *link* element defines a reference from an *entry* or *feed* to a Web resource.

class ll.xist.ns.atom.**logo**

Bases: *Element*

The *logo* element's content is an IRI reference that identifies an image that provides visual identification for a *feed*.

class ll.xist.ns.atom.**published**

Bases: *Element*

The *published* element indicatesg an instant in time associated with an event early in the life cycle of the *entry*.

class ll.xist.ns.atom.**rights**

Bases: *Element*

The *rights* element contains text that conveys information about rights held in and over an *entry* or *feed*.

class ll.xist.ns.atom.**source**

Bases: *Element*

If an *entry* is copied from one *feed* into another *feed*, then the source *feed*'s metadata (all child elements of *feed* other than the *entry* elements) may be preserved within the copied entry by adding a *source* child element, if it is not already present in the *entry*, and including some or all of the source *feed*'s Metadata elements as the *source* element's children.

class ll.xist.ns.atom.**subtitle**

Bases: *Element*

The *subtitle* element contains text that conveys a human-readable description or subtitle for a *feed*.

class ll.xist.ns.atom.**summary**

Bases: *Element*

The *summary* element contains text that conveys a short summary, abstract, or excerpt of an entry.

class ll.xist.ns.atom.**title**

Bases: *Element*

The *title* element contains text that conveys a human-readable title for an *entry* or *feed*.

class ll.xist.ns.atom.**updated**

Bases: *Element*

The *updated* element contains a date indicating the most recent instant in time when an *entry* or *feed* was modified in a way the publisher considers significant.

class ll.xist.ns.atom.**email**

Bases: *Element*

The *email* element's content conveys an e-mail address associated with the person.

class ll.xist.ns.atom.**uri**

Bases: *Element*

The *uri* element's content conveys an IRI associated with the person.

class ll.xist.ns.atom.**name**

Bases: *Element*

The *name* element's content conveys a human-readable name for the person.

struts_html – Jakarta Struts HTML tags

A module that allows you to embed tags from Struts html custom tag library.

class ll.xist.ns.struts_html.**taglib**

Bases: *ProcInst*

Creates a standard struts taglib header

class ll.xist.ns.struts_html.**Element**

Bases: *Element*

Common base class for all the struts html elements

class ll.xist.ns.struts_html.**MouseElement**

Bases: PartMouseElement

Common base class for all the struts elements which have mouse attributes

class ll.xist.ns.struts_html.**base**

Bases: *Element*

Document base URI

class ll.xist.ns.struts_html.**button**

Bases: *MouseElement*

A button

class ll.xist.ns.struts_html.**cancel**

Bases: *MouseElement*

A cancel button

class ll.xist.ns.struts_html.**checkbox**

Bases: *MouseElement*

A html checkbox element

class ll.xist.ns.struts_html.**errors**

Bases: *Element*

Displays error messages which have been generated from an action or a validation method

class ll.xist.ns.struts_html.**file**

Bases: *MouseElement*

HTML input element of type file

class ll.xist.ns.struts_html.**form**

Bases: *Element*

HTML form

class ll.xist.ns.struts_html.**frame**

Bases: *Element*

Render an HTML frame element

class ll.xist.ns.struts_html.**hidden**

Bases: PartMouseElement

hidden form field

class ll.xist.ns.struts_html.**html**

Bases: *Element*

Render a HTML html element

class ll.xist.ns.struts_html.**image**

Bases: *MouseElement*

image input

class ll.xist.ns.struts_html.**img**

Bases: *Element*

html img tag

class ll.xist.ns.struts_html.**javascript**

Bases: *Element*

Render JavaScript validation based on the validation rules loaded by the ValidatorPlugIn.

class ll.xist.ns.struts_html.**link**

Bases: *Element*

html link

class ll.xist.ns.struts_html.**messages**

Bases: *Element*

Conditionally display a set of accumulated messages.

class ll.xist.ns.struts_html.**multibox**

Bases: *MouseElement*

multiple checkbox element

class ll.xist.ns.struts_html.**option**

Bases: *Element*

option element

class ll.xist.ns.struts_html.**options**

Bases: *Element*

struts html options element

class ll.xist.ns.struts_html.**optionsCollection**

Bases: *Element*

Render a collection of select options

class ll.xist.ns.struts_html.**password**

Bases: *MouseElement*

a password text input field

class ll.xist.ns.struts_html.**radio**

Bases: *MouseElement*

html input radio

class ll.xist.ns.struts_html.**reset**

Bases: *MouseElement*

a reset button

class ll.xist.ns.struts_html.**rewrite**

Bases: *Element*

render a request uri like html link

class ll.xist.ns.struts_html.**select**

Bases: PartMouseElement

a select element text input field

class ll.xist.ns.struts_html.**submit**

Bases: *MouseElement*

a submit button

class ll.xist.ns.struts_html.**text**

Bases: *MouseElement*

a text input field

class ll.xist.ns.struts_html.**textarea**

Bases: *MouseElement*

a textarea

class ll.xist.ns.struts_html.**xhtml**

Bases: *Element*

Render HTML tags as XHTML

struts_config – Struts configuration files

Namespace module for Struts configuration files: http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd.

6.1.9 parse – Parsing XML/HTML

This module contains everything you need to create XIST objects by parsing files, strings, URLs etc.

Parsing XML is done with a pipelined approach. The first step in the pipeline is a source object that provides the input for the rest of the pipeline. The next step is the XML parser. It turns the input source into an iterator over parsing events (an "event stream"). Further steps in the pipeline might resolve namespace prefixes (*NS*), and instantiate XIST classes (*Node*). The final step in the pipeline is either building an XML tree via *tree()* or an iterative parsing step (similar to ElementTrees `iterparse()` function) via *itertree()*.

Parsing a simple HTML string might e.g. look like this:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import html
>>> source = b"<a href='http://www.python.org/'>Python</a>"
>>> doc = parse.tree(
...   parse.String(source),
...   parse.Expat(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(html)),
... )
>>> doc.string()
'<a href="http://www.python.org/">Python</a>'
```

A source object is an iterable object that produces the input byte string for the parser (possibly in multiple chunks) (and information about the URL of the input):

```
>>> from ll.xist import parse
>>> list(parse.String(b"<a href='http://www.python.org/'>Python</a>"))
[('url', URL('STRING')),
 ('bytes', "<a href='http://www.python.org/'>Python</a>")]
```

All subsequent objects in the pipeline are callable objects, get the input iterator as an argument and return an iterator over events themselves. The following code shows an example of an event stream:

```
>>> from ll.xist import parse
>>> source = b"<a href='http://www.python.org/'>Python</a>"
>>> list(parse.events(parse.String(source), parse.Expat()))
[('url', URL('STRING')),
 ('position', (0, 0)),
 ('enterstarttag', 'a'),
 ('enterattr', 'href'),
 ('text', 'http://www.python.org/'),
 ('leaveattr', 'href'),
 ('leavestarttag', 'a'),
 ('position', (0, 39)),
 ('text', 'Python'),
 ('endtag', 'a')]
```

An event is a tuple consisting of the event type and the event data. Different stages in the pipeline produce different event types. The following event types can be produced by source objects:

"url"
The event data is the URL of the source. Usually such an event is produced only once at the start of the event stream. For sources that have no natural URL (like strings or streams) the URL can be specified when creating the source object.

"bytes"
This event is produced by source objects (and *Transcoder* objects). The event data is a byte string.

"str"
The event data is a string. This event is produced by *Decoder* or source objects. Note that the only predefined pipeline objects that can handle "str" events are *Encoder* objects, i.e. normally a parser handles "bytes" events, but not "str" events.

The following type of events are produced by parsers (in addition to the "url" event from above):

"position"
The event data is a tuple containing the line and column number in the source (both starting with 0). All the following events should use this position information until the next position event.

"xmldecl"
The XML declaration. The event data is a dictionary containing the keys "version", "encoding" and "standalone". Parsers may omit this event.

"begindoctype"
The begin of the doctype. The event data is a dictionary containing the keys "name", "publicid" and "systemid". Parsers may omit this event.

"enddoctype"
The end of the doctype. The event data is None. (If there is no internal subset, the "enddoctype" event immediately follows the "begindoctype" event). Parsers may omit this event.

"comment"
A comment. The event data is the content of the comment.

"text"
Text data. The event data is the text content. Parsers should try to avoid outputting multiple text events in sequence.

"cdata"
A CDATA section. The event data is the content of the CDATA section. Parsers may report CDATA sections as "text" events instead of "cdata" events.

"enterstarttag"
The beginning of an element start tag. The event data is the element name.

"leavestarttag"
The end of an element start tag. The event data is the element name. The parser will output events for the attributes between the "enterstarttag" and the "leavestarttag" event.

"enterattr"
The beginning of an attribute. The event data is the attribute name.

"leaveattr"
The end of an attribute. The event data is the attribute name. The parser will output events for the attribute value between the "enterattr" and the "leaveattr" event. (In almost all cases this is one text event).

"endtag"
An element end tag. The event data is the element name.

"procinst"
A processing instruction. The event data is a tuple consisting of the processing instruction target and the data.

"entity"
An entity reference. The event data is the entity name.

The following events are produced for elements and attributes in namespace mode (instead of those without the ns suffix). They are produced by *NS* objects or by *Expat* objects when the ns argument is true (i.e. the expat parser performs the namespace resolution):

"enterstarttagns"
The beginning of an element start tag in namespace mode. The event data is an (namespace name, element name) tuple.

"leavestarttagns"
The end of an element start tag in namespace mode. The event data is an (namespace name, element name) tuple.

"enterattrns"
The beginning of an attribute in namespace mode. The event data is an (namespace name, element name) tuple.

"leaveattrns"
The end of an attribute in namespace mode. The event data is an (namespace name, element name) tuple.

"endtagns"
An element end tag in namespace mode. The event data is an (namespace name, element name) tuple.

Once XIST nodes have been instantiated (by *Node* objects) the following events are used:

"xmldeclnode"
The XML declaration. The event data is an instance of *ll.xist.ns.xml.XML*.

"doctypenode"
The doctype. The event data is an instance of *ll.xist.xsc.DocType*.

"commentnode"
A comment. The event data is an instance of *ll.xist.xsc.Comment*.

"textnode"
Text data. The event data is an instance of *ll.xist.xsc.Text*.

"enterelementnode"
The beginning of an element. The event data is an instance of *ll.xist.xsc.Element* (or one of its subclasses). The attributes of the element object are set, but the element has no content yet.

"leaveelementnode"
The end of an element. The event data is an instance of *ll.xist.xsc.Element*.

"procinstnode"
A processing instruction. The event data is an instance of *ll.xist.xsc.ProcInst*.

"entitynode"
An entity reference. The event data is an instance of *ll.xist.xsc.Entity*.

For consuming event streams there are three functions:

events()
This generator simply outputs the events.

tree()
This function builds an XML tree from the events and returns it.

itertree()
This generator builds a tree like *tree()*, but returns events during certain steps in the parsing process.

Example

The following example shows a custom generator in the pipeline that lowercases all element and attribute names:

```
from ll.xist import xsc, parse
from ll.xist.ns import html

def lowertag(input):
   for (event, data) in input:
      if event in {"enterstarttag", "leavestarttag", "endtag", "enterattr", "leaveattr
↪"}:
         data = data.lower()
      yield (event, data)

e = parse.tree(
   parse.String(b"<A HREF='gurk'><B>gurk</B></A>"),
   parse.Expat(),
   lowertag,
   parse.NS(html),
   parse.Node(pool=xsc.Pool(html))
)

print(e.string())
```

This scripts outputs:

```
<a href="gurk"><b>gurk</b></a>
```

exception ll.xist.parse.**UnknownEventError**

Bases: TypeError

This exception is raised when a pipeline object doesn't know how to handle an event.

class ll.xist.parse.**String**

Bases: object

Provides parser input from a string.

__init__(*data*, *url=None*)
Create a *String* object. `data` must be a `bytes` or `str` object. `url` specifies the URL for the source (defaulting to `"STRING"`).

__iter__()

Produces an event stream of one "url" event and one "bytes" or "str" event for the data.

class ll.xist.parse.**Iter**

Bases: object

Provides parser input from an iterator over strings.

__init__(*iterable*, *url=None*)

Create a *Iter* object. iterable must be an iterable object producing bytes or str objects. url specifies the URL for the source (defaulting to "ITER").

__iter__()

Produces an event stream of one "url" event followed by the "bytes"/"str" events for the data from the iterable.

class ll.xist.parse.**Stream**

Bases: object

Provides parser input from a stream (i.e. an object that provides a read() method).

__init__(*stream*, *url=None*, *bufsize=8192*)

Create a *Stream* object. stream must have a read() method (with a size argument). url specifies the URL for the source (defaulting to "STREAM"). bufsize specifies the chunksize for reads from the stream.

__iter__()

Produces an event stream of one "url" event followed by the "bytes"/"str" events for the data from the stream.

class ll.xist.parse.**File**

Bases: object

Provides parser input from a file.

__init__(*filename*, *bufsize=8192*)

Create a *File* object. filename is the name of the file and may start with ~ or ~user for the home directory of the current or the specified user. bufsize specifies the chunksize for reads from the file.

__iter__()

Produces an event stream of one "url" event followed by the "bytes" events for the data from the file.

class ll.xist.parse.**URL**

Bases: object

Provides parser input from a URL.

__init__(*name*, *bufsize=8192*, **args*, ***kwargs*)

Create a *URL* object. name is the URL. bufsize specifies the chunksize for reads from the URL. args and kwargs will be passed on to the open() method of the URL object.

The URL for the input will be the final URL for the resource (i.e. it will include redirects).

__iter__()

Produces an event stream of one "url" event followed by the "bytes" events for the data from the URL.

class ll.xist.parse.**ETree**

Bases: object

Produces a (namespaced) event stream from an object that supports the ElementTree API.

__init__(*data*, *url=None*, *defaultxmlns=None*)

Create an *ETree* object. Arguments have the following meaning:

data
An object that supports the ElementTree API.

url
The URL of the source. Defaults to "ETREE".

defaultxmlns
The namespace name (or a namespace module containing a namespace name) that will be used for all elements that don't have a namespace.

__iter__()

Produces an event stream of namespaced parsing events for the ElementTree object passed as data to the constructor.

class ll.xist.parse.**Decoder**

Bases: object

Decode the bytes object produced by the previous object in the pipeline to str object.

This input object can be a source object or any other pipeline object that produces bytes objects.

__init__(*encoding=None*)

Create a *Decoder* object. encoding is the encoding of the input. If encoding is None it will be automatically detected from the XML data.

class ll.xist.parse.**Encoder**

Bases: object

Encode the str objects produced by the previous object in the pipeline to bytes objects.

This input object must be a pipeline object that produces string output (e.g. a *Decoder* object).

This can e.g. be used to parse a str object instead of a bytes object like this:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import html
>>> source = "<a href='http://www.python.org/'>Python</a>"
>>> doc = parse.tree(
...   parse.String(source),
...   parse.Encoder(encoding="utf-8"),
...   parse.Expat(encoding="utf-8"),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(html)),
... )
>>> doc.string()
'<a href="http://www.python.org/">Python</a>'
```

__init__(*encoding=None*)

Create an *Encoder* object. encoding will be the encoding of the output. If encoding is None it will be automatically detected from the XML declaration in the data.

class ll.xist.parse.**Transcoder**

Bases: object

Transcode the bytes object of the input object into another encoding.

This input object can be a source object or any other pipeline object that produces bytes events.

__init__(*fromencoding=None*, *toencoding=None*)

Create a *Transcoder* object. fromencoding is the encoding of the input. toencoding is the encoding of the output. If any of them is None the encoding will be detected from the data.

class ll.xist.parse.**Parser**

Bases: object

Basic parser interface.

class ll.xist.parse.**Expat**

Bases: *Parser*

A parser using Pythons builtin expat parser.

__init__(*encoding=None*, *xmldecl=False*, *doctype=False*, *loc=True*, *cdata=False*, *ns=False*)

Create an *Expat* parser. Arguments have the following meaning:

encoding

[string or None]
Forces the parser to use the specified encoding. The default None results in the encoding being detected from the XML itself.

xmldecl

[bool]
Should the parser produce events for the XML declaration?

doctype

[bool]
Should the parser produce events for the document type?

loc

[bool]
Should the parser produce "location" events?

cdata

[bool]
Should the parser output CDATA sections as "cdata" events? (If cdata is false "text" events are output instead.)

ns

[bool]
If ns is true, the parser performs namespace processing itself, i.e. it will emit "enterstarttagns", "leavestarttagns", "endtagns", "enterattrns" and "leaveattrns" events instead of "enterstarttag", "leavestarttag", "endtag", "enterattr" and "leaveattr" events.

__call__(*input*)

Return an iterator over the events produced by input.

class ll.xist.parse.**SGMLOP**

Bases: *Parser*

A parser based on sgmlop.

__init__(*encoding=None*, *cdata=False*)

Create a *SGMLOP* parser. Arguments have the following meaning:

encoding

[string or None]
Forces the parser to use the specified encoding. The default None results in the encoding being detected from the XML itself.

cdata

[bool]
Should the parser output CDATA sections as "cdata" events? (If cdata is false output "text" events instead.)

__call__(*input*)

Return an iterator over the events produced by input.

class ll.xist.parse.**NS**

Bases: object

An *NS* object is used in a parsing pipeline to add support for XML namespaces. It replaces the "enterstarttag", "leavestarttag", "endtag", "enterattr" and "leaveattr" events with the appropriate namespace version of the events (i.e. "enterstarttagns" etc.) where the event data is a (namespace, name) tuple.

The output of an *NS* object in the stream looks like this:

```
>>> from ll.xist import parse
>>> from ll.xist.ns import html
>>> list(parse.events(
...   parse.String(b"<a href='http://www.python.org/'>Python</a>"),
...   parse.Expat(),
...   parse.NS(html)
... ))
[('url', URL('STRING')),
 ('position', (0, 0)),
 ('enterstarttagns', ('http://www.w3.org/1999/xhtml', 'a')),
 ('enterattrns', (None, 'href')),
 ('text', 'http://www.python.org/'),
 ('leaveattrns', (None, 'href')),
 ('leavestarttagns', ('http://www.w3.org/1999/xhtml', 'a')),
 ('position', (0, 39)),
 ('text', 'Python'),
 ('endtagns', ('http://www.w3.org/1999/xhtml', 'a'))]
```

__init__(*prefixes=None*, ***kwargs*)

Create an *NS* object. prefixes (if not None) can be a namespace name (or module), which will be used for the empty prefix, or a dictionary that maps prefixes to namespace names (or modules). kwargs maps prefixes to namespaces names too. If a prefix is in both prefixes and kwargs, kwargs wins.

class ll.xist.parse.**Node**

Bases: object

A *Node* object is used in a parsing pipeline to instantiate XIST nodes. It consumes a namespaced event stream:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import html
>>> list(parse.events(
...   parse.String(b"<a href='http://www.python.org/'>Python</a>"),
...   parse.Expat(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(html))
... ))
[('enterelementnode',
  <element ll.xist.ns.html.a xmlns='http://www.w3.org/1999/xhtml' (no children/1␣
↪attr) location='STRING:0:0' at 0x10a683550>),
 ('textnode',
  <ll.xist.xsc.Text content='Python' location='STRING:0:39' at 0x10a5e1170>),
 ('leaveelementnode',
  <element ll.xist.ns.html.a xmlns='http://www.w3.org/1999/xhtml' (no children/1␣
↪attr) location='STRING:0:0' at 0x10a683550>)
]
```

The event data of all events are XIST nodes. The element node from the "enterelementnode" event already has all attributes set. There will be no events for attributes.

__init__(*pool=None*, *base=None*, *loc=True*)

Create a *Node* object.

pool may be None or a xsc.Pool object and specifies which classes used for creating element, entity and processsing instruction instances.

base specifies the base URL for interpreting relative links in the input.

loc specified whether location information should be attached to the nodes that get generated (the startloc attribute (and endloc attribute for elements))

class ll.xist.parse.**Tidy**

Bases: object

A *Tidy* object parses (potentially ill-formed) HTML from a source into a (non-namespaced) event stream by using lxml's HTML parser:

```
>>> from ll.xist import parse
>>> list(parse.events(parse.URL("http://www.yahoo.com/"), parse.Tidy()))
[('url', URL('http://de.yahoo.com/?p=us')),
 ('enterstarttag', 'html'),
 ('enterattr', 'class'),
 ('text', 'y-fp-bg y-fp-pg-grad  bkt708'),
 ('leaveattr', 'class'),
 ('enterattr', 'lang'),
 ('text', 'de-DE'),
 ('leaveattr', 'lang'),
 ('enterattr', 'style'),
 ('leaveattr', 'style'),
 ('leavestarttag', 'html'),
...
```

__init__(*encoding=None*, *xmldecl=False*, *doctype=False*)

Create a new *Tidy* object. Parameters have the following meaning:

encoding
[string or None]
The encoding of the input. If encoding is None it will be automatically detected by the HTML parser.

xmldecl
[bool]
Should the parser produce events for the XML declaration?

doctype
[bool]
Should the parser produce events for the document type?

ll.xist.parse.**events**(**pipeline*)

Return an iterator over the events produced by the pipeline objects in pipeline.

ll.xist.parse.**tree**(**pipeline*, *validate=False*)

Return a tree of XIST nodes from the event stream pipeline.

pipeline must output only events that contain XIST nodes, i.e. the event types "xmldeclnode", "doctypenode", "commentnode", "textnode", "enterelementnode", "leaveelementnode", "procinstnode" and "entitynode".

If validate is true, the tree is validated, i.e. it is checked if the structure of the tree is valid (according to the model attribute of each element node), if no undeclared elements or attributes have been encountered, all required attributes are specified and all attributes have allowed values.

The node returned from *tree()* will always be a Frag object.

Example:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("http://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )

>>> doc[0]
<element ll.xist.ns.html.html
   xmlns='http://www.w3.org/1999/xhtml'
   (7 children/3 attrs)
   location='https://www.python.org/:?:?'
   at 0x110a4ecd0>
```

`ll.xist.parse.itertree`(**pipeline*, *entercontent=True*, *enterattrs=False*, *enterattr=False*, *enterelementnode=False*, *leaveelementnode=True*, *enterattrnode=True*, *leaveattrnode=False*, *selector=None*, *validate=False*)

Parse the event stream `pipeline` iteratively.

itertree() still builds a tree, but it returns an iterator of `xsc.Cursor` objects that tracks changes to the tree as it is built.

`validate` specifies whether each node should be validated after it has been fully parsed.

The rest of the arguments can be used to control when *itertree()* returns to the calling code. For an explanation of their meaning see the class *ll.xist.xsc.Cursor*.

Example:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import xml, html, chars
>>> for c in parse.itertree(
...   parse.URL("http://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars)),
...   selector=html.a/html.img
... ):
...   print(c.path[-1].attrs.src, "-->", c.path[-2].attrs.href)

https://www.python.org/static/img/python-logo.png --> https://www.python.org/
```

6.1.10 present – Screen output of XML trees

This module contains presenter classes, which are used for formatting XIST trees into various formats.

class `ll.xist.present.Presenter`

Bases: `object`

This class is the base of the presenter classes. It is abstract and only serves as documentation for the methods.

A *Presenter* generates a specific string representation of a node to be printed on the screen.

presentText(*node*)

Present a *ll.xist.xsc.Text* node.

presentFrag(*node*)

Present a *ll.xist.xsc.Frag* node.

presentComment(*node*)

Present a *ll.xist.xsc.Comment* node.

presentDocType(*node*)

Present a *ll.xist.xsc.DocType* node.

presentProcInst(*node*)

Present a *ll.xist.xsc.ProcInst* node.

presentAttrs(*node*)

Present an *ll.xist.xsc.Attrs* node.

presentElement(*node*)

Present an *ll.xist.xsc.Element* node.

presentEntity(*node*)

Present a *ll.xist.xsc.Entity* node.

presentNull(*node*)

Present the `ll.xist.xsc.Null` node.

presentAttr(*node*)

Present an *ll.xist.xsc.Attr* node.

class ll.xist.present.**TreePresenter**

Bases: *Presenter*

This presenter shows the object as a nested tree.

__init__(*node*, *indent=None*, *defaultxmlns=None*)

Create a *TreePresenter* object for the XIST node `node` using `indent` for indenting each tree level. If `indent` is `None` use the value of the environment variable `LL_XIST_INDENT` as the indent string (falling back to a tab if the environment variable doesn't exist).

If `defaultxmlns` is not `None`, elements from this namespace will be output without any namespace name.

class ll.xist.present.**CodePresenter**

Bases: *Presenter*

This presenter formats the object as a nested Python object tree.

This makes it possible to quickly convert HTML/XML files to XIST constructor calls.

__init__(*node*, *indent=None*)

Create a *CodePresenter* object for the XIST node `node` using `indent` for indenting each tree level. If `indent` is `None` use the value of the environment variable `LL_XIST_INDENT` as the indent string (falling back to a tab if the environment variable doesn't exist).

6.1.11 sims – Simple schema validation

This module contains classes for a very simple validation model.

Validation is specified like this:

```
class inner(xsc.Element):
   model = sims.NoElements()

class outer(xsc.Element):
   model = sims.Elements(inner)
```

With this configuration `inner` elements may only contain text and `outer` elements may only contain `inner` elements. Everything else will issue warnings when parsing or publishing.

exception `ll.xist.sims.`**`SIMSWarning`**

Bases: *`Warning`*

Base class for all warning classes in this module.

exception `ll.xist.sims.`**`EmptyElementWithContentWarning`**

Bases: *`SIMSWarning`*

Warning that is issued when an element has content, but it shouldn't (i.e. `model` is *`Empty`*)

exception `ll.xist.sims.`**`WrongElementWarning`**

Bases: *`SIMSWarning`*

Warning that is issued when an element contains another element of a certain type, but shouldn't.

exception `ll.xist.sims.`**`ElementWarning`**

Bases: *`SIMSWarning`*

Warning that is issued when an element contains another element but shouldn't contain any.

exception `ll.xist.sims.`**`IllegalTextWarning`**

Bases: *`SIMSWarning`*

Warning that is issued when an element contains a text node but shouldn't.

exception `ll.xist.sims.`**`AnyWarning`**

Bases: *`SIMSWarning`*

Warning that is issued when an element contains a text node but shouldn't.

`ll.xist.sims.`**`badtext`**(*node*)

Return whether `node` is a text node (i.e. *`ll.xist.xsc.Text`* that does not consist of whitespace only).

class `ll.xist.sims.`**`Empty`**

Bases: `object`

This validator checks that an element has no content.

class `ll.xist.sims.`**`Transparent`**

Bases: `object`

This validator implements the "transparent" content model of HTML5.

class `ll.xist.sims.`**`NoElements`**

Bases: `object`

This validator checks that an element does not have child elements from the same namespace.

`validate`(*path*)

check that the content of `node` is valid.

class `ll.xist.sims.`**`NoElementsOrText`**

Bases: `object`

This validator checks that an element does have neither child elements from the same namespace nor real (i.e. not-whitespace) text nodes.

`validate`(*path*)

check that the content of `node` is valid.

class ll.xist.sims.**Elements**

Bases: object

This validator checks that an element does have neither child elements from any of the namespaces of those elements specified in the constructor except for those elements itself nor real (i.e. not-whitespace) text nodes.

__init__(**elements*)

Every element in elements may be in the content of the node to which this validator is attached. Any other element from one of the namespaces of those elements is invalid. Elements from other namespaces are OK.

validate(*path*)

check that the content of node is valid.

class ll.xist.sims.**ElementsOrText**

Bases: *Elements*

This validator checks that an element doesn't have child elements from the same namespace except those specified in the constructor.

__init__(**elements*)

Every element in elements may be in the content of the node to which this validator is attached. Any other element from one of the namespaces of those elements is invalid. Elements from other namespaces are OK.

validate(*path*)

Check that the content of node is valid.

class ll.xist.sims.**NotElements**

Bases: object

This validator checks that an element doesn't contain any of the specified elements.

__init__(**elements*)

Every element in elements may not be in the content of the node to which this validator is attached.

class ll.xist.sims.**All**

Bases: object

This meta validator checks that all its child validators declare the content of the element to be valid.

class ll.xist.sims.**Any**

Bases: object

This meta validator checks that at least one of its child validators declares the content of the element to be valid.

6.1.12 xfind – Tree traversal and filtering

This module contains XFind selectors and related classes and functions.

A selector specifies a condition that a node in an XIST tree must satisfy to match the selector. For example the method Node.walk() will only output nodes that match the specified selector.

Selectors can be combined with various operations and form a language comparable to XPath but implemented as Python expressions.

ll.xist.xfind.**filter**(*iter*, **selectors*)

Filter an iterator over xsc.Cursor objects against a *Selector* object.

Example:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> [c.node.string() for c in xfind.filter(doc.walk(), html.b, html.title)]
[
   '<title>Welcome to Python.org</title>',
   '<b>Web Programming</b>',
   '<b>GUI Development</b>',
   '<b>Scientific and Numeric</b>',
   '<b>Software Development</b>',
   '<b>System Administration</b>'
]
```

`ll.xist.xfind.selector(*objs)`

Create a *Selector* object from `objs`.

If `objs` is empty (i.e. `selector()` is called without arguments) `any` is returned (which matches every node).

If more than one argument is passed (or the argument is a tuple), an *OrCombinator* is returned.

Otherwise the following steps are taken for the single argument `obj`:

- if `obj` already is a *Selector* object it is returned unchanged;
- if `obj` is a `Node` subclass, an *IsInstanceSelector* is returned (which matches if the node is an instance of this class);
- if `obj` is a `Node` instance, an *IsSelector* is returned (which matches only `obj`);
- if `obj` is callable a *CallableSelector* is returned (where matching is done by calling `obj`);
- if `obj` is `None` `any` will be returned;
- otherwise `selector()` will raise a `TypeError`.

class `ll.xist.xfind.Selector`

Bases: `object`

A selector specifies a condition that a node in an XIST tree must satisfy to match the selector.

Whether a node matches the selector can be specified by overwriting the *__contains__()* method. Selectors can be combined with various operations (see methods below).

__contains__(*path*)

Return whether `path` (which is a list of XIST nodes from the root of the tree to the node in question) matches the selector.

__truediv__(*other*)

Create a *ChildCombinator* with `self` as the left hand selector and `other` as the right hand selector.

__rtruediv__(*other*)

Create a *ChildCombinator* with `other` as the left hand selector and `self` as the right hand selector.

__floordiv__(*other*)

Create a *DescendantCombinator* with `self` as the left hand selector and `other` as the right hand selector.

__rfloordiv__(*other*)

Create a *DescendantCombinator* with `other` as the left hand selector and `self` as the right hand selector.

__mul__(*other*)

Create an *AdjacentSiblingCombinator* with `self` as the left hand selector and `other` as the right hand selector.

__rmul__(*other*)

Create an *AdjacentSiblingCombinator* with `other` as the left hand selector and `self` as the right hand selector.

__pow__(*other*)

Create a *GeneralSiblingCombinator* with `self` as the left hand selector and `other` as the right hand selector.

__rpow__(*other*)

Create a *GeneralSiblingCombinator* with `other` as the left hand selector and `self` as the right hand selector.

__and__(*other*)

Create an *AndCombinator* from `self` and `other`.

__rand__(*other*)

Create an *AndCombinator* from `other` and `self`.

__or__(*other*)

Create an *OrCombinator* from `self` and `other`.

__ror__(*other*)

Create an *OrCombinator* from `other` and `self`.

__invert__()

Create a *NotCombinator* inverting `self`.

class ll.xist.xfind.**AnySelector**

Bases: *Selector*

Selector that selects all nodes.

An instance of this class named `any` is created as a module global, i.e. you can use `xfind.any`.

class ll.xist.xfind.**IsInstanceSelector**

Bases: *Selector*

Selector that selects all nodes that are instances of the specified type. You can either create an `IsInstanceSelector` object directly or simply pass a class to a function that expects a selector (this class will be automatically wrapped in an `IsInstanceSelector`):

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(html.a):
...   print(node.attrs.href, node.attrs.title)
...
https://www.python.org/#content Skip to content
https://www.python.org/#python-network
https://www.python.org/ The Python Programming Language
https://www.python.org/psf-landing/ The Python Software Foundation
...
```

__getitem__(*index*)

Return an *nthoftype* selector that uses `index` as the index and `self.types` as the types.

class ll.xist.xfind.**element**

Bases: *Selector*

Selector that selects all elements that have a specified namespace name and element name:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.element(html, "img")):
...   print(node.string())
...
<img alt="python™" class="python-logo" src="https://www.python.org/static/img/
↪python-logo.png" />
```

class ll.xist.xfind.**procinst**

Bases: *Selector*

Selector that selects all processing instructions that have a specified name.

class ll.xist.xfind.**entity**

Bases: *Selector*

Selector that selects all entities that have a specified name.

class ll.xist.xfind.**IsSelector**

Bases: *Selector*

Selector that selects one specific node in the tree. This can be combined with other selectors via *ChildCombinator* or *DescendantCombinator* selectors to select children of this specific node. You can either create an `IsSelector` directly or simply pass a node to a function that expects a selector:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(doc[0]/xsc.Element):
...   print(repr(node))
...
<element ll.xist.ns.html.head xmlns='http://www.w3.org/1999/xhtml' (89 children/
↪no attrs) location='https://www.python.org/:?:?' at 0x104ad7630>
<element ll.xist.ns.html.body xmlns='http://www.w3.org/1999/xhtml' (14 children/
↪2 attrs) location='https://www.python.org/:?:?' at 0x104cc1f28>
```

class ll.xist.xfind.**IsRootSelector**

Bases: *Selector*

Selector that selects the node that is the root of the traversal.

An instance of this class named `isroot` is created as a module global, i.e. you can use `xfind.isroot`.

class ll.xist.xfind.**IsEmptySelector**

Bases: *Selector*

Selector that selects all empty elements or fragments.

An instance of this class named `empty` is created as a module global, i.e. you can use `xfind.empty`:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.empty):
...   print(node.string())
...
<meta charset="utf-8" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<link href="https://ajax.googleapis.com/" rel="prefetch" />
<meta name="application-name" content="Python.org" />
...
```

class ll.xist.xfind.**OnlyChildSelector**

Bases: *Selector*

Selector that selects all nodes that are the only child of their parents.

An instance of this class named `onlychild` is created as a module global, i.e. you can use `xfind.onlychild`:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.onlychild & html.a):
...   print(node.string())
...
<a class="text-shrink" href="javascript:;" title="Make Text Smaller">Smaller</a>
<a class="text-grow" href="javascript:;" title="Make Text Larger">Larger</a>
<a class="text-reset" href="javascript:;" title="Reset any font size changes I␣
↪have made">Reset</a>
<a href="http://plus.google.com/+Python"><span aria-hidden="true" class="icon-
↪google-plus"></span>Google+</a>
...
```

class ll.xist.xfind.**OnlyOfTypeSelector**

Bases: *Selector*

Selector that selects all nodes that are the only nodes of their type among their siblings.

An instance of this class named `onlyoftype` is created as a module global, i.e. you can use `xfind.onlyoftype`:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.onlyoftype & xsc.Element):
...   print(repr(node))
...
<element ll.xist.ns.html.html xmlns='http://www.w3.org/1999/xhtml' (7 children/3␣
↪attrs) location='https://www.python.org/:?:?' at 0x108858d30>
<element ll.xist.ns.html.head xmlns='http://www.w3.org/1999/xhtml' (89 children/
↪no attrs) location='https://www.python.org/:?:?' at 0x108858630>
<element ll.xist.ns.html.title xmlns='http://www.w3.org/1999/xhtml' (1 child/no␣
↪attrs) location='https://www.python.org/:?:?' at 0x108c547b8>
<element ll.xist.ns.html.body xmlns='http://www.w3.org/1999/xhtml' (14 children/
↪2 attrs) location='https://www.python.org/:?:?' at 0x108c54eb8>
...
```

class ll.xist.xfind.**hasattr**

Bases: *Selector*

Selector that selects all element nodes that have an attribute with one of the specified names. (Names can be strings, (attribute name, namespace name) tuples or attribute classes or instances):

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.hasattr("id")):
...   print(node.xmlname, node.attrs.id)
...
body homepage
div touchnav-wrapper
div top
a close-python-network
...
```

class ll.xist.xfind.**attrhasvalue**

Bases: *Selector*

Selector that selects all element nodes where an attribute with the specified name has one of the specified values. (Names can be strings, (attribute name, namespace name) tuples or attribute classes or instances). Note that "fancy" attributes (i.e. those containing non-text) will not be considered:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
```

(continues on next page)

(continued from previous page)

```
... )
>>> for node in doc.walknodes(xfind.attrhasvalue("rel", "stylesheet")):
...   print(node.attrs.href)
...
https://www.python.org/static/stylesheets/style.css
https://www.python.org/static/stylesheets/mq.css
```

class ll.xist.xfind.**attrcontains**

Bases: *Selector*

Selector that selects all element nodes where an attribute with the specified name contains one of the specified substrings in its value. (Names can be strings, (attribute name, namespace name) tuples or attribute classes or instances). Note that "fancy" attributes (i.e. those containing non-text) will not be considered:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.attrcontains("rel", "stylesheet")):
...   print(node.attrs.rel, node.attrs.href)
...
stylesheet https://www.python.org/static/stylesheets/style.css
stylesheet https://www.python.org/static/stylesheets/mq.css
```

class ll.xist.xfind.**attrstartswith**

Bases: *Selector*

Selector that selects all element nodes where an attribute with the specified name starts with any of the specified strings. (Names can be strings, (attribute name, namespace name) tuples or attribute classes or instances). Note that "fancy" attributes (i.e. those containing non-text) will not be considered:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.attrstartswith("class", "icon-")):
...   print(node.bytes())
...
b'<span aria-hidden="true" class="icon-arrow-down"><span>\xe2\x96\xbc</span></
↪span>'
b'<span aria-hidden="true" class="icon-arrow-up"><span>\xe2\x96\xb2</span></span>
↪'
b'<span aria-hidden="true" class="icon-search"></span>'
b'<span aria-hidden="true" class="icon-facebook"></span>'
...
```

class ll.xist.xfind.**attrendswith**

Bases: *Selector*

Selector that selects all element nodes where an attribute with the specified name ends with one of the

specified strings. (Names can be strings, (attribute name, namespace name) tuples or attribute classes or instances). Note that "fancy" attributes (i.e. those containing non-text) will not be considered:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.attrendswith("href", ".css")):
...   print(node.attrs.href)
...
https://www.python.org/static/stylesheets/style.css
https://www.python.org/static/stylesheets/mq.css
```

class ll.xist.xfind.**hasid**

Bases: *Selector*

Selector that selects all element nodes where the `id` attribute has one if the specified values:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.hasid("id-search-field")):
...   print(node.string())
...
<input class="search-field" id="id-search-field" name="q" placeholder="Search"␣
↪role="textbox" tabindex="1" type="search" />
```

class ll.xist.xfind.**hasclass**

Bases: *Selector*

Selector that selects all element nodes where the `class` attribute contains one of the specified values:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.hasclass("tier-1")/html.a):
...   print(node.string())
...
A A
Socialize
Sign In
About
Downloads
...
```

class ll.xist.xfind.**InAttrSelector**

Bases: *Selector*

Selector that selects all attribute nodes and nodes inside of attributes:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for path in doc.walkpaths(xfind.inattr & xsc.Text, enterattrs=True,␣
↪enterattr=True):
...   print(path[-3].xmlname, path[-2].xmlname, path[-1].string())
...
html class no-js
html dir ltr
html lang en
meta charset utf-8
meta content IE=edge
meta http-equiv X-UA-Compatible
...
```

class ll.xist.xfind.**Combinator**

Bases: *Selector*

A `Combinator` is a selector that transforms one or combines two or more other selectors in a certain way.

class ll.xist.xfind.**BinaryCombinator**

Bases: *Combinator*

A `BinaryCombinator` is a combinator that combines two selector: the left hand selector and the right hand selector.

class ll.xist.xfind.**ChildCombinator**

Bases: *BinaryCombinator*

A `ChildCombinator` is a *BinaryCombinator*. To match the `ChildCombinator` the node must match the right hand selector and its immediate parent must match the left hand selector (i.e. it works similar to the > combinator in CSS or the / combinator in XPath).

`ChildCombinator` objects can be created via the division operator (/):

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(html.a/html.img):
...   print(node.string())
...
<img alt="python™" class="python-logo" src="https://www.python.org/static/img/
↪python-logo.png" />
```

class ll.xist.xfind.**DescendantCombinator**

Bases: *BinaryCombinator*

A DescendantCombinator is a *BinaryCombinator*. To match the DescendantCombinator the node must match the right hand selector and any of its ancestor nodes must match the left hand selector (i.e. it works similar to the descendant combinator in CSS or the // combinator in XPath).

DescendantCombinator objects can be created via the floor division operator (//):

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(html.div//html.img):
...   print(node.string())
...
<img alt="python™" class="python-logo" src="https://www.python.org/static/img/
↪python-logo.png" />
```

class ll.xist.xfind.**AdjacentSiblingCombinator**

Bases: *BinaryCombinator*

A AdjacentSiblingCombinator is a *BinaryCombinator*. To match the AdjacentSiblingCombinator the node must match the right hand selector and the immediately preceding sibling must match the left hand selector.

AdjacentSiblingCombinator objects can be created via the multiplication operator (*). The following example outputs all *span* elements that immediately follow a *form* element:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(html.form*html.span):
...   print(node.string())
...
<span class="breaker"></span>
```

class ll.xist.xfind.**GeneralSiblingCombinator**

Bases: *BinaryCombinator*

A GeneralSiblingCombinator is a *BinaryCombinator*. To match the GeneralSiblingCombinator the node must match the right hand selector and any of the preceding siblings must match the left hand selector.

AdjacentSiblingCombinator objects can be created via the exponentiation operator (**). The following example outputs all *meta* elements that come after a *link* elements:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
```

(continues on next page)

(continued from previous page)

```
... )
>>> for node in doc.walknodes(html.link**html.meta):
...   print(node.string())
...
<meta name="application-name" content="Python.org" />
<meta name="msapplication-tooltip" content="The official home of the Python␣
↪Programming Language" />
<meta name="apple-mobile-web-app-title" content="Python.org" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style" content="black" />
...
```

class ll.xist.xfind.**ChainedCombinator**

Bases: *Combinator*

A ChainedCombinator combines any number of other selectors.

class ll.xist.xfind.**OrCombinator**

Bases: *ChainedCombinator*

An OrCombinator is a *ChainedCombinator* where the node must match at least one of the selectors to match the OrCombinator. An OrCombinator can be created with the binary or operator (|):

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.hasattr("href") | xfind.hasattr("src")):
...   print(node.attrs.href if "href" in node.Attrs else node.attrs.src)
...
https://ajax.googleapis.com/
https://www.python.org/static/js/libs/modernizr.js
https://www.python.org/static/stylesheets/style.css
https://www.python.org/static/stylesheets/mq.css
https://www.python.org/static/favicon.ico
...
```

class ll.xist.xfind.**AndCombinator**

Bases: *ChainedCombinator*

An AndCombinator is a *ChainedCombinator* where the node must match all of the combined selectors to match the AndCombinator. An AndCombinator can be created with the binary and operator (&):

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(html.input & xfind.hasattr("id")):
...   print(node.string())
...
```

(continues on next page)

(continued from previous page)

```
<input class="search-field" id="id-search-field" name="q" placeholder="Search"␣
↪role="textbox" tabindex="1" type="search" />
```

class ll.xist.xfind.**NotCombinator**

Bases: *Combinator*

A `NotCombinator` inverts the selection logic of the underlying selector, i.e. a node matches only if it does not match the underlying selector. A `NotCombinator` can be created with the unary inversion operator (~).

The following example outputs all internal scripts:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(html.script & ~xfind.hasattr("src")):
...   print(node.string())
...
<script type="text/javascript">
    var _gaq = _gaq || [];
    _gaq.push(['_setAccount', 'UA-39055973-1']);
    _gaq.push(['_trackPageview']);

    (function() {
        var ga = document.createElement('script'); ga.type = 'text/javascript';␣
↪ga.async = true;
        ga.src = ('https:' == document.location.protocol ? 'https://ssl' :
↪'http://www') + '.google-analytics.com/ga.js';
        var s = document.getElementsByTagName('script')[0]; s.parentNode.
↪insertBefore(ga, s);
    })();
    </script>
<script>window.jQuery || document.write('&lt;script src="/static/js/libs/jquery-
↪1.8.2.min.js"&gt;&lt;\/script&gt;')</script>
```

class ll.xist.xfind.**CallableSelector**

Bases: *Selector*

A `CallableSelector` is a selector that calls a user specified callable to select nodes. The callable gets passed the path and must return a bool specifying whether this path is selected. A `CallableSelector` is created implicitely whenever a callable is passed to a method that expects a selector.

The following example outputs all links that point outside the `python.org` domain:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> def isextlink(path):
...   return isinstance(path[-1], html.a) and not str(path[-1].attrs.href).
```

(continues on next page)

(continued from previous page)

```
↪startswith("https://www.python.org")
...
>>> for node in doc.walknodes(isextlink):
...   print(node.string())
...
<a href="http://docs.python.org/" title="Python Documentation">Docs</a>
<a href="https://pypi.python.org/" title="Python Package Index">PyPI</a>
<a class="text-shrink" href="javascript:;" title="Make Text Smaller">Smaller</a>
<a class="text-grow" href="javascript:;" title="Make Text Larger">Larger</a>
..
```

class ll.xist.xfind.**nthchild**

Bases: *Selector*

An `nthchild` object is a selector that selects every node that is the n-th child of its parent. E.g. `nthchild(0)` selects every first child, `nthchild(-1)` selects each last child. Furthermore `nthchild("even")` selects each first, third, fifth, ... child and `nthchild("odd")` selects each second, fourth, sixth, ... child.

class ll.xist.xfind.**nthoftype**

Bases: *Selector*

An `nthoftype` object is a selector that selects every node that is the n-th node of a specified type among its siblings. Similar to *nthchild* `nthoftype` supports negative and positive indices as well as `"even"` and `"odd"`. Which types are checked can be passed explicitly. If no types are passed the type of the node itself is used:

```
>>> from ll.xist import xsc, parse, xfind
>>> from ll.xist.ns import xml, html, chars
>>> doc = parse.tree(
...   parse.URL("https://www.python.org/"),
...   parse.Tidy(),
...   parse.NS(html),
...   parse.Node(pool=xsc.Pool(xml, html, chars))
... )
>>> for node in doc.walknodes(xfind.nthoftype(0, html.h2)):
...   print(node.string())
...
<h2 class="widget-title"><span aria-hidden="true" class="icon-get-started"></
↪span>Get Started</h2>
<h2 class="widget-title"><span aria-hidden="true" class="icon-download"></span>
↪Download</h2>
<h2 class="widget-title"><span aria-hidden="true" class="icon-documentation"></
↪span>Docs</h2>
<h2 class="widget-title"><span aria-hidden="true" class="icon-jobs"></span>Jobs</
↪h2>
...
```

6.1.13 css – CSS related functions

This module contains functions related to the handling of CSS.

`ll.xist.css.replaceurls(stylesheet, replacer)`

Replace all URLs appearing in the `CSSStyleSheet` `stylesheet`. For each URL the function `replacer` will be called and the URL will be replaced with the result.

`ll.xist.css.geturls(stylesheet)`

Return a list of all URLs appearing in the `CSSStyleSheet` `stylesheet`.

`ll.xist.css.iterrules(node, base=None, media=None, title=None)`

Return an iterator for all CSS rules defined in the HTML tree `node`. This will parse the CSS defined in any `html.style` or `html.link` element (and recursively in those stylesheets imported via the `@import` rule). The rules will be returned as `CSSStyleRule` objects from the `cssutils` package (so this requires `cssutils`).

The `base` argument will be used as the base URL for parsing the stylesheet references in the tree (so `None` means the URLs will be used exactly as they appear in the tree). All URLs in the style properties will be resolved.

If `media` is given, only rules that apply to this media type will be produced.

`title` can be used to specify which stylesheet group should be used. If `title` is `None` only the persistent and preferred stylesheets will be used. If `title` is a string only the persistent stylesheets and alternate stylesheets with that style name will be used.

For a description of "persistent", "preferred" and "alternate" stylesheets see <http://www.w3.org/TR/2002/WD-xhtml2-20020805/mod-styleSheet.html#sec_20.1.2.>

`ll.xist.css.applystylesheets(node, base=None, media=None, title=None)`

applystylesheets() modifies the XIST tree `node` by removing all CSS (from `html.link` and `html.style` elements and their `@imported` stylesheets) and putting the resulting style properties into the `style` attribute of every affected element instead.

For the meaning of `base`, `media` and `title` see *iterrules()*.

class `ll.xist.css.CSSWeightedSelector`

Bases: *Selector*

Base class for all CSS pseudo-class selectors.

class `ll.xist.css.CSSHasAttributeSelector`

Bases: *CSSWeightedSelector*

A *CSSHasAttributeSelector* selector selects all element nodes that have an attribute with the specified XML name.

class `ll.xist.css.CSSAttributeListSelector`

Bases: *CSSWeightedSelector*

A *CSSAttributeListSelector* selector selects all element nodes where an attribute with the specified XML name has the specified word among the white space-separated list of words in the attribute value.

class `ll.xist.css.CSSAttributeLangSelector`

Bases: *CSSWeightedSelector*

A *CSSAttributeLangSelector* selector selects all element nodes where an attribute with the specified XML name either is exactly the specified value or starts with the specified value followed by `"-"`.

class `ll.xist.css.CSSFirstChildSelector`

Bases: *CSSWeightedSelector*

A *CSSFirstChildSelector* selector selects all element nodes that are the first child of its parent.

class ll.xist.css.**CSSLastChildSelector**

Bases: *CSSWeightedSelector*

A *CSSLastChildSelector* selector selects all element nodes that are the last child of its parent.

class ll.xist.css.**CSSFirstOfTypeSelector**

Bases: *CSSWeightedSelector*

A *CSSLastChildSelector* selector selects all element nodes that are the first of its type among their siblings.

class ll.xist.css.**CSSLastOfTypeSelector**

Bases: *CSSWeightedSelector*

A *CSSLastChildSelector* selector selects all element nodes that are the last of its type among their siblings.

class ll.xist.css.**CSSOnlyChildSelector**

Bases: *CSSWeightedSelector*

A *CSSOnlyChildSelector* selector selects all element nodes that are the only element among its siblings.

class ll.xist.css.**CSSOnlyOfTypeSelector**

Bases: *CSSWeightedSelector*

A *CSSOnlyOfTypeSelector* selector selects all element nodes that are the only element of its type among its siblings.

class ll.xist.css.**CSSEmptySelector**

Bases: *CSSWeightedSelector*

A *CSSEmptySelector* selector selects all element nodes that are empty (i.e. they contain no elements or non-whitespace text).

class ll.xist.css.**CSSRootSelector**

Bases: *CSSWeightedSelector*

A *CSSRootSelector* selector selects the root element.

class ll.xist.css.**CSSLinkSelector**

Bases: *CSSWeightedSelector*

A *CSSLinkSelector* selector selects all HTML links.

class ll.xist.css.**CSSDisabledSelector**

Bases: *CSSWeightedSelector*

A *CSSDisabledSelector* selector selects all HTML elements where the attribute `disabled` is set.

Note that that is not 100% what the pseudo class *:disabled* means, but since we have no live DOM this is the best we can do.

class ll.xist.css.**CSSFunctionSelector**

Bases: *CSSWeightedSelector*

Base class of all CSS selectors that require an argument.

class ll.xist.css.**CSSNthChildSelector**

Bases: *CSSFunctionSelector*

A *CSSNthChildSelector* selector selects all element nodes that are the n-th element among their siblings.

class ll.xist.css.**CSSNthLastChildSelector**

Bases: *CSSFunctionSelector*

A *CSSNthLastChildSelector* selector selects all element nodes that are the n-th last element among their siblings.

class ll.xist.css.**CSSNthOfTypeSelector**

Bases: *CSSFunctionSelector*

A *CSSNthOfTypeSelector* selector selects all element nodes that are the n-th of its type among their siblings.

class ll.xist.css.**CSSNthLastOfTypeSelector**

Bases: *CSSFunctionSelector*

A *CSSNthOfTypeSelector* selector selects all element nodes that are the n-th last of its type among their siblings.

class ll.xist.css.**CSSAdjacentSiblingCombinator**

Bases: *BinaryCombinator*

A *CSSAdjacentSiblingCombinator* works similar to an *AdjacentSiblingCombinator* except that only preceding *elements* are considered.

class ll.xist.css.**CSSGeneralSiblingCombinator**

Bases: *BinaryCombinator*

A *CSSGeneralSiblingCombinator* works similar to an `xfind.GeneralSiblingCombinator` except that only preceding *elements* are considered.

ll.xist.css.**selector**(*selectors*, *prefixes=None*)

Create a `xfind.Selector` object that matches all nodes that match the specified CSS selector expression. `selectors` can be a string or a `cssutils.css.selector.Selector` object. `prefixes` may be a mapping mapping namespace prefixes to namespace names.

ll.xist.css.**parsestring**(*data*, *base=None*, *encoding=None*)

Parse the string `data` into a `cssutils` stylesheet. `base` is the base URL for the parsing process, `encoding` can be used to force the parser to use the specified encoding.

ll.xist.css.**parsestream**(*stream*, *base=None*, *encoding=None*)

Parse a `cssutils` stylesheet from the stream `stream`. `base` is the base URL for the parsing process, `encoding` can be used to force the parser to use the specified encoding.

ll.xist.css.**parsefile**(*filename*, *base=None*, *encoding=None*)

Parse a `cssutils` stylesheet from the file named `filename`. `base` is the base URL for the parsing process (defaulting to the filename itself), `encoding` can be used to force the parser to use the specified encoding.

ll.xist.css.**parseurl**(*name*, *base=None*, *encoding=None*, **args*, ***kwargs*)

Parse a `cssutils` stylesheet from the URL `name`. `base` is the base URL for the parsing process (defaulting to the final URL of the response, i.e. including redirects), `encoding` can be used to force the parser to use the specified encoding. `arg` and `kwargs` are passed on to `URL.openread()`, so you can pass POST data and request headers.

6.1.14 `scripts` – XIST related scripts

This package contains the following scripts:

dtd2xsc

dtd2xsc creates a skeleton XIST namespace module from a DTD.

xml2xsc

xml2xsc is a script that generates a skeleton XIST namespace module from one or more XML files.

tld2xsc

tld2xsc converts a Java tag library description XML file to a skeleton XIST namespace.

doc2txt

doc2txt creates a plain text file from a XML file using XISTs doc XML vocabulary.

uhpp

uhpp is a script for pretty printing HTML files. It is URL-enabled, so you can specify local file names and URLs (and remote files via `ssh` URLs).

These scripts can either be called via Pythons `-m` option:

```
$ python -mll.xist.scripts.xml2xsc --help
```

or as a simple script installed in the search path:

```
$ xml2xsc --help
```

`dtd2xsc` – Creating XIST namespaces from DTDs

Purpose

dtd2xsc is a script that helps create XIST namespace modules from DTDs. It reads one or more DTDs and outputs a skeleton namespace module.

Options

dtd2xsc supports the following options:

urls

Zero or more URLs (or filenames) of DTDs to be parsed. If no URL is given stdin will be read.

-x <name>, **--xmlns** <name>

The default namespace name. All elements that don't belong to any namespace will be assigned to this namespace.

-s <value>, **--shareattrs** <value>

Should attributes be shared among the elements? `none` means that each element will have its own standalone `Attrs` class directly derived from `ll.xist.Elements.Attrs`. For `dupes` each attribute that is used by more than one element will be moved into its own `Attrs` class. For `all` this will be done for all attributes.

-m <value>, **--model** <value>

Add model information to the namespace. `no` doesn't add any model information. `simple` only adds `model = False` or `model = True` (i.e. only the information whether the element must be empty or not). `fullall` adds a *`ll.xist.sims`* model object to each element class. `fullonce` adds full model information to, but reuses model objects for elements which have the same model.

-d <flag>, **--defaults** <flag>

Should default values for attributes specified in the DTD be added to the XIST namespace (as the `default` specification in the attribute class)? (Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`)

--duplicates <value>

If more that one DTD is specified on the command line, some elements might be specified in more than one DTD. *`--duplicates`* specifies how to handle this case: `reject` doesn't allow multiple element specifications. `allow` allows them, but only if both specifications are identical (i.e. have the same attributes). `merge` allows them and adds the attribute specification of all element specifications to the resulting XIST namespace.

Note that **dtd2xsc** requires lxml to work.

Example

Suppose we have the following DTD file (named `foo.dtd`):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT persons (person*)>
<!ELEMENT person (firstname?, lastname?)>
<!ATTLIST person id CDATA #REQUIRED>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
```

Then we can generate a skeleton XIST namespace from it with the following command:

```
$ dtd2xsc ~/gurk.dtd -xhttp://xmlns.example.org/ -mfullall
```

The output will be:

```
# -*- coding: ascii -*-

from ll.xist import xsc, sims

xmlns = 'http://xmlns.example.org/'

class firstname(xsc.Element): xmlns = xmlns

class lastname(xsc.Element): xmlns = xmlns

class person(xsc.Element):
   xmlns = xmlns
   class Attrs(xsc.Element.Attrs):
      class id(xsc.TextAttr): required = True

class persons(xsc.Element): xmlns = xmlns

person.model = sims.Elements(lastname, firstname)
persons.model = sims.Elements(person)
firstname.model = sims.NoElements()
lastname.model = sims.NoElements()
```

`xml2xsc` – Creating XIST namespaces from XML

Purpose

`xml2xsc` is a script that generates an XIST namespace module from one or more XML files. **`xml2xsc`** will output an XIST element class for each element it encounters in any of the XML files. The attributes and model information **`xml2xsc`** assigns to an element will be collected from each occurrence of the element in the XML files, so the XML files should cover as many different cases as possible.

Options

xml2xsc supports the following options:

urls

Zero or more URLs (or filenames) of XML files to be parsed. If no URL is given stdin will be read.

-p <parser>, **--parser** <parser>

Which XML parser should be used from parsing the XML files? (`etree` is the default, `lxml` requires that lxml is installed)

-s <mode>, **--shareattrs** <mode>

Should attributes be shared among the elements? `none` means that each element will have its own standalone `Attrs` class directly derived from `ll.xist.xsc.Elements.Attrs`. For `dupes` each attribute that is used by more than one element will be moved into its own `Attrs` class. For `all` this will be done for all attributes.

-m <mode>, **--model** <mode>

Add model information to the namespace. `no` doesn't add any model information. `simple` only adds `model = False` or `model = True` (i.e. only the information whether the element must be empty or not). `fullall` adds a *`ll.xist.sims`* model object to each element class. `fullonce` adds full model information to, but reuses model objects for elements which have the same model.

-x <name>, **--defaultxmlns** <name>

The default namespace name. All elements that don't belong to any namespace will be assigned to this namespace.

Example

Suppose we have the following XML file (named `foo.xml`):

```
<x a="0"><x b="1"/><y/></x>
```

Then we can generate a skeleton XIST namespace from it with the following command:

```
$ xml2xsc foo.xml -xhttp://xmlns.example.org/ -mfullonce
```

The output will be:

```
# -*- coding: ascii -*-

from ll.xist import xsc, sims

xmlns = 'http://xmlns.example.org/'

class x(xsc.Element):
   xmlns = xmlns
   class Attrs(xsc.Element.Attrs):
      class a(xsc.TextAttr): pass
      class b(xsc.TextAttr): pass

class y(xsc.Element): xmlns = xmlns

x.model = sims.Elements(y, x)
y.model = sims.Empty()
```

tld2xsc – Creating XIST namespaces from Java tag libraries

Purpose

tld2xsc is a script that converts a JSP Tag Library Descriptor XML file into a skeleton XIST namespace module. The tld file is read from stdin and the namespace module is printed to stdout.

Options

tld2xsc supports the following options:

-s <value>, **--shareattrs** <value>

Should attributes be shared among the elements? `none` means that each element will have its own standalone `Attrs` class directly derived from `ll.xist.xsc.Elements.Attrs`. For `dupes` each attribute that is used by more than one element will be moved into its own `Attrs` class. For `all` this will be done for all attributes.

-m <model>, **--model** <model>

Add model information to the namespace. `no` doesn't add any model information. `simple` only adds `model = False` or `model = True` (i.e. only the information whether the element must be empty or not). `fullall` adds a *`ll.xist.sims`* model object to each element class. `fullonce` adds full model information to, but reuses model objects for elements which have the same model.

doc2txt – Creating text files from XIST doc files

doc2txt is a script that converts an XML files using XIST doc vocabulary (i.e. the *`ll.xist.ns.doc`* namespace module) into plain text (by using *`ll.xist.ns.html.astext()`*).

doc2txt supports the following options:

-t <title>, **--title** <title>

The title for the document

-w <width>, **--width** <width>

The width of the formatted text output (default 72)

The input is read from stdin and printed to stdout.

Example

The following generates `spam.txt` from `spam.xml` formatted to 80 columns:

```
$ doc2txt <spam.xml >spam.txt -w80
```

uhpp – Pretty printing HTML

Purpose

uhpp is a script for pretty printing HTML files. It is URL-enabled, so you can specify local file names and URLs (and remote files via `ssh` URLs).

Options

uhpp supports the following options:

urls

Zero or more URLs to be printed. If no URL is given, stdin is read.

-c <flag>, **--compact** <flag>

Compact HTML before printing (i.e. remove whitespace nodes)? (Allowed valued are `false`, `no`, `0`, `true`, `yes` or `1`)

Examples

Pretty print stdin:

```
$ cat foo.html | uhpp
```

Pretty print a local HTML file:

```
$ uhpp foo.html
```

Pretty print a remote HTML file:

```
$ uhpp ssh://user@www.example.org/~/foo.html
```

6.2 UL4 – A templating language

ll.ul4c provides templating for XML/HTML as well as any other text-based format. A template defines placeholders for data output and basic logic (like loops and conditional blocks), that define how the final rendered output will look.

ll.ul4c compiles a template to an internal format, which makes it possible to implement renderers for these templates in multiple programming languages.

Note: Apart from this Python implementation there are implementations for Java (both a compiler and renderer) and Javascript (renderer only).

In the template source any text surrounded by <? and ?> is a "template tag". The first word inside the tag is the tag type. It defines what the tag does. For example `<?print foo?>` is a print tag (it prints the value of the variable `foo`). A complete example template looks like this:

```
<?if data?>
   <ul>
      <?for item in data?>
         <li><?print xmlescape(item)?></li>
      <?end for?>
   </ul>
<?end if?>
```

A complete Python program that compiles a template and renders it might look like this:

```
from ll import ul4c

code = '''
   <?if data?>
```

(continues on next page)

(continued from previous page)

```
      <ul>
         <?for item in data?>
            <li><?print item?></li>
         <?end for?>
      </ul>
   <?end if?>
'''

template = ul4c.Template(code)

print(template.renders(data=["Python", "Java", "Javascript", "PHP"]))
```

The variables that should be available to the template code can be passed to the method *renders()* as keyword arguments. *renders()* returns the final rendered output as a string. Alternatively the method *render()* can be used, which is a generator and returns the output piecewise.

For more information see the following chapters:

6.2.1 Literals

The following object types can be created and used insided templates:

- strings
- integers
- floats
- date objects
- color objects
- The "null" value (`None`)
- boolean values (`True` and `False`)
- lists
- dictionaries
- sets

Note that depending on the implementation language of the renderer additional types might be supported, e.g. a Python renderer will probably support both tuples and lists and anything supporting `__getitem__()` (or `__iter__()` when the list is used in a loop) for lists, Java might support anything implementing the `List` interface (or the `Collection` interface if the list is used in a loop).

Objects of these types can either be passed to the template in the call to the render function, or the template can create objects of thoses types itself. The syntax for creating such a constant is very similar to Python's syntax.

The "null" object

The "null" object can be referred to via `None`.

Boolean literals

The boolean constants can be referred to via `True` and `False`.

Integer literals

Integer constants can be written in decimal, hexadecimal, octal and binary: `42`, `0x2a`, `0o52` and `0b101010` all refer to the integer value 42.

Float literals

Float constants must contain a decimal point or an exponential specifier, e.g. `42.`, `4e23`.

String literals

Strings are delimited with single or double quotes and support all escape sequences that Python supports (except `\N{}`). Strings constants allow `\uXXXX` escaping. Examples:

- `"abc"` and `'abc'`;
- `"'"` and `'\''` are single quotes;
- `'"'` and `"\""` are double quotes;
- `"\n"` is a line feed and `"\t"` is a tab;
- `"\x61"` and `"\u0061"` are lowercase "a"s;

Strings can also be delimited with triple single or double quotes like in Python. These strings support embedded line feeds.

Date literals

Date objects have a year, month and day component and can be created like this:

- `@(2008-12-24)`

Datetime literals

Datetime objects have a date and time including microseconds and can be created like this:

- `@(2008-12-24T12:34)`
- `@(2008-12-24T12:34:56)`
- `@(2008-12-24T12:34:56.987654)`

Color literals

Color values are 8 bit red, green, blue and alpha values. Color constants can be created like this:

- `#fff`
- `#fff8`
- `#0063a8`
- `#0063a880`

The variants with 3 or 6 hex digits will create a color object with an alpha value of 255.

Lists

Lists can be created like this:

- `[]`
- `[1, 2, 3]`
- `[None, 42, "foo", [False, True]]`

* expressions can be used to expand other lists inplace, so

```
[1, *[2, 3], 4, *[5, 6]]
```

is equivalent to

```
[1, 2, 3, 4, 5, 6]
```

It is also possible to create a list with a list comprehension:

```
["(" + c.upper() + ")" for c in "hurz" if c < "u"]
```

This will create the list

```
["(H)", "(R)"]
```

The `if` condition is optional, i.e.

```
["(" + c.upper() + ")" for c in "hurz"]
```

will create the list

```
["(H)", "(U)", "(R)", "(Z)"]
```

Dictionaries

Dictionaries can be created like this:

- `{}`
- `{1: 2, 3: 4}`
- `{"foo": 17, "bar": 23}`

** expressions can be used to expand other dictionaries inplace, so:

```
{"foo": 17, **{"bar": 23, "baz": 42}}
```

is equivalent to

```
{"foo": 17, "bar": 23, "baz": 42}
```

The ** expression must be a dictionary or a list of key/value pairs.

It is also possible to create a dictionary with a dictionary comprehension:

```
{ c.upper() : "(" + c + ")" for c in "hurz" if c < "u"}
```

This will create the dictionary

```
{ "H": "(h)", "R": "(r)"}
```

The `if` condition is optional, i.e.

```
{ c.upper() : "(" + c + ")" for c in "hurz"}
```

will create the dictionary

```
{ "H": "(h)", "U": "(u)", "R": "(r)", "Z": "(z)"}
```

Sets

Sets can be created like this:

- `{/}` (this is the empty set)
- `{1, 2, 3}`
- `{"foo", "bar"}`

The empty set can also be created with the function `set`:

```
set()
```

* expressions are also supported:

```
{1, *[2, 3], 4, *[5, 6]}
```

It is also possible to create a set with a set comprehension:

```
{c.upper() for c in "hurz" if c < "u"}
```

This will create the set

```
{"H", "R"}
```

The `if` condition is optional, i.e.

```
{c.upper() for c in "hurz"}
```

will create the dictionary

```
{"H", "R", "U", "Z"}
```

The `Undefined` object

The object `Undefined` will be returned when a nonexistent variable, a nonexistent dictionary entry or an index that is out of range for a list/string is accessed.

6.2.2 Tags

The template code tries to mimic Python syntax as far as possible, but is limited to what is required for templates and does not allow executing arbitrary Python statements. In some spots it also borrows Javascript semantics.

`ll.ul4c` supports the following tag types:

<?print?>

The `print` tag outputs the value of a variable or any other expression. If the expression doesn't evaluate to a string it will be converted to a string first. The format of the string depends on the renderer, but should follow Python's `str()` output as much as possible:

```
<h1><?print person.lastname?>, <?print person.firstname?></h1>
```

Printing `None` or undefined objects produces no output.

Hint: The `<?print?>` tag is implemented by *`ll.ul4c.PrintAST`*.

<?printx?>

The `printx` tag outputs the value of a variable or any other expression and escapes the characters <, >, &, ' and " with the appropriate character or entity references for XML or HTML output.

Hint: The `<?printx?>` tag is implemented by *`ll.ul4c.PrintXAST`*.

<?for?>

The `for` tag can be used to loop over the items in a list, the characters in a string, the keys in a dictionary or any other iterable object. The end of the loop body must be marked with an `<?end for?>` tag:

```
<ul>
   <?for person in data.persons?>
      <li><?print person.lastname?>, <?print person.firstname?></li>
   <?end for?>
</ul>
```

In `for` loops variable unpacking is supported, so you can do the following:

```
<?for (key, value) in dict.items()?>
```

if `dict` is a dictionary.

This unpacking can be arbitrarily nested, i.e. the following is possible too:

```
<?for (i, (key, value)) in enumerate(dict.items())?>
```

Hint: The `<?for?>` tag is implemented by *`ll.ul4c.ForBlockAST`*.

<?break?>

The `break` tag can be used to break out of the innermost running loop.

Hint: The `<?break?>` tag is implemented by *`ll.ul4c.BreakAST`*.

<?continue?>

The `continue` tag can be used to skip the rest of the loop body of the innermost running loop and continue with the next iteration of the loop.

Hint: The `<?continue?>` tag is implemented by *`ll.ul4c.ContinueAST`*.

<?if?>

The `if` tag can be used to output a part of the template only when a condition is true. The end of the `if` block must be marked with an `<?end if?>` tag. The truth value of an object is mostly the same as in Python:

- `None` is false.
- The integer `0` and the float value `0.0` are false.
- Empty strings, lists and dictionaries are false.
- `timedelta` and `monthdelta` objects for an empty timespan (i.e. `timedelta(0, 0, 0)` and `monthdelta(0)`) are false.
- `False` is false.
- `Undefined` is false.
- Anything else is true.

For example we can output the person list only if there are any persons:

```
<?if persons?>
   <ul>
      <?for person in persons?>
         <li><?print person.lastname?>, <?print person.firstname?></li>
      <?end for?>
   </ul>
<?end if?>
```

`elif` and `else` are supported too:

```
<?if persons?>
   <ul>
      <?for person in persons?>
         <li><?print person.lastname?>, <?print person.firstname?></li>
      <?end for?>
   </ul>
<?else?>
   <p>No persons found!</p>
<?end if?>
```

or:

```
<?if len(persons)==0?>
   No persons found!
<?elif len(persons)==1?>
   One person found!
<?else?>
   <?print len(persons)?> persons found!
<?end if?>
```

Hint: The `<?if?>`, `<?elif?>` and `<?else?>` tags are implemented by *ll.ul4c.ConditionalBlocksAST*, *ll.ul4c.IfBlockAST*, *ll.ul4c.ElIfBlockAST* and *ll.ul4c.ElseBlockAST*.

`<?code?>`

The `code` tag can contain statements that define or modify variables or expressions which will be evaluated for their side effects. Apart from the assigment operator =, the following augmented assignment operators are supported:

- `+=` (adds a value to the variable)
- `-=` (subtracts a value from the variable)
- `*=` (multiplies the variable by a value)
- `/=` (divides the variable by a value)
- `//=` (divides the variable by a value, rounding down to the next smallest integer)
- `%=` (Does a modulo operation and replaces the variable value with the result)
- `<<=` (Does bitwise "shift left" operation and replaces the variable value with the result)
- `>>=` (Does bitwise "shift right" operation and replaces the variable value with the result)
- `&=` (Does bitwise "and" operation and replaces the variable value with the result)
- `|=` (Does bitwise "or" operation and replaces the variable value with the result)
- `^=` (Does bitwise "exclusive-or" operation and replaces the variable value with the result)

For example the following template will output `40`:

```
<?code x = 17?>
<?code x += 23?>
<?print x?>
```

Hint: The content of `<?code?>` tags is implemented as *UL4 expressions*.

`<?render?>`

The `render` tag allows one template to call other templates. The following Python code demonstrates this:

```
from ll import ul4c

# Template 1
source1 = """\
<?if data?>\
<ul>
<?for i in data?><?render itemtmpl(item=i)?><?end for?>\
</ul>
<?end if?>\
"""

tmpl1 = ul4c.Template(source1)

# Template 2
source2 = "<li><?print xmlescape(item)?></li>\n"
```

(continues on next page)

(continued from previous page)

```
tmpl2 = ul4c.Template(source2)

# Data object for the outer template
data = ["Python", "Java", "Javascript", "PHP"]

print(tmpl1.renders(itemtmpl=tmpl2, data=data))
```

This will output:

```
<ul>
<li>Python</li>
<li>Java</li>
<li>Javascript</li>
<li>PHP</li>
</ul>
```

I.e. templates can be passed just like any other object as a variable. `<?render itemtmpl(item=i)?>` renders the `itemtmpl` template and passes the `i` variable, which will be available in the inner template under the name `item`.

Hint: The `<?render?>` tag is implemented by *`ll.ul4c.RenderAST`*.

`<?renderx?>`

The `renderx` tag works similar to the `render` tag, except that the output of the template called will be XML escaped (like `printx` does). The following Python code demonstrates this:

```
from ll import ul4c

# Template 1
tmpl1 = ul4c.Template("<&>")

# Template 2
tmpl2 = ul4c.Template("<?renderx tmpl()?>\n")

print(tmpl1.renders(tmpl=tmpl2))
```

This will output:

```
&lt;&amp;&gt;
```

Hint: The `<?renderx?>` tag is implemented by *`ll.ul4c.RenderXAST`*.

<?render_or_print?>

The render_or_print tag combines the functionality of the render and the print tag, so for example

```
<?render_or_print foo(bar)?>
```

is more or less equivalent to

```
<?if istemplate(foo)?>
   <?render foo(bar)?>
<?else?>
   <?print foo?>
<?end if?>
```

i.e. if foo is renderable, it will be rendered, otherwise it will be printed. Furthermore the arguments to the call will always be evaluated even if foo isn't renderable, so for example:

```
<?render_or_print 'foo'(None+None)?>
```

will fail with:

```
<?render_or_print 'foo'(None+None)?>
                        ~~~~~~~~~
TypeError: unsupported operand type(s) for +: 'NoneType' and 'NoneType'
```

Hint: The <?render_or_print?> tag is implemented by *ll.ul4c.RenderOrPrintAST*.

<?render_or_printx?>

The render_or_printx tag is similar to render_or_print except that the object will be output via <?printx?> instead of <?print?> if it isn't renderable.

Hint: The <?render_or_printx?> tag is implemented by *ll.ul4c.RenderOrPrintXAST*.

<?renderx_or_print?>

The renderx_or_print tag is similar to render_or_print except that the object will be rendered via <?renderx?> instead of <?render?> if it is renderable.

Hint: The <?renderx_or_print?> tag is implemented by *ll.ul4c.RenderXOrPrintAST*.

<?renderx_or_printx?>

The renderx_or_printx tag is similar to renderx_or_print except that the object will be output via <?printx?> instead of <?print?> if it isn't renderable.

Hint: The <?renderx_or_printx?> tag is implemented by *ll.ul4c.RenderXOrPrintXAST*.

<?def?>

The `def` tag defines a new template as a variable. Usage looks like this:

```
<?def quote?>
   "<?print text?>"
<?end def?>
```

This defines a local variable `quote` that is a template object. This template can be rendered like any other template that has been passed to the outermost template:

```
<?render quote(text="foo")?>
```

It's also possible to include a signature in the definition of the template. This makes it possible to define default values for template variables and to call templates with positional arguments:

```
<?def quote(text='foo')?>
   "<?print text?>"
<?end def?>
<?render quote()?> and <?render quote("bar")?>
```

This will output `"foo" and "bar"`.

`*` and `**` arguments are also supported:

```
<?def weightedsum(*args)?>
   <?print sum(i*arg for (i, arg) in enumerate(args, 1))?>
<?end def?>
<?render weightedsum(17, 23, 42)?>
```

This will print `189` (i.e. `1 * 17 + 2 * 23 + 3 * 42`).

Hint: The `<?def?>` tag simply creates a *`Template`* object inside another *`Template`* object.

<?renderblocks?>

The `renderblocks` tag is syntactic sugar for rendering a template and passing other templates as arguments in the call. For example if we have the following template:

```
<?def page(head, body, lang="en", doctype=False)?>
   <?if doctype?>
      <!DOCTYPE html>
   <?end if?>
   <html lang="<?printx lang?>">
      <head>
         <?render head()?>
      </head>
      <body>
         <?render body()?>
      </body>
   </html>
<?end def?>
```

then we can render this template in the following way:

```
<?renderblocks page(lang="de", doctype=True)?>
   <?def head?>
```

(continues on next page)

(continued from previous page)

```
      <title>Foo</title>
   <?end def?>
   <?def body?>
      <h1>Bar!</h1>
   <?end def?>
<?end renderblocks?>
```

This is syntactic sugar for:

```
<?def head?>
   <title>Foo</title>
<?end def?>
<?def body?>
   <h1>Bar!</h1>
<?end def?>
<?render page(lang="de", doctype=True, head=head, body=body)?>
```

In both cases the output will be:

```
<!DOCTYPE html>
<html lang="de">
   <head>
      <title>Foo</title>
   </head>
   <body>
      <h1>Bar!</h1>
   </body>
</html>
```

All variables defined between `<?renderblocks page(...)?>` and `<?end renderblocks?>` are passed as additional keyword arguments in the render call to `page`. (But note that those variables will be local to the `<?renderblocks?>` block, i.e. they will not leak into the surrounding code.)

Hint: The `<?renderblocks?>` tag is implemented by *`ll.ul4c.RenderBlocksAST`*.

`<?renderblock?>`

The `renderblock` is a special version of `renderblocks`. The complete content of the `renderblock` block will be wrapped in a signatureless template named `content` and this template will be passed as the keyword argument `content` to the render call. With this we can define a generic template for HTML links:

```
<?def a(content, **attrs)?>
   <a<?for (an, av) in attrs.items()?> <?print an?>="<?printx av?>"<?end for?>>
      <?render content()?>
   </a>
<?end def?>
```

and then use it like this:

```
<?renderblock a(class="extern", href="http://www.python.org/")?>
   Link to the Python homepage
<?end renderblock?>
```

The output will be:

```
<a class="extern" href="http://www.python.org/">
   Link to the Python homepage
</a>
```

Hint: The `<?renderblock?>` tag is implemented by *`ll.ul4c.RenderBlockAST`*.

`<?return?>`

The `return` tag returns a value from the template when the template is called as a function. For more info see *Templates as functions*.

Hint: The `<?return?>` tag is implemented by *`ll.ul4c.ReturnAST`*.

`<?ul4?>`

The `ul4` tag can be used to specify a name and a signature for the template itself. This overwrites the name and signature specified in the `ul4c.Template` constructor:

```
>>> from ll import ul4c
>>> t = ul4c.Template("<?ul4 foo(x)?><?print x?>")
>>> t.name
'foo'
>>> t.signature
<Signature (x)>
```

Hint: The `<?ul4?>` tag has no corresponding AST nodes. Its content will set attributes of the template instead.

`<?note?>`

A `note` tag is a comment and can be used to explain the template code. When the template gets executed, the content of the tag will be completely ignored.

The `<?note?>` tag supports two variants:

- The comment can be included as the content of the tag:

  ```
  <?note comment?>
  ```

- The comment can be included between a `<?note?>` and an `<?end note?>` tag:

  ```
  <?note?>
  comment
  <?end note?>
  ```

  This second variant makes it possible to include UL4 source code in `<?note?>` tags.

Hint: A `<?note?>` tag has no corresponding AST nodes.

<?doc?>

A doc tag contains the documentation of the template itself. The content of the <?doc?> tag is available as the doc attribute:

```
>>> from ll import ul4c
>>> t = ul4c.Template("<?doc foo?><?print x?>")
>>> t.doc
'foo'
```

Each <?doc?> contains the documentation for the template to which the <?doc?> tag belongs, i.e. if the <?doc?> tag is at the outermost level, it belongs to the outermost template. If the <?doc?> tag is inside a local template, it is the documentation for the local template. If multiple <?doc?> tags are given, only the first one will be used, all later ones will be ignored.

The <?doc?> tag supports two variants:

- The description can be included as the content of the tag:

  ```
  <?doc description?>
  ```

- The description can be included between a <?doc?> and an <?end doc?> tag:

  ```
  <?doc?>
  description
  <?end doc?>
  ```

  This second variant makes it possible to include UL4 source code in <?doc?> tags.

Note that the template name, documentation and signature are accessible inside the templates themselves, i.e.:

```
<?def f(x=17, y=23)?>
   <?doc return the sum of x and y?>
   <?return x+y?>
<?end def?>
<?print f.name?>
<?print f.doc?>
<?print f.signature?>
```

will output:

```
f
return the sum of x and y
(x=17, y=23)
```

Hint: A <?doc?> tag has no corresponding AST nodes. Its content will set the doc property of the template instead.

`<?ignore?>`

An `ignore` tag can be used to "comment out" template code, so that the code will never be executed. `<?ignore?>` and `<?end ignore?>` tags nest, so code that already contains `<?ignore?>` and `<?end ignore?>` tags can be ignored by added additional `<?ignore?>` and `<?end ignore?>` tags around it.

It is not required that the content between the `<?ignore?>` and `<?end ignore?>` tag is proper UL4 code.

For example the follow template won't output anything:

```
<?ignore?>
   <?for i in range(20)?>
      <?print i?>
   <?end for?>
   <?ignore?>
      <?note Unfinished if?>
      <?if 42?>
   <?end ignore?>
<?end ignore?>
```

Hint: An `<?ignore?>` tag has no corresponding AST nodes.

`<?whitespace?>`

The `whitespace` tag can be used to overwrite the handling of whitespace in the template. For more info see *Whitespace handling*.

Hint: A `<?whitespace?>` tag has no corresponding AST nodes. Its content will set the `whitespace` attribute of the template instead.

6.2.3 Nested scopes

UL4 templates support lexical scopes. This means that a template that is defined (via `<?def?>`) inside another template has access to the local variables of the outer template. The inner template sees the state of the variables at the point in time when the inner templates gets called. The following example will output `2`:

```
<?code i = 1?>
<?def x?>
   <?print i?>
<?end def?>
<?code i = 2?>
<?render x()?>
```

6.2.4 Expressions

ll.ul4c supports many of the operators supported by Python. The following subchapters describe all expressions/operators that UL4 supports and are ordered from highest precedence to lowest.

Generator expressions

UL4 supports generator expressions with look like list comprehensions without the square brackets. Generator expression do not create lists in memory but instead return an iterable that can be iterated once. Function and methods that require an iterable argument can directly consume such iterables:

```
<?print ", ".join("(" + c + ")" for c in "gurk")?>
```

will output

```
(g), (u), (r), (k)
```

Outside of function/method arguments (or when more that one argument is passed) parentheses are required around generator expressions:

```
<?code ge = ("(" + c + ")" for c in "gurk")?>
<?print ", ".join(ge)?>
```

Hint: Generator expressions are implemented by *ll.ul4c.GeneratorExpressionAST*.

Index/slice access

Index and slice access is available for all container types, i.e. in the expression `a[b]` the following type combinations are supported:

- string, integer: Returns the `b`th character from the string `a`. Note that negative `b` values are supported and are relative to the end, so `a[-1]` is the last character.
- list, integer: Returns the `b`th list entry of the list `a`. Negative `b` values are supported too.
- dict, string: Return the value from the dictionary `a` corresponding to the key `b`. Note that some implementations might support keys other than strings too. (The Python and Java implementations do for example. The Javascript implementation does too, if `Map` is supported.)

If the specified key doesn't exist or the index is out of range for the string or list, a special "undefined" object is returned.

Slices are also supported (for list and string objects). As in Python one or both of the indexes may be missing to start at the first or end after the last character/item. Negative indexes are relative to the end. Indexes that are out of bounds are simply clipped, so

```
<?print "Hello, World!"[7:-1]?>
```

prints `World` and

```
<?print "Hello, World!"[:-8]?>
```

prints `Hello`.

Hint: Index/slice access is implemented by *ll.ul4c.ItemAST*.

Attribute access

For string keys it's also possible to access dictionary entries via the attribute access operator `.`, i.e. `foo.key` is the same as `foo["key"]` if `foo` is a dictionary.

Hint: Attribute access is implemented by *`ll.ul4c.AttrAST`*.

Function calls

A function call in UL4 looks like this: `date(2014, 10, 9, 17, 29)`. (this returns the date object `@(2014-10-09T17:29)`). Some of the trailing arguments in a function call might be optional and have default values. For example the first three arguments for the `date` function (`year`, `month` and `day`) are required, the remaining four arguments (`hour`, `minute`, `second` and `microsecond`) are optional and default to `0`.

Parameter values can also be passed via keyword arguments, i.e. `date(2014, 10, 9)` could also be written as `date(day=9, month=10, year=2014)`.

Furthermore Python's `*` and `**` syntax is supported for passing additional positional or keyword arguments. For example:

```
<?code args = [2014, 10, 9, 17, 29]?>
<?code d = date(*args)?>
```

is the same as:

```
<?code d = date(2014, 10, 9, 17, 29)?>
```

The same can also be done with a keyword dictionary and the `**` syntax:

```
<?code kwargs = {"day": 9, "month": 10, "year": 2014, "hour": 17: "minute": 29}?>
<?code d = date(**kwargs)?>
```

Of course it's also possible to mix argument passing mechanics:

```
<?code d = date(2014, *[10, 9], **{"hour": 17, "minute": 29})?>
```

or

```
<?code d = date(2014, month=10, day=9, **{'hour': 17, 'minute': 29})?>
```

However the `*` and `**` arguments can only be use at the end of the argument list and positional arguments must always be before keyword arguments.

A list of builtin functions can be found in *Functions*.

Hint: This documentation uses Python's `/` and `*` notation to specify positional-only and keyword-only arguments. So

```
<?ul4 f(x, /, y, *, z)?>
```

means that the function `f` accepts the parameter `x` only when passed by position, `y` can be passed either by position or by keyword and `z` will only be accepted when passed by keyword.

Hint: Function calls are implemented by *`ll.ul4c.CallAST`*.

Unary operators

Arithmetic negation

The unary operator `-` inverts the sign of its operand, which must be an integer, float of boolean value:

```
<?code x = 42?><?print -x?>
```

prints `-42`. For `-` boolean values are treated as the numbers `0` and `1`, i.e.:

```
<?code x = True?><?print -x?>
```

prints `-1`.

Hint: Arithmetic negation is implemented by *`ll.ul4c.NegAST`*.

Binary negation

The unary operator `~` inverts the bits of an integer or boolean value. Non-negative numbers are interpreted as having an unlimited number of leading `0` bits and negative numbers are interpreted as having an unlimited number of leading `1` bits. The means that `~x` will be negative if `x` is non-negative and vice versa.

Hint: Arithmetic negation is implemented by *`ll.ul4c.BitNotAST`*.

Multiplicative binary operators

Multiplication

The multiplication operator `*` returns the arithmetic product of its operands (which must be integer, float or boolean values). Furthermore it's possible to multiply a sequence (i.e. a string or list) with a non-negative integer to get a new sequences that repeats the items of the original sequence a number of times, e.g. `"foo" * 2` returns `"foofoo"` and `[1, 2, 3] * 3` return `[1, 2, 3, 1, 2, 3, 1, 2, 3]`. Multiplying with `0` returns an empty string or list.

Hint: Multiplication is implemented by *`ll.ul4c.MulAST`*.

True division

The true division operator `/` returns the quotient of its operands (which must be integer, float or boolean values). The result is always a float value. `1/2` returns `0.5`.

Hint: True division is implemented by *`ll.ul4c.TrueDivAST`*.

Floor division

The float division operator `//` returns the quotient of its operands (which must be integer, float or boolean values) rounded down to an integer (rounding is always done towards -infinity, i.e. `(-25)/10` returns `-3`). If any of the operands is a float the result is a float too, otherwise it's an integer.

Hint: Floor division is implemented by *`ll.ul4c.FloorDivAST`*.

Modulo

The modulo operator `%` returns the remainder from the division of the first operand by the second, e.g. `15 % 7` returns 1.

Hint: The modulo operator is implemented by *`ll.ul4c.ModAST`*.

Additive binary operators

Addition

The addition operator + returns the sum of its operands (which must be integer, float or boolean values). Furthermore sequences of the same type can be added, so `"foo" + "bar"` returns `"foobar"` and `[1, 2] + [3, 4]` returns `[1, 2, 3, 4]`.

Hint: Addition is implemented by *`ll.ul4c.AddAST`*.

Subtraction

The subtraction operator `-` returns the difference of its operands (which must be integer, float or boolean values).

Hint: Subtraction is implemented by *`ll.ul4c.SubAST`*.

Bit shift operators

Binary left shift operator

The binary left shift operator << shifts the bits of its first operand (an integer or boolean) to the left by the number of positions given by the second operand (which must also be an integer or boolean).

Hint: The binary left shift operator is implemented by *`ll.ul4c.ShiftLeftAST`*.

Binary right shift operator

The binary right shift operator >> shifts the bits of its first operand (an integer or boolean) to the right by the number of positions given by the second operand (which must also be an integer or boolean).

Hint: The binary right shift operator is implemented by *ll.ul4c.ShiftRightAST*.

Binary bitwise "and" operator

The bitwise and operator & returns the bitwise "and" combination of its operands (which must be integer or boolean values). E.g. `6 & 3` returns `2`.

As with the unary operator ~, negative numbers are interpreted as having an unlimited number of leading `1` bits.

Hint: The binary bitwise "and" operator is implemented by *ll.ul4c.BitAndAST*.

Binary bitwise "exclusive or" operator

The bitwise exclusive or operator ^ returns the bitwise exclusive "or" combination of its operands (which must be integer or boolean values). E.g. `6 ^ 3` returns `5`.

Negative numbers are again interpreted as having an unlimited number of leading `1` bits.

Hint: The binary bitwise "exclusive or" operator is implemented by *ll.ul4c.BitXOrAST*.

Binary bitwise "inclusive or" operator

The bitwise inclusive or operator | returns the bitwise inclusive "or" combination of its operands (which must be integer or boolean values). E.g. `6 | 3` returns `7`.

Negative numbers are again interpreted as having an unlimited number of leading `1` bits.

Hint: The binary bitwise "inclusive or" operator is implemented by *ll.ul4c.BitOrAST*.

Binary comparison operators

The comparison operators ==, !=, <, <=, > and >= compare the value of the two operands. == and != support comparison of all types of object. All others support comparison of "compatible" objects, which means all "number" objects (integer, float and boolean) can be compared with each other, all other objects can only be compared to objects of the same type.

Hint: These operators are implemented by *ll.ul4c.EQAST*, *ll.ul4c.NEAST*, *ll.ul4c.LTAST*, *ll.ul4c.LEAST*, *ll.ul4c.GTAST* and *ll.ul4c.GEAST*.

Identity comparison operators

The comparison operators `is` and `is not` test whether both operands refer to the same object or not.

Note that the behaviour of these operators for "atomic" immutable objects (like integers, floats and strings) is implementation defined.

Hint: These operators are implemented by *ll.ul4c.IsAST* and *ll.ul4c.IsNotAST*.

Containment tests

The `in` operator

The `in` operator tests whether the first operand is contained in the second operand. In the expression `a in b` the following type combinations are supported:

- string, string: Checks whether `a` is a substring of `b`.
- any object, list: Checks whether the object `a` is in the list `b` (comparison is done by value not by identity)
- string, dict: Checks whether the key `a` is in the dictionary `b`. (Note that some implementations might support keys other than strings too. E.g. Python and Java do, Javascript does only for `Map` objects.)

Hint: The `in` operator is implemented by *ll.ul4c.ContainsAST*.

The `not in` operator

The `not in` operator returns the inverted result of the `in` operator, i.e. it tests whether the first operand is not contained in the second operand.

Hint: The `not in` operator is implemented by *ll.ul4c.NotContainsAST*.

Boolean negation

The unary operator `not` inverts the truth value of its operand. I.e. `not x` is `True` for `None`, `False`, the undefined value, `0`, `0.0`, empty lists, strings, dictionaries and other empty containers and `False` for everything else.

Hint: The boolean negation operator is implemented by *ll.ul4c.NotAST*.

Boolean "and" operator

The binary operator `and` returns whether both of its operands are true. It works like Python `and` operator by short-circuiting operand evaluation, i.e. if the result is clear from the first operand the seconds won't be evaluated.

Furthermore `and` always return one of the operands.

So `a and b` first evaluates `a`; if `a` is false, its value is returned; otherwise, `b` is evaluated and the resulting value is returned.

Hint: The boolean "and" operator is implemented by *ll.ul4c.AndAST*.

Boolean "or" operator

The binary operator `or` returns whether any of its operands is true. Like `and` evaluation is short-circuited and one of the operands is returned.

For example, the following code will output the `data.title` object if it's true, else `data.id` will be output:

```
<?printx data.title or data.id?>
```

Hint: The boolean "or" operator is implemented by *ll.ul4c.OrAST*.

Conditional expression

The conditional expression (also called an "inline if") `a if c else b` first evaluates the condition `c`. If it is true `a` is evaluated and returned else `b` is evaluated and returned.

Hint: The "inline if" operator is implemented by *ll.ul4c.IfAST*.

6.2.5 Functions

ll.ul4c supports a number of functions.

`today()`

`today()` returns the current date as a date object.

`now()`

`now()` returns the current date and time as a datetime object.

`utcnow()`

`utcnow()` returns the current date and time as a datetime object in UTC.

`isundefined(obj, /)`

`isundefined(foo)` returns `True` if `foo` is `Undefined`, else `False` is returned:

```
data is <?if isundefined(data)?>undefined<?else?>defined<?end if?>!
```

isdefined(obj, /)

isdefined(foo) returns False if foo is Undefined, else True is returned:

```
data is <?if isdefined(data)?>defined<?else?>undefined<?end if?>!
```

isnone(obj, /)

isnone(foo) returns True if foo is None, else False is returned:

```
data is <?if isnone(data)?>None<?else?>something else<?end if?>!
```

isbool(obj, /)

isbool(foo) returns True if foo is True or False, else False is returned.

isint(obj, /)

isint(foo) returns True if foo is an integer object, else False is returned.

isfloat(obj, /)

isfloat(foo) returns True if foo is a float object, else False is returned.

isstr(obj, /)

isstr(foo) returns True if foo is a string object, else False is returned.

isdate(obj, /)

isdate(foo) returns True if foo is a date object, else False is returned.

istimedelta(obj, /)

istimedelta(foo) returns True if foo is a timedelta object, else False is returned.

ismonthdelta(obj, /)

ismonthdelta(foo) returns True if foo is a monthdelta object, else False is returned.

islist(obj, /)

islist(foo) returns True if foo is a list object, else False is returned.

isdict(obj, /)

`isdict(foo)` returns `True` if `foo` is a dictionary object, else `False` is returned.

isset(obj, /)

`isset(foo)` returns `True` if `foo` is a set object, else `False` is returned.

isexception(obj, /)

`isexception(foo)` returns `True` if `foo` is an exception object, else `False` is returned.

iscolor(obj, /)

`iscolor(foo)` returns `True` if `foo` is a color object, else `False` is returned.

istemplate(obj, /)

`istemplate(foo)` returns `True` if `foo` is a template object, else `False` is returned.

isinstance(obj, type, /)

`istemplate(obj, type)` returns `True` if `obj` is a instance of the type `type`. `type` must be a type object. For type objects see *Types*. For example

```
<?print isinstance("gurk", str)?>
```

prints `True`.

repr(obj, /)

`repr(foo)` converts `foo` to a string representation that is useful for debugging proposes. The output in most cases looks that the UL4 constant that could be used to recreate the object.

ascii(obj, /)

`ascii(foo)` produces the same output as `repr(foo)` except that all non-ASCII characters in the output for strings will be escaped.

format(value, spec, lang="en")

`format` formats a value. Currently `format` supports the following types for `value`: `date`, `int` and `float` (`float` is only supported in the Python version).

The second argument `spec` is a format specification string (whose format is specific to the type of `value`).

The third (optional) argument `lang` is the target language.

So for example

```
<?print format(@(2000-02-29), "%a, %d. %b. %Y", "de")?>
```

outputs `Di, 29. Feb. 2000` and

```
<?print format(42, "08b")?>
```

outputs `00101010`.

UL4 tries to follow Pythons convention for the format string specification, so for more information see the documentation for Pythons `format()` function.

slice(iterable, start=None, stop, step=None, /)

`slice` returns a slice from a sequence or iterator. You can either pass the stop index (i.e. `slice(foo, 10)` is an iterator over the first 10 items from `foo`), or a start and stop index (`slice(foo, 10, 20)` return the 11th upto to 20th item from `foo`) or a start and stop index and a step size. If given start and stop must be non-negative and step must be positive.

asjson(obj, /)

`asjson(foo)` returns a JSON representation of the object `foo`. (Date objects, color objects and templates are not supported by JSON, but `asjson` will output the appropriate Javascript code for those objects).

fromjson(string, /)

`fromjson(foo)` decodes the JSON string `foo` and returns the resulting object. (Date objects, color objects and templates are not supported by `fromjson`).

asul4on(obj, /, indent=None)

`asul4on(foo)` returns the UL4ON representation of the object `foo`.

fromul4on(dump, /)

`fromul4on(foo)` decodes the UL4ON string `foo` and returns the resulting object.

csv(obj, /)

`csv(foo)` formats the value `foo` for output into a CSV file.

len(obj, /)

`len(foo)` returns the length of a string, or the number of items in a list or dictionary.

round(number, /, digits=0)

Returns `number` rounded to `digits` precision after the decimal point. If `digits` is non-positive the returned value will always be of type `int`.

For example `round(42.123, 2)` returns `42.12` and `round(485, -2)` returns 500.

floor(number, /, digits=0)

Returns number rounded down (i.e. towards -∞) to digits precision after the decimal point. If digits is non-positive the returned value will always be of type int.

For example floor(42.567, 2) returns 42.56 and floor(485, -2) returns 400.

ceil(number, /, digits=0)

Returns number rounded up (i.e. towards ∞) to digits precision after the decimal point. If digits is non-positive the returned value will always be of type int.

For example ceil(42.567, 2) returns 42.57 and ceil(485, -2) returns 500.

any(iterable, /)

any(foo) returns True if any of the items in the iterable foo is true. Otherwise False is returns. If foo is empty False is returned.

all(iterable, /)

all(foo) returns True if all of the items in the iterable foo are true. Otherwise False is returns. If foo is empty True is returned.

enumerate(iterable, start=0)

Enumerates the items of the argument (which must be iterable, i.e. a string, a list or dictionary) and for each item in the original iterable returns a two item list containing the item position and the item itself. For example the following code:

```
<?for (i, c) in enumerate("foo")?>
   (<?print c?>=<?print i?>)
<?end for?>
```

prints

```
(f=0)(o=1)(o=2)
```

isfirstlast(iterable, /)

Iterates through items of the argument (which must be iterable, i.e. a string, a list or dictionary) and gives information about whether the item is the first and/or last in the iterable. For example the following code:

```
<?for (first, last, c) in isfirstlast("foo")?>
   <?if first?>[<?end if?>
   (<?print c?>)
   <?if last?>]<?end if?>
<?end for?>
```

prints

```
[(f)(o)(o)]
```

isfirst(iterable, /)

Iterates through items of the argument (which must be iterable, i.e. a string, a list or dictionary) and gives information about whether the item is the first in the iterable. For example the following code:

```
<?for (first, c) in isfirst("foo")?>
   <?if first?>[<?end if?>
   (<?print c?>)
<?end for?>
```

prints

```
[(f)(o)(o)
```

islast(iterable, /)

Iterates through items of the argument (which must be iterable, i.e. a string, a list or dictionary) and gives information about whether the item is the last in the iterable. For example the following code:

```
<?for (last, c) in islast("foo")?>
   (<?print c?>)
   <?if last?>]<?end if?>
<?end for?>
```

prints

```
(f)(o)(o)]
```

enumfl(iterable, /)

This function is a combination of `enumerate` and `isfirstlast`. It iterates through items of the argument (which must be iterable, i.e. a string, a list or dictionary) and gives information about whether the item is the first and/or last in the iterable and its position. For example the following code:

```
<?for (index, first, last, c) in enumfl("foo")?>
   <?if first?>[<?end if?>
   (<?print c?>=<?print index?>)
   <?if last?>]<?end if?>
<?end for?>
```

prints

```
[(f=0)(o=1)(o=2)]
```

first(iterable, /, default=None)

`first` returns the first element produced by an iterable object. If the iterable is empty the default value (which is the second parameter and defaults to `None`) is returned.

last(iterable, /, default=None)

`last` returns the last element produced by an iterable object. If the iterable is empty the default value (which is the second parameter and defaults to `None`) is returned.

xmlescape(obj, /)

`xmlescape` takes a string as an argument. It returns a new string where the characters &, <, >, ' and " have been replaced with the appropriate XML entity or character reference. For example:

```
<?print xmlescape("<'foo' & 'bar'>")?>
```

prints

```
&lt;&#39;foo&#39; &amp; ;&#39;bar&#39&gt;
```

If the argument is not a string, it will be converted to a string first.

`<?printx foo?>` is a shortcut for `<?print xmlescape(foo)?>`.

min(*args, default=<nodefault>, key=None)

`min` returns the minimum value of its two or more arguments. If it's called with one argument, this argument must be iterable and `min` returns the minimum value of this argument. if called with one empty argument the value of `default` will be returned (if given, else an exception will be raised).

If `key` is given, it will be used for extracting comparison keys, i.e. those keys will be compared instead of the items themselves for determining the minimal item.

If multiple items are minimal, the function returns the first one encountered.

max(*args, default=<nodefault>, key=None)

`max` returns the maximum value of its two or more arguments. If it's called with one argument, this argument must be iterable and `max` returns the maximum value of this argument. The arguments `default` and `key` work the same way as for `min()`.

sum(iterable, /, start=0)

`sum` returns the sum of the number from the iterable passed in. The second parameter is the start value (i.e. the value that will be added to the total sum) and defaults to 0. For example the template `<?print sum(range(101))?>` will output `5050`.

sorted(iterable, /, key=None, reverse=False)

`sorted` returns a sorted list with the items from its argument. For example:

```
<?for c in sorted('abracadabra')?><?print c?><?end for?>
```

prints

```
aaaaabbcdrr
```

Supported arguments are iterable objects, i.e. strings, lists, dictionaries and colors.

If `key` is given, it will be used for extracting comparison keys, i.e. those keys will be compared instead of the items themselves for determining the final order.

If `reverse` is true, the sort order will be reversed.

`chr(i, /)`

`chr(i)` returns a one-character string containing the character with the code point `i`. `i` must be an integer. For example `<?print chr(0x61)?>` outputs `a`.

`ord(c, /)`

This is the inverse function to `chr` The argument for `ord` must be a one-character string. `ord` returns the code point of that character as an integer. For example `<?print ord('a')?>` outputs `97`.

`hex(number, /)`

Return the hexadecimal representation of the integer argument (with a leading `0x`). For example `<?print hex(42)?>` outputs `0x2a`.

`oct(number, /)`

Return the octal representation of the integer argument (with a leading `0o`). For example `<?print oct(42)?>` outputs `0o52`.

`bin(number, /)`

Return the binary representation of the integer argument (with a leading `0b`). For example `<?print bin(42)?>` outputs `0b101010`.

`range(start=None, stop, step=None, /)`

`range` returns an object that can be iterated and will produce consecutive integers up to the specified argument. With two arguments the first is the start value and the second is the stop value. With three arguments the third one is the step size (which can be negative). For example the following template:

```
<?for i in range(4, 10, 2)?>(<?print i?>)<?end for?>
```

prints

```
(4)(6)(8)
```

`rgb(r, g, b, a=1.0)`

`rgb` returns a color object. It can be called with

- three arguments, the red, green and blue values. The alpha value will be set to 255;
- four arguments, the red, green, blue and alpha values.

Arguments are treated as values from 0 to 1 and will be clipped accordingly. For example:

```
<?print rgb(1, 1, 1)?>
```

prints `#fff`.

md5(string, /)

`md5(s)` returns the MD5 hash of the string `s`.

scrypt(string, /, salt)

`scrypt(str, salt)` returns the scrypt hash of the string `str` using the salt value `salt`. The returned string contains 256 hex digits.

For more info on scrypt, see https://en.wikipedia.org/wiki/Scrypt

Note: `scrypt` is not implemented in the Javascript version of UL4.

random()

`random()` returns a random float value between 0 (included) and 1 (excluded).

randrange(start=None, stop, step=None, /)

`randrange(start, stop, step)` returns a random integer value between `start` (included) and `stop` (excluded). `step` specifies the step size (i.e. when `r` is the random value, `(r-start) % step` will always be `0`). `step` and `start` can be omitted.

randchoice(seq)

`randchoice(seq)` returns a random item from the sequence `seq`.

urlquote(string)

`urlquote` escaped special characters for including the output in URLs. For example:

```
<?print urlquote("/\xff")?>
```

prints

```
%2F%C3%BF
```

urlunquote(string)

`urlunquote` is the inverse function to `urlquote`. So:

```
<?print urlunquote("%2F%C3%BC")?>
```

prints

```
/ü
```

type(obj, /)

type returns the type of an object as a type object. For type object see the following description.

getattr(obj, attrname, default=?, /)

getattr returns the attribute named attrname of the object obj. If obj doesn't have an attribute with that name default will returned (when passed, else an UndefinedKey object will returned).

hasattr(obj, attrname, /)

hasattr returns whether the object obj has an attribute named attrname.

6.2.6 Types

A type object can be used for testing whether an object is an instance of that type with the function *isinstance()*. Some type objects are callable to create new instances of that type (all of the following builtin type objects are).

A type object has the attributes __module__, __name__ and __doc__.

bool(obj=False, /)

bool(foo) converts foo to an boolean. I.e. True or False is returned according to the truth value of foo. Calling bool without arguments returns False.

int(obj=0, /, base=None)

int(foo) converts foo to an integer. foo can be a string, a float, a boolean or an integer. int can also be called with two arguments. In this case the first argument must be a string and the second is the number base for the conversion. Calling int without arguments returns 0.

float(obj=0.0, /)

float(foo) converts foo to a float. foo can be a string, a float, a boolean or an integer. Calling float without arguments returns 0.0.

str(obj="", /)

str(foo) converts foo to a string. If foo is None or Undefined the result will be the empty string. For lists and dictionaries the exact format is undefined, but should follow Python's repr format. For color objects the result is a CSS expression (e.g. "#fff"). Calling str without arguments returns the empty string.

date(year, month, day, hour=0, minute=0, second=0, microsecond=0)

date() creates a date object from the parameter passed in. date() supports from three parameters (year, month, day) upto seven parameters (year, month, day, hour, minute, second, microsecond).

timedelta(days=0, seconds=0, microseconds=0)

timedelta returns an object that represents a timespan. timedelta allows from zero to three arguments specifying the numbers of days, seconds and microseconds. Passing negative values or values that are out of bounds (e.g. 24*60*60+1 seconds) is allowed. Arguments default to 0, i.e. timedelta() returns the timespan for "0 days, 0 seconds, 0 microseconds". In a boolean context this object is treated as false (i.e. bool(timedelta())) returns False). The following arithmetic operations are supported:

- date + timedelta
- date - timedelta
- timedelta + timedelta
- timedelta - timedelta
- number * timedelta
- timedelta * number
- timedelta / number
- timedelta // int

monthdelta(months=0, /)

monthdelta returns an object that represents a timespan of a number of months. monthdelta allows zero or one arguments. With zero arguments monthdelta returns the timespan for "0 months". In a boolean context this object is treated as false (i.e. bool(monthdelta())) or bool(monthdelta(0))) return False). The following arithmetic operations are supported:

- date + monthdelta
- date - monthdelta
- monthdelta + monthdelta
- monthdelta - monthdelta
- int * monthdelta
- monthdelta // int

For operations involving date objects, if the resulting day falls out of the range of valid days for the target month, the last day for the target month will be used instead, i.e. <?print @(2000-01-31) + monthdelta(1)?> prints 2000-02-29.

color(r=0, g=0, b=0, a=255)

color returns a color object. Each argument must be an integer between 0 and 255. Values will be clipped accordingly. For example

```
<?print color(0x33, 0x66, 0x99)?>
```

prints #369 and

```
<?print color(0x33, 0x66, 0x99, 0xcc)?>
```

prints rgba(51,102,153,0.800).

list(iterable=[], /)

list(foo) converts foo to a list. This works for lists, strings and all iterable objects. Calling list without arguments returns an empty list.

dict(*args, **kwargs)

dict(foo) converts foo to a dictionary. For this, foo must be either a dictionary itself, or an iterable of (key, value) pairs. dict() supports multiple positional arguments. Later entries overwrite earlier ones. (i.e. dict({17: 23}, {17: 42}) returns {17: 42}). dict also supports arbitrary keyword arguments, which create dictionary entries with the name of the argument as a string key, so dict(foo=42) returns {'foo': 42}. Calling dict without arguments returns an empty dictionary.

set(iterable=[], /)

set(foo) converts foo to a set. This works for lists, strings and all iterable objects. Calling set without arguments returns an empty set.

6.2.7 Modules

A module is an object that collects a number of constants, functions and types under a common name. Also a module always has the attributes __name__ and __doc__.

operator

The operator module contains the type attrgetter. The object attrgetter(n) is callable and when called with an object a returns as attribute named n. This can be used for sorting objects by their attributes without having to create local templates.

For example

```
<?print operator.attrgetter('year')(now())()?>
```

prints 2021.

math

This module contains constants and functions related to mathematical operations:

- math.e is the base of the natural logarithm (2.718281828...).
- math.pi is the ratio of a circle's circumference to its diameter (3.14159265...).
- math.tau is 2 * math.pi.
- The trigonometric function math.cos() return the cosine of it argument (which must be in radians).
- The trigonometric function math.sin() return the sine of it argument (which must be in radians).
- The trigonometric function math.tan() return the tangent of it argument (which must be in radians).
- math.sqrt() which returns the square root of its argument.
- math.isclose(a, b) returns True if the values a and b are close to each other and False otherwise. Whether or not two values are considered close is determined according to given absolute and relative tolerances (via the keyword arguments rel_tol and abs_tol).

ul4on

This module contains functions and types for working with *ll.ul4on*:

- `ul4on.dumps(foo)` returns the UL4ON representation of the object `foo`.
- `ul4on.loads(foo)` decodes the UL4ON string `foo` and returns the resulting object.
- `ul4on.Encoder(indent)` creates an encoder object that can be used to create multiple UL4ON dump using the same context. An `ul4on.Encoder` object has a method `dumps` that creates an UL4ON dump for the passed in object.
- `ul4on.Decoder()` creates a decoder object that can be used to recreate objects from multiple UL4ON dumps using the same context. An `ul4on.Decoder` object has a method `loads` that recreates an object from the passed in UL4ON dump.

ul4

This module contains all the types of the nodes in the abstract syntax tree that comprises an UL4 template. The available types are: `TextAST`, `IndentAST`, `LineEndAST`, `CodeAST`, `ConstAST`, `SeqItemAST`, `UnpackSeqItemAST`, `DictItemAST`, `UnpackDictItemAST`, `PositionalArgumentAST`, `KeywordArgumentAST`, `UnpackListArgumentAST`, `UnpackDictArgumentAST`, `ListAST`, `ListComprehensionAST`, `SetAST`, `SetComprehensionAST`, `DictAST`, `DictComprehensionAST`, `GeneratorExpressionAST`, `VarAST`, `BlockAST`, `ConditionalBlocksAST`, `IfBlockAST`, `ElIfBlockAST`, `ElseBlockAST`, `ForBlockAST`, `WhileBlockAST`, `BreakAST`, `ContinueAST`, `AttrAST`, `SliceAST`, `UnaryAST`, `NotAST`, `NegAST`, `BitNotAST`, `PrintAST`, `PrintXAST`, `ReturnAST`, `BinaryAST`, `ItemAST`, `IsAST`, `IsNotAST`, `EQAST`, `NEAST`, `LTAST`, `LEAST`, `GTAST`, `GEAST`, `ContainsAST`, `NotContainsAST`, `AddAST`, `SubAST`, `MulAST`, `FloorDivAST`, `TrueDivAST`, `ModAST`, `ShiftLeftAST`, `ShiftRightAST`, `BitAndAST`, `BitXOrAST`, `BitOrAST`, `AndAST`, `OrAST`, `IfAST`, `ChangeVarAST`, `SetVarAST`, `AddVarAST`, `SubVarAST`, `MulVarAST`, `FloorDivVarAST`, `TrueDivVarAST`, `ModVarAST`, `ShiftLeftVarAST`, `ShiftRightVarAST`, `BitAndVarAST`, `BitXOrVarAST`, `BitOrVarAST`, `CallAST`, `RenderAST`, `RenderXAST`, `RenderBlockAST`, `RenderBlocksAST`, `SignatureAST`, `Template` and `TemplateClosure`.

The only callable type in this list is `Template` which can be used to create an UL4 template from source. Its signature is

```
(
   source=None,
   name=None,
   *,
   whitespace="keep",
   signature=None
)
```

Note: `ul4.Template` is not callable in the Javascript version of UL4.

color

This module contains the following types and functions:

`color.Color(r=0, g=0, b=0, a=255)`

Return a color object with the red component `r`, the green component `g`, the blue component `b` and the alpha (opacity) component `a`. All values must be integers in the range 0 to 255. Float values will be rounded to integers and values out of range will be clipped to the allowed range.

`color.css(value, default)`

Return a color object for the CSS color specification `value`. If `value` can't be interpreted as a CSS color, `default` will be returned if given, otherwise an exception will be raised.

color.mix(*values)

Calculates a weighted mix of the colors from `values`. Items in `values` are either colors or weights. The following example mixes two parts black with one part white:

```
<?print color.mix(2, #000, 1, #fff)?>
```

which will print:

```
#555
```

6.2.8 Methods

Objects in *ll.ul4c* support some methods too (depending on the type of the object).

str

Strings support the following methods:

upper()

The `upper` method of strings returns an uppercase version of the string for which it's called:

```
<?print 'foo'.upper()?>
```

prints FOO.

lower()

The `lower` method of strings returns an lowercase version of the string for which it's called.

capitalize()

The `capitalize` method of strings returns a copy of the string with its first letter capitalized.

startswith(prefix, /)

`x.startswith(y)` returns `True` if the string `x` starts with the string `y` and `False` otherwise. `y` may also be a list of string. In this case `x.startswith(y)` returns `True` if `x` starts with any of the strings in `y`.

endswith(suffix, /)

`x.endswith(y)` returns `True` if the string `x` ends with the string `y` and `False` otherwise. `y` may also be a list of string. In this case `x.endswith(y)` returns `True` if `x` ends with any of the strings in `y`.

strip(chars=None, /)

The string method `strip` returns a copy of the string with leading and trailing whitespace removed. If an argument `chars` is given and not `None`, characters in `chars` will be removed instead.

lstrip(chars=None, /)

The string method `lstrip` returns a copy of the string with leading whitespace removed. If an argument `chars` is given and not `None`, characters in `chars` will be removed instead.

rstrip(chars=None, /)

The string method `rstrip` returns a copy of the string with trailing whitespace removed. If an argument `chars` is given and not `None`, characters in `chars` will be removed instead.

split(sep=None, maxsplit=None)

The string method `split` splits the string into separate "words" and returns the resulting list. Without any arguments, the string is split on whitespace characters. With one argument the argument specifies the separator to use. The second optional argument specifies the maximum number of splits to do.

rsplit(sep=None, maxsplit=None)

The string method `rsplit` works like `split`, except that splitting starts from the end (which is only relevant when the maximum number of splits is given).

splitlines(keepends=False)

The string method `splitlines` splits the string into a list of lines, using Unicode line ending characters, i.e. the following character sequences terminate a line: `"\n"`, `"\r"`, `"\r\n"`, `"\x0b"`, `"\x0c"`, `"\x1c"`, `"\x1d"`, `"\x1e"`, `"\x85"`, `"\u2028"` and `"\u2029"`. Line breaks are not included in the resulting list unless a second parameter is given and true.

count(sub, start=None, end=None, /)

This method counts non-overlapping occurrences of a substring in a string. For example `"abababa".count("aba")` returns 2. The optional second and third argument specify the start and end position for the search.

find(sub, start=None, end=None, /)

This method searches for a substring of the string and returns the position of the first appearance of the substring or -1 if the substring can't be found. For example `"foobar".find("bar")` returns 3. The optional second and third argument specify the start and end position for the search.

rfind(sub, start=None, end=None, /)

This method works like `find` but searches from the end.

replace(old, new, count=-1, /)

The string method `replace` has two arguments. It returns a new string where each occurrence of the first argument is replaced by the second argument, i.e. `"abracadabra".replace("ab", "ba")` returns `"baracadbara"`. If the third argument `count` non-negative is specifies the maximum number of replacements.

join(iterable, /)

`join` is a string method. It returns a concatentation of the strings in the argument sequence with the string itself as the separator, i.e.:

```
<?print "+".join("1234")?>
```

outputs `1+2+3+4`.

list

List objects support the following methods:

count(sub, start=None, end=None, /)

This method counts occurrences of an item in a list. The optional second and third argument specify the start and end position for the search.

find(sub, start=None, end=None, /)

This method searches for an item in a list and returns the position of the first appearance of the item or -1 if the item can't be found. The optional second and third argument specify the start and end position for the search.

rfind(sub, start=None, end=None, /)

This method works like `find` but searches from the end.

append(*items)

`append` its arguments to the end of the list for which it is called:

```
<?code v = [1, 2]?>
<?code v.append(3, 4)?>
<?print v?>
```

prints `[1, 2, 3, 4]`.

insert(pos, *items)

`inserts` first argument in the insert position, the remaining arguments are the items that will be inserted at that position, so:

```
<?code v = [1, 4]?>
<?code v.insert(1, 2, 3)?>
<?print v?>
```

prints `[1, 2, 3, 4]`.

dict

Dictionaries have the following methods:

keys()

Return an iterator over the keys ob the dictionary (this is the same as iterating over the dictionary itself).

items()

Return an iterator over entries of the dictionary as (key, value) pairs.

values()

Return an iterator over values of the dictionary.

get(key, default=None, /)

`get` is a dictionary method. `d.get(k, v)` returns `d[k]` if the key `k` is in `d`, else `v` is returned. If `v` is not given, it defaults to `None`.

update(*args, **kwargs)

`update` supports an arbitrary number of positional and keyword arguments. Each positional argument may be a dictionary, all the items in the dictionary will be copied to the target dictionary. A positional argument may also be an iterable of `(key, value)` pairs. These will also be copied to the target dictionary. After each positional argument is copied over in a last step the keyword arguments will be copied to the target dictionary.

clear()

`clear()` makes a dictionary empty.

set

Set object have the following methods:

add(*items)

add() adds all arguments to the set.

clear()

clear() makes a set empty.

pop(pos=-1)

pop removes the last item of the list and returns it. If an index is passed the item at that position is removed and returned. A negative index is treated as relative to the end of the list.

Templates

Templates have the following method:

renders(...)

The renders method of template objects renders the template and returns the output as a string. Parameters can be passed via keyword arguments or via the ** syntax:

```
<?code output = template.renders(a=17, b=23)?>
<?code data = {'a': 17, 'b': 23}?>
<?code output = template.renders(**data)?>
```

(Also if the template has a signature, positional arguments and the * syntax are supported.)

date and datetime

date and datetime objects have the following methods:

isoformat()

isoformat returns the date/datetime object in ISO 8601 format, i.e.:

```
<?print now().isoformat()?>
```

might output 2010-02-22T18:30:29.569639,

and:

```
<?print today().isoformat()?>
```

might output 2010-02-22.

mimeformat()

`mimeformat` returns the date/datetime object in MIME format (assuming the object is in UTC), i.e.:

```
<?print utcnow().mimeformat()?>
```

might output `Mon, 22 Feb 2010 17:38:40 GMT`,

and:

```
<?print today().mimeformat()?>
```

might output `Mon, 22 Feb 2010`.

day(), month(), year(), hour(), minute(), second(), microsecond() and weekday()

Those methods return a specific attribute of a date or datetime object. For example the following reproduces the `mimeformat` output from above (except for the linefeeds of course):

```
<?code weekdays = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']?>
<?code months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
↪ 'Nov', 'Dec']?>
<?code t = @(2010-02-22T17:38:40.123456)?>
<?print weekdays[t.weekday()]?>,
<?print format(t.day(), '02')?>
<?print months[t.month()-1]?>
<?print format(t.year(), '04')?>
<?print format(t.hour(), '02')?>:
<?print format(t.minute(), '02')?>:
<?print format(t.second(), '02')?>.
<?print format(t.microsecond(), '06')?> GMT
```

date()

For date objects `date()` returns the object unmodified, for datetime objects a date object containing the date portion of the object is returned, so:

```
<?print @(2000-02-29T12:34:56.987654).date()?>
```

prints `2000-02-29`.

timestamp()

The method `timestamp()` of date and datetime objects returns the number of seconds (with microseconds precision) between this date and 1970-01-01T00:00:00.

calendar(firstweekday=0, mindaysinfirstweek=4)

`d.calendar()` returns a list containing:

1. the calendar year `d` belongs to;
2. the calendar week number of `d`;
3. the weekday of `d`

(A day might belong to a different calender year, if it is in week 1 but before January 1, or if belongs to week 1 of the following year).

`firstweekday` defines what a week is (i.e. which weekday is considered the start of the week, `0` is Monday and `6` is Sunday). `mindaysinfirstweek` defines how many days must be in a week to be considered the first week in the year.

For example for the ISO week number (according to https://en.wikipedia.org/wiki/ISO_week_date) the week starts on Monday (i.e. `firstweekday == 0`) and a week is considered the first week if it's the first week that contains a Thursday (which means that this week contains at least four days in January, so `mindaysinfirstweek == 4`).

For the US `firstweekday == 6` and `mindaysinfirstweek == 1`, i.e. the week starts on Sunday and January the first is always in week 1.

There's also the convention that the week 1 is the first complete week in January. For this `mindaysinfirstweek == 7`.

week(firstweekday=0, mindaysinfirstweek=4)

`week` returns the calendar week number of the date for which it is called. For more information see the method `calendar`.

yearday()

`yearday` returns the number of days since the beginning of the year, so:

```
<?print @(2010-01-01).yearday()?>
```

prints `1` and:

```
<?print @(2010-12-31).yearday()?>
```

prints `365`.

color

Color objects support the following methods:

r()
Return the red component of the color (as an 8-bit integer).

g()
Return the green component of the color (as an 8-bit integer).

b()
Return the blue component of the color (as an 8-bit integer).

a()
Return the alpha (opacity) component of the color (as an 8-bit integer).

hsv()
Return the color as an HSV tuple ("hue", "saturation", "value"). All three values are between 0.1 and 1.0.

hsva()
Similar to `hsv()`, but returns the alpha (opacity) as the fourth tuple item.

hls()
Return the color as an HLS tuple ("hue", "saturation", "lightness"). All three values are between 0.1 and 1.0.

hlsa()
Similar to `hls()`, but returns the alpha (opacity) as the fourth tuple item.

For more info about the HSV and HSV color models see Wikipedia.

hue()
Return the hue value of the HLS color tuple.

sat()
Return the saturation value of the HLS color tuple.

light()
Return the lightness value of the HLS color tuple.

lum()
Return the luminance value of the color tuple, which is similar to lightness but is the following **weighted** sum of the components:

```
(0.2126 * r() + 0.7152 * g() + 0.0722 * b())/255
```

withhue(hue)
Return a new color with the HLS hue replaced by `hue`.

withlight(light)
Return a new color with the HLS lightness replaced by `light`.

withsat(sat)
Return a new color with the HLS saturation replaced by `sat`.

witha(a)
Return a new color with the alpha (opacity) component replaced by `a`.

withlum(lum)
Return a new color with the luminance replaced by `lum`.

abslight(f)
Return a new color with `f` added to the HLS lightness of the original color.

rellight(f)
Return a new color where the HLS lightness of the original color has been modified.

If `f` is positive the lightness will be increased, with `f==1` giving a lightness of 1. If `f` is negative, the lightness will be decreased with `f==-1` giving a lightness of 0. `f==0` will leave the lightness unchanged. All other values return a linear interpolation.

abslum(f)
Return a new color with `f` added to the luminance of the original color. I.e. for a color `c` the following should always print `True`:

```
<?print c.abslum(f).lum() == c.lum() + f?>
```

(except for rounding errors and when the modified luminance would be smaller than 0 or larger than 1).

rellum(f)
Return a new color with `f` used to modify the luminance of the original color.

If `f` is positive the luminance will be increased, with `f==1` giving a luminance of 1. If `f` is negative, the luminance will be decreased with `f==-1` giving a luminance of 0. `f==0` will leave the luminance unchanged. All other values return a linear interpolation.

combine(r=None, g=None, b=None, a=None)

Return a new color with some of its components replaced by the arguments. If a component is not passed (or the value `None` is given) the component will be unchanged in the resulting color.

invert(f=1.0)

Return an inverted version of the original color, i.e. for each color `c` the following prints `True` three times:

```
<?print c.invert().r() == 255 - c.r()?>
<?print c.invert().g() == 255 - c.g()?>
<?print c.invert().b() == 255 - c.b()?>
```

`f` specifies the amount of inversion, with 1 returning a complete inversion, and 0 returning the original color. Values between 0 and 1 return an interpolation of both extreme values. (So 0.5 always returns medium grey).

6.2.9 Templates as functions

UL4 templates can be called as functions too. Calling a template as a function will ignore any output from the template. The return value will be the value of the first `<?return?>` tag encountered:

```
from ll import ul4c

code = """
   <?for item in data?>
      <?if "i" in item?>
         <?return item?>
      <?end if?>
   <?end for?>
"""

function = ul4c.Template(code)

output = function(data=["Python", "Java", "Javascript", "PHP"]))
```

With this, `output` will be the string `"Javascript"`.

When no `<?return?>` tag is encountered, `None` will be returned.

When a `<?return?>` tag is encountered when the template is used as a template, output will simply stop and the return value will be ignored.

6.2.10 Global variables

UL4 templates support global variables. To be able to pass parameters and global variables to an UL4 template a second set of methods is available, so that a list of positional arguments, a dictionary with keyword arguments and a dictionary with global variables can be passed.

These methods are *render_with_globals()*, *renders_with_globals()* and *call_with_globals()*.

An example using *renders_with_globals()* looks like this:

```
from ll import ul4c

t1 = ul4c.Template("<?print x?>")
t2 = ul4c.Template("<?render t1()?>")

output = t2.renders_with_globals((), {"t1": t1}, {"x": 42})
```

With this `output` will be the string `"42"`.

And an example using *call_with_globals()* looks like this:

```
from ll import ul4c

t1 = ul4c.Template("<?return x?>")
t2 = ul4c.Template("<?return t1()?>")

result = t2.call_with_globals((), {"t1": t1}, {"x": 42})
```

With this `result` will be `42`.

6.2.11 Interfacing with Python

Exposing attributes

It is possible to expose attributes of a Python object to UL4 templates. This is done by setting the class attribute `ul4_attrs`:

```
from ll import ul4c

class Person:
   ul4_attrs = {"firstname", "lastname"}

   def __init__(self, firstname, lastname, age):
      self.firstname = firstname
      self.lastname = lastname
      self.age = age

p = Person("John", "Doe", 42)

template = ul4c.Template("<?print p.lastname?>, <?print p.firstname?>")
print(template.renders(p=p))
```

This will output `Doe, John`.

Attributes not in `ul4_attrs` will not be visible:

```
template = ul4c.Template("<?print p.age?>")
print(template.renders(p=p))
```

This will output nothing, as the `age` attribute is not visible and thus `p.age` returns an `undefined` object.

Exposing methods

It is also possible to expose methods of a Python object to UL4 templates. This is done by including the method name in the `ul4_attrs` class attribute:

```
from ll import ul4c

class Person:
   ul4_attrs = {"fullname"}

   def __init__(self, firstname, lastname):
      self.firstname = firstname
      self.lastname = lastname

   def fullname(self):
      return self.firstname + " " + self.lastname
```

(continues on next page)

(continued from previous page)

```
p = Person("John", "Doe")

template = ul4c.Template("<?print p.fullname()?>")
print(template.renders(p=p))
```

This will output `John Doe`.

Furthermore it's possible to specify that the method needs access to the rendering context (which stores the local variables and the UL4 call stack):

```
class Person:
   ul4_attrs = {"fullname", "varcount"}

   @ul4c.withcontext
   def varcount(self, context):
      return len(context.vars)
```

Custom attributes

To customize getting and setting object attributes from UL4 templates the methods `ul4_getattr()` and `ul4_setattr()` can be implemented:

```
from ll import ul4c

class Person:
   ul4_attrs = {"firstname", "lastname"}

   def __init__(self, firstname, lastname, age):
      self.firstname = firstname
      self.lastname = lastname
      self.age = age

   def ul4_getattr(self, name):
      return getattr(self, name).upper()

p = Person("John", "Doe", 42)

template = ul4c.Template("<?print p.lastname?>, <?print p.firstname?>")
print(template.renders(p=p))
```

This will output `DOE, JOHN`.

If the object has an attribute `ul4_attrs` `ul4_getattr()` will only be called for the attributes in `ul4_attrs`, otherwise `ul4_getattr()` will be called for all attributes (and should raise an `AttributeError` for nonexistent attributes)

Attributes can be made writable by implementing the method `ul4_setattr()`:

```
from ll import ul4c

class Person:
   ul4_attrs = {"firstname", "lastname"}

   def __init__(self, firstname, lastname, age):
      self.firstname = firstname
      self.lastname = lastname
```

(continues on next page)

(continued from previous page)

```
        self.age = age

    def ul4_setattr(self, name, value):
        return setattr(self, name, value.upper())

p = Person("John", "Doe", 42)

template = ul4c.Template("<?code p.lastname = 'Doe'?><?print p.lastname?>, <?print p.
↪firstname?>")
print(template.renders(p=p))
```

This will output `DOE, John`.

If the object has an attribute `ul4_attrs` `ul4_setattr()` will only be called for the attributes in `ul4_attrs`, otherwise `ul4_setattr()` will be called for all attributes (and should raise an `AttributeError` for nonexistent or readonly attributes)

Without a `ul4_setattr()` method, attributes will never be made writable.

6.2.12 Exceptions

Exception objects can not be created directly by UL4 templates, but UL4 templates can work with exceptions and access their attributes. The function `isexception` returns `True` if the argument is an exception object and exception objects have an attribute `context` that exposed the `__cause__` or `__context__` attribute of the Python exception object.

Exceptions that happen in UL4 templates use exception chaining to add information about the location of the error while the exception bubbles up the Python call stack. So the exception will be e.g. a `TypeError` object and its `__cause__` attribute (which is accessible as the UL4 attribute `context`) specifies the immediate location inside the UL4 source code where the exception happened (and its `__cause__` is the location that called that one etc.). So if we have the following UL4 template:

```
<?def x(i)?>
   Print: <?print 1/i?>
   Render: <?render x(i-1)?>
<?end def?>
Initial render: <?render x(3)?>
```

Calling the template will result in a `ZeroDivisionError` exception. We can format a nice UL4 stacktrace (in HTML) for this exception with the following UL4 code:

```
<?def frame(exc)?>
   <?if exc.context?>
      <?render frame(exc.context)?>
   <?end if?>
   <?if exc.location?>
      <li>
         <p>
            <b><?printx type(exc)?></b>
            in template <b><?printx exc.location.template.name?></b>
            : line <?printx exc.location.line?>
            ; col <?printx exc.location.col?>
         </p>
         <p>
            <?print exc.location.sourceprefix?>
            <b><?print exc.location.source?></b>
            <?print exc.location.sourcesuffix?>
```

(continues on next page)

(continued from previous page)

```
        </p>
      </li>
   <?else?>
      <li>
         <p>
            <b><?printx type(exc).__name__?></b><?if str(exc)?>: <?print str(exc)?>
         </p>
      </li>
   <?end def?>
<?end def?>

<ul>
   <?render frame(exc)?>
</ul>
```

6.2.13 Whitespace handling

Normally the literal text between template tags will be output as it is. This behaviour can be changed by passing a different value to the `whitespace` parameter in the constructor. The possible values are:

"keep"
The default behaviour: literal text will be output as it is.

"strip"
Linefeeds and the following indentation in literal text will be ignored:

```
>>> from ll import ul4c
>>> t = ul4c.Template("""
...   <?for i in range(10)?>
...      <?print i?>
...      ;
...   <?end for?>
... """, whitespace="strip")
>>> t.renders()
'0;1;2;3;4;5;6;7;8;9;'
```

However trailing whitespace at the end of the line will still be honored.

"smart"
If a line contains only indentation and one tag that isn't a `print`, `printx` or `render` tag, the indentation and the linefeed after the tag will be stripped from the text.

Furthermore the additional indentation that might be introduced by a `for`, `if`, `elif`, `else` or `def` block will be ignored. So for example the output of:

```
<?code langs = ["Python", "Java", "Javascript"]?>
<?if langs?>
   <?for lang in langs?>
      <?print lang?>
   <?end for?>
<?end if?>
```

will simply be

```
Python
Java
Javascript
```

without any additional empty lines or indentation.

Rendering a template `B` inside an template `A` will reindent the output of `B` to the indentation level of the `<?render?>` tag in the template `A`.

It is also possible to specify the whitespace behaviour in the template itself with the `<?whitespace?>` tag, so:

```
<?whitespace smart?>
```

anywhere in the template source will switch on smart whitespace handling.

A `<?whitespace?>` tag overwrites the `whitespace` parameter specified in the constructor.

6.2.14 Module documentation

`ll.ul4c` provides templating for XML/HTML as well as any other text-based format. A template defines placeholders for data output and basic logic (like loops and conditional blocks), that define how the final rendered output will look.

`ll.ul4c` compiles a template to an internal format, which makes it possible to implement template renderers in multiple programming languages.

`ll.ul4c.`**`withcontext`**`(`*`f`*`)`

Normally when a function is exposed to UL4 this function will be called directly.

However when the function needs access to the rendering context (i.e. the local variables or information about the call stack), the function must have an additional first parameter, and UL4 must be told that this parmeter is required.

This can be done with this decorator.

`exception` `ll.ul4c.`**`LocationError`**

Bases: `Exception`

Exception class that provides a location inside an UL4 template.

If an exception happens inside an UL4 template, this exception will propagate outwards and will be decorated with *`LocationError`* instances which will be chained via the `__cause__` attribute. Only the original exception will be reraised again and again, so these *`LocationError`* will never have a traceback attached to them.

The first `__cause__` attribute marks the location in the UL4 source where the exception happened and the last `__cause__` attribute at the end of the exception chain marks the outermost call.

`exception` `ll.ul4c.`**`BlockError`**

Bases: `Exception`

Exception that is raised by the compiler when an illegal block structure is detected (e.g. an `<?end if?>` without a previous `<?if?>`).

`class` `ll.ul4c.`**`Context`**

Bases: `object`

A *`Context`* object stores the context of a call to a template. This consists of local, global and builtin variables and the indent stack.

`class` `ll.ul4c.`**`AST`**

Bases: `object`

Base class for all UL4 syntax tree nodes.

eval(*context*)

This evaluates the node.

For most nodes this is a normal function that returns the result of evaluating the node. (For these nodes the class attribute `output` is `False`.). For nodes that produce output (like literal text, *PrintAST*, *PrintXAST* or *RenderAST* etc.) it is a generator which yields the text output of the node. For blocks (which might contain nodes which produce output) this is also a generator. (For these nodes the class attribute `output` is `True`.)

class ll.ul4c.**TextAST**

Bases: *AST*

AST node for literal text (i.e. the stuff between tags).

Attributes are:

text

[str]
The text

class ll.ul4c.**IndentAST**

Bases: *TextAST*

AST node for literal text that is an indentation at the start of the line.

Attributes are:

text

[str]
The indentation text (i.e. a string that consists solely of whitespace).

class ll.ul4c.**LineEndAST**

Bases: *TextAST*

AST node for literal text that is the end of a line.

Attributes are:

text

[str]
The text of the linefeed (i.e. `"\n"` or `"\r\n"`).

class ll.ul4c.**Tag**

Bases: *AST*

A *Tag* object is the location of a template tag in a template.

class ll.ul4c.**CodeAST**

Bases: *AST*

The base class of all AST nodes that are not literal text.

These nodes appear inside a *Tag*.

class ll.ul4c.**ConstAST**

Bases: *CodeAST*

AST node for a constant value.

Attributes are:

value

The constant to be loaded.

class ll.ul4c.**SeqItemAST**

Bases: *CodeAST*

AST node for an item in a list/set "literal" (e.g. `{x, y}` or `[x, y]`)

Attributes are:

value

[*AST*]
The list/set item (`x` and `y` in the above examples).

class ll.ul4c.**UnpackSeqItemAST**

Bases: *CodeAST*

AST node for an * unpacking expression in a list/set "literal" (e.g. the `y` in `{x, *y}` or `[x, *y]`)

Attributes are:

value

[*AST*]
The item to be unpacked into list/set items (`y` in the above examples).

class ll.ul4c.**DictItemAST**

Bases: *CodeAST*

AST node for a dictionary entry in a dict expression (*DictAST*).

Attributes are:

key

[*AST*]
The key of the entry.

value

[*AST*]
The value of the entry.

class ll.ul4c.**UnpackDictItemAST**

Bases: *CodeAST*

AST node for ** unpacking expressions in dict "literal" (e.g. the `**u` in `{k: v, **u}`).

Attributes are:

item

[*AST*]
The argument that must evaluate to a dictionary or an iterable of (key, value) pairs.

class ll.ul4c.**PositionalArgumentAST**

Bases: *CodeAST*

AST node for a positional argument. (e.g. the `x` in `f(x)`).

Attributes are:

value

[*AST*]
The value of the argument (`x` in the above example).

class ll.ul4c.**KeywordArgumentAST**

Bases: *CodeAST*

AST node for a keyword argument in a *CallAST* (e.g. the `x=y` in the function call `f(x=y)`).

Attributes are:

name

[`str`]
The keyword argument name (`"x"` in the above example).

value
[*AST*]
The keyword argument value (y in the above example).

class ll.ul4c.**UnpackListArgumentAST**

Bases: *CodeAST*

AST node for an * unpacking expressions in a *CallAST* (e.g. the *x in f(*x)).

Attributes are:

item
[*AST*]
The argument that must evaluate an iterable.

class ll.ul4c.**UnpackDictArgumentAST**

Bases: *CodeAST*

AST node for an ** unpacking expressions in a *CallAST* (e.g. the **x in f(**x)).

Attributes are:

item
[*AST*]
The argument that must evaluate to a dictionary or an iterable of (key, value) pairs.

class ll.ul4c.**ListAST**

Bases: *CodeAST*

AST node for creating a list object (e.g. [x, y, *z]).

Attributes are:

items
[list]
The items that will be put into the newly created list as a list of *SeqItemAST* (x and y in the above example) and *UnpackSeqItemAST* objects (z in the above example).

class ll.ul4c.**ListComprehensionAST**

Bases: *CodeAST*

AST node for a list comprehension (e.g. [v for (a, b) in w if c].

Attributes are:

item
[*AST*]
The expression for the item in the newly created list (v in the above example).

varname
[nested tuple of *VarAST* objects]
The loop variable (or variables) (a and b in the above example).

container
[*AST*]
The container or iterable object over which to loop (w in the above example).

condition
[*AST* or None]
The condition (as an *AST* object if there is one, or None if there is not) (c in the above example).

class ll.ul4c.**SetAST**

Bases: *CodeAST*

AST node for creating a set object (e.g. {x, y, *z}.

Attributes are:

items
[list]
The items that will be put into the newly created set as a list of *SeqItemAST* (x and y in the above example) and *UnpackSeqItemAST* objects (z in the above example).

class ll.ul4c.**SetComprehensionAST**

Bases: *CodeAST*

AST node for a set comprehension (e.g. `{v for (a, b) in w if c}`.

Attributes are:

item
[*AST*]
The expression for the item in the newly created set (v in the above example).

varname
[nested tuple of *VarAST* objects]
The loop variable (or variables) (a and b in the above example).

container
[*AST*]
The container or iterable object over which to loop (w in the above example).

condition
[*AST* or None]
The condition (as an *AST* object if there is one, or None if there is not) (c in the above example).

class ll.ul4c.**DictAST**

Bases: *CodeAST*

AST node for creating a dict object (e.g. *{k: v, **u}*.

Attributes are:

items
[list]
The items that will be put into the newly created dictionary as a list of *DictItemAST* (for k and v in the above example) and *UnpackDictItemAST* objects (for u in the above example).

class ll.ul4c.**DictComprehensionAST**

Bases: *CodeAST*

AST node for a dictionary comprehension (e.g. `{k: v for (a, b) in w if c}`.

Attributes are:

key
[*AST*]
The expression for the keys in the newly created dictionary (k in the above example).

value
[*AST*]
The expression for the values in the newly created dictionary (v in the above example).

varname
[nested tuple of *VarAST* objects]
The loop variable (or variables) (a and b in the above example).

container
[*AST*]
The container or iterable object over which to loop (w in the above example).

condition
[*AST* or None]
The condition (as an *AST* object if there is one, or None if there is not) (c in the above example).

class ll.ul4c.**GeneratorExpressionAST**

Bases: *CodeAST*

AST node for a generator expression (e.g. `(x for (a, b) in w if c)`).

Attributes are:

item
[*AST*]
An expression for the item that looping over the generator expression will produce (`x` in the above example).

varname
[nested `tuple` of *VarAST* objects]
The loop variable (or variables) (`a` and `b` in the above example).

container
[*AST*]
The container or iterable object over which to loop (`w` in the above example).

condition
[*AST* or `None`]
The condition (as an *AST* object if there is one, or `None` if there is not) (`c` in the above example).

class ll.ul4c.**VarAST**

Bases: *CodeAST*

AST node for getting a variable.

Attributes are:

name
[`str`]
The name of the variable.

class ll.ul4c.**BlockAST**

Bases: *CodeAST*

Base class for all AST nodes that are blocks.

A block contains a sequence of AST nodes that are executed sequencially. A block may execute its content zero (e.g. an `<?if?>` block) or more times (e.g. a `<?for?>` block).

Attributes are:

content
[`list` of *AST* objects]
The content of the block.

class ll.ul4c.**ConditionalBlocksAST**

Bases: *BlockAST*

AST node for a conditional `<?if?>`/`<?elif?>`/`<?else?>` block.

Attributes are:

content
[`list`]
The content of the *ConditionalBlocksAST* block is one *IfBlockAST* followed by zero or more *ElIfBlockAST*s followed by zero or one *ElseBlockAST*.

class ll.ul4c.**IfBlockAST**

Bases: *BlockAST*

AST node for an `<?if?>` block in an `<?if?>`/`<?elif?>`/`<?else?>` block.

Attributes are:

condition
[*AST*]
The condition in the <?if?> block.

content
[list of *AST* objects]
The content of the <?if?> block.

class ll.ul4c.**ElIfBlockAST**

Bases: *BlockAST*

AST node for an <?elif?> block.

Attributes are:

condition
[*AST*]
The condition in the <?elif?> block.

content
[list of *AST* objects]
The content of the <?elif?> block.

class ll.ul4c.**ElseBlockAST**

Bases: *BlockAST*

AST node for an <?else?> block.

Attributes are:

content
[list of *AST* objects]
The content of the <?else?> block.

class ll.ul4c.**ForBlockAST**

Bases: *BlockAST*

AST node for a <?for?> loop.

For example

```
<?for (a, b) in w?>
   body
<?end for?>
```

Attributes are:

varname
[nested tuple of *VarAST* objects]
The loop variable (or variables) (a and b in the above example).

container
[*AST*]
The container or iterable object over which to loop (w in the above example).

content
[list of *AST* objects]
The loop body (body in the above example).

class ll.ul4c.**WhileBlockAST**

Bases: *BlockAST*

AST node for a <?while?> loop.

For example

```
<?while c?>
   body
<?end for?>
```

Attributes are:

condition
[*AST*]
The condition which must be true to continue executing the loops booy (`c` in the above example).

content
[`list` of *AST* objects]
The loop body (`body` in the above example).

class `ll.ul4c.`**BreakAST**

Bases: *CodeAST*

AST node for a `<?break?>` tag inside a `<?for?>` loop.

class `ll.ul4c.`**ContinueAST**

Bases: *CodeAST*

AST node for a `<?continue?>` tag inside a `<?for?>` block.

class `ll.ul4c.`**AttrAST**

Bases: *CodeAST*

AST node for an expression that gets or sets an attribute of an object. (e.g. `x.y`).

Attributes are:

obj
[*AST*]
The object from which to get the attribute (`x` in the above example);

attrname
[`str`]
The name of the attribute (`"y"` in the above example).

class `ll.ul4c.`**SliceAST**

Bases: *CodeAST*

AST node for creating a slice object (used in `obj[index1:index2]`).

Attributes are:

index1
[*AST* or `None`]
The start index (`index1` in the above example).

index2
[*AST* or `None`]
The stop index (`index2` in the above example).

`index1` and `index2` may also be `None` (for missing slice indices, which default to 0 for the start index and the length of the sequence for the stop index).

class `ll.ul4c.`**UnaryAST**

Bases: *CodeAST*

Base class for all AST nodes implementing unary expressions (i.e. operators with one operand).

Atttributes are:

obj
[*AST*]
The operand of the unary operator.

class `ll.ul4c.`**NotAST**

Bases: *UnaryAST*

AST node for a unary "not" expression (e.g. `not x`).

Attributes are:

obj

[*AST*]
The operand (`x` in the above example).

class `ll.ul4c.`**NegAST**

Bases: *UnaryAST*

AST node for a unary negation expression (e.g. `-x`).

Attributes are:

obj

[*AST*]
The operand (`x` in the above example).

class `ll.ul4c.`**BitNotAST**

Bases: *UnaryAST*

AST node for a bitwise unary "not" expression that returns its operand with its bits inverted (e.g. `~x`).

Attributes are:

obj

[*AST*]
The operand (`x` in the above example).

class `ll.ul4c.`**PrintAST**

Bases: *UnaryAST*

AST node for a `<?print?>` tag (e.g. `<?print x?>`).

Attributes are:

obj

[*AST*]
The object to be printed (`x` in the above example).

class `ll.ul4c.`**PrintXAST**

Bases: *UnaryAST*

AST node for a `<?printx?>` tag (e.g. `<?printx x?>`).

Attributes are:

obj

[*AST*]
The object to be printed (`x` in the above example).

class `ll.ul4c.`**ReturnAST**

Bases: *UnaryAST*

AST node for a `<?return?>` tag (e.g. `<?return x?>`).

Attributes are:

obj

[*AST*]
The operand (`x` in the above example).

class `ll.ul4c.`**BinaryAST**

Bases: *CodeAST*

Base class for all UL4 AST nodes implementing binary expressions (i.e. operators with two operands).

obj1

[*AST*]
The first operand.

obj2

[*AST*]
The second operand.

class `ll.ul4c.`**ItemAST**

Bases: *BinaryAST*

AST node for subscripting expression (e.g. `x[y]`).

Attributes are:

obj1

[*AST*]
The container object, which must be a list, string or dict (`x` in the above example).

obj2

[*AST*]
The index/key object (`y` in the above example).

class `ll.ul4c.`**IsAST**

Bases: *BinaryAST*

AST node for a binary `is` comparison expression (e.g. `x is y`).

Attributes are:

obj1

[*AST*]
The left operand (`x` in the above example).

obj2

[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**IsNotAST**

Bases: *BinaryAST*

AST node for a binary `is not` comparison expression (e.g. `x is not y`).

Attributes are:

obj1

[*AST*]
The left operand (`x` in the above example).

obj2

[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**EQAST**

Bases: *BinaryAST*

AST node for the binary equality comparison (e.g. `x == y`.

Attributes are:

obj1

[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**NEAST**

Bases: *BinaryAST*

AST node for a binary inequality comparison (e.g. `x != y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**LTAST**

Bases: *BinaryAST*

AST node for the binary "less than" comparison (e.g. `x < y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**LEAST**

Bases: *BinaryAST*

AST node for the binary "less than or equal" comparison (e.g. `x <= y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**GTAST**

Bases: *BinaryAST*

AST node for the binary "greater than" comparison (e.g. `x > y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**GEAST**

Bases: *BinaryAST*

AST node for the binary "greater than or equal" comparison (e.g. `x >= y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**ContainsAST**

Bases: *BinaryAST*

AST node for the binary containment testing operator (e.g. `x in y`).

Attributes are:

obj1
[*AST*]
The item/key object (`x` in the above example).

obj2
[*AST*]
The container object (`y` in the above example).

class `ll.ul4c.`**NotContainsAST**

Bases: *BinaryAST*

AST node for an inverted containment testing expression (e.g. `x not in y`).

Attributes are:

obj1
[*AST*]
The item/key object (`x` in the above example).

obj2
[*AST*]
The container object (`y` in the above example).

class `ll.ul4c.`**AddAST**

Bases: *BinaryAST*

AST node for a binary addition expression (e.g. `x + y`).

Attributes are:

obj1
[*AST*]
The left summand (`x` in the above example).

obj2
[*AST*]
The right summand (`y` in the above example).

class `ll.ul4c.`**SubAST**

Bases: *BinaryAST*

AST node for the binary subtraction expression (e.g. `x - y`).

class `ll.ul4c.`**MulAST**

Bases: *BinaryAST*

AST node for the binary multiplication expression (e.g. `x * y`).

Attributes are:

obj1
[*AST*]
The left factor (`x` in the above example).

obj2
[*AST*]
The right factor (`y` in the above example).

class `ll.ul4c.`**FloorDivAST**

Bases: *BinaryAST*

AST node for a binary truncating division expression (e.g. `x // y`).

Attributes are:

obj1
[*AST*]
The dividend (`x` in the above example).

obj2
[*AST*]
The divisor (`y` in the above example).

class `ll.ul4c.`**TrueDivAST**

Bases: *BinaryAST*

AST node for a binary true division expression (e.g. `x / y`).

Attributes are:

obj1
[*AST*]
The dividend (`x` in the above example).

obj2
[*AST*]
The divisor (`y` in the above example).

class `ll.ul4c.`**ModAST**

Bases: *BinaryAST*

AST node for a binary modulo expression (e.g. `x % y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**ShiftLeftAST**

Bases: *BinaryAST*

AST node for a bitwise left shift expression (e.g. `x << y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class ll.ul4c.**ShiftRightAST**

Bases: *BinaryAST*

AST node for a bitwise right shift expression (e.g. `x >> y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class ll.ul4c.**BitAndAST**

Bases: *BinaryAST*

AST node for a binary bitwise "and" expression (e.g `x & y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class ll.ul4c.**BitXOrAST**

Bases: *BinaryAST*

AST node for the binary bitwise "exclusive or" expression (e.g. `x ^ y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class ll.ul4c.**BitOrAST**

Bases: *BinaryAST*

AST node for a binary bitwise "or" expression (e.g. `x | y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class ll.ul4c.**AndAST**

Bases: *BinaryAST*

AST node for the binary "and" expression (i.e. `x and y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class `ll.ul4c.`**OrAST**

Bases: *BinaryAST*

AST node for a binary "or" expression (e.g. `x or y`).

Attributes are:

obj1
[*AST*]
The item/key object (`x` in the above example).

obj2
[*AST*]
The container object (`y` in the above example).

class `ll.ul4c.`**IfAST**

Bases: *CodeAST*

AST node for the ternary inline `if/else` operator (e.g. `x if y else z`).

Attributes are:

objif
[*AST*]
The value of the `if/else` expression when the condition is true (`x` in the above example).

objcond
[*AST*]
The condition (`y` in the above example).

objelse
[*AST*]
The value of the `if/else` expression when the condition is false (`z` in the above example).

class `ll.ul4c.`**ChangeVarAST**

Bases: *CodeAST*

Base class for all AST nodes that are assignment operators, i.e. that set or modify a variable/attribute or item.

Attributes are:

lvalue
[*AST*]
The left hand side, i.e. the value that will be modified.

value
[*AST*]
The right hand side, the value that will be assigned or be used to modify the intial value.

class `ll.ul4c.`**SetVarAST**

Bases: *ChangeVarAST*

AST node for setting a variable, attribute or item to a value (e.g. `x = y`).

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class `ll.ul4c.`**AddVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that adds a value to a variable (e.g. `x += y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class `ll.ul4c.`**SubVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that subtracts a value from a variable/attribute/item. (e.g. `x -= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class `ll.ul4c.`**MulVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that assigns the result of a multiplication to its left operand. (e.g. `x *= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class `ll.ul4c.`**FloorDivVarAST**

Bases: *ChangeVarAST*

AST node for augmented assignment expression that divides a variable by a value, truncating to an integer value (e.g. `x //= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class ll.ul4c.**TrueDivVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that assigns the result of a truncation division to its left operand. (e.g. `x //= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class ll.ul4c.**ModVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that assigns the result of a modulo expression to its left operand. (e.g. `x %= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class ll.ul4c.**ShiftLeftVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that assigns the result of a "shift left" expression to its left operand. (e.g. `x <<= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class ll.ul4c.**ShiftRightVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that assigns the result of a "shift right" expression to its left operand. (e.g. `x >>= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class ll.ul4c.**BitAndVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that assigns the result of a binary bitwise "and" expression to its left operand. (e.g. `x &= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class ll.ul4c.**BitXOrVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that assigns the result of a binary bitwise "exclusive or" expression to its left operand. (e.g. `x ^= y`).

Attributes are:

obj1
[*AST*]
The left operand (`x` in the above example).

obj2
[*AST*]
The right operand (`y` in the above example).

class ll.ul4c.**BitOrVarAST**

Bases: *ChangeVarAST*

AST node for an augmented assignment expression that assigns the result of a binary bitwise "or" expression to its left operand. (e.g. `x |= y`).

Attributes are:

lvalue
[*AST*]
The left hand side (`x` in the above example).

value
[*AST*]
The right hand side (`y` in the above example).

class ll.ul4c.**CallAST**

Bases: *CodeAST*

AST node for calling an object (e.g. `f(x, y)`).

Attributes are:

obj
[*AST*]
The object to be called (`f` in the above example) (or rendered/printed in the subclass *RenderAST* and its subclasses);

args
[`list`]
The arguments to the call as a `list` of *PositionalArgumentAST*, *KeywordArgumentAST*, *UnpackListArgumentAST* or *UnpackDictArgumentAST* objects (`x` and `y` in the above example).

class ll.ul4c.**RenderAST**

Bases: *CallAST*

AST node for rendering a template (e.g. `<?render t(x)?>`.

For a list of attribute see *CallAST*.

class ll.ul4c.**RenderXAST**

Bases: *RenderAST*

AST node for rendering a template and XML-escaping the output (e.g. `<?renderx t(x)?>`.

For a list of attribute see *CallAST*.

class ll.ul4c.**RenderOrPrintAST**

Bases: *RenderAST*

AST node for rendering a template or printing an object.

```
<?render_or_print t(x)?>
```

is equivalent to

```
<?if istemplate(t)?>
   <?render t(x)?>
<?else?>
   <?print t?>
<?end if?>
```

except that even if `t` is not renderable, all argument in the call will be evaluated.

For a list of attribute see *CallAST*.

class ll.ul4c.**RenderOrPrintXAST**

Bases: *RenderAST*

AST node for rendering a template or printing an object (e.g. `<?render_or_printx t(x)?>`.

For a list of attribute see *CallAST*.

class ll.ul4c.**RenderXOrPrintAST**

Bases: *RenderAST*

AST node for rendering a template or printing an object (e.g. `<?renderx_or_print t(x)?>`.

For a list of attribute see *CallAST*.

class ll.ul4c.**RenderXOrPrintXAST**

Bases: *RenderAST*

AST node for rendering a template or printing an object (e.g. `<?renderx_or_printx t(x)?>`.

For a list of attribute see *CallAST*.

class ll.ul4c.**RenderBlockAST**

Bases: *RenderAST*

AST node for rendering a template via a `<?renderblock?>` block and passing the content of the block as one additional keyword argument named `content`.

For example

```
<?renderblock t(a, b)?>
   content
<?end renderblock?>
```

Attributes are:

obj
[*AST*]
The object to be rendered (`t` in the above example);

args
[`list`]
The arguments to the call as a `list` of *PositionalArgumentAST*, *KeywordArgumentAST*, *UnpackListArgumentAST* or *UnpackDictArgumentAST* objects (`a` and `b` in the above example).

content
[`list` of *AST* objects]
The content of the `<?renderblock?>` tag (`content` in the above example) that will be passed as a signatureless template as the keyword argument `content` to the object.

class `ll.ul4c.`**RenderBlocksAST**

Bases: *RenderAST*

AST node for rendering a template and passing additional arguments via nested variable definitions, e.g.:

```
<?renderblocks t(a, b)?>
   <?code x = 42?>
   <?def n?>
      ...
   <?end def?>
<?end renderblocks?>
```

Attributes are:

obj
[*AST*]
The object to be rendered (`t` in the above example);

args
[`list`]
The arguments to the call as a `list` of *PositionalArgumentAST*, *KeywordArgumentAST*, *UnpackListArgumentAST* or *UnpackDictArgumentAST* objects (`a` and `b` in the above example).

content
[`list` of *AST* objects]
The content of the `<?renderblocks?>` tag. These must be *AST* nodes that define variables (e.g. *SetVarAST* (the `<?code x = 42?>` in the above example), or *Template* (the `<?def n?>...<?end def?>` in the above example)).

class `ll.ul4c.`**Template**

Bases: *BlockAST*

A UL4 template.

A template object is normally created by passing the template source to the constructor. It can also be loaded from the compiled format via the class methods *load()* (from a stream) or *loads()* (from a string).

The compiled format can be generated with the methods *dump()* (which dumps the format to a stream) or *dumps()* (which returns a string with the compiled format).

Rendering the template can be done with the methods *render()* (which is a generator) or *renders()* (which returns a string).

A *Template* can also be called as a function (returning the result of the first `<?return?>` tag encountered). In this case all output of the template will be ignored.

For rendering and calling a template with global variables the following methods are available: *render_with_globals()*, *renders_with_globals()* and *call_with_globals()*.

A *Template* object is itself an AST node. Evaluating it will store a *TemplateClosure* object for this template under the template's name in the local variables.

__init__(*source=None*, *name=None*, *, *namespace=None*, *whitespace='keep'*, *signature=None*)

Create a `Template` object.

If `source` is `None`, the `Template` remains uninitialized, otherwise `source` will be compiled.

`name` is the name of the template. It will be used in exception messages and should be a valid Python identifier.

`namespace` is the namespace name. It defaults to None.

`whitespace` specifies how whitespace is handled in the literal text in templates (i.e. the text between the tags):

"keep"
Whitespace is kept as it is.

"strip"
Strip linefeeds and the following indentation from the text. However trailing whitespace at the end of the line will still be honored.

"smart"
If a line contains only indentation and one tag that isn't a `print` or `printx` tag, the indentation and the linefeed after the tag will be stripped from the text. Furthermore the additional indentation that might be introduced by a `for`, `if`, `elif`, `else` or `def` block will be ignored. So for example the output of:

```
<?code langs = ["Python", "Java", "Javascript"]?>
<?if langs?>
   <?for lang in langs?>
      <?print lang?>
   <?end for?>
<?end if?>
```

will simply be:

```
Python
Java
Javascript
```

without any additional empty lines or indentation.

(Output will always be ignored when calling a template as a function.)

`signature` is the signature of the template. For a top level template it can be:

None
The template will accept all keyword arguments.

An `inspect.Signature` object
This signature will be used as the signature of the template.

A callable
The signature of the callable will be used.

A string
The signature as a string, i.e. something like `"x, y=42, *args, **kwargs"`. This string will be parsed and evaluated to create the signature for the template.

If the template is a locally defined subtemplate (i.e. a template defined by another template via `<?def t?>...<?end def?>`), `signature` can be:

None
The template will accept all arguments.

A *SignatureAST* object

This AST node will be evaluated at the point of definition of the subtemplate to create the final signature of the subtemplate.

classmethod loads(*data*)

Loads a template as an UL4ON dump from the string `data`.

classmethod load(*stream*)

Loads the template as an UL4ON dump from the stream `stream`. format.

dump(*stream*)

Dump the template in compiled UL4ON format to the stream `stream`.

dumps()

Return the template in compiled UL4ON format (as a string).

render(**args*, ***kwargs*)

Render the template iteratively (i.e. this is a generator).

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. Positional arguments will only be supported if the template has a signature.

render_with_globals(*args*, *kwargs*, *globals*)

Render the template iteratively (i.e. this is a generator).

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. `globals` contains global variables. Positional arguments will only be supported if the template has a signature.

renders(**args*, ***kwargs*)

Render the template as a string.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. Positional arguments will only be supported if the template has a signature.

renders_with_globals(*args*, *kwargs*, *globals*)

Render the template as a string.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. `globals` contains global variables. Positional arguments will only be supported if the template has a signature.

ul4_call(*context*, /, **args*, ***kwargs*)

Call the template as a function and return the resulting value.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. Positional arguments will only be supported if the template has a signature.

__call__(**args*, ***kwargs*)

Call the template as a function and return the resulting value.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. Positional arguments will only be supported if the template has a signature.

call_with_globals(*args*, *kwargs*, *globals*)

Call the template as a function and return the resulting value.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. `globals` contains global variables. Positional arguments will only be supported if the template has a signature.

jssource()

Return the template as the source code of a Javascript function.

javasource()

Return the template as Java source code.

class ll.ul4c.**SignatureAST**

Bases: *CodeAST*

AST node for the signature of a locally defined subtemplate.

Attributes are:

params

[list]
The parameter. Each list item is a tuple with three items:

name

[str]
The name of the argument.

type

[str]
The type of the argument. One of:

- pk (positional or keyword argument without default)
- pk= (positional or keyword argument with default)
- p (positional-only argument without default)
- p= (positional-only argument with default)
- k (keyword-only argument without default)
- k= (keyword-only argument with default)
- * (argument that collects addition positional arguments)
- ** (argument that collects addition keyword arguments)

default

[*AST* or None]
The default value for the argument (or None if the argument has no default value).

class ll.ul4c.**TemplateClosure**

Bases: *BlockAST*

A locally defined subtemplate.

A TemplateClosure is a closure, i.e. it can use the local variables of the template it is defined in.

class ll.ul4c.**BoundTemplate**

Bases: object

A template bound to an object.

Calling or rendering a *BoundTemplate* instance passes the object to which the template is bound as the first positional argument.

render(**args*, ***kwargs*)

Render the template iteratively (i.e. this is a generator).

args and kwargs contain the top level positional and keyword arguments available to the template code. Positional arguments will only be supported if the template has a signature.

render_with_globals(*args*, *kwargs*, *globals*)

Render the template iteratively (i.e. this is a generator).

args and kwargs contain the top level positional and keyword arguments available to the template code. globals contains global variables. Positional arguments will only be supported if the template has a signature.

renders(**args*, ***kwargs*)

Render the template as a string.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. Positional arguments will only be supported if the template has a signature.

renders_with_globals(*args*, *kwargs*, *globals*)

Render the template as a string.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. `globals` contains global variables. Positional arguments will only be supported if the template has a signature.

__call__(**args*, ***kwargs*)

Call the template as a function and return the resulting value.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. Positional arguments will only be supported if the template has a signature.

call_with_globals(*args*, *kwargs*, *globals*)

Call the template as a function and return the resulting value.

`args` and `kwargs` contain the top level positional and keyword arguments available to the template code. `globals` contains global variables. Positional arguments will only be supported if the template has a signature.

6.3 `ll.ul4on` – Object serialization

This module provides functions for encoding and decoding a lightweight text-based format for serializing the object types supported by UL4.

It is extensible to allow encoding/decoding arbitrary instances (i.e. it is basically a reimplementation of `pickle`, but with string input/output instead of bytes and with an eye towards cross-plattform support).

There are implementations for Python (this module), Java and Javascript (as part of the UL4 packages for those languages).

Furthermore there's an Oracle package that can be used for generating UL4ON encoded data.

Basic usage follows the API design of `pickle`, `json`, etc. and supports most builtin Python types:

```
>>> from ll import ul4on
>>> ul4on.dumps(None)
'n'
>>> ul4on.loads('n')
>>> ul4on.dumps(False)
'bF'
>>> ul4on.loads('bF')
False
>>> ul4on.dumps(42)
'i42'
>>> ul4on.loads('i42')
42
>>> ul4on.dumps(42.5)
'f42.5'
>>> ul4on.loads('f42.5')
42.5
>>> ul4on.dumps('foo')
"S'foo'"
>>> ul4on.loads("S'foo'")
'foo'
```

`date`, `datetime` and `timedelta` objects are supported too:

```
>>> import datetime
>>> ul4on.dumps(datetime.date.today())
'X i2014 i11 i3'
>>> ul4on.dumps(datetime.datetime.now())
'Z i2014 i11 i3 i18 i16 i45 i314157'
>>> ul4on.loads('X i2014 i11 i3')
datetime.date(2014, 11, 3)
>>> ul4on.loads('Z i2014 i11 i3 i18 i16 i45 i314157')
datetime.datetime(2014, 11, 3, 18, 16, 45, 314157)
>>> ul4on.dumps(datetime.timedelta(days=1))
'T i1 i0 i0'
>>> ul4on.loads('T i1 i0 i0')
datetime.timedelta(1)
```

`ll.ul4on` also supports *`Color`* objects from *`ll.color`*:

```
>>> from ll import color
>>> ul4on.dumps(color.red)
'C i255 i0 i0 i255'
>>> ul4on.loads('C i255 i0 i0 i255')
Color(0xff, 0x00, 0x00)
```

Lists, dictionaries and sets are also supported:

```
>>> ul4on.dumps([1, 2, 3])
'L i1 i2 i3 ]'
>>> ul4on.loads('L i1 i2 i3 ]')
[1, 2, 3]
>>> ul4on.dumps(dict(one=1, two=2))
"D S'two' i2 S'one' i1 }"
>>> ul4on.loads("D S'two' i2 S'one' i1 }")
{'one': 1, 'two': 2}
>>> ul4on.dumps({1, 2, 3})
'Y i1 i2 i3 }'
>>> ul4on.loads('Y i1 i2 i3 }')
{1, 2, 3}
```

6.3.1 Recursive data structures

`ll.ul4on` can also handle recursive data structures:

```
>>> r = []
>>> r.append(r)
>>> ul4on.dumps(r)
'L ^0 ]'
>>> r2 = ul4on.loads('L ^0 ]')
>>> r2
[[...]]
>>> r2 is r2[0]
True
>>> r = {}
>>> r['recursive'] = r
>>> ul4on.dumps(r)
"D S'recursive' ^0 }"
>>> r2 = ul4on.loads("D S'recursive' ^0 }")
```

(continues on next page)

(continued from previous page)

```
>>> r2
{'recursive': {...}}
>>> r2['recursive'] is r2
True
```

Note: The `^0` part in the dump is a so called "back reference", it tells the decoder that in this spot an object is referenced that has already been part of the dump (The `0` indicates where in the dump the object can be found).

6.3.2 Extensibility

UL4ON is extensible. It supports serializing arbitrary instances by registering the class with the UL4ON serialization machinery:

```
from ll import ul4on

@ul4on.register("com.example.person")
class Person:
   def __init__(self, firstname=None, lastname=None):
      self.firstname = firstname
      self.lastname = lastname

   def __repr__(self):
      return f"<Person firstname={self.firstname!r} lastname={self.lastname!r}>"

   def ul4ondump(self, encoder):
      encoder.dump(self.firstname)
      encoder.dump(self.lastname)

   def ul4onload(self, decoder):
      self.firstname = decoder.load()
      self.lastname = decoder.load()

jd = Person("John", "Doe")
output = ul4on.dumps(jd)
print("Dump:", output)
jd2 = ul4on.loads(output)
print("Loaded:", jd2)
```

This script outputs:

```
Dump: O S'com.example.person' S'John' S'Doe' )
Loaded: <Person firstname='John' lastname='Doe'>
```

It is also possible to pass a custom registry to *load()* and *loads()*:

```
from ll import ul4on

class Person:
   ul4onname = "com.example.person"

   def __init__(self, firstname=None, lastname=None):
      self.firstname = firstname
      self.lastname = lastname
```

(continues on next page)

(continued from previous page)

```
    def __repr__(self):
        return f"<Person firstname={self.firstname!r} lastname={self.lastname!r}>"

    def ul4ondump(self, encoder):
        encoder.dump(self.firstname)
        encoder.dump(self.lastname)

    def ul4onload(self, decoder):
        self.firstname = decoder.load()
        self.lastname = decoder.load()

jd = Person("John", "Doe")
output = ul4on.dumps(jd)
print("Dump:", output)
jd2 = ul4on.loads(output, {"com.example.person": Person})
print("Loaded:", jd2)
```

Any type name not found in the registry dict passed in will be looked up in the global registry.

Note: If a class isn't registered with the UL4ON serialization machinery, you have to set the class attribute `ul4onname` yourself for serialization to work.

For deserialization the class **must** be registered either in the local registry passed to the *Decoder* or globally via *register()*.

6.3.3 Object content mismatch

In situations where an UL4ON API is updated frequently it is useful to be able to update the writing side and the reading side independently. To support this, *Decoder* has a method *loadcontent()* that is a generator that reads the content items of an object from the input stream and yields those items.

This allows to handle both situations:

- When the writing side outputs more items that the reading side expects, exhausting the iterator returned by `loadcontent()` will read and ignore the unrecognized items and leave the input stream in a consistent state.
- When the writing side outputs less items then the reading side expects, the remaining items can by initialized with default values.

For our example class it could be used like this:

```
from ll import ul4on

class Person:
    ul4onname = "com.example.person"

    def __init__(self, firstname=None, lastname=None):
        self.firstname = firstname
        self.lastname = lastname

    def __repr__(self):
        return f"<Person firstname={self.firstname!r} lastname={self.lastname!r}>"

    def ul4ondump(self, encoder):
        encoder.dump(self.firstname)
        encoder.dump(self.lastname)
```

(continues on next page)

(continued from previous page)

```
    def ul4onload(self, decoder):
        index = -1
        for (index, item) in enumerate(decoder.loadcontent()):
            if index == 0:
                self.firstname = item
            elif index == 1:
                self.lastname = item
        # Initialize attributes that were not loaded by ``loadcontent``
        if index < 1:
            self.lastname = None
            if index < 0:
                self.firstname = None

output = """o s'com.example.person' s'John' )"""
j = ul4on.loads(output, {"com.example.person": Person})
print("Loaded:", j)
```

This outputs:

```
Loaded: <Person firstname='John' lastname=None>
```

6.3.4 Chunked UL4ON

`ll.ul4on` also provides access to the classes that implement UL4ON encoding and decoding. This can be used to create multiple UL4ON dumps using the same encoding context, or recreate multiple objects from those multiple UL4ON dumps (using the same decoding context).

An example for encoding:

```
encoder = ul4on.Encoder()
obj = "spam"
print(encoder.dumps(obj))
print(encoder.dumps(obj))
```

This prints:

```
S'spam'
^0
```

The second call outputs a back reference, since the encoder remembers that the string `"spam"` has already been output.

An example for decoding:

```
decoder = ul4on.Decoder()
print(decoder.loads("S'spam'"))
print(decoder.loads("^0"))
```

This prints:

```
spam
spam
```

since the decoder remembers which object has been decoded as the first object from the first dump.

One application of this is embedding multiple related UL4ON dumps as data attributes in HTML and then deserializing those UL4ON chuncks back into the appropriate Javascript objects. For example:

```
from ll import ul4on
from ll.misc import xmlencode as xe

encoder = ul4on.Encoder()

counter = 0

def dump(obj):
   global counter
   counter += 1
   return f"{counter} {encoder.dumps(obj)}"

data = ["gurk", "hurz", "hinz", "kunz"]

def f(s):
   return f"<li data-ul4on='{xe(dump(s))}'>{xe(s.upper())}</li>"

items = "\n".join(f(s) for s in data)
html = f"<ul data-ul4on='{xe(dump(data))}'>\n{items}\n</ul>"
print(html)
```

This outputs:

```
<ul data-ul4on='5 L ^0 ^1 ^2 ^3 ]'>
<li data-ul4on='1 S&#39;gurk&#39;'>GURK</li>
<li data-ul4on='2 S&#39;hurz&#39;'>HURZ</li>
<li data-ul4on='3 S&#39;hinz&#39;'>HINZ</li>
<li data-ul4on='4 S&#39;kunz&#39;'>KUNZ</li>
</ul>
```

By iterating through the `data-ul4on` attributes in the correct order and feeding each UL4ON chunk to the same decoder, all objects can be recreated and attached to their appropriate HTML elements.

6.3.5 Incremental UL4ON and persistent objects

Objects that have an attribute `ul4onid` are considered "persistent" objects. The combination of `ul4onname` and `ul4onid` uniquely identifies each persistent object (even across multiple unrelated UL4ON dumps).

An *Encoder* will dump those objects differently than other objects without an `ul4onid` attribute.

A *Decoder* will remember all persistent objects it has loaded (under their `ul4onname` and `ul4onid`). If the decoder encounters the `ul4onname` and `ul4onid` of an object it has remembered, it will not create a new object, instead `ul4onload()` will be called for the existing object. If the decoder encounters a persistent objects it hasn't remembered, it will create a new object (passing the `ul4onid` as the only argument to the constructor) and then call `ul4onload()` on the new object.

This means that with this approach it's possible to use one *Decoder* object to load multiple unrelated UL4ON dumps "incrementally" one after the other, but still merge the persistent objects in the subsequent dumps into the those created by previous dumps.

Note: For persistent objects `ul4onload()` and `ul4ondump()` don't have the dump/load the `ul4onid` attribute, as this is done by the *Encoder*/*Decoder*.

Note: If the value of the attribute `ul4onid` is `None` the object will be treated as an "ordinary" (i.e. non-persistent) object.

Note: For this approach, the method `reset()` must be called between calls to `load()` or `loads()` to reset the information about back references.

6.3.6 Module documentation

`ll.ul4on.register`(*name: str*)

This decorator can be used to register the decorated class with the `ll.ul4on` serialization machinery.

`name` must be a globally unique name for the class. To avoid name collisions Java's class naming system should be used (i.e. an inverted domain name like `com.example.foo.bar`).

`name` will be stored in the class attribute `ul4onname`.

class `ll.ul4on.Encoder`

Bases: `object`

An `Encoder` is used for serializing an object into an UL4ON dump.

It manages the internal state required for handling backreferences and other stuff.

`__init__`(*indent: str = None*)

Create an encoder for serializing objects.

When `indent` is not `None`, it is used as an indentation string for pretty printing the output.

`dumps`(*obj: Any*) → str

Serialize `obj` and return the resulting dump as a string.

`dump`(*obj: Any, stream: TextIO | None = None*) → None

Serialize `obj` into the stream `stream` as an UL4ON formatted dump.

`stream` must provide a `write()` method.

Passing `None` for `stream` may only be done by objects that call `dump()` to implement UL4ON serialization in their own `ul4ondump()` method.

class `ll.ul4on.Decoder`

Bases: `object`

A `Decoder` is used for deserializing an UL4ON dump.

It manages the internal state required for handling backreferences, persistent objects and other stuff.

`__init__`(*registry: Dict[str, Callable[[...], Any]] | None = None*)

Create a decoder for deserializing objects from an UL4ON dump.

`registry` is used as a "custom type registry". It must map UL4ON type names to callables that create new empty instances of those types. Any type not found in `registry` will be looked up in the global registry (see `register()`).

`loads`(*dump: str*) → Any

Deserialize the object in the string `dump` and return it.

`load`(*stream: TextIO | None = None*) → Any

Deserialize the next object from the stream `stream` and return it.

`stream` must provide a `read()` method.

Passing `None` for `stream` may only be done by objects that call `load()` to implement UL4ON deserialization in their own `ul4onload()` method.

loadcontent() → Generator[Any, None, None]

Load the content of an object until the "object terminator" is encountered.

This is a generator and might produce fewer or more items than expected. The caller must be able to handle both cases (e.g. by ignoring additional items or initializing missing items with a default value).

The iterator should always be exhausted when it is read, otherwise the stream will be in an undefined state.

loadcontentitems() → Generator[Tuple[str, Any], None, None]

Similar to *loadcontent()*, but will load the content of an object as (key, value) pairs.

For further info see *loadcontent()*.

reset() → None

Clear the internal cache for backreferences so that a new unrelated UL4ON dump can be loaded.

However the cache for persistent objects will not be cleared.

store_persistent_object(*object*) → None

Add a persistent object to the cache of persistent objects.

forget_persistent_object(*object*) → None

Remove a persistent object from the cache of persistent objects.

persistent_object(*name: str*, *id: str*) → Any

Return the persistent object with the type `name` and the id `id`, or `None`, when the decoder hasn't encountered that object yet.

persistent_objects() → ValuesView[Any]

Return an iterator over all persistent objects the decoder has encountered so far.

ll.ul4on.**dumps**(*obj: Any*, /, *indent: str | None = None*) → str

Serialize `obj` as an UL4ON formatted string.

ll.ul4on.**dump**(*obj: Any*, /, *stream: TextIO*, *indent: str | None = None*) → None

Serialize `obj` as an UL4ON formatted stream to `stream`.

`stream` must provide a `write()` method.

ll.ul4on.**load**(*stream: TextIO*, /, *registry: Dict[str, Callable[[...], Any]] | None = None*) → Any

Deserialize `stream` (which must be file-like object with a `read()` method containing an UL4ON formatted object) to a Python object.

For the meaning of `registry` see *Decoder.__init__()*.

ll.ul4on.**loads**(*dump: str*, /, *registry: Dict[str, Callable[[...], Any]] | None = None*) → Any

Deserialize `dump` (which must be a string containing an UL4ON formatted object) to a Python object.

For the meaning of `registry` see *Decoder.__init__()*.

ll.ul4on.**loadclob**(*clob*, /, *bufsize: int = 1048576*, *registry: Dict[str, Callable[[...], Any]] | None = None*) → Any

Deserialize `clob` (which must be an `cx_Oracle` `CLOB` variable containing an UL4ON formatted object) to a Python object.

`bufsize` specifies the chunk size for reading the underlying `CLOB` object.

For the meaning of `registry` see *Decoder.__init__()*.

6.4 ll.url – RFC 2396 compliant URLs

ll.url contains an RFC 2396 compliant implementation of URLs and classes for accessing resource metadata as well as file like classes for reading and writing resource data.

These three levels of functionality are implemented in three classes:

URL
: *URL* objects are the names of resources and can be used and modified, regardless of the fact whether these resources actually exits. *URL* objects never hits the hard drive or the net.

Connection
: *Connection* objects contain functionality that accesses and changes file metadata (like last modified date, permission bits, directory structure etc.). A connection object can be created by calling the connect() method on a *URL* object.

Resource
: *Resource* objects are file like objects that work with the actual bytes that make up the file data. This functionality lives in the *Resource* class and its subclasses. Creating a resource is done by calling the open() method on a *Connection* or a *URL*.

6.4.1 Module documentation

ll.url.**httpdate**(*dt*)

Return a string suitable for a "Last-Modified" and "Expires" header.

dt is a datetime.datetime object in UTC.

class ll.url.**Context**

Bases: object

Working with URLs (e.g. calling *URL.open()* or *URL.connect()*) involves *Connection* objects. To avoid constantly creating new connections you can pass a Context object to those methods. Connections will be stored in the Context object and will be reused by those methods.

A Context object can also be used as a context manager. This context object will be used for all open() and connect() calls inside the with block. (Note that after the end of the with block all connections will be closed.)

closeall()

Close and drop all connections in this context.

class ll.url.**Cursor**

Bases: object

A Cursor object is used by the walk() method during directory traversal. It contains information about the state of the traversal and can be used to influence which directories are traversed and in which order.

Information about the state of the traversal is provided in the following attributes:

rooturl
: The URL where traversal has been started (i.e. the object for which the walk() method has been called)

url
: The current URL being traversed.

event
: A string that specifies which event is currently handled. Possible values are: "beforedir", "afterdir" and "file". A "beforedir" event is emitted before a directory is entered. "afterdir" is emitted after a directory has been entered. "file" is emitted when a file is encountered.

isdir
: True if url refers to a directory.

isfile
Tur if `url` refers to a regular file.

The following attributes specify which part of the tree should be traversed:

beforedir
Should the generator yield `"beforedir"` events?

afterdir
Should the generator yield `"afterdir"` events?

file
Should the generator yield `"file"` events?

enterdir
Should the directory be entered?

Note that if any of these attributes is changed by the code consuming the generator, this new value will be used for the next traversal step once the generator is resumed and will be reset to its initial value (specified in the constructor) afterwards.

__init__(*url*, *beforedir=True*, *afterdir=False*, *file=True*, *enterdir=False*)
Create a new `Cursor` object for a tree traversal rooted at the node `node`.

The arguments `beforedir`, `afterdir`, `file` and `enterdir` are used as the initial values for the attributes of the same name. (see the class docstring for info about their use).

restore()
Restore the attributes `beforedir`, `afterdir`, `file` and `enterdir` to their initial value.

class ll.url.**Connection**

Bases: `object`

A `Connection` object is used for accessing and modifying the metadata associated with a file. It is created by calling the *connect()* method on a *URL* object.

stat(*url*)
Return the result of a *stat()* call on the file `url`.

lstat(*url*)
Return the result of a *stat()* call on the file `url`. Like *stat()*, but does not follow symbolic links.

chmod(*url*, *mode*)
Set the access mode of the file `url` to `mode`.

chown(*url*, *owner=None*, *group=None*)
Change the owner and/or group of the file `url`.

lchown(*url*, *owner=None*, *group=None*)
Change the owner and/or group of the file `url` (ignoring symbolic links).

uid(*url*)
Return the user id of the owner of the file `url`.

gid(*url*)
Return the group id the file `url` belongs to.

owner(*url*)
Return the name of the owner of the file `url`.

group(*url*)
Return the name of the group the file `url` belongs to.

mimetype(*url*)
Return the mimetype of the file `url`.

exists(*url*)

Test whether the file `url` exists.

isfile(*url*)

Test whether the resource `url` is a file.

isdir(*url*)

Test whether the resource `url` is a directory.

islink(*url*)

Test whether the resource `url` is a link.

ismount(*url*)

Test whether the resource `url` is a mount point.

access(*url*, *mode*)

Test for access to the file/resource `url`.

size(*url*)

Return the size of the file `url`.

imagesize(*url*)

Return the size of the image `url` (if the resource is an image file) as a `(width, height)` tuple. This requires the PIL.

cdate(*url*)

Return the "metadate change" date of the file/resource `url` as a `datetime.datetime` object in UTC.

adate(*url*)

Return the last access date of the file/resource `url` as a `datetime.datetime` object in UTC.

mdate(*url*)

Return the last modification date of the file/resource `url` as a `datetime.datetime` object in UTC.

resheaders(*url*)

Return the MIME headers for the file/resource `url`.

remove(*url*)

Remove the file `url`.

rmdir(*url*)

Remove the directory `url`.

rename(*url*, *target*)

Renames `url` to `target`. This might not work if `target` has a different scheme than `url` (or is on a different server).

link(*url*, *target*)

Create a hard link from `url` to `target`. This will not work if `target` has a different scheme than `url` (or is on a different server).

symlink(*url*, *target*)

Create a symbolic link from `url` to `target`. This will not work if `target` has a different scheme than `url` (or is on a different server).

chdir(*url*)

Change the current directory to `url`.

mkdir(*url*, *mode=511*)

Create the directory `url`.

makedirs(*url*, *mode=511*)

Create the directory `url` and all intermediate ones.

walk(*url*, *beforedir=True*, *afterdir=False*, *file=True*, *enterdir=True*)

Return an iterator for traversing the directory hierarchy rooted at the directory `url`.

Each item produced by the iterator is a *Cursor* object. It contains information about the state of the traversal and can be used to influence which parts of the directory hierarchy are traversed and in which order.

The arguments `beforedir`, `afterdir`, `file` and `enterdir` specify how the directory hierarchy should be traversed. For more information see the *Cursor* class.

Note that the *Cursor* object is reused by *walk()*, so you can't rely on any attributes remaining the same across calls to `next()`.

The following example shows how to traverse the current directory, print all files except those in certain directories:

```
from ll import url

for cursor in url.here().walk(beforedir=True, afterdir=False, file=True):
   if cursor.isdir:
      if cursor.url.path[-2] in (".git", "build", "dist", "__pycache__"):
         cursor.enterdir = False
   else:
      print(cursor.url)
```

listdir(*url*, *include=None*, *exclude=None*, *ignorecase=False*)

Iterates over items in the directory `url`. The items produced are *URL* objects relative to `url`.

With the optional `include` argument, this only lists items whose names match the given pattern. Items matching the optional pattern `exclude` will not be listed. `include` and `exclude` can be strings (which will be interpreted as `fnmatch` style filename patterns) or lists of strings. If `ignorecase` is true case-insensitive name matching will be performed.

files(*url*, *include=None*, *exclude=None*, *ignorecase=False*)

Iterates over files in the directory `url`. The items produced are *URL* objects relative to `url`.

With the optional `include` argument, this only lists files whose names match the given pattern. Files matching the optional pattern `exclude` will not be listed. `include` and `exclude` can be strings (which will be interpreted as `fnmatch` style filename patterns) or lists of strings. If `ignorecase` is true case-insensitive name matching will be performed.

dirs(*url*, *include=None*, *exclude=None*, *ignorecase=False*)

Iterates over directories in the directory `url`. The items produced are *URL* objects relative to `url`.

With the optional `include` argument, this only directories items whose names match the given pattern. Directories matching the optional pattern `exclude` will not be listed. `include` and `exclude` can be strings (which will be interpreted as `fnmatch` style filename patterns) or lists of strings. If `ignorecase` is true case-insensitive name matching will be performed.

walkall(*url*, *include=None*, *exclude=None*, *enterdir=None*, *skipdir=None*, *ignorecase=False*)

Recursively iterate over files and subdirectories. The iterator yields *URL* objects naming each child URL of the directory `url` and its descendants relative to `url`. This performs a depth-first traversal, returning each directory before all its children.

With the optional `include` argument, only yield items whose names match the given pattern. Items matching the optional pattern `exclude` will not be listed. Directories that don't match the optional pattern `enterdir` or match the pattern `skipdir` will not be traversed. `include`, `exclude`, `enterdir` and `skipdir` can be strings (which will be interpreted as `fnmatch` style filename patterns) or lists of strings. If `ignorecase` is true case-insensitive name matching will be performed.

walkfiles(*url*, *include=None*, *exclude=None*, *enterdir=None*, *skipdir=None*, *ignorecase=False*)

Return a recursive iterator over files in the directory `url`.

With the optional `include` argument, only yield files whose names match the given pattern. Files matching the optional pattern `exclude` will not be listed. Directories that don't match the optional pattern `enterdir` or match the pattern `skipdir` will not be traversed. `include`, `exclude`, `enterdir` and `skipdir` can be strings (which will be interpreted as `fnmatch` style filename patterns) or lists of strings. If `ignorecase` is true case-insensitive name matching will be performed.

walkdirs(*url*, *include=None*, *exclude=None*, *enterdir=None*, *skipdir=None*, *ignorecase=False*)

Return a recursive iterator over subdirectories in the directory `url`.

With the optional `include` argument, only yield directories whose names match the given pattern. Items matching the optional pattern `exclude` will not be listed. Directories that don't match the optional pattern `enterdir` or match the pattern `skipdir` will not be traversed. `include`, `exclude`, `enterdir` and `skipdir` can be strings (which will be interpreted as `fnmatch` style filename patterns) or lists of strings. If `ignorecase` is true case-insensitive name matching will be performed.

open(*url*, **args*, ***kwargs*)

Open `url` for reading or writing. *`open()`* returns a *`Resource`* object.

Which additional parameters are supported depends on the actual resource created. Some common parameters are:

mode
[`str`]
A string indicating how the file is to be opened (just like the mode argument for the builtin *`open()`* (e.g. `"rb"` or `"wb"`).

headers
[`dict`]
Additional headers to use for an HTTP request.

data
[`bytes`]
Request body to use for an HTTP POST request.

python
[`str` or `None`]
Name of the Python interpreter to use on the remote side (used by `ssh` URLs)

nice
[`int` or `None`]
Nice level for the remote python (used by `ssh` URLs)

check
[`bool` or `None`]
Whether `ssh` host keys should be checked (used by `ssh` URLs where it defaults to `True` and `ssh-nocheck` URLs where it defaults to `False`).

class `ll.url.`**LocalConnection**

Bases: *`Connection`*

A `LocalConnection` object is used for accessing and modifying the metadata associated with a file in the local filesystem. It is created by calling the *`connect()`* method on a *`URL`* object with no scheme or the `file` or `root` scheme.

class `ll.url.`**SshConnection**

Bases: *`Connection`*

A `SshConnection` object is used for accessing and modifying the metadata associated with a file on a remote filesystem. Remote files will be accessed via code executed remotely on the target host via `execnet`.

`SshConnection` objects are created by calling the `connect()` method on a *`URL`* object with the `ssh` or `ssh-nocheck` scheme.

Note: Using the scheme `ssh-nocheck` disables checks of the host key, i.e. it passes `-o "StrictHostKeyChecking=no"` to the underlying `ssh` command.

If you need to use further options (e.g. when your `known_hosts` file isn't writable), you should configure that in your `~/.ssh/config` file, for example:

```
Host foo
   Hostname foo.example.org
   StrictHostKeyChecking no
   UserKnownHostsfile /dev/null
```

or for Windows:

```
Host foo
   Hostname foo.example.org
   StrictHostKeyChecking no
   UserKnownHostsfile nul:
```

class `ll.url.`**URLConnection**

Bases: *Connection*

A `URLConnection` object is used for accessing and modifying the metadata associated any other resource specified by a URL (except those handled by the other *Connection* subclasses).

`ll.url.`**here**(*scheme='file'*)

Return the current directory as an *URL* object.

`ll.url.`**home**(*user=''*, *scheme='file'*)

Return the home directory of the current user (or the user named `user`, if `user` is specified) as an *URL* object:

```
>>> url.home()
URL('file:/home/walter/')
>>> url.home("andreas")
URL('file:/home/andreas/')
```

`ll.url.`**root**()

Return a blank `root` *URL*, i.e. `URL("root:")`.

`ll.url.`**File**(*name*, *scheme='file'*)

Turn a filename into an *URL* object:

```
>>> url.File("a#b")
URL('file:a%23b')
```

`ll.url.`**Dir**(*name*, *scheme='file'*)

Turns a directory name into an *URL* object, just like *File()*, but ensures that the path is terminated with a `/`:

```
>>> url.Dir("a#b")
URL('file:a%23b/')
```

`ll.url.`**Ssh**(*user*, *host*, *path='~/'*)

Return a ssh *URL* for the user `user` on the host `host` with the path `path`. `path` (defaulting to the users home directory) must be a path in URL notation (i.e. use `/` as directory separator):

```
>>> url.Ssh("root", "www.example.com", "~joe/public_html/index.html")
URL('ssh://root@www.example.com/~joe/public_html/index.html')
```

If the path starts with `~/` it is relative to this users home directory, if it starts with `~user` it's relative to the home directory of the user `user`. In all other cases the path is considered to be absolute.

`ll.url.first(urls)`

Return the first URL from `urls` that exists as a real file or directory. `None` entries in `urls` will be skipped.

`ll.url.firstdir(urls)`

Return the first URL from `urls` that exists as a real directory. `None` entries in `urls` will be skipped.

`ll.url.firstfile(urls)`

Return the first URL from `urls` that exists as a real file. `None` entries in `urls` will be skipped.

class `ll.url.Resource`

Bases: `object`

A `Resource` is a base class that provides a file-like interface to local and remote files, URLs and other resources.

Each resource object has the following attributes:

url
The URL for which this resource has been opened (i.e. `foo.open().url is foo` if `foo` is a *URL* object);

name
A string version of `url`;

closed
A `bool` specifying whether the resource has been closed (i.e. whether the `close()` method has been called).

In addition to file methods (like `read()`, `readlines()`, `write()` and `close()`) a resource object might provide the following methods:

finalurl()
Return the real URL of the resource (this might be different from the `url` attribute in case of a redirect).

size()
Return the size of the file/resource.

mdate()
Return the last modification date of the file/resource as a `datetime.datetime` object in UTC.

mimetype()
Return the mimetype of the file/resource.

imagesize()
Return the size of the image (if the resource is an image file) as a `(width, height)` tuple. This requires the PIL.

class `ll.url.FileResource`

Bases: *Resource*

A subclass of *Resource* that handles local files.

class `ll.url.RemoteFileResource`

Bases: *Resource*

A subclass of *Resource* that handles remote files (i.e. those using the `ssh` scheme).

class `ll.url.URLResource`

Bases: *Resource*

A subclass of *Resource* that handles HTTP, FTP and other URLs (i.e. those that are not handled by *FileResource* or *RemoteFileResource*.

class ll.url.**SchemeDefinition**

Bases: object

A SchemeDefinition instance defines the properties of a particular URL scheme.

__init__(*scheme*, *usehierarchy*, *useserver*, *usefrag*, *islocal=False*, *isremote=False*, *defaultport=None*)

Create a new SchemeDefinition instance. Arguments are:

- scheme: The name of the scheme;
- usehierarchy: Specifies whether this scheme uses hierarchical URLs or opaque URLs (i.e. whether hier_part or opaque_part from the BNF in RFC 2396 is used);
- useserver: Specifies whether this scheme uses an Internet-based server authority component or a registry of naming authorities (only for hierarchical URLs);
- usefrag: Specifies whether this scheme uses fragments (according to the BNF in RFC 2396 every scheme does, but it doesn't make sense for e.g. "javascript", "mailto" or "tel");
- islocal: Specifies whether URLs with this scheme refer to local files;
- isremote: Specifies whether URLs with this scheme refer to remote files (there may be schemes which are neither local nor remote, e.g. "mailto");
- defaultport: The default port for this scheme (only for schemes using server based authority).

connect(*url*, *context=None*, ***kwargs*)

Create a *Connection* for the *URL* url (which must have self as the scheme).

closeall(*context*)

Close all connections active for this scheme in the context context.

class ll.url.**URL**

Bases: object

An RFC 2396 compliant URL.

__init__(*url=None*)

Create a new URL instance. url may be a str object, or an URL (in which case you'll get a copy of url), or None (which will create an URL referring to the "current document").

clone()

Return an identical copy self.

withext(*ext*)

Return a new *URL* where the filename extension has been replaced with ext.

withoutext()

Return a new *URL* where the filename extension has been removed.

withfile(*file*)

Return a new *URL* where the filename (i.e. the name of last component of path_segments) has been replaced with file.

withfrag(*frag*)

Return a new *URL* where the fragment has been replaced with frag.

withoutfrag()

Return a new *URL* where the frag has been dropped.

__truediv__(*other*)

Join self with another (possible relative) *URL* other, to form a new *URL*.

other may be a str or *URL* object. It may be None (referring to the "current document") in which case self will be returned. It may also be a list or other iterable. For this case a list (or iterator) will be returned where __div__() will be applied to every item in the list/iterator. E.g. the following

expression returns all the files in the current directory as absolute URLs (see the method `files()` and the function *here()* for further explanations):

```
>>> here = url.here()
>>> for f in here/here.files():
...   print(f)
```

__rtruediv__(*other*)

Right hand version of `__div__()`. This supports lists and iterables as the left hand side too.

relative(*baseurl*, *allowschemerel=False*)

Return an relative *URL* `rel` such that `baseurl/rel == self`, i.e. this is the inverse operation of `__div__()`.

If `self` is relative, has a different `scheme` or `authority` than `baseurl` or a non-hierarchical scheme, an identical copy of `self` will be returned.

If `allowschemerel` is true, scheme relative URLs are allowed, i.e. if both `self` and `baseurl` use the same hierarchical scheme, but a different authority (i.e. server), a scheme relative url (`//server/path/file.html`) will be returned.

__bool__()

Return whether the *URL* is not empty, i.e. whether it is not the *URL* referring to the start of the current document.

__eq__(*other*)

Return whether two *URL* objects are equal. Note that only properties relevant for the current scheme will be compared.

__ne__(*other*)

Return whether two *URL* objects are *not* equal.

__hash__()

Return a hash value for `self`, to be able to use *URL* objects as dictionary keys. You must be careful not to modify an *URL* as soon as you use it as a dictionary key.

abs(*scheme=-1*)

Return an absolute version of `self` (works only for local URLs).

If the argument `scheme` is specified, it will be used for the resulting URL otherwise the result will have the same scheme as `self`.

real(*scheme=-1*)

Return the canonical version of `self`, eliminating all symbolic links (works only for local URLs).

If the argument `scheme` is specified, it will be used for the resulting URL otherwise the result will have the same scheme as `self`.

islocal()

Return whether `self` refers to a local file, i.e. whether `self` is a relative *URL* or the scheme is `root` or `file`).

local()

Return `self` as a local filename (which will only works if `self` is local (see *islocal()*).

connect(*context=None*, ***kwargs*)

Return a *Connection* object for accessing and modifying the metadata of `self`.

Whether you get a new connection object, or an existing one depends on the scheme, the URL itself, and the context passed in (as the `context` argument).

open(**args*, ***kwargs*)

Open `self` for reading or writing. *open()* returns a *Resource* object.

Which additional parameters are supported depends on the actual resource created. Some common parameters are:

mode (supported by all resources)
A string indicating how the file is to be opened (just like the mode argument for the builtin *open()*; e.g. `"rb"` or `"wb"`).

context (supported by all resources)
open() needs a *Connection* for this URL which it gets from a *Context* object.

headers
Additional headers to use for an HTTP request.

data
Request body to use for an HTTP POST request.

python
Name of the Python interpreter to use on the remote side (used by `ssh` URLs)

nice
Nice level for the remove python (used by `ssh` URLs)

import_(*name=None*)

Import the content of the URL `self` as a Python module.

`name` can be used the specify the module name (i.e. the `__name__` attribute of the module). The default determines it from the URL.

6.4.2 Special features of `ll.url`

The class *ll.url.URL* supports many common schemes and one additional special scheme named `root` that deserves an explanation.

A `root` URL is supposed to be an URL that is relative to a "project" directory instead to a base URL of the document that contains the URL.

Suppose we have a document with the following base URL:

```
>>> from ll import url
>>> base = url.URL("root:company/it/about/index.html")
```

Now, if we have the following relative URL in this document:

```
>>> url1 = url.URL("images/logos/spam.png")
```

the combined URL will be:

```
>>> base/url1
URL('root:company/it/about/images/logos/spam.png')
```

Now it we use this combined URL and interpret it relative to the base URL we get back our original relative URL:

```
>>> (base/url1).relative(base)
URL('images/logos/spam.png')
```

Let's try a `root` URL now:

```
>>> url2 = url.URL("root:images/logos/spam.png")
```

Combining this URL with the base URL gives us the same as `url2`:

```
>>> base/url2
URL('root:images/logos/spam.png')
```

But if we interpret this result relative to `base`, we'll get:

```
>>> (base/url2).relative(base)
URL('../../../images/logos/spam.png')
```

I.e. this gives us a relative URL that references `url2` from `base` when both URLs are relative to the same root directory.

6.5 `ll.make` – Object oriented make replacement

`ll.make` provides tools for building projects.

Like make it allows you to specify dependencies between files and actions to be executed when files don't exist or are out of date with respect to one of their sources. But unlike make you can do this in an object oriented way and targets are not only limited to files.

Relevant classes are:

- *`Project`*, which is the container for all actions in a project and
- *`Action`* (and subclasses), which are used to transform input data and read and write files (or other entities like database records).

A simple script that copies a file `foo.txt` to `bar.txt` reencoding it from `"latin-1"` to `"utf-8"` in the process looks like this:

```
from ll import make

class MyProject(make.Project):
   name = "Acme.MyProject"

   def create(self):
      make.Project.create(self)
      source = self.add(make.FileAction("foo.txt"))
      temp = source.callattr("decode", "iso-8859-1")
      temp = temp.callattr("encode", "utf-8")
      target = self.add(make.FileAction("bar.txt", temp))
      self.writecreatedone()

p = MyProject()
p.create()

if __name__ == "__main__":
   p.build("bar.txt")
```

`ll.make.`**`filechanged`**(*key*)

Get the last modified date (or `bigbang`, if the file doesn't exist).

class `ll.make.`**`Level`**

Bases: `object`

Stores information about the recursive execution of *`Action`*s.

`ll.make.`**`report`**(*func*)

Standard decorator for *`Action.get()`* methods.

This decorator handles proper reporting of nested action calls. If it isn't used, only the output of calls to *Project.writestep()* will be visible to the user.

exception ll.make.**RedefinedTargetWarning**

Bases: Warning

Warning that will be issued when a target is added to a project and a target with the same key already exists.

exception ll.make.**UndefinedTargetError**

Bases: KeyError

Exception that will be raised when a target with the specified key doesn't exist within the project.

ll.make.**getoutputs**(*project*, *since*, *input*)

Recursively iterate through the object input (if it's a tuple, list or dict) and return a tuple containing:

- An object (data) of the same structure as input, where every action object encountered is replaced with the output of that action;
- A timestamp (changed) which the newest timestamp among all the change timestamps of the actions encountered.

If none of the actions has any data newer than since (i.e. none of the actions produced any new data) data will be nodata.

class ll.make.**Action**

Bases: object

An *Action* is responsible for transforming input data into output data. It may have no, one or many inputs which themselves may be other actions. It fetches, combines and transforms the output data of those actions and returns its own output data.

__init__()

Create a new *Action* instance.

get(*project*, *since*)

This method (i.e. the implementations in subclasses) is the workhorse of ll.make. *get()* must return the output data of the action if this data has changed since since (which is a datetime.datetime object in UTC). If the data hasn't changed since since the special object nodata must be returned.

In both cases the action must make sure that the data is internally consistent, i.e. if the input data is the output data of other actions self has to ensure that those other actions update their data too, independent from the fact whether *get()* will return new data or not.

Two special values can be passed for since:

bigbang

This timestamp is older than any timestamp that can appear in real life. Since all data is newer than this, *get()* must always return output data.

bigcrunch

This timestamp is newer than any timestamp that can appear in real life. Since there can be no data newer than this, *get()* can only return output data in this case if ensuring internal consistency resulted in new data.

In all cases *get()* must set the instance attribute changed to the timestamp of the last change to the data before returning. In most cases this if the newest changed timestamp of the input actions.

execute(*project*, **args*, ***kwargs*)

Execute the action: transform the input data in args and kwargs and return the resulting output data. This method must be implemented in subclasses.

getkey()

Get the nearest key from self or its inputs. This is used by *ModuleAction* for the filename.

call(**args*, ***kwargs*)

Return a *CallAction* for calling `selfs` output with positional arguments from `args` and keyword arguments from `kwargs`.

getattr(*attrname*)

Return a *GetAttrAction* for getting `selfs` attribute named `attrname`.

callattr(*attrname*, **args*, ***kwargs*)

Return a *CallAttrAction* for calling `selfs` attribute named `attrname` with positional arguments from `args` and keyword arguments from `kwargs`.

__iter__()

Return an iterator over the input actions of `self`.

iterallinputs()

Return an iterator over all input actions of `self` (i.e. recursively).

findpaths(*input*)

Find dependency paths leading from `self` to the other action `input`. I.e. if `self` depends directly or indirectly on `input`, this generator will produce all paths `p` where `p[0] is self` and `p[-1] is input` and `p[i+1] in p[i]` for all `i` in `range(len(p)-1)`.

class ll.make.**ObjectAction**

Bases: *Action*

An *ObjectAction* returns an object.

class ll.make.**TransformAction**

Bases: *Action*

A *TransformAction* depends on exactly one input action and transforms the input data into output data.

class ll.make.**CollectAction**

Bases: *TransformAction*

A *CollectAction* is a *TransformAction* that simply outputs its input data unmodified, but updates a number of other actions in the process.

addinputs(**otherinputs*)

Register all actions in `otherinputs` as additional actions that have to be updated before `self` is updated.

class ll.make.**PhonyAction**

Bases: *Action*

A *PhonyAction* doesn't do anything. It may depend on any number of additional input actions which will be updated when this action gets updated. If there's new data from any of these actions, a *PhonyAction* will return `None` (and `nodata` otherwise as usual).

__init__(**inputs*, ***kwargs*)

Create a *PhonyAction* object. `doc` describes the action and is printed by the method *Project.writephonytargets()*.

addinputs(**inputs*)

Register all actions in `inputs` as additional actions that have to be updated once `self` is updated.

class ll.make.**FileAction**

Bases: *TransformAction*

A *FileAction* is used for reading and writing files (and other objects providing the appropriate interface).

__init__(*key*, *input=None*, *encoding=None*, *errors=None*)

Create a `FileAction` object with `key` as the "filename". `key` must be an object that provides a method `open()` for opening readable and writable streams to the file. `input` is the data written to the file (or the action producing the data). `encoding` is the encoding to be used for reading/writing. If `encoding` is `None` binary i/o will be used. `errors` is the codec error handling name for encoding/decoding text.

write(*project*, *data*)

Write `data` to the file and return it.

read(*project*)

Read the content from the file and return it.

get(*project*, *since*)

If a `FileAction` object doesn't have an input action it reads the input file and returns the content if the file has changed since `since` (otherwise `nodata` is returned).

If a `FileAction` object does have an input action and the output data from this input action is newer than the file `self.key` the data will be written to the file. Otherwise (i.e. the file is up to date) the data will be read from the file.

chmod(*mode=420*)

Return a `ModeAction` that will change the file permissions of `self` to `mode`.

chown(*user=None*, *group=None*)

Return an `OwnerAction` that will change the user and/or group ownership of `self`.

class ll.make.**MkDirAction**

Bases: `TransformAction`

This action creates the a directory (passing through its input data).

__init__(*key*, *mode=511*)

Create a `MkDirAction` instance. `mode` (which defaults to `0o777`) will be used as the permission bit pattern for the new directory.

execute(*project*, *data*)

Create the directory with the permission bits specified in the constructor.

class ll.make.**PipeAction**

Bases: `TransformAction`

This action pipes the input through an external shell command.

__init__(*input*, *command*)

Create a `PipeAction` instance. `command` is the shell command to be executed (which must read it's input from stdin and write its output to stdout).

class ll.make.**CacheAction**

Bases: `TransformAction`

A `CacheAction` is a `TransformAction` that passes through its input data, but caches it, so that it can be reused during the same build round.

class ll.make.**GetAttrAction**

Bases: `TransformAction`

This action gets an attribute from its input object.

class ll.make.**CallAction**

Bases: `Action`

This action calls a function or any other callable object with a number of arguments. Both positional and keyword arguments are supported and the function and the arguments can be static objects or actions.

class ll.make.**CallAttrAction**

Bases: *Action*

This action calls an attribute of an object with a number of arguments. Both positional and keyword arguments are supported and the object, the attribute name and the arguments can be static objects or actions.

class ll.make.**CommandAction**

Bases: *TransformAction*

This action executes a system command (via `os.system()`) and passes through the input data.

__init__(*command*, *input=None*)

Create a new *CommandAction* object. `command` is the command that will executed when `execute()` is called.

class ll.make.**ModeAction**

Bases: *TransformAction*

ModeAction changes file permissions and passes through the input data.

__init__(*input=None*, *mode=420*)

Create an *ModeAction* object. `mode` (which defaults to `0644`) will be use as the permission bit pattern.

execute(*project*, *data*, *mode*)

Change the permission bits of the file `self.getkey()`.

class ll.make.**OwnerAction**

Bases: *TransformAction*

OwnerAction changes the user and/or group ownership of a file and passes through the input data.

__init__(*input=None*, *user=None*, *group=None*)

Create a new *OwnerAction* object. `user` can either be a numerical user id or a user name or `None`. If it is `None` no user ownership will be changed. The same applies to `group`.

execute(*project*, *data*, *user*, *group*)

Change the ownership of the file `self.getkey()`.

class ll.make.**ModuleAction**

Bases: *TransformAction*

This action will turn the input string into a Python module.

__init__(*input=None*)

Create an *ModuleAction*.

This object must have an input action (which might be a *FileAction* that creates the source file).

addinputs(**inputs*)

Register all actions in `inputs` as modules used by this module. These actions must be *ModuleAction* objects too.

Normally it isn't necessary to call the method directly. Instead fetch the required module inside your module like this:

```
from ll import make

mymodule = make.currentproject.get("mymodule.py")
```

This will record your module as depending on `mymodule`, so if `mymodule` changes, your module will be reloaded too. For this to work you need to have an *ModuleAction* added to the project with the key `"mymodule.py"`.

class ll.make.**FOPAction**

Bases: *TransformAction*

This action transforms an XML string (containing XSL-FO) into PDF. For it to work Apache FOP is required. The command line is hardcoded but it's simple to overwrite the class attribute `command` in a subclass.

class ll.make.**AlwaysAction**

Bases: *Action*

This action always returns `None` as new data.

class ll.make.**NeverAction**

Bases: *Action*

This action never returns new data.

class ll.make.**Project**

Bases: `dict`

A *Project* collects all *Action* objects from a project. It is responsible for initiating the build process and for generating a report about the progress of the build process.

strtimedelta(*delta*)

Return a nicely formatted and colored string for the `datetime.timedelta` value `delta`. `delta` may also be `None` in with case `"0"` will be returned.

strdatetime(*dt*)

Return a nicely formatted and colored string for the `datetime.datetime` value `dt`.

strcounter(*counter*)

Return a nicely formatted and colored string for the counter value `counter`.

strerror(*text*)

Return a nicely formatted and colored string for the error text `text`.

strkey(*key*)

Return a nicely formatted and colored string for the action key `key`.

straction(*action*)

Return a nicely formatted and colored string for the action `action`.

__setitem__(*key*, *target*)

Add the action `target` to `self` as a target and register it under the key `key`.

add(*target*, *key=None*)

Add the action `target` as a target to `self`. If `key` is not `None`, `target` will be registered under this key, otherwise it will be registered under its own key (i.e. `target.key`).

__getitem__(*key*)

Return the target with the key `key`. If an key can't be found, it will be wrapped in a *ll.url.URL* object and retried. If `key` still can't be found a *UndefinedTargetError* will be raised.

has_key(*key*)

Return whether the target with the key `key` exists in the project.

__contains__(*key*)

Return whether the target with the key `key` exists in the project.

create()

Create all dependencies for the project. Overwrite in subclasses.

This method should only be called once, otherwise you'll get lots of *RedefinedTargetWarning*s. But you can call `clear()` to remove all targets before calling *create()*. You can also use the method *recreate()* for that.

recreate()

Calls `clear()` and *create()* to recreate all project dependencies.

argparser()

Return an `argparse` parser for parsing the command line arguments. This can be overwritten in subclasses to add more arguments.

parseargs(*args=None*)

Use the parser returned by *argparser()* to parse the argument sequence `args`, modify `self` accordingly and return the result of the parsers `parse_args()` call.

get(*target*)

Get up-to-date output data from the target `target` (which must be an action registered with `self` (or the id of one). During the call the global variable `currentproject` will be set to `self`.

build(**targets*)

Rebuild all targets in `targets`. Items in `targets` must be actions registered with `self` (or their ids).

buildwithargs(*args=None*)

For calling make scripts from the command line. `args` defaults to `sys.argv`. Any positional arguments in the command line will be treated as target ids. If there are no positional arguments, a list of all registered *PhonyAction* objects will be output.

write(**texts*)

All screen output is done through this method. This makes it possible to redirect the output (e.g. to logfiles) in subclasses.

writeln(**texts*)

All screen output is done through this method. This makes it possible to redirect the output (e.g. to logfiles) in subclasses.

writeerror(**texts*)

Output an error.

warn(*warning*, *stacklevel*)

Issue a warning through the Python warnings framework

writestacklevel(*level*, **texts*)

Output `texts` indented `level` levels.

writestack(**texts*)

Output `texts` indented properly for the current nesting of action execution.

writestep(*action*, **texts*)

Output `texts` as the description of the data transformation done by the action `arction`.

writenote(*action*, **texts*)

Output `texts` as the note for the data transformation done by the action `action`.

writecreatedone()

Can be called at the end of an overwritten *create()* to report the number of registered targets.

writephonytargets()

Show a list of all *PhonyAction* objects in the project and their documentation.

findpaths(*target*, *source*)

Find dependency paths leading from the action `target` to the action `source`.

6.6 ll.daemon – Forking daemon processes

This module can be used on UNIX to fork a daemon process. It is based on Jürgen Hermann's Cookbook recipe.

An example script might look like this:

```
from ll import daemon

counter = daemon.Daemon(
   stdin="/dev/null",
   stdout="/tmp/daemon.log",
   stderr="/tmp/daemon.log",
   pidfile="/var/run/counter/counter.pid",
   user="nobody"
)

if __name__ == "__main__":
   if counter.service():
      import sys, os, time
      sys.stdout.write(f"Daemon started with pid {os.getpid()}\n")
      sys.stdout.write(f"Daemon stdout output\n")
      sys.stderr.write(f"Daemon stderr output\n")
      c = 0
      while True:
         sys.stdout.write(f"{c}: {time.ctime(time.time())}\n")
         sys.stdout.flush()
         c += 1
         time.sleep(1)
```

class ll.daemon.**Daemon**

Bases: object

The *Daemon* class provides methods for starting and stopping a daemon process as well as handling command line arguments.

__init__(*stdin='/dev/null'*, *stdout='/dev/null'*, *stderr='/dev/null'*, *pidfile=None*, *user=None*, *group=None*)

The stdin, stdout, and stderr arguments are file names that will be opened and be used to replace the standard file descriptors in sys.stdin, sys.stdout, and sys.stderr. These arguments are optional and default to "/dev/null". Note that stderr is opened unbuffered, so if it shares a file with stdout then interleaved output may not appear in the order that you expect.

pidfile must be the name of a file. *start()* will write the pid of the newly forked daemon to this file. *stop()* uses this file to kill the daemon.

user can be the name or uid of a user. *start()* will switch to this user for running the service. If user is None no user switching will be done.

In the same way group can be the name or gid of a group. *start()* will switch to this group.

openstreams()

Open the standard file descriptors stdin, stdout and stderr as specified in the constructor.

handlesighup(*signum*, *frame*)

Handle a SIG_HUP signal: Reopen standard file descriptors.

handlesigterm(*signum*, *frame*)

Handle a SIG_TERM signal: Remove the pid file and exit.

switchuser(*user*, *group*)

Switch the effective user and group. If `user` and `group` are both `None` nothing will be done. `user` and `group` can be an `int` (i.e. a user/group id) or `str` (a user/group name).

start()

Daemonize the running script. When this method returns the process is completely decoupled from the parent environment.

stop()

Send a `SIGTERM` signal to a running daemon. The pid of the daemon will be read from the pidfile specified in the constructor.

argparser()

Return an `argparse` parser for parsing the command line arguments. This can be overwritten in subclasses to add more arguments.

parseargs(*parser*, *args=None*)

Use the parser returned by *argparser()* to parse the argument sequence `args`, modify `self` accordingly and return the result of the parsers `parse_args()` call.

service(*args=None*)

Handle command line arguments and start or stop the daemon accordingly.

`args` must be a list of command line arguments (including the program name in `args[0]`). If `args` is `None` or unspecified `sys.argv` is used.

The return value is true when a starting option has been specified as the command line argument, i.e. if the daemon should be started.

The `argparse` arguments are available afterwards as `self.args`.

6.7 `ll.sisyphus` – Writing jobs with Python

`sisyphus` simplifies running Python stuff as jobs.

This can either be done under the direction of a cron daemon or a similar process runner, then `sisyphus` makes sure that there will be no more than one job of a certain name running at any given time.

Or `sisyphus` can be used as its own minimal cron daemon and can execute the job repeatedly.

A job has a maximum allowed runtime. If this maximum is exceeded, the job will kill itself. In addition to that, job execution can be logged and in case of job failure an email can be sent, a message can be posted to a Mattermost chat channel or an event can be emitted to a Sentry server.

To use this module, you must derive your own class from *Job*, implement the *execute()* method and then call the module level function *execute()* or *executewithargs()* with your job object (preferably in an `if __name__ == "__main__"` block).

Logs will (by default) be created in the `~/ll.sisyphus` directory. This can be changed by overwriting the appropriate methods in the subclass.

To execute a job, use the module level function *execute()* (or *executewithargs()* when you want to support command line arguments).

6.7.1 Example

The following example illustrates the use of this module:

```
import os
import urllib.request
from ll import sisyphus

class Fetch(sisyphus.Job):
   projectname = "ACME.FooBar"
   jobname = "Fetch"
   argdescription = "fetch http://www.python.org/ and save it to a local file"
   maxtime = 3 * 60

   def __init__(self):
      self.url = "http://www.python.org/"
      self.tmpname = f"Fetch_Tmp_{os.getpid()}.html"
      self.officialname = "Python.html"

   def execute(self):
      self.log(f"fetching data from {self.url!r}")
      data = urllib.request.urlopen(self.url).read()
      datasize = len(data)
      self.log(f"writing file {self.tmpname!r} ({datasize:,} bytes)")
      with open(self.tmpname, "wb") as f:
         f.write(data)
      self.log(f"renaming file {self.tmpname!r} to {self.officialname!r}")
      os.rename(self.tmpname, self.officialname)
      return f"cached {self.url!r} as {self.officialname!r} ({datasize:,} bytes)"

if __name__ == "__main__":
   sisyphus.executewithargs(Fetch())
```

You will find the log files for this job in `~/ll.sisyphus/ACME.FooBar/Fetch/`.

6.7.2 Result status of a job run

The method *`Job.execute()`* (which must be overwritten to implement the jobs main functionality) should return a one-line summary of what the job did (this is called a "successful run"). It can also return `None` to report that the job had nothing to do (this is called an "uneventful run").

Apart from "uneventful" and "successful" runs, the following results are possible:

"interrupted"
: The job failed with an `KeyboardInterrupt`.

"failed"
: The job failed with an exception (other than `KeyboardInterrupt`).

"timeout"
: The job ran longer than that the allowed maximum runtime.

6.7.3 Repeat mode

Normally sisyphus jobs run under the control of a cron daemon or similar process runner. In this mode the method *Job.execute()* is executed once and after that, execution of the Python script ends.

However it is possible to activate repeat mode with the class/instance attribute `repeat` (or the command line option *--repeat*). If `repeat` is true, execution of the job will be repeated indefinitely.

By default the next job run starts immediately after the end of the previous run, but it is possible to delay the next run. For this the class/instance attribute `nextrun` (or the command line option *--nextrun*) can be used. In its simplest form this is the number of seconds to wait until the next job run is started. It can also be a `datetime.timedelta` object that specifies the delay, or it can be a `datetime.datetime` object specifying the next job run. Furthermore `nextrun` can be callable (so it can be implemented as a method) and can return any of the types `int`, `float`, `datetime.timedelta` or `datetime.datetime`. And, if `Job.nextrun` is `None`, the job run will be repeated immediately.

6.7.4 Logging and tags

Logging itself is done by calling `log()`:

```
self.log(f"can't parse XML file {filename}")
```

This logs the argument without tagging the line.

It is possible to add tags to the logging call. This is done by accessing attributes of the `log` pseudo method. I.e. to add the tags `xml` and `warning` to a log call you can do the following:

```
self.log.xml.warning(f"can't parse XML file {filename}")
```

It's also possible to do this via `__getitem__` calls, i.e. the above can be written like this:

```
self.log['xml']['warning'](f"can't parse XML file {filename}")
```

`sisyphus` itself uses the following tags:

`sisyphus`
This tag will be added to all log lines produced by `sisyphus` itself.

`init`
This tag is used for the log lines output at the start of the job.

`report`
This tag will be added for all log messages related to sending the failure report email.

`result`
This tag is used for the final line written to the log files that shows a summary of what the job did (or why it failed).

`fail`
This tag is used in the result line if the job failed with an exception.

`errors`
This tag is used in the result line if the job ran to completion, but some exceptions where logged.

`ok`
This tag is used in the result line if the job ran to completion without any exceptions.

`kill`
This tag is used in the result line if the job was killed because it exceeded the maximum allowed runtime.

`info`
This tag is used for all other informational log messages output by `sisyphus` itself (like log file cleanup etc.).

6.7.5 Exceptions

When an exception object is passed to `self.log` the tag `exc` will be added to the log call automatically.

6.7.6 Delayed logs

If a log message has the tag `delay` it is considered a delayed message. Delayed messages will be buffered up until the first log message that isn't delayed is encountered (`sisyphuss` messages all are delayed). Then all buffered messages will be output. If only delayed messages are output during the complete job run, only the result of the job run will be output. If this output is `None` nothing will be output. This means that you will get no log entries until something "interesting" happens.

6.7.7 Log files

By default logging is done to the log file (whose name changes from run to run as it includes the start time of the job).

However logging to `stdout` and `stderr` can also be activated.

Logfiles for uneventful runs wil be deleted after the run.

Multiple links will be created that automatically point to the last log file. The "current" link (by default named `current.sisyphuslog`) will always point to the log file of the currently running job. If no job is running, but the last run was eventful, it will point to the newest log file. If the last run was uneventful the link will point to a nonexistent log file (whose name can be used to determine the date of the last run).

The following links will be created at the end of the job run and will only start to point to non-existent files when the log files they point to get cleaned up:

- The "last successful" link (by default named `last_successful.sisyphuslog`) will always point to the last successful job run,
- `last_failed.sisyphuslog` points to the last failed run,
- `last_interrupted.sisyphuslog` points to the last interrupted run and
- `last_timeout.sisyphuslog` points to the last run that timed out.

6.7.8 Email

It is possible to send an email when a job fails. For this, the options *`--fromemail`*, *`--toemail`* and *`--smtphost`* (or the appropriate class attributes) have to be set. If the job terminates because of an exception or exceeds its maximum runtime (and the option *`--noisykills`* is set) or any of the calls to `log()` include the tag `email` or `external`, an email will be sent. This email includes the last 10 logging calls and the final exception (if there is any) in plain text and HTML format as well as as a JSON attachment.

6.7.9 Mattermost

It is possible to send log entries to a Mattermost chat channel. For this the options *`--mattermost_url`*, *`--mattermost_channel`* and *`--mattermost_token`* (or the appropriate class attributes) must be specified. All log entries including the tag `mattermost` or `external`, as well as all exceptions that abort the job will be sent to the Mattermost channel.

6.7.10 Sentry

It is possible to send log entries to a Sentry server. For this the option *--sentry_dsn* (or the appropriate class attribute) must be specified. All log entries including the tag `sentry` or `external`, as well as all exceptions that abort the job will be sent to the Sentry server.

If the logging call includes any of the tags `fatal`, `error`, `warning`, `info`, `debug` this will be used as the event level. If the log argument is an exception the event level will be `fatal`. Otherwise it wil default to `info`.

All tags will be converted to Sentry tags like this: A sisyphus tag `foo` will be converted into a Sentry tag `sisypus.tag.foo` with a value of `true`.

Active tasks will be converted into Sentry breadcrumbs (See the methods *task()* and *tasks()* for more info).

6.7.11 Health checks

When a job is started with the option *--healthcheck*, instead of running the job normally a health check is done. This bypasses the normal mechanism that prevents multiple instances of the job from running (i.e. you can have a normal job execution and a health check running in parallel).

If the job is healthy this will exit with an exit status of 0, otherwise it will exit with an exit status of 1 and an error message on `stdout` stating the reason why the job is considered unhealthy. There are three possible scenarios for this:

1. The job has never been run.
2. The last run has ended with an error.
3. The last run was too long ago.

To configure how scenario 3 is handled the class/instance attribute `maxhealthcheckage` (or the command line option *--maxhealthcheckage*) can be used. In its simplest form this is a number of seconds or a `datetime.timedelta` object. A job run that is older that this value triggers scenario 3. `maxhealthcheckage` can be also be a `datetime.datetime` object specifying the cut-off date.

Furthermore `maxhealthcheckage` can be callable (so it can be implemented as a method) and can return any of the types `int`, `float`, `datetime.timedelta` or `datetime.datetime`.

And if `Job.maxhealthcheckage` is `None`, scenario 3 will never trigger.

6.7.12 Requirements

To reliably stop the job after the allowed maximum runtime, `sisyphus` forks the process and kills the child process after the maximum runtime is expired (via `os.fork()` and `signal.signal()`). This won't work on Windows. So on Windows the job will always run to completion without being killed after the maximum runtime.

To make sure that only one job instance runs concurrently, `sisyphus` uses `fcntl` to create an exclusive lock on the file of the running script. This won't work on Windows either. So on Windows you might have multiple running instances of the job.

`sisyphus` uses the module `setproctitle` to change the process title during various phases of running the job. If `setproctitle` is not available the process title will not be changed.

If the module `psutil` is available it will be used to kill the child process and any of its own child processes after the maximum runtime of the job is exceeded. If `psutil` isn't available just the child process will be killed (which is no problem as long as the child process doesn't spawn any other processes).

If logging to Mattermost is used, `requests` has to be installed.

If logging to Sentry is used, `sentry_sdk` has to be installed.

For compressing the log files one of the modules `gzip`, `bz2` or `lzma` is required (which might not be part of your Python installation).

6.7.13 Module documentation

class ll.sisyphus.**Status**

Bases: IntEnum

The result status of a job run.

Possible values are:

- UNEVENTFUL,
- SUCCESSFUL,
- FAILED,
- INTERRUPTED,
- TIMEOUT.

__format__(*format_spec*, /)

Convert to a string according to format_spec.

class ll.sisyphus.**Process**

Bases: Enum

The type of a running sisyphus process.

Possible values are:

- SOLO (when in non-forking mode),
- PARENT (the parent process in forking mode),
- CHILD (the child process in forking mode).

class ll.sisyphus.**Job**

Bases: object

A Job object executes a task (either once or repeatedly).

To use this class, derive your own class from it and overwrite the *execute()* method.

The job can be configured in three ways: By class attributes in the *Job* subclass, by attributes of the *Job* instance (e.g. set in __init__()) and by command line arguments (if *executewithargs()* is used). The following command line arguments are supported (the name of the attribute is the same as the long command line argument name):

-p <projectname>, **--projectname** <projectname>

The name of the project this job belongs to. This might be a dot-separated hierarchical project name (e.g. including customer names or similar stuff).

-j <jobname>, **--jobname** <jobname>

The name of the job itself (defaulting to the name of the class if none is given).

--identifier <identifier>

An additional identifier that will be added to the failure report email.

--fromemail <emailadress>

The sender email address for the failure report email.

This email will only be sent if the options *--fromemail*, *--toemail* and *--smtphost* are set (and any error or output to the email log occured, which only happens when the log entry has the tag email or external, or if it is an exception that aborts the job run).

--toemail <emailadress>

An email address where an email will be sent in case of a failure.

--smtphost <servername>

The SMTP server to be used for sending the failure report email.

--smtpport <integer>

The port number used for the connection to the SMTP server.

--smtpuser <username>

The user name used to log into the SMTP server. (Login will only be done if both *--smtpuser* and *--smtppassword* are given)

--smtppassword <password>

The password used to log into the SMTP server.

--mattermost_url <url>

The URL where log entries can be posted to a Mattermost chat. For example:

```
https://mattermost.example.org/api/v4/posts
```

A log entry will only be posted to the Mattermost chat channel if the options *--mattermost_url*, *--mattermost_channel* and *--mattermost_token* are set (and the log entry has the tag `mattermost` or `external` or is an exception that aborts the job run).

Note that using this feature requires `requests`.

--mattermost_channel <id>

The channel id of the Mattermost chat channel where log entries should be posted. For example:

```
4cnszmopr3ntjexi4qmx499inc
```

--mattermost_token <auth>

The "Personal Access Token" used for authorizing the post with the Mattermost server. For example:

```
9xuqwrwgstrb3mzrxb83nb357a
```

--sentry_dsn <dsn>

Sentry DSN for logging to a Sentry server. Something like:

```
https://examplePublicKey@o0.ingest.sentry.io/0
```

--sentry_environment <environment>

Environment reported to Sentry.

--sentry_release <release>

Release reported to Sentry.

--sentry_debug <flag>

Activates/deactivates Sentry debug mode.

(Allowed <flag> values are `false`, `no`, `0`, `true`, `yes` or `1`)

A log entry will only be sent to Sentry if the options *--sentry_dsn* is are set (and the log entry has the tag `sentry` or `external`, or is an exception that aborts the job run).

-m <seconds>, **--maxtime** <seconds>

Maximum allowed runtime for the job (as the number of seconds). If the job runs longer than that it will kill itself.

(The instance attribute will always be converted to the type `datetime.timedelta`)

--fork <flag>

Forks the process and does the work in the child process. The parent process is responsible for monitoring the maximum runtime (this is the default). In non-forking mode the single process does both the work and the runtime monitoring.

(Allowed <flag> values are `false`, `no`, `0`, `true`, `yes` or `1`)

--noisykills <flag>

Should a message be printed/a failure email be sent when the maximum runtime is exceeded?

(Allowed <flag> values are `false`, `no`, `0`, `true`, `yes` or `1`)

--exit_on_error <flag>

End job execution even in repeat mode when an exception is thrown?

(Allowed <flag> values are `false`, `no`, `0`, `true`, `yes` or `1`)

-n <flag>, **--notify** <flag>

Should a notification be issued to the OS X Notification center? (done via terminal-notifier).

(Allowed <flag> values are `false`, `no`, `0`, `true`, `yes` or `1`)

-r <flag>, **--repeat** <flag>

Should job execution be repeated indefinitely?

(This means that the job basically functions as its own cron daemon).

(Allowed <flag> values are `false`, `no`, `0`, `true`, `yes` or `1`)

--nextrun <seconds>

How many seconds should we wait after a job run before the next run gets started (only when *--repeat* is set)?

The class/instance attribute can also be a callable (i.e. it's possible to implement this as a method). Also `datetime.datetime` is supported and specifies the start date for the next job run.

--healthcheck <flag>

Instead of normally executing the job, run a health check instead.

(Allowed <flag> values are `false`, `no`, `0`, `true`, `yes` or `1`)

--maxhealthcheckage <seconds>

If the last uneventful or successful job run is older then this number of seconds, consider the job to be unhealthy.

-f <flag>, **--log2file** <flag>

Should a logfile be written at all?

(Allowed <flag> values are `false`, `no`, `0`, `true`, `yes` or `1`)

--formatlogline <format>

An UL4 template for formatting each line in the logfile. Available variables are `time` (current time), `starttime` (start time of the job), `tags` (list of tags for the line) and `line` (the log line itself).

--keepfilelogs <days>

The number of days the logfiles are kept. Old logfiles (i.e. all files in the same directory as the current logfile that are more than `keepfilelogs` days old) will be removed at the end of the job.

(The instance attribute will always be converted to the type `datetime.timedelta`)

--compressfilelogs <days>

The number of days after which log files are compressed (if they aren't deleted via *--keepfilelogs*).

(The instance attribute will always be converted to the type `datetime.timedelta`)

--compressmode <mode>

How to compress the logfiles. Possible values are: "gzip", "bzip2" and "lzma". The default is "bzip2".

--encoding <encodingname>

The encoding to be used for the logfile. The default is "utf-8".

--errors <errorhandlingname>

Encoding error handler name (goes with *--encoding*). The default is "strict".

--maxemailerrors <integer>

This options limits the number of exceptions and errors messages that will get attached to the failure email. The default is 10.

--proctitle <flag>

When this options is specified, the process title will be modified during execution of the job, so that the **ps** command shows what the processes are doing. The default is True. (This requires setproctitle.)

(Allowed <flag> values are false, no, 0, true, yes or 1)

Command line arguments take precedence over instance attributes (if *executewithargs()* is used) and those take precedence over class attributes.

Furthermore the following class attribute can be set to customize the help message:

argdescription

Description for the help message of the command line argument parser.

basedir() → Path

Return the base directory where all log files will be kept.

The path must be absolute.

logfilename() → Path | None

Return the filename of the logfile for this job.

The value must by an absolute pathlib.Path object (or None to disable creating the logfile).

currentloglinkname() → Path | None

Return the filename of the link to the currently active logfile.

The value must by an absolute pathlib.Path object (or None to disable creating the link).

lastsuccessfulloglinkname() → Path | None

Return the filename of the link that points to the logfile of the last successful run of the job.

The value must by an absolute pathlib.Path object (or None to disable creating the link).

lastfailedloglinkname() → Path | None

Return the filename of the link that points to the logfile of the last failed run of the job.

The value must by an absolute pathlib.Path object (or None to disable creating the link).

lastinterruptedloglinkname() → Path | None

Return the filename of the link that points to the logfile of the last interrupted run of the job.

The value must by an absolute pathlib.Path object (or None to disable creating the link).

lasttimeoutloglinkname() → Path | None

Return the filename of the link that points to the logfile of the last run of the job with a timeout.

The value must by an absolute pathlib.Path object (or None to disable creating the link).

healthfilename() → Path

Return the filename where the health of the last job run is stored.

The value must by an absolute pathlib.Path object and may not be None.

emailfilename(*process: Process | None = None*) → Path

Return the filename where the parent and child process can log message that should be part of the email report.

The value must by an absolute `pathlib.Path` object and may not be `None`.

execute() → str | None

Execute the job once.

Overwrite in subclasses to implement your job functionality.

The return value is a one line summary of what the job did.

When this method returns `None` instead this tells the job machinery that the run of the job was uneventful and that the logfile can be deleted.

healthcheck() → str | None

Called in parallel to a running job to check whether the job is healthy.

Returns `None` if everything is ok, or an error message otherwise.

argparser() → ArgumentParser

Return an `argparse` parser for parsing the command line arguments. This can be overwritten in subclasses to add more arguments.

parseargs(*args: List[str] | None*) → Namespace

Use the parser returned by `argparser()` to parse the argument sequence `args`, modify `self` accordingly and return the result of the parsers `parse_args()` call.

task(*type: str | None = None, name: str | None = None, index: int | None = None, count: int | None = None, **data*) → *Task*

`task()` is a context manager and can be used to specify subtasks.

Arguments have the following meaning:

type
[`str` or `None`]
The type of the task.

name
[`str` or `None`]
The name of the task.

index
[`int` or `None`]
If this task is one in a sequence of similar tasks, `index` should be the index of this task, i.e. the first task of this type has `index==0`, the second one `index==1` etc.

count
[`int` or `None`]
If this task is one in a sequence of similar tasks and the total number of tasks is known, `count` should be the total number of tasks.

****data**
Additional information about the task. This will be added to the Sentry breadcrumbs when logging to Sentry. Otherwise this is ignored.

tasks(*iterable: Iterable[T], type: str | None | Callable[[...], str | None] = None, name: str | None | Callable[[...], str | None] = None, data: dict | None | Callable[[...], dict | None] = None*) → Generator[T, None, None]

`tasks()` iterates through `iterable` and calls `task()` for each item. `index` and `count` will be passed to `task()` automatically. `type`, `name` and `data` will be used for the type, name and additional data of the task. They can either be constants (in which case they will be passed as is) or callables (in which case they will be called with the item to get the type/name/data).

Example:

```
import sys, operator

items = list(sys.modules.items())
for (name, module) in self.tasks(items, "module", operator.itemgetter(0)):
   self.log(f"module is {module}")
```

The log output will look something like the following:

```
[2019-05-06 18:52:31.366810]=[t+0:00:00.263849] :: parent 19448 :: {sisyphus}
↪{init} >> /Users/walter/x/gurk.py (max time 0:01:40)
[2019-05-06 18:52:31.367831]=[t+0:00:00.264870] :: parent 19448 :: {sisyphus}
↪{init} >> logging to <stdout>, /Users/walter/ll.sisyphus/Test/Job/2019-05-
↪06-18-52-31-102961.sisyphuslog
[2019-05-06 18:52:31.371690]=[t+0:00:00.268729] :: [1] child 19451 ::
↪{sisyphus}{init} >> forked worker child
[2019-05-06 18:52:31.376598]=[t+0:00:00.273637] :: [1] child 19451 :: [1/
↪226] module sys >> module is <module 'sys' (built-in)>
[2019-05-06 18:52:31.378561]=[t+0:00:00.275600] :: [1] child 19451 :: [2/
↪226] module builtins >> module is <module 'builtins' (built-in)>
[2019-05-06 18:52:31.380381]=[t+0:00:00.277420] :: [1] child 19451 :: [3/
↪226] module _frozen_importlib >> module is <module 'importlib._bootstrap'␣
↪(frozen)>
[2019-05-06 18:52:31.382248]=[t+0:00:00.279287] :: [1] child 19451 :: [4/
↪226] module _imp >> module is <module '_imp' (built-in)>
[2019-05-06 18:52:31.384064]=[t+0:00:00.281103] :: [1] child 19451 :: [5/
↪226] module _thread >> module is <module '_thread' (built-in)>
[2019-05-06 18:52:31.386047]=[t+0:00:00.283086] :: [1] child 19451 :: [6/
↪226] module _warnings >> module is <module '_warnings' (built-in)>
[2019-05-06 18:52:31.388009]=[t+0:00:00.285048] :: [1] child 19451 :: [7/
↪226] module _weakref >> module is <module '_weakref' (built-in)>
[...]
[2019-05-06 18:52:31.847315]=[t+0:00:00.744354] :: [1] child 19451 ::
↪{sisyphus}{result}{ok} >> done
```

class ll.sisyphus.**Task**

Bases: object

A subtask of a *Job*.

__init__(*job:* Job, *type:* *str* | *None* = *None*, *name:* *str* | *None* = *None*, *index:* *int* | *None* = *None*, *count:* *int* | *None* = *None*, ***data*)

Create a Task object. For the meaning of the parameters see *Job.task()*.

class ll.sisyphus.**Tag**

Bases: object

A Tag object can be used to call a function with an additional list of tags. Tags can be added via __getattr__() or __getitem__() calls.

class ll.sisyphus.**Logger**

Bases: object

A *Logger* is called by the *Job* for each logging event.

name() → str | None

A name for the logger (using in reporting)

log(*timestamp:* *datetime*, *tags:* *Tuple[str, ...]*, *tasks:* *List[*Task*]*, *text:* *str*) → None

Called by the *Job* when a log entry has to be made.

Arguments have the following meaning:

timestamp
[datetime.datetime]
The moment when the logging call was made.

tags
[List of strings]
The tags that were part of the logging call. For example for the logging call:

```
self.log.xml.warning("Skipping foobar")
```

the list of tags is:

```
["xml", "warning"]
```

tasks
[List of *Task* objects]
The currently active stack of *Task* objects.

text
[Any object]
The log text. This can be any object. If it's not a string it will be converted to a string via `pprint.pformat()` (or `traceback.format_exception()` if it's an exception)

taskstart(*tasks: List[Task]*) → None

Called by the *Job* when a new subtask has been started.

`tasks` is the stack of currently active tasks (so `tasks[-1]` is the task that has been started).

taskend(*tasks: List[Task]*) → None

Called by the *Job* when a subtask is about to end.

`tasks` is the stack of currently active tasks (so `tasks[-1]` is the task that's about to end).

close(*status: Status*) → bool

Called by the *Job* when job execution has finished.

`status` (a *Status*) is the result status of the job run.

Return whether the logfile has been closed. (All normal loggers will close except `stdout` and `stderr` loggers).

class ll.sisyphus.**StreamLogger**

Bases: *Logger*

Logger that writes logging events into an open file-like object. Is is used for logging to `stdout` and `stderr`.

class ll.sisyphus.**FileLogger**

Bases: *StreamLogger*

Logger that writes logging events into a file specified via an *URL* object. This is used for logging to the standard log file.

class ll.sisyphus.**LinkLogger**

Bases: *Logger*

Baseclass of all loggers that handle links to the log file.

class ll.sisyphus.**CurrentLinkLogger**

Bases: *LinkLogger*

Logger that handles the link to the current log file.

class ll.sisyphus.**LastStatusLinkLogger**

Bases: *LinkLogger*

Logger that handles the link to the log file for a specific job status.

class ll.sisyphus.**EmailLogger**

Bases: *Logger*

Logger that handles sending an email report of the job run.

class ll.sisyphus.**MattermostLogger**

Bases: *Logger*

Logger that logs messages to a Mattermost chat channel.

class ll.sisyphus.**SentryLogger**

Bases: *Logger*

Logger that logs messages and exceptions to Sentry.

ll.sisyphus.**execute**(*job:* Job) → None

Execute the job `job` once or repeatedly.

ll.sisyphus.**executewithargs**(*job:* Job, *args:* *List[str] | None = None*) → None

Execute the job `job` once or repeatedly with command line arguments.

`args` are the command line arguments (`None` results in `sys.argv` being used).

6.8 ll.color – RGB colors and color model conversion

`ll.color` provides classes and functions for handling RGBA colors.

class ll.color.**Color**

Bases: `tuple`

A *Color* object represents a color with 8-bit red, green and blue components and opacity.

static **__new__**(*cls*, *r:* *int = 0*, *g:* *int = 0*, *b:* *int = 0*, *a:* *int = 255*)

Create a `Color` with the 8 bit red, green, blue and alpha components `r`, `g`, `b` and `a`. Values will be clipped to the range [0; 255].

classmethod **fromcss**(*s:* *str*) → *Color*

Create a *Color* object from the CSS color string `s`. All formats from CSS2 are supported (i.e. `'#xxx'`, `'#xxxxxx'`, `rgb(r, g, b)`, `rgb(r%, g%, b%)`, `rgba(r, g, b, a)`, `rgba(r%, g%, b%, a)` and color names like `'red'`).

classmethod **fromrgb**(*r:* *int | float*, *g:* *int | float*, *b:* *int | float*, *a:* *int | float = 1.0*) → *Color*

Create a *Color* object from the red, green, blue and alpha values `r`, `g`, `b` and `a`. All values will be clipped to the range [0; 1].

classmethod **fromhsv**(*h:* *int | float*, *s:* *int | float*, *v:* *int | float*, *a:* *int | float = 1.0*) → *Color*

Create a *Color* object from the hue, saturation and value values `h`, `s` and `v` and the alpha value `a`. The hue value will be used modulo 1.0, saturation, value and alpha will be clipped to the range [0; 1].

classmethod **fromhls**(*h:* *int | float*, *l:* *int | float*, *s:* *int | float*, *a:* *int | float = 1.0*) → *Color*

Create a *Color* object from the hue, luminance and saturation values `h`, `l` and `s` and the alpha value `a`. The hue value will be used modulo 1.0, luminance, saturation and alpha will be clipped to the range [0; 1].

__str__() → str

`self` formatted as a CSS color string.

r() → int

The red value as an int between 0 and 255.

g() → int

The green value as an int between 0 and 255.

b() → int

The blue value as an int between 0 and 255.

a() → int

The alpha value as an int between 0 and 255.

rgb() → Tuple[float, float, float]

The red, green and blue value as a float tuple with values between 0.0 and 1.0.

rgba() → Tuple[float, float, float, float]

The red, green, blue and alpha value as a float tuple with values between 0.0 and 1.0.

hsv() → Tuple[float, float, float]

`self` as a HSV tuple ("hue", "saturation", "value"). All three values are between 0.0 and 1.0.

hsva() → Tuple[float, float, float, float]

`self` as a HSV+alpha tuple ("hue", "saturation", "value", "alpha"). All four values are between 0.0 and 1.0.

hls() → Tuple[float, float, float]

`self` as a HLS tuple ("hue, luminance, saturation"). All three values are between 0.0 and 1.0.

hlsa() → Tuple[float, float, float, float]

`self` as a HLS+alpha tuple ("hue, luminance, saturation, alpha"). All four values are between 0.0 and 1.0.

hue() → float

The hue value from *`hls()`*.

light() → float

The lightness value from *`hls()`*.

sat() → float

The saturation value from *`hls()`*.

lum() → float

Luminance according to sRGB:

```
(0.2126*r + 0.7152*g + 0.0722*b)/255
```

withhue(*hue: int | float*) → *Color*

Return a new color with the HLS hue replaced by `hue`.

withlight(*light: int | float*) → *Color*

Return a new color with the HLS lightness replaced by `light`

withsat(*sat: int | float*) → *Color*

Return a new color with the HLS saturation replaced by `sat`

withlum(*lum: int | float*) → *Color*

Return a copy of `self` where the luminance has been replace with `lum`.

witha(*a: int*) → *Color*

Return a copy of `self` with the alpha value replaced with `a`.

abslight(*f: int | float*) → *Color*

Return a copy of `self` with `f` added to the HLS lightness.

rellight(*f: int | float*) → *Color*

Return a copy of `self` where the lightness has been modified: If `f` is positive the lightness will be increased, with `f==1` giving a lightness of 1. If `f` is negative, the lightness will be decreased with `f==-1` giving a lightness of 0. `f==0` will leave the lightness unchanged.

abslum(*f: int | float*) → *Color*

Return a copy of `self` where `f` has been added to the lum value.

rellum(*f: int | float*) → *Color*

Return a copy of `self` where the luminance has been modified: If `f` is positive the luminance will be increased, with `f==1` giving a luminance of 1. If `f` is negative, the luminance will be decreased with `f==-1` giving a luminance of 0. `f==0` will leave the luminance unchanged. All other values return a linear interpolation.

combine(*r: int | float = None, g: int | float = None, b: int | float = None, a: int | float = None*) → *Color*

Return a copy of `self` with some of its components replaced by the arguments. If a component is nont passed (or the value `None` is given) the component will be unchanged in the resulting color.

invert(*f: int | float = 1.0*) → *Color*

Return an inverted version of `self`, i.e. for each color `c` the following prints `True` three times:

```
<?print c.invert().r() == 255 - c.r()?>
<?print c.invert().g() == 255 - c.g()?>
<?print c.invert().b() == 255 - c.b()?>
```

`f` specifies the amount of inversion, with 1 returning a complete inversion, and 0 returning the original color. Values between 0 and 1 return an interpolation of both extreme values. (And 0.5 always returns medium grey).

__mod__(*other: Color*) → *Color*

Blends `self` with the background color `other` according to the CSS specification

`ll.color.`**css**(*value: str, default: str | None = <object object>, /*) → *Color*

Create a *`Color`* object from the CSS color string `value` via *`Color.fromcss()`*.

If `value` is no valid CSS color string and `default` is given, return `default` instead.

`ll.color.`**dist**(*c1: Color, c2: Color, /*) → *float*

Return the distance between two colors.

`ll.color.`**multiply**(*c1: Color, c2: Color, /*) → *Color*

Multiplies the colors `c1` and `c2`.

`ll.color.`**screen**(*c1: Color, c2: Color, /*) → *Color*

Does a negative multiplication of the colors `c1` and `c2`.

`ll.color.`**mix**(**args*) → *Color*

Calculates a weighted mix of the colors from `args`. Items in `args` are either colors or weights. The following example mixes two parts black with one part white:

```
>>> from ll import color
>>> color.mix(2, color.black, 1, color.white)
Color(0x55, 0x55, 0x55)
```

6.9 ll.misc – Utility functions and classes

ll.misc contains various utility functions and classes used by the other LivingLogic modules and packages.

ll.misc.**item**(*iterable*, *index*, /, *default=None*)

Returns the index'th item from the iterable. index may be negative to count from the end. E.g. 0 returns the first item produced by the iterator, 1 the second, -1 the last one etc. If index is negative the iterator will be completely exhausted, if it's positive it will be exhausted up to the index'th item. If the iterator doesn't produce that many items default will be returned.

index may also be an iterable of indexes, in which case item() will be applied recursively, i.e. item(["foo", "bar"], (1, -1)) returns 'r'.

ll.misc.**first**(*iterable*, /, *default=None*)

Return the first item from the iterable. If the iterator doesn't produce any items default will be returned.

ll.misc.**last**(*iterable*, /, *default=None*)

Return the last item from the iterable. If the iterator doesn't produce any items default will be returned.

ll.misc.**count**(*iterable*, /)

Count the number of items produced by the iterable. Calling this function will exhaust the iterator.

ll.misc.**isfirst**(*iterable*, /)

Iterate through items of the iterable and give information about whether the item is the first in the iterable:

```
>>> list(misc.isfirst("foo"))
[(True, 'f'), (False, 'o'), (False, 'o')]
```

ll.misc.**islast**(*iterable*, /)

Iterate through items of the iterable and give information about whether the item is the last in the iterable:

```
>>> list(misc.islast("foo"))
[(False, 'f'), (False, 'o'), (True, 'o')]
```

ll.misc.**isfirstlast**(*iterable*, /)

Iterate through items of the iterable and give information about whether the item is the first and/or last in the iterable:

```
>>> list(misc.isfirstlast("foo"))
[(True, False, 'f'), (False, False, 'o'), (False, True, 'o')]
```

ll.misc.**notimplemented**(*function*)

A decorator that raises NotImplementedError when the method is called.

This saves you the trouble of formatting the error message yourself for each implementation.

ll.misc.**withdoc**(*doc*)

A decorator that adds a docstring to the function it decorates.

This can be useful if the docstring is not static, and adding it afterwards is not possible.

class ll.misc.**Enum**

Bases: Enum

Subclass of enum.Enum where class and instance repr() output include the module and fully qualified class name.

class ll.misc.**IntEnum**

Bases: IntEnum

Subclass of enum.IntEnum where class and instance repr() output includes the module and fully qualified class name.

__format__(*format_spec*, /)

Convert to a string according to format_spec.

class ll.misc.**propclass**

Bases: property

propclass provides an alternate way to define properties.

Subclassing propclass and defining methods __get__(), __set__() and __delete__() will automatically generate the appropriate property:

```
class name(misc.propclass):
   """
   The name property
   """
   def __get__(self):
      return self._name
   def __set__(self, name):
      self._name = name.lower()
   def __delete__(self):
      self._name = None
```

ll.misc.**format_class**(*obj*)

Format the name of the class of obj.

Example:

```
>>> misc.format_class(42)
'int'
>>> misc.format_class(open('README.rst', 'rb'))
'_io.BufferedReader'
```

ll.misc.**format_exception**(*exc*)

Format an exception object.

Example:

```
>>> misc.format_exception(ValueError("bad value"))
'ValueError: bad value'
```

ll.misc.**exception_chain**(*exc*)

Traverses the chain of exceptions. This is a generator.

class ll.misc.**Pool**

Bases: object

A Pool object can be used as an inheritable alternative to modules. The attributes of a module can be put into a pool and each pool can have base pools where lookup continues if an attribute can't be found.

register(*object*)

Register object in the pool. object can be a module, a dictionary or a Pool objects (with registers the pool as a base pool). If object is a module and has an attribute __bases__ (being a sequence of other modules) this attribute will be used to initialize selfs base pool.

clear()

Make self empty.

clone()

Return a copy of self.

ll.misc.**iterone**(*item*)

Return an iterator that will produce one item: item.

class ll.misc.**Iterator**

Bases: object

Iterator adds __getitem__() support to an iterator. This is done by calling *item()* internally.

get(*index*, *default=None*)

Return the index'th item from the iterator (or default if there's no such item).

class ll.misc.**Queue**

Bases: object

Queue provides FIFO queues: The method *write()* writes to the queue and the method *read()* read from the other end of the queue and remove the characters read.

write(*chars*)

Write the string chars to the buffer.

read(*size=-1*)

Read up to size character from the buffer (or all if size is negative). Those characters will be removed from the buffer.

class ll.misc.**Const**

Bases: object

This class can be used for singleton constants.

class ll.misc.**FlagAction**

Bases: Action

FlagAction can be use with argparse for options that represent flags. An options can have a value like yes or no for the corresponding boolean value, or if the value is omitted it is the inverted default value (i.e. specifying the option toggles it).

ll.misc.**tokenizepi**(*string*)

Tokenize the string object string according to the processing instructions in the string. tokenize() will generate tuples with the first item being the processing instruction target and the second being the PI data. "Text" content (i.e. anything other than PIs) will be returned as (None, data).

ll.misc.**itersplitat**(*string*, *positions*)

Split string at the positions specified in positions.

For example:

```
>>> from ll import misc
>>> import datetime
>>> datetime.datetime(*map(int, misc.itersplitat("20090609172345", (4, 6, 8, 10,␣
↪12))))
datetime.datetime(2009, 6, 9, 17, 23, 45)
```

This is a generator.

ll.misc.**parse_header**(*line*)

Parse a Content-type like header.

Return the main content-type and a dictionary of options.

ll.misc.**module**(*source*, *filename='unnamed.py'*, *name=None*)

Create a module from the Python source code source. filename will be used as the filename for the module and name as the module name (defaulting to the filename part of filename).

ll.misc.**javaexpr**(*obj*)

Return a Java expression for the object obj.

Example:

```
>>> print(misc.javaexpr([1, 2, 3]))
java.util.Arrays.asList(1, 2, 3)
```

class ll.misc.**SysInfo**

Bases: object

A SysInfo object contains information about the host, user, python version and script. Available attributes are host_name, host_fqdn, host_ip, host_sysname, host_nodename, host_release, host_version, host_machine, user_name, user_uid, user_gid, user_gecos, user_dir, user_shell, python_executable, python_version, pid, script_name, short_script_name and script_url.

SysInfo object also support a mimimal dictionary interface (i.e. __getitem__() and __iter__()).

One module global instance named sysinfo is created at module import time.

class ll.misc.**monthdelta**

Bases: object

monthdelta objects can be used to add months/years to a datetime.datetime or datetime.date object. If the resulting day falls out of the range of valid days for the target month, the last day for the target month will be used instead:

```
>>> import datetime
>>> from ll import misc
>>> datetime.date(2000, 1, 31) + misc.monthdelta(1)
datetime.date(2000, 2, 29)
```

exception ll.misc.**Timeout**

Bases: Exception

Exception that is raised when a timeout in *timeout()* occurs.

ll.misc.**timeout**(*seconds*)

A context manager that limits the runtime of the wrapped code.

As this uses signal, this won't with threads and only on UNIX.

ll.misc.**notifystart**()

Notify OS X of the start of a process by removing the previous notification.

ll.misc.**notifyfinish**(*title*, *subtitle*, *message*)

Notify OS X of the end of a process.

ll.misc.**prettycsv**(*rows*, *padding=' '*)

Format table rows.

rows must be a list of lists of strings (e.g. as produced by the csv module). padding is the padding between columns.

prettycsv() is a generator.

ll.misc.**jsmin**(*input*)

Minimizes the Javascript source input.

6.10 `ll.pysql` – Database import script

6.10.1 Overview

The module/script `pysql` can be used to import data into one or more Oracle or Postgres databases. It reads `pysql` files which are an extension of normal Oracle or Postgres SQL files.

A PySQL file can contain different types of commands.

SQL commands

A PySQL file may contain normal SQL commands. For the `pysql` script to be able to execute these commands they must be terminated with a comment line `-- @@@`. `pysql` will prepare the command for execution and execute it. Any exception that is raised as a result of executing the command will stop the script and be reported. This is in contrast to how Oracle's `sqlplus` executes SQL commands. `sqlplus` would continue after an error and exit with status code 0 even if there were errors. (For Oracle `pysql` can also explicitely ignore any exception raised by commands by specifying a different exception handling mode.)

A PySQL file that only contains SQL commands is still a valid Oracle or Postgres SQL file, so it still can be executed via `sqlplus` or `psql`.

Literal Python blocks

A literal Python block starts with a line that only contains `#>>>` and ends with a line that only contains `#<<<`. Python code within the block gets executed when the block is encountered. The following objects are available within the block as global variables:

`connection`
The active database connection (or `None` if there is no active database connection).

`DB_TYPE_CLOB`
`cx_Oracle.DB_TYPE_CLOB`, i.e. `cx_Oracle`s type for `CLOB` parameters;

`DB_TYPE_NCLOB`
`cx_Oracle.DB_TYPE_NCLOB`, i.e. `cx_Oracle`s type for `NCLOB` parameters;

`DB_TYPE_BLOB`
`cx_Oracle.DB_TYPE_BLOB`, i.e. `cx_Oracle`s type for `BLOB` parameters;

`sqlexpr`
Can be used to specify that an argument for a *`procedure`* should be an SQL expression instead of a Python value or a *`var`* object;

`datetime`
Python's datetime module;

Furthermore all PySQL commands (see below) are available.

Variables that get set within a literal Python block will be available (and retain their value) in subsequent literal Python blocks or other PySQL commands.

PySQL commands

A PySQL file may also contain PySQL commands. A PySQL command looks and behaves like a Python function call. This function call must either be contained in a single line (i.e. start with `name(` and end with `)` or it must start with a line that only contains `name(` and end at a line that only contains `)`. (`name` must be the name of a PySQL command).

The following commands are available:

include
: Include another PySQL file;

connect
: Connect to a database;

disconnect
: Disconnect from the active database connection;

procedure
: Call a procedure in the database (and handle OUT parameters via *var* objects);

sql
: Execute an SQL statement in the database (and handle OUT parameter via *var* objects);

literalsql
: Execute an SQL statement in the database (this is what SQL commands get converted to);

commit
: Commit the transaction in the active database connection;

rollback
: Roll back the transaction in the active database connection;

literalpy
: Execute Python code (this is what literal Python blocks get converted to);

setvar
: Set a variable;

unsetvar
: Delete a variable;

raise_exceptions
: Set the exception handling mode;

push_raise_exceptions
: Temporarily modifies the exception handling mode;

pop_raise_exceptions
: Reverts to the previously active exception handling mode;

check_errors
: Checks whether there are invalid database objects;

scp
: Create a file on a remote host via **scp**;

file
: Create a file on the local machine;

reset_sequence
: Resets a database sequence to the maximum value of a field in a table;

user_exists
: Test whether a database user exists;

schema_exists
: Tests whether a database schema exists (which is the same as a user for Oracle);

object_exists
: Test whether a database object (table, package, procedure, etc.) exists;

constraint_exists
: Test whether a database constraint (primary key, foriegn key, unique or check constraint) exists;

drop_types
: Drop all database objects of a certain type;

comment
: A comment;

loadbytes
: Load the binary content of a file;

loadstr
: Load the text content of a file;

var
: Mark an argument for a *procedure* or *sql* command as being an OUT parameter (or pass the value of the variable in subsequent *procedure*/*sql* commands);

env
: Return the value of an environment variable.

Comments

A line starting with # (outside of a SQL command or literal Python block) is considered a comment and will be ignored.

6.10.2 Example

The following is a complete PySQL file that will create a sequence, table and procedure and will call the procedure to insert data into the table:

```
create sequence person_seq
   increment by 10
   start with 10
   maxvalue 1.0e28
   minvalue 10
   nocycle
   cache 20
   noorder
;

-- @@@

create sequence contact_seq
   increment by 10
   start with 10
   maxvalue 1.0e28
   minvalue 10
   nocycle
   cache 20
   noorder
;

-- @@@
```

(continues on next page)

(continued from previous page)

```
create table person
(
   per_id integer not null,
   per_firstname varchar2(200),
   per_lastname varchar2(200)
);

-- @@@

alter table person add constraint person_pk primary key(per_id);

-- @@@

create table contact
(
   con_id integer not null,
   per_id integer not null,
   con_type varchar2(200),
   con_value varchar2(200)
);

-- @@@

alter table contact add constraint contact_pk primary key(con_id);

-- @@@

create or replace procedure person_insert
(
   c_user in varchar2,
   p_per_id in out integer,
   p_per_firstname in varchar2 := null,
   p_per_lastname in varchar2 := null
)
as
begin
   if p_per_id is null then
      select person_seq.nextval into p_per_id from dual;
   end if;

   insert into person
   (
      per_id,
      per_firstname,
      per_lastname
   )
   values
   (
      p_per_id,
      p_per_firstname,
      p_per_lastname
   );
end;
/

-- @@@
```

(continues on next page)

(continued from previous page)

```
create or replace procedure contact_insert
(
   c_user in varchar2,
   p_con_id in out integer,
   p_per_id in integer := null,
   p_con_type in varchar2 := null,
   p_con_value in varchar2 := null
)
as
begin
   if p_con_id is null then
      select contact_seq.nextval into p_con_id from dual;
   end if;

   insert into contact
   (
      con_id,
      per_id,
      con_type,
      con_value
   )
   values
   (
      p_con_id,
      p_per_id,
      p_con_type,
      p_con_value
   );
end;
/

-- @@@

# import data

procedure(
   'person_insert',
   args=dict(
      c_user='import',
      p_per_id=var('per_id_max'),
      p_per_firstname='Max',
      p_per_lastname='Mustermann',
   )
)

procedure(
   'contact_insert',
   args=dict(
      c_user='import',
      p_per_id=var('per_id_max'),
      p_con_id=var('con_id_max'),
      p_con_type='email',
      p_con_value='max@example.org',
   )
)
```

(continues on next page)

(continued from previous page)

```
file(
   'portrait_{per_id_max}.png',
   b'\x89PNG\r\n\x1a\n...',
)

reset_sequence(
   'person_seq',
   table='person',
   field='per_id',
}

check_errors()
```

This file can then be imported into an Oracle database with the following command:

```
python -m ll.pysql -d user/pwd@database data.pysql
```

This will create two sequences, two tables and two procedures. Then it will import two records, one by calling `person_insert` and one by calling `contact_insert`. The PL/SQL equivalent of procedure calls is:

```
declare
   v_per_id_max integer;
   v_con_id_max integer;
begin
   person_insert(
      per_id=v_per_id_max,
      per_firstname='Max',
      per_lastname='Mustermann'
   );
   contact_insert(
      con_id=v_con_id_max,
      per_id=v_per_id_max,
      con_type='email',
      con_value='max@example.org'
   )
end;
```

Furthermore it will create one file (named something like `portrait_42.png`) and reset the sequence `person_seq` to the maximum value of the field `per_id` in the table `person`. Finally it will make sure that no errors exist in the schema.

6.10.3 Multiple database connections

PySQL can handle multiple database connections. New database connections can be opened with the *`connect`* command. This command opens a new database connection. Subsequent commands that talk to the database will use this connection until a *`disconnect`* command disconnects from the database and reverts to the previous connection (or `None` if this was the outermost open database connection). An example looks like this:

```
connect("oracle:user/pwd@db")
procedure("test")
disconnect()
```

for Oracle or like this:

```
connect("postgres:host=localhost dbname=db user=me password=secret")
procedure("test")
disconnect()
```

for Postgres.

6.10.4 Variables

Variable objects can be used to receive OUT parameters of procedure calls or SQL statements. A variable object can be specified like this: `var("foo")`. `"foo"` is the "name" of the variable. When a variable object is passed to a procedure the first time (i.e. the variable object is uninitialized), the resulting value after the call will be stored under the name of the variable. When the variable is used in a later command the stored value will be used instead. (Note that it's not possible to use the same variable twice in the same procedure call, if it hasn't been used before, however in later commands this is no problem).

The type of the variable defaults to `int`, but a different type can be passed when creating the object by passing the Python type like this: `var("foo", str)`.

It is also possible to create variable objects via command line parameters.

As a PySQL command is a Python literal, it is possible to use Python expressions inside a PySQL command.

6.10.5 External files

Inside a PySQL command it is possible to load values from external files. The *`loadbytes`* command loads a `bytes` object from an external file like this:

```
loadbytes("path/to/file.png")
```

A `str` object can be loaded with the *`loadstr`* command like this:

```
loadstr("path/to/file.txt", encoding="utf-8", errors="replace")
```

The second and third argument are the encoding and error handling name respectively.

The filename is treated as being relative to the file containing the *`loadbytes`* or *`loadstr`* call.

This file content can then be used in other PySQL commands (e.g. as parameters in *`procedure`* commands, or as file content in *`scp`* or *`file`* commands).

6.10.6 Command line usage

`pysql.py` has no external dependencies except for `cx_Oracle` (for Oracle) or `psycopg` (for Postgres) and can be used as a script for importing a PySQL file into the database (However some commands require *`ll.orasql`* for an Oracle database). As a script it supports the following command line options:

`file`
: The name of one or more PySQL files that will be read and imported. If no filename is given, commands are read from `stdin`.

`-v, --verbose`
: Gives different levels of output while data is being imported to the database. The default is no output (unless an exception occurs). Possible modes are: `dot` (one dot for each command), `type` (each command type), `file` (the file names and line numbers from which code gets executed), `log` (the log messages output by the commands) or `full` (source code that will be executed and the log messages output by the commands).

-d, --database
The initial database connection that will be used before any additional *connect* commands.

For Postgres the value must start with `postgres:` the rest of the value will be passed to `psycopg.Connection.connect()` as a positional argument. For example:

```
postgres:host=localhost dbname=test user=me password=secret
```

For Oracle the value may start with `oracle:`. The rest can be a standard Oracle connectstring. For example:

```
me/secret@database
```

-z, --summary
Give a summary of the number of commands executed and procedures called.

-r, --rollback
Specifies that transactions should be rolled back at the end of the script run, or when a *disconnect* command disconnects from a database. The default is to commit at the end or on each disconnect. (But note that for Oracle when a DDL statement is in the script, Oracle will still implicitely commit everything up to the statement.)

-s, --scpdirectory
The base directory for *scp* file copy commands. As files are copied via **scp** this can be a remote filename (like `root@www.example.org:~/uploads/`) and must include a trailing `/`.

If it is a local directory it should be absolute (otherwise PySQL scripts included from other directories won't work).

-f, --filedirectory
The base directory for *file* file save commands. It must include a trailing `/`.

--tabsize
The tab size when PySQL source is printed in `full` mode.

--context
The number of lines at the start and end of the source code of a block to print in `full` mode. The default is to print the complete source code.

-D, --define
Can be used multiple times to define variables. Supported formats are:

name
Defines a string variable named `name` and sets the value to the empty string.

name=value
Defines a string variable named `name` and sets the value to `value`.

name:type
Defines a variable named `name` of type `type` and sets the value to `False`, `0`, `0.0` or the empty string depending on the type. Supported types are `str`, `bool`, `int` and `float`.

name:type=value
Defines a variable named `name` of type `type` and sets the value to `value`. For type `bool` supported values are `0`, `no`, `false`, `False`, `1`, `yes`, `true` and `True`.

`ll.pysql.`**format_class**(*obj*)

Return the name for the class `obj`.

`ll.pysql.`**shortrepr**(*value*)

Return a short "repr" output for a `str` or `bytes` value.

If the value `value` is sort enough, the normal `repr()` output will be returned, otherwise an abbreviated value will be returned, i.e. something like this:

```
<'foobarbaz' ... (1234 characters)>
```

class ll.pysql.**Handler**

Bases: object

A Handler object is responsible for executing PySQL commands.

Handler can not execute commands that require a database connection. That is the job of the subclasses *OracleHandler*, *OraSQLHandler* and *PostgresHandler*.

static from_connectstring(*connectstring*, *mode=None*)

Create an appropriate Handler from a connectstring.

If connectstring is None, a *Handler* object will be returned.

If connectstring starts with postgres:, a *PostgresHandler* will be returned.

Otherwise on *OracleHandler* will be returned.

static from_command(*command*)

Create an appropriate Handler from *connect* command.

connectstring()

Return a string identifying the database connection for this handler.

connect(*context*, *command*)

Execute the *connect* command command.

include(*context*, *command*)

Execute the *include* command command.

user_exists(*context*, *command*)

Execute the *user_exists* command command.

schema_exists(*context*, *command*)

Execute the *schema_exists* command command.

object_exists(*context*, *command*)

Execute the *object_exists* command command.

constraint_exists(*context*, *command*)

Execute the *constraint_exists* command command.

check_errors(*context*, *command*)

Execute the *check_errors* command command.

literalsql(*context*, *command*)

Execute the *literalsql* command command.

literalpy(*context*, *command*)

Execute the *literalpy* command command.

procedure(*context*, *command*)

Execute the *procedure* command command.

sql(*context*, *command*)

Execute the *sql* command command.

reset_sequence(*context*, *command*)

Execute the *reset_sequence* command command.

drop_types(*context*, *command*)

Execute the *drop_types* command command.

rollback(*context*, *command*)

Execute the *rollback* command command.

commit(*context*, *command*)

Execute the *commit* command command.

disconnect(*context*, *command*)

Execute the *disconnect* command command.

setvar(*context*, *command*)

Execute the *setvar* command command.

unsetvar(*context*, *command*)

Execute the *unsetvar* command command.

raise_exceptions(*context*, *command*)

Execute the *raise_exceptions* command command.

push_raise_exceptions(*context*, *command*)

Execute the *push_raise_exceptions* command command.

pop_raise_exceptions(*context*, *command*)

Execute the *pop_raise_exceptions* command command.

scp(*context*, *command*)

Execute the *scp* command command.

file(*context*, *command*)

Execute the *file* command command.

comment(*context*, *command*)

Execute the *comment* command command.

loadbytes(*context*, *command*)

Execute the *loadbytes* command command.

loadstr(*context*, *command*)

Execute the *loadstr* command command.

var(*context*, *command*)

Execute the *var* command command.

env(*context*, *command*)

Execute the *env* command command.

log(*context*, *command*)

Execute the *log* command command.

class ll.pysql.**DBHandler**

Bases: *Handler*

Subclass of *Handler* that has a real database connection.

class ll.pysql.**OracleHandler**

Bases: *DBHandler*

Subclass of *DBHandler* that executes database commands via cx_Oracle.

However *drop_types* is not supported, for this *OraSQLHandler* is required, which requires that *ll.orasql* is available).

class ll.pysql.**OraSQLHandler**

Bases: *OracleHandler*

Subclass of *DBHandler* that executes database commands via *ll.orasql*.

class ll.pysql.**PostgresHandler**

Bases: *DBHandler*

Subclass of *DBHandler* that executes database commands for Postgres.

class ll.pysql.**Command**

Bases: object

The base class of all commands. A *Command* object is created from a function call in a PySQL file and then immediatly the method `execute()` will be called to execute the command.

The only parameters in the call that is supported by all commands are the following:

raise_exceptions
[bool (optional)]
Specifies whether exceptions that happen during the execution of the command should be reported and terminate the script (`True`), or should be ignored (`False`). `None` (the default) uses the global configuration.

cond
[bool (optional)]
Specifies whether this command should be executed or not. If `cond` is `True` (the default), the command will be executed, else it won't.

ll.pysql.**register**(*cls*)

Register a *Command* subclass as a PySQL command.

This is used as a class decorator.

class ll.pysql.**include**

Bases: *Command*

The `include` command includes another PySQL file. The filename is passed in the first parameter `filename`. This filename is interpreted as being relative to the directory with the file containing the `include` command.

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class ll.pysql.**connect**

Bases: *Command*

The `connect` command connects to the database given in the connectstring in the parameter `connectstring`. After the `connect` command until the matching *disconnect* command, all commands that talk to the database will use this connection. After a *disconnect* command `pysql` will revert back to the previously active database connection. Parameter have the following meaning:

mode
[string or `None` (optional)]
The connection mode: This can be either `'sysdba'` or `None` (the default).

retry
[int (optional)]
The number of times PySQL tries to get a database connection.

retrydelay
[int (optional)]
The number of seconds to wait between connection tries.

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class ll.pysql.**disconnect**

Bases: *Command*

The `disconnect` command disconnects from the active database connection and reverts back to the previously active database connection.

`commit` specifies whether the transaction should be committed. If `commit` is `None`, the default commit mode is used (which can be changed on the command line via the `-r`/`--rollback` option).

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class `ll.pysql.`**procedure**

Bases: `_SQLCommand`

A `procedure` command calls an Oracle procedure in the database. The following parameters are supported:

name

[string (required)]
The name of the procedure to be called (This may include `.` for calling a procedure in a package or one owned by a different user).

args

[dictionary (optional)]
A dictionary with the names of the parameters as keys and the parameter values as values. PySQL supports all types as values that `cx_Oracle` supports. In addition to those, three special classes are supported:

- *sqlexpr* objects can be used to specify that the paramater should be literal SQL. So e.g. `sqlexpr("sysdate")` will be the date when the PySQL script was executed.
- *var* objects can be used to hold values that are `OUT` parameters of the procedure. For example on first use of `var("foo_10")` the value of the `OUT` parameter will be stored under the key `"foo_10"`. The next time `var("foo_10")` is encountered the value stored under the key `"foo_10"` will be passed to the procedure. The type of the variable defaults to `int`. If a different type is required it can be passed as the second argument to *var*, e.g. `var("foo_10", str)`.
- Finally all other commands can be called to get a value (for example the two commands *loadbytes* and *loadstr* to load values from external files (as long as they are of type `bytes` or `str`). `loadbytes("foo/bar.txt")` will return with the content of the external file `foo/bar.txt` (as a `bytes` object). If a `str` object is required, *loadstr* can be used. Encoding info can be passed like this:

```
loadstr("foo/bar.txt", encoding="utf-8", errors="replace")
```

argtypes

[dictionary (optional)]
A dictionary with the names of the parameters as keys and Postgres datatypes as the values. This is used for adding a cast to the parameter value in the call to guide Postgres to find the correct overloaded version of the procedure. For Oracle `argtypes` will be ignored.

For the rest of the parameters see the base class `_DatabaseCommand`.

class `ll.pysql.`**sql**

Bases: `_SQLCommand`

An `sql` command directly executes an SQL statement in the Oracle database. The following parameters are supported:

sql

[string (required)]
The SQL to be executed. This may contain parameters in the form of `:paramname`. The values for those parameters will be taken from `args`.

args

[dictionary (optional)]
A dictionary with the names of the parameters as keys and the parameter values as values. Similar to procedure calls *var*, *loadbytes* and *loadstr* objects are supported. However *sqlexpr* objects are not supported (they will be ignored).

argtypes

[dictionary (optional)]

A dictionary with the names of the parameters as keys and Postgres datatypes as the values. This is used for adding a cast to the parameter value in the call to try to convert the value to the proper Postgres datatype. For Oracle `argtypes` will be ignored.

For the rest of the parameters see the base class `_DatabaseCommand`.

If you have arguments you can reference them in Oracle code like this:

```
sql(
   "insert into foo (bar) values (:bar)",
   args=dict(
      bar="bar",
   )
)
```

or (if you want to use a variable) like this:

```
sql(
   "insert into foo (bar) values (:bar)",
   args=dict(
      bar=var("bar", str),
   )
)
```

Or like this when you wnat to get a value out from an SQL command:

```
sql(
   "begin; :now := to_char(sysdate); end;",
   args=dict(
      now=var("now", date),
   )
)
```

For Postgres you must reference parameters in the query like this:

```
sql(
   "insert into foo (bar) values (%(bar)s)",
   args=dict(
      bar="bar",
   )
)
```

or (if you want to use a variable) like this:

```
sql(
   "insert into foo (bar) values (%(bar)s)",
   args=dict(
      bar=var("bar", str),
   )
)
```

However to get variables out of a Postgres SQL statement you must use a `select`:

```
sql(
   "select baz as baz from foo where bar = %(bar)s",
   args=dict(
      bar=var("bar", str),
   )
)
```

Specify the target variable name via the output name of the field expressions.

In the about example a variable `baz` will be set.

class `ll.pysql.`**`literalsql`**

Bases: `_SQLCommand`

A `literalsql` is used for SQL that appears literally in the PySQL file. Apart from the `sql` attribute it supports no further parameters.

class `ll.pysql.`**`commit`**

Bases: `_SQLCommand`

A `commit` command commits the current transaction in the activate database connection (or the one specified via the `connection` parameter).

For the rest of the parameters see the base class `_DatabaseCommand`.

class `ll.pysql.`**`rollback`**

Bases: `_SQLCommand`

A `rollback` command rolls back the current transaction in the activate database connection (or the one specified via the `connection` parameter).

For the rest of the parameters see the base class `_DatabaseCommand`.

class `ll.pysql.`**`literalpy`**

Bases: `_DatabaseCommand`

A `literalpy` is used for Python code that appears literally in the PySQL file. Apart from the `code` attribute it supports the no further parameters.

class `ll.pysql.`**`setvar`**

Bases: *Command*

The `setvar` command sets a variable to a fixed value. The following parameters are supported:

name
[string (required)]
The name of the variable to set.

value
[object (required)]
The value of the variable.

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class `ll.pysql.`**`unsetvar`**

Bases: *Command*

The `unsetvar` command deletes a variable. The parameter `name` must be given and must contain the name of the variable.

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class `ll.pysql.`**`raise_exceptions`**

Bases: *Command*

The `raise_exceptions` command changes the global error reporting mode for all subsequent commands. After:

```
raise_exceptions(False)
```

for all subsequent commands any exception will be ignored and reported and command execution will continue with the next command.

```
raise_exceptions(True)
```

will switch back to aborting the execution of the PySQL script once an exception is encountered.

Note that the global configuration will only be relevant for commands that don't specify the `raise_exceptions` parameter themselves.

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class `ll.pysql.`**`push_raise_exceptions`**

Bases: *Command*

The `push_raise_exceptions` command changes the global error reporting mode for all subsequent commands, but remembers the previous exception handling mode. After:

```
push_raise_exceptions(False)
```

for all subsequent commands any exception will be ignored and reported and command execution will continue with the next command. It is possible to switch back to the previous exception handling mode via:

```
pop_raise_exceptions()
```

Note that this global configuration will only be relevant for commands that don't specify the `raise_exceptions` parameter themselves.

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class `ll.pysql.`**`pop_raise_exceptions`**

Bases: *Command*

The *pop_raise_exceptions* command restores the previously active exception handling mode (i.e. the one active before the last *push_raise_exceptions* command).

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class `ll.pysql.`**`check_errors`**

Bases: `_DatabaseCommand`

The `check_errors` command checks that there are no compilation errors in the active database schema. If there are, an exception will be raised.

For the rest of the parameters see the base class `_DatabaseCommand` (but the value of the `raise_exceptions` key will be ignored).

class `ll.pysql.`**`scp`**

Bases: *Command*

The `scp` command creates a file by copying it via the **scp** program. The following parameters are supported:

name
[string (required)]
The name of the file to be created. It may contain `format()` style specifications containing any variable (for example those that appeared in a *procedure* or *sql* command). These specifiers will be replaced by the correct variable values. As these files will be copied via the **scp** program, ssh file names can be used.

content
[bytes (required)]
The content of the file to be created. This can also be a *loadbytes* command to load the content from an external file.

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class ll.pysql.**file**

Bases: *Command*

The `file` command creates a file by directly saving it from Python. The following parameters are supported:

name
[string (required)]
The name of the file to be created. It may contain `format()` style specifications containing any variable (for example those that appeared in a *procedure* or *sql* command). These specifiers will be replaced by the correct variable values.

content
[bytes (required)]
The content of the file to be created. This can also be a *loadbytes* command to load the content from an external file.

mode
[integer (optional)]
The file mode for the new file. If the mode is specified, `os.chmod()` will be called on the file.

owner
[integer or string (optional)]
The owner of the file (as a user name or a uid).

group
[integer or string (optional)]
The owning group of the file (as a group name or a gid). If `owner` or `group` is given, `os.chown()` will be called on the file.

For the parameters `raise_exceptions` and `cond` see the base class *Command*.

class ll.pysql.**reset_sequence**

Bases: `_DatabaseCommand`

The `reset_sequence` command resets a sequence in the database to the maximum value of a field in a table. The following parameters are supported:

sequence
[string (required)]
The name of the sequence to reset.

table
[string (required)]
The name of the table that contains the field.

field
[string (required)]
The name of the field in the table `table`. The sequence will be reset to a value so that fetching the next value from the sequence will deliver a value that is larger than the maximum value of the field `field` in the table `table`.

minvalue
[integer (optional, default taken from sequence)]
The minimum value for the sequence.

increment
[integer (optional, default taken from sequence)]
The increment (i.e. the step size) for the sequence.

For the rest of the parameters see the base class `_DatabaseCommand`.

class ll.pysql.**user_exists**

Bases: `_DatabaseCommand`

The `user_exists` command returns whether a user with a specified name exists in the database. It supports the following parameters:

name
[string (required)]
The name of the user to be checked for existence.

For the rest of the parameters see the base class `_DatabaseCommand`.

class `ll.pysql.`**`schema_exists`**

Bases: `_DatabaseCommand`

The `schema_exists` command returns whether a schema with a specified name exists in the database. It supports the following parameters:

name: string (required)
The name of the schema to be checked for existence.

For the rest of the parameters see the base class `_DatabaseCommand`.

class `ll.pysql.`**`object_exists`**

Bases: `_DatabaseCommand`

The `object_exists` command returns whether an object with a specified name exists in the database. It supports the following parameters:

name
[string (required)]
The name of the object to be checked for existence.

owner
[string (optional)]
The owner of the object (defaults to the current user if not specified or `None`).

For the rest of the parameters see the base class `_DatabaseCommand`.

Note that `object_exists` won't test for constraints. For this use *`constraint_exists`*.

class `ll.pysql.`**`constraint_exists`**

Bases: `_DatabaseCommand`

The `constraint_exists` command returns whether a constraint (i.e. a primary key, foreign key, unique or check constraint) with a specified name exists in the database. It supports the following parameters:

name
[string (required)]
The name of the object to be checked for existence.

owner
[string (optional)]
The owner of the constraint (defaults to the current user if not specified or `None`).

For the rest of the parameters see the base class `_DatabaseCommand`.

class `ll.pysql.`**`drop_types`**

Bases: `_DatabaseCommand`

The `drop_types` command drops database objects.

Unlike all other commands this command requires the *`ll.orasql`* module.

`drop_types` supports the following parameters:

drop
[list of strings (optional)]
The types of objects to drop (value must be names for *`ll.orasql`* object types.

keep
[list of strings (optional)]
The types of objects to keep (value must be names for *`ll.orasql`* object types.

drop and keep are mutually exclusive. When neither of them is specified *all* database objects will be dropped.

For the rest of the parameters see the base class _DatabaseCommand.

class ll.pysql.**comment**

Bases: *Command*

The comment command does nothing.

class ll.pysql.**loadbytes**

Bases: *Command*

The loadbytes command can be used to load a bytes object from an external file. The following parameters are supported:

filename
[string (required)]
The name of the file to be loaded. The filename is treated as being relative to the directory containing the PySQL file that contains *loadbytes* command.

For the parameters raise_exceptions and cond see the base class *Command*.

class ll.pysql.**loadstr**

Bases: *Command*

The loadstr command can be used to load a str object from an external file. The following parameters are supported:

filename
[string (required)]
The name of the file to be loaded. The filename is treated as being relative to the directory containing the PySQL file that contains the loadstr command.

encoding
[string (optional)]
The encoding used for decoding the bytes in the file to text.

errors
[string (optional)]
The error handling mode for decoding.

For the parameters raise_exceptions and cond see the base class *Command*.

__init__(*filename*, *, *encoding=None*, *errors='strict'*, *raise_exceptions=None*, *cond=True*)

Create a new *loadstr* object.

class ll.pysql.**var**

Bases: *Command*

var commands are used to mark procedure values that are OUT parameters. On first use the parameter is used as an OUT parameter and PySQL will remembers the OUT value as a local variable under the unique name specified in the constructor. When a *var* object is used a second time a variable object will be passed to the procedure with it's value set to the value of the local variable. The following parameters are supported:

key
[string (required)]
A unique name for the value.

type
[class (optional)]
The type of the value (defaulting to int).

Note that when the key is None, PySQL will *not* remember the value, instead each use of var(None) will create a new OUT parameter. This can be used for OUT parameters whose values is not required by subsequent commands.

class ll.pysql.**env**

Bases: *Command*

A *env* command returns the value of an environment variable.

The following parameters are supported:

name
[string (required)]
The name of the environment variable.

default
[string (optional)]
The default to use, if the environment variable isn't set. This defaults to None.

class ll.pysql.**log**

Bases: *Command*

log commands generate logging output.

The following parameters are supported:

objects
[Any]
The objects to log. Strings will be logged directly. For all other objects repr() will be called.

class ll.pysql.**CommandExecutor**

Bases: object

A CommandExecutor object wraps executing a *Command* object in a callable. Calling the CommandExecutor object executes the command using the specified context and returns the command result.

This class exists because *Command* objects serve two purposes:

1. They can be created to print them to a file (via the method Command.source());
2. They can be put into a PySQL file which will then be read and executed, with must then create the *Command* object and execute it immediately. This is the job of CommandExecutor objects.

class ll.pysql.**Context**

Bases: object

A *Context* objects contains the configuration and run time information required for importing a PySQL file.

executeall(**filenames*)

Execute all commands in the PySQL files specified by filenames. If filenames is empty sys.stdin is read.

class ll.pysql.**sqlexpr**

Bases: object

An *sqlexpr* object can be used to specify an SQL expression as a procedure parameter instead of a fixed value. For example passing the current date (i.e. the date of the import) can be done with sqlexpr("sysdate").

class ll.pysql.**pyexpr**

Bases: object

A *pyexpr* object can be used to embed literal Python source code in a PySQL file.

Note: As PySQL source code is evaluated via eval()/exec() anyway, it it always possible to embed Python expressions in PySQL source code. However this doesn't roundtrip, i.e. printing the PySQL command via source() outputs the value of a "literal" Python expression.

exception ll.pysql.**CompilationError**

Bases: Exception

Exception raised by *check_errors* when invalid database objects are encountered.

exception ll.pysql.**SCPError**

Bases: Exception

Exception raised by *scp* when a call to the scp command fails.

class ll.pysql.**Location**

Bases: object

The location of a PySQL/SQL command in a PySQL file.

6.11 ll.orasql – Utilities for cx_Oracle

ll.orasql contains utilities for working with cx_Oracle:

- It allows calling procedures and functions with keyword arguments (via the classes *Procedure* and *Function*).
- Query results will be put into *Record* objects, where database fields are accessible as object attributes.
- The *Connection* class provides methods for iterating through the database metadata.
- Importing this module adds support for URLs with the scheme oracle to *ll.url*. Examples of these URLs are:

```
oracle://user:pwd@db/
oracle://user:pwd@db/view/
oracle://user:pwd@db/view/USER_TABLES.sql
oracle://sys:pwd:sysdba@db/
```

class ll.orasql.**Args**

Bases: dict

An *Args* object is a subclass of dict that is used for passing arguments to procedures and functions. Both item and attribute access (i.e. __getitem__() and __getattr__()) are available. Names are case insensitive.

class ll.orasql.**LOBStream**

Bases: object

A *LOBStream* object provides streamlike access to a BLOB or CLOB.

readall()

Read all remaining data from the stream and return it.

readchunk()

Read a chunk of data from the stream and return it. Reading is done in optimally sized chunks.

read(*size=None*)

Read size bytes/characters from the stream and return them. If size is None all remaining data will be read.

reset()

Reset the stream so that the next *read()* call starts at the beginning of the LOB.

seek(*offset*, *whence=0*)

Seek to the position offset in the LOB. The whence argument is optional and defaults to 0 (absolute file positioning); The other allowed value is 1 (seek relative to the current position).

class ll.orasql.**Record**

Bases: tuple, Mapping

A *Record* is a subclass of tuple that is used for storing results of database fetches and procedure and function calls. Both item and attribute access (i.e. __getitem__() and __getattr__()) are available. Field names are case insensitive.

get(*name*, *default=None*)

Return the value for the field named name. If this field doesn't exist in self, return default instead.

keys()

Return an iterator over field names.

items()

Return an iterator over (field name, field value) tuples.

replace(***kwargs*)

Return a new *Record* with the same fields as self, except for those fields given new values by whichever keyword arguments are specified.

class ll.orasql.**SessionPool**

Bases: SessionPool

SessionPool is a subclass of cx_Oracle.SessionPool.

ll.orasql.**owned**(*obj*, *owner*)

Do we own the object obj according to the owner specification owner?

owner can be:

None
The current user (i.e. via the view USER_OBJECTS);

ALL
Any user (via the views ALL_OBJECTS or DBA_OBJECTS);

username
[string]
The specified user

set or tuple
A set or tuple of usernames. An object must belong to any of the users to be considered owned.

class ll.orasql.**Connection**

Bases: Connection

Connection is a subclass of cx_Oracle.Connection.

__init__(**args*, ***kwargs*)

Create a new connection. In addition to the parameters supported by cx_Oracle.connect() the following keyword argument is supported.

readlobs
[bool or integer]
If readlobs is False all cursor fetches return *LOBStream* objects for LOB objects. If readlobs is an int LOBs with a maximum size of readlobs will be returned as bytes/str objects. If readlobs is True all LOB values will be returned as bytes/str objects.

decimal
[bool]
If decimal is True numbers will be returned as decimal.Decimal objects, else float will be used.

Furthermore the clientinfo will be automatically set to the name of the currently running script (except if the clientinfo keyword argument is given and None).

cursor(*readlobs=None*)

Return a new cursor for this connection. For the meaning of `readlobs` see `__init__()`.

tables(*owner=None*, *mode='flat'*)

Generator that yields all table definitions in the current users schema (or all users schemas). `mode` specifies the order in which tables will be yielded:

"create"
Create order, inserting records into the table in this order will not violate foreign key constraints.

"drop"
Drop order, deleting records from the table in this order will not violate foreign key constraints.

"flat"
Unordered.

`owner` specifies from which user tables should be yielded. It can be `None` (for the current user), `ALL` (for all users (the default)) or a user name.

Tables that are materialized views will be skipped in all cases.

sequences(*owner=None*)

Generator that yields sequences. `owner` can be `None` (the default), `ALL` or a user name.

fks(*owner=None*)

Generator that yields all foreign key constraints. `owner` can be `None` (the default), `ALL` or a user name.

privileges(*owner=None*)

Generator that yields object privileges. `owner` can be `None` (the default), `ALL`, a user name or a set or tuple of user names.

synonyms(*owner=None*, *object_owner=ll.orasql.ALL*)

Generator that yields synonyms. `owner` and `object_owner` can be `None` (the default), `ALL`, a user name or a set or tuple of user names.

users()

Generator that yields all users.

objects(*owner=None*, *mode='create'*)

Generator that yields the sequences, tables, primary keys, foreign keys, table and columns comments, unique constraints, indexes, views, functions, procedures, packages, types and jobsin the current users schema (or all users schemas) in a specified order.

`mode` specifies the order in which objects will be yielded:

"create"
Create order, i.e. recreating the objects in this order will not lead to errors;

"drop"
Drop order, i.e. dropping the objects in this order will not lead to errors;

"flat"
Unordered.

`owner` specifies from which schema objects should be yielded. For more information see `owned()`.

objects_named(*name*, *owner=None*)

Return all objects named `name` from the schema. If `owner` is `None` the current schema is queried, else the specified one is used. `name` and `owner` are treated case insensitively.

There might be multiple object with the same name, if these names only differ in casing. Also there *will* be multiple object with the same name for packages and package bodies.

object_named(*name*, *owner=None*)

Return the object named `name` from the schema. If `owner` is `None` the current schema is queried, else the specified one is used. `name` and `owner` are treated case insensitively.

If there are multiple objects with the same name, which one gets returned is undefined.

If there is no such object an `SQLNoSuchObjectError` exception will be raised.

`ll.orasql.`**connect**(**args*, ***kwargs*)

Create a connection to the database and return a *Connection* object.

class `ll.orasql.`**Cursor**

Bases: `Cursor`

A subclass of the cursor class in `cx_Oracle`. The "fetch" methods will return records as *Record* objects and `LOB` values will be returned as *LOBStream* objects or `str`/`bytes` objects (depending on the cursors `readlobs` attribute).

__init__(*connection*, *readlobs=None*)

Return a new cursor for the connection `connection`. For the meaning of `readlobs` see *Connection.__init__()*.

ddprefix()

Return whether the user has access to the `DBA_*` views (`"dba"`) or not (`"all"`).

ddprefixargs()

Return whether the user has access to the `DBA_ARGUMENTS` view (`"dba"`) or not (`"all"`).

class `ll.orasql.`**MixinNormalDates**

Bases: `object`

Mixin class that provides methods for determining creation and modification dates for objects.

class `ll.orasql.`**MixinCodeSQL**

Bases: `object`

Mixin class that provides methods returning the create and drop statements for various objects.

class `ll.orasql.`**SchemaObject**

Bases: `object`

The base class for all Python classes modelling schema objects in the database.

createsql(*connection=None*, *term=True*)

Return SQL code to create this object.

dropsql(*connection=None*, *term=True*)

Return SQL code to drop this object

fixname(*code*)

Replace the name of the object in the SQL code `code` with the name of `self`.

exists(*connection=None*)

Return whether the object `self` really exists in the database specified by `connection`.

cdate(*connection=None*)

Return a `datetime.datetime` object with the creation date of `self` in the database specified by `connection` (or `None` if that information is not available).

udate(*connection=None*)

Return a `datetime.datetime` object with the last modification date of `self` in the database specified by `connection` (or `None` if that information is not available).

references(*connection=None*, *done=None*)

Objects directly used by `self`.

If `connection` is not `None` it will be used as the database connection from which to fetch data. If `connection` is `None` the connection from which `self` has been extracted will be used. If there is not such connection, you'll get an exception.

referencesall(*connection=None*, *done=None*)

All objects used by `self` (recursively).

For the meaning of `connection` see *references()*.

`done` is used internally and shouldn't be passed.

referencedby(*connection=None*)

Objects using `self`.

For the meaning of `connection` see *references()*.

referencedbyall(*connection=None*, *done=None*)

All objects depending on `self` (recursively).

For the meaning of `connection` see *references()*.

`done` is used internally and shouldn't be passed.

class ll.orasql.**OwnedSchemaObject**

Bases: *SchemaObject*

The base class for all Python classes modelling schema objects in the database.

classmethod names(*connection*, *owner=None*)

Generator that yields the names of all objects of this type. The argument `owner` specifies whose objects are yielded. For more information see *owned()*.

Names will be in ascending order.

classmethod objects(*connection*, *owner=None*)

Generator that yields all objects of this type in the current users schema. The argument `owner` specifies whose objects are yielded. For more information see *owned()*.

synonyms(*connection=None*)

Generator that yields all synonyms for this object.

privileges(*connection=None*)

Generator that yields all privileges on this object.

class ll.orasql.**Sequence**

Bases: *MixinNormalDates*, *OwnedSchemaObject*

Models a sequence in the database.

createsqlcopy(*connection=None*, *term=True*)

Return SQL code to create an identical copy of this sequence.

class ll.orasql.**Table**

Bases: *MixinNormalDates*, *OwnedSchemaObject*

Models a table in the database.

mview(*connection=None*)

The materialized view this table belongs to (or `None` if it's a real table).

ismview(*connection=None*)

Is this table a materialized view?

organization(*connection=None*)

Return the organization of this table: either "heap" (for "normal" tables) or "index" (for index organized tables).

logging(*connection=None*)

Return whether the table is in logging mode or not.

compression(*connection=None*)

Return the compression mode of the table.

(None, "BASIC" or "ADVANCED").

columns(*connection=None*)

Generator that yields all column objects of this table.

records(*connection=None*)

Generator that yields all records of this table.

comment(*connection=None*)

Return the table comment

comments(*connection=None*)

Generator that yields all column comments of this table.

constraints(*connection=None*)

Generator that yields all constraints for this table.

pk(*connection=None*)

Return the primary key constraint for this table (or None if the table has no primary key constraint).

fks(*connection=None*)

Return the foreign key constraints for this table.

uniques(*connection=None*)

Return the unique constraints for this table.

checks(*connection=None*)

Return the unique constraints for this table.

class ll.orasql.**TableComment**

Bases: *OwnedSchemaObject*

Models a table comment in the database.

comment(*connection=None*)

Return the comment text for this table.

class ll.orasql.**ColumnComment**

Bases: *OwnedSchemaObject*

Models a column comment in the database.

comment(*connection=None*)

Return the comment text for this column.

class ll.orasql.**Constraint**

Bases: *OwnedSchemaObject*

Base class of all constraints (primary key constraints, foreign key constraints, unique constraints and check constraints).

isenabled(*connection=None*)

Return whether this constraint is enabled.

table(*connection=None*)

Return the *Table* `self` belongs to.

class ll.orasql.**PrimaryKey**

Bases: *Constraint*

Models a primary key constraint in the database.

columns(*connection=None*)

Return an iterator over the columns this primary key consists of.

class ll.orasql.**ForeignKey**

Bases: *Constraint*

Models a foreign key constraint in the database.

refconstraint(*connection=None*)

Return the constraint referenced by `self`.

In most cases this is a *PrimaryKey*, but it also might be a *UniqueConstraint*.

columns(*connection=None*)

Return an iterator over the columns this foreign key consists of.

class ll.orasql.**UniqueConstraint**

Bases: *Constraint*

Models a unique constraint in the database.

class ll.orasql.**CheckConstraint**

Bases: *Constraint*

Models a check constraint in the database.

class ll.orasql.**Index**

Bases: *MixinNormalDates*, *OwnedSchemaObject*

Models an index in the database.

rebuildsql(*connection=None*, *term=True*)

Return SQL code to rebuild this index.

constraint(*connection=None*)

If this index is generated by a constraint, return the constraint otherwise return `None`.

isconstraint(*connection=None*)

Is this index generated by a constraint?

table(*connection=None*)

Return the *Table* `self` belongs to.

columns(*connection=None*)

Return an iterator over the columns this index consists of.

class ll.orasql.**Synonym**

Bases: *OwnedSchemaObject*

Models a synonym in the database.

object(*connection=None*)

Return the object for which `self` is a synonym.

classmethod names(*connection*, *owner=None*, *object_owner=ll.orasql.ALL*)

Generator that yields the names of all synonyms. For the meaning of `owner` and `object_owner` see *objects()*.

Names will be in ascending order.

classmethod objects(*connection*, *owner=None*, *object_owner=ll.orasql.ALL*)

Generator that yields all synonym in the current users schema. The argument `owner` specifies to which owner the synonym must belong to to be yielded. The argument `object_owner` specifies to which owner the object must belong to to be yielded. For more information see *owned()*.

class ll.orasql.**View**

Bases: *MixinNormalDates*, *OwnedSchemaObject*

Models a view in the database.

class ll.orasql.**MaterializedView**

Bases: *View*

Models a meterialized view in the database.

class ll.orasql.**Library**

Bases: *OwnedSchemaObject*

Models a library in the database.

class ll.orasql.**Argument**

Bases: `object`

Argument objects hold information about the arguments of a stored procedure.

class ll.orasql.**Callable**

Bases: *MixinNormalDates*, *MixinCodeSQL*, *OwnedSchemaObject*

Models a callable object in the database, i.e. functions and procedures.

arguments(*connection=None*)

Generator that yields all arguments of the function/procedure `self`.

class ll.orasql.**Procedure**

Bases: *Callable*

Models a procedure in the database. A *Procedure* object can be used as a wrapper for calling the procedure with keyword arguments.

__call__(*cursor*, **args*, ***kwargs*)

Call the procedure with arguments `args` and keyword arguments `kwargs`. `cursor` must be a *ll.orasql.Cursor* object. This will return a *Record* object containing the result of the call (i.e. this record will contain all specified and all out parameters).

class ll.orasql.**Function**

Bases: *Callable*

Models a function in the database. A *Function* object can be used as a wrapper for calling the function with keyword arguments.

__call__(*cursor*, **args*, ***kwargs*)

Call the function with arguments `args` and keyword arguments `kwargs`. `cursor` must be an *ll.orasql.Cursor* object. This will return a tuple containing the result and a *Record* object containing the modified parameters (i.e. this record will contain all specified and out parameters).

class ll.orasql.**Package**

Bases: *MixinNormalDates*, *MixinCodeSQL*, *OwnedSchemaObject*

Models a package in the database.

class ll.orasql.**PackageBody**

Bases: *MixinNormalDates*, *MixinCodeSQL*, *OwnedSchemaObject*

Models a package body in the database.

class ll.orasql.**Type**

Bases: *MixinNormalDates*, *MixinCodeSQL*, *OwnedSchemaObject*

Models a type definition in the database.

class ll.orasql.**TypeBody**

Bases: *MixinNormalDates*, *MixinCodeSQL*, *OwnedSchemaObject*

Models a type body in the database.

class ll.orasql.**Trigger**

Bases: *MixinNormalDates*, *MixinCodeSQL*, *OwnedSchemaObject*

Models a trigger in the database.

class ll.orasql.**JavaSource**

Bases: *MixinNormalDates*, *OwnedSchemaObject*

Models Java source code in the database.

class ll.orasql.**Privilege**

Bases: *object*

Models a database object privilege (i.e. a grant).

A *Privilege* object has the following attributes:

privilege
[string]
The type of the privilege (`EXECUTE` etc.)

name
[string]
The name of the object for which this privilege grants access

owner
[string or `None`]
the owner of the object

grantor
[string or `None`]
Who granted this privilege?

grantee
[string or `None`]
To whom has this privilege been granted?

connection
[*Connection* or `None`]
The database connection

object(*connection=None*)

Return the object on which `self` grants a privilege.

classmethod objects(*connection*, *owner=None*)

Generator that yields object privileges. For the meaning of `owner` see *owned()*.

grantsql(*connection=None*, *term=True*, *mapgrantee=True*)

Return SQL code to grant this privilege. If `mapgrantee` is a list or a dictionary and `self.grantee` is not in this list (or dictionary) no command will be returned. If it's a dictionary and `self.grantee` is in it, the privilege will be granted to the user specified as the value instead of the original one. If `mapgrantee` is true (the default) the privilege will be granted to the original grantee.

class ll.orasql.**Column**

Bases: *OwnedSchemaObject*

Models a single column of a table in the database. This is used to output `ALTER TABLE` statements for adding, dropping and modifying columns.

datatype(*connection=None*)

The SQL type of this column.

default(*connection=None*)

The SQL default value for this column.

nullable(*connection=None*)

Is this column nullable?

compression(*connection=None*)

The compression mode for this LOB column.

Return `None` if this is not a LOB column, or it isn't compressed.

comment(*connection=None*)

The comment for this column.

class ll.orasql.**User**

Bases: *SchemaObject*

Models a user in the database.

classmethod names(*connection*)

Generator that yields the names of all users in ascending order

classmethod objects(*connection*)

Generator that yields all user objects.

class ll.orasql.**Preference**

Bases: *OwnedSchemaObject*

Models a preference in the database.

classmethod names(*connection*, *owner=None*)

Generator that yields the names of all preferences.

classmethod objects(*connection*, *owner=None*)

Generator that yields all preferences.

class ll.orasql.**JobClass**

Bases: *SchemaObject*

Models a job class (from the `dbms_scheduler` package) in the database.

classmethod names(*connection*)

Generator that yields the names of all job classes.

classmethod objects(*connection*)

Generator that yields all job classes.

class ll.orasql.**Job**

Bases: *OwnedSchemaObject*

Models a job (from the `dbms_scheduler` package) in the database.

classmethod names(*connection*, *owner=None*)

Generator that yields the names of all jobs.

classmethod objects(*connection*, *owner=None*)

Generator that yields all jobs.

class ll.orasql.**OracleFileResource**

Bases: *Resource*

An *OracleFileResource* wraps an Oracle database object (like a table, view, function, procedure etc.) in a file-like API for use with *ll.url*.

6.11.1 scripts – Oracle related scripts

This package contains the following scripts:

oracreate
: **oracreate** prints (or executes in another schema) the SQL of all objects in an Oracle database schema (i.e. all tables, procedures, functions, views, etc.)

oradrop
: **oradrop** prints (or executes) drop statements for all objects in an Oracle database schema.

oradelete
: **oradelete** prints (or executes) SQL for deleting all records from all tables in an Oracle database schema.

oragrant
: **oragrant** prints (or executes) grants statements from an Oracle database schema.

orafind
: **orafind** can be used to search for a string in all fields of all tables in an Oracle database schema.

oradiff
: **oradiff** can be used for finding the differences between two Oracle database schemas.

oramerge
: **oramerge** can be used for merging the changes between two Oracle database schemas into a third one.

These scripts can either be called via Pythons `-m` option:

```
$ python -mll.orasql.scripts.oracreate --help
```

or as a simple script installed in the search path:

```
$ oracreate --help
```

oracreate – Printing a schema definition

Purpose

oracreate prints the SQL of all objects in an Oracle database schema in a way that can be used to recreate the schema (i.e. objects will be ordered so that no errors happen for nonexistent objects during script execution). **oracreate** can also be used to actually recreate the schema.

Options

oracreate supports the following options:

connectstring

An Oracle connectstring.

-v, --verbose

Produces output (on stderr) while the database is read or written. (Allowed values are false, no, 0, true, yes or 1)

-c <mode>**, --color** <mode>

Should the output (when the -v option is used) be colored? If auto is specified (the default) then the output is colored if stderr is a terminal. Valid modes are yes, no or auto.

-s <flag>**, --seqcopy** <flag>

Outputs CREATE SEQUENCE statements for the existing sequences that have the current value of the sequence as the starting value (otherwise the sequences will restart with their initial value). (Valid flag values are false, no, 0, true, yes or 1)

-x <connectstring>**, --execute** <connectstring>

When the -x argument is given the SQL script isn't printed on stdout but executed in the database specfied as the -x argument.

-k <flag>**, --keepjunk** <flag>

If false (the default), database objects that have $ or SYS_EXPORT_SCHEMA_ in their name will be skipped (otherwise these objects will be included). (Valid flag values are false, no, 0, true, yes or 1)

-i <flag>**, --ignore** <flag>

If true, any exception that occurs while the database is read or written will be ignored. (Valid flag values are false, no, 0, true, yes or 1)

--format <format>

If --execute is not given, this determines the output format: Plain SQL (format sql), or PySQL (format pysql) which can be piped into *ll.pysql*.

--include <regexp>

Only include objects in the output if their name contains the regular expression.

--exclude <regexp>

Exclude objects from the output if their name contains the regular expression.

Examples

Print the content of the database schema user@db:

```
$ oracreate user/pwd@db >db.sql
```

Copy the database schema user@db to user2@db2:

```
$ oracreate user/pwd@db -x user2/pwd2@db2 -v
```

`oradrop` – Deleting a schema definition

Purpose

`oradrop` prints the drop statements for all objects in an Oracle database schema in the correct order (i.e. objects will be dropped so that no errors happen during script execution). **`oradrop`** can also be used to actually make the schema empty.

Options

`oradrop` supports the following options:

`connectstring`

An Oracle connectstring.

`-v` `<flag>`, **`--verbose`** `<flag>`

Produces output (on stderr) while the database is read or written. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-c` `<mode>`, **`--color`** `<mode>`

Should the output (when the *`-v`* option is used) be colored? If `auto` is specified (the default) then the output is colored if stderr is a terminal. Valid modes are `yes`, `no` or `auto`.

`-f` `<mode>`, **`--fks`** `<mode>`

Specifies how foreign keys from other schemas pointing to this schema should be treated: `keep` will not change the foreign keys in any way (this *will* lead to errors); `disable` will disable the foreign keys and `drop` will drop them completely.

`-x` `<flag>`, **`--execute`** `<flag>`

When the *`-x`* argument is given the SQL script isn't printed on stdout, but is executed directly in the schema specified via the *`connectstring`* option. Be careful with this: You *will* have an empty schema after `oradrop -x`. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-k` `<flag>`, **`--keepjunk`** `<flag>`

If false (the default), database objects that have `$` or `SYS_EXPORT_SCHEMA_` in their name will be skipped (otherwise these objects will be included in the output). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-i` `<flag>`, **`--ignore`** `<flag>`

If true, any exception that occurs while the database is read or written will be ignored. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`--format` `<format>`

If `--execute` is not given, this determines the output format: Plain SQL (format `sql`), or PySQL (format `pysql`) which can be piped into *`ll.pysql`*.

`--include` `<regexp>`

Only include objects in the output if their name contains the regular expression.

`--exclude` `<regexp>`

Exclude objects from the output if their name contains the regular expression.

`oradelete` – Deleting all records

Purpose

`oradelete` prints the delete statements for all tables in an Oracle database schema in the correct order (i.e. records will be deleted so that no errors happen during script execution). **`oradelete`** can also be used to actually make all tables empty.

Options

`oradelete` supports the following options:

`connectstring`

An Oracle connectstring.

`-v` `<flag>`, **`--verbose`** `<flag>`

Produces output (on stderr) while the database is read or written. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-c` `<flag>`, **`--color`** `<flag>`

Should the output (when the `-v` option is used) be colored? If `auto` is specified (the default) then the output is colored if stderr is a terminal. Valid modes are `yes`, `no` or `auto`.

`-s` `<flag>`, **`--sequences`** `<flag>`

Should sequences be reset to their initial values? (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-x` `<flag>`, **`--execute`** `<flag>`

When the *`-x`* argument is given the SQL script isn't printed on stdout, but is executed directly in the schema specified via the *`connectstring`* option. Be careful with this: You *will* have empty tables after `oradelete -x`. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-k` `<flag>`, **`--keepjunk`** `<flag>`

If false (the default), database objects that have `$` or `SYS_EXPORT_SCHEMA_` in their name will be skipped (otherwise these objects will be included in the output). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-i` `<flag>`, **`--ignore`** `<flag>`

If true, any exception that occurs while the database is read or written will be ignored. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-t` `<flag>`, **`--truncate`** `<flag>`

If given the script uses the `TRUNCATE` command instead of the `DELETE` command. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`--format` `<format>`

If `--execute` is not given, this determines the output format: Plain SQL (format `sql`), or PySQL (format `pysql`) which can be piped into *`ll.pysql`*.

`--include` `<regexp>`

Only include objects in the output if their name contains the regular expression.

`--exclude` `<regexp>`

Exclude objects from the output if their name contains the regular expression.

`oragrant` – Printing permissions for a schema

Purpose

`oragrant` prints all existing grants in an Oracle database schema. It can also be used to execute these grant statements directly.

Options

`oragrant` supports the following options:

`connectstring`

An Oracle connectstring.

`-v` `<flag>`, **`--verbose`** `<flag>`

Produces output (on stderr) while the database is read or written. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-c` `<mode>`, **`--color`** `<mode>`

Should the output (when the *`-v`* option is used) be colored? If `auto` is specified (the default) then the output is colored if stderr is a terminal. Valid modes are `yes`, `no` or `auto`.

`-x` `<connectstring>`, **`--execute`** `<connectstring>`

When the *`-x`* argument is given the SQL script isn't printed on stdout, but executed in the database specfied as the *`-x`* argument.

`-k` `<flag>`, **`--keepjunk`** `<flag>`

If false (the default), database objects that have `$` or `SYS_EXPORT_SCHEMA_` in their name will be skipped (otherwise these objects will be included in the output). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-i` `<flag>`, **`--ignore`** `<flag>`

If true, any exception that occurs while the database is read or written will be ignored. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-m` `<expr>`, **`--mapgrantee`** `<expr>`

A Python `dict` or `list` literal which will be evaluated. If the grantee is not in this list (or dictionary) no grant statement will be returned. If it's a dictionary and the grantee exists as a key, the privilege will be granted to the user specified as the value instead of the original one. The default is to grant all privileges to the original grantee.

`--format` `<format>`

If *`--execute`* is not given, this determines the output format: Plain SQL (format `sql`), or PySQL (format `pysql`) which can be piped into *`ll.pysql`*.

`--include` `<regexp>`

Only include objects in the output if their name contains the regular expression.

`--exclude` `<regexp>`

Exclude objects from the output if their name contains the regular expression.

Example

Grant all privileges that `alice` has in the schema `user@db` to `bob` in `user2@db2`:

```
$ oragrant user/pwd@db -x user2/pwd2@db2 -m '{"alice": "bob"}' -v
```

`orafind` – Finding records in a schema

Purpose

`orafind` can be used to search all tables in an Oracle database schema for a string.

Options

`orafind` supports the following options:

`connectstring`
An Oracle connectstring.

`searchstring`
The text to be searched for.

`tables`
Zero or more tables names. If any table names are specified the search will be limited to those tables. Otherwise all tables will be searched.

`-v` `<flag>`, **`--verbose`** `<flag>`
Produces output (on stderr) while the database is read or written. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-c` `<mode>`, **`--color`** `<mode>`
Should the output (when the `-v` option is used) be colored? If `auto` is specified (the default) then the output is colored if stderr is a terminal. Valid modes are `yes`, `no` or `auto`.

`-i` `<flag>`, **`--ignore-case`** `<flag>`
If true, the search will be case insensitve. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-r` `<flag>`, **`--read-lobs`** `<flag>`
If true, `CLOB`s will be read when printing search results. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

Example

Search for `spam` in all tables in the schema `user@db`. The search is case insensitive and `CLOB`s will be printed:

```
$ orafind user/pwd@db spam -i -r
```

`oradiff` – Diffing two schemas

Purpose

`oradiff` can be used to find the difference between two Oracle database schemas.

Options

`oradiff` supports the following options:

`connectstring1`
Oracle connectstring for the first database schema.

`connectstring2`
Oracle connectstring for the second database schema.

`-v` `<flag>`, **`--verbose`** `<flag>`
Produces output (on stderr) while the database is read. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-c` `<mode>`, **`--color`** `<mode>`
Should the output (when the *`-v`* option is used) be colored? If `auto` is specified (the default) then the output is colored if stderr is a terminal. Valid modes are `yes`, `no` or `auto`.

`-m` `<mode>`, **`--mode`** `<mode>`
Specifies how the differences should be shown. `brief` only prints whether objects are different (or which ones exist in only one of the databases); `udiff` outputs the differences in "unified diff" format and `full` outputs the object from the second schema if they differ (i.e. it outputs the script that must be executed to copy the differences from schema 2 to schema 1).

`--format` `<format>`
If *`--mode`* is `full`, this determines the output format: Plain SQL (format `sql`), or PySQL (format `pysql`) which can be piped into *`ll.pysql`*.

`-n` `<context>`, **`--context`** `<context>`
The number of context lines in unified diff mode (i.e. the number of unchanged lines above and below each block of changes; the default is 2)

`-k` `<flag>`, **`--keepjunk`** `<flag>`
If false (the default), database objects that have `$` or `SYS_EXPORT_SCHEMA_` in their name will be skipped (otherwise these objects will be included in the output). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

`-b` `<mode>`, **`--blank`** `<mode>`
The *`-b`* option specifies how whitespace in the database objects should be compared. With `literal` all whitespace is significant, with `trail` trailing whitespace will be ignore, with `lead` leading whitespace will be ignored, with `both` trailing and leading whitespace will be ignored and with `collapse` trailing and leading whitespace will be ignored and stretches of whitespace will be treated as a single space.

Example

Compare the schemas `user@db` and `user2@db2`, collapsing whitespace and using unified diff mode with 5 context lines:

```
$ oradiff user/pwd@db user2/pwd2@db2 -bcollapse -mudiff -n5 -v >db.diff
```

`oramerge` – Three-way merging of schemas

Purpose

oramerge can be used for merging the changes between two Oracle database schemas into a third one. Depending on the existance/non-existance of schema objects in the three schemas **oramerge** does the right thing. If a schema objects exists in all three schemas, the external tool **merge3** will be used for creating a merged version of the object (except for tables where the appropriate `ALTER TABLE` statements will be output if possible).

Options

oramerge supports the following options:

connectstring1
: Old version of database schema

connectstring2
: New version of database schema

connectstring3
: Schema into which changes should be merged

-v <flag>, **--verbose** <flag>
: Produces output (on stderr) while the database is read or written. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-c <mode>, **--color** <mode>
: Should the output (when the `-v` option is used) be colored? If `auto` is specified (the default) then the output is colored if stderr is a terminal. Valid modes are `yes`, `no` or `auto`.

-k <flag>, **--keepjunk** <flag>
: If false (the default), database objects that have `$` or `SYS_EXPORT_SCHEMA_` in their name will be skipped (otherwise these objects will be considered as merge candidates). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

Example

Output a script that merges the changes between `user@db` and `user2@db2` into `user3@db3`:

```
$ oramerge user/pwd@db user2/pwd2@db2 user3/pwd3@db3 -v >db.sql
```

`orareindex` – Recreating indexes/constraints

Purpose

orareindex recreates/rebuilds all indexes and unique constraints in an Oracle database schema.

Options

orareindex supports the following options:

connectstring

An Oracle connectstring.

-v <flag>, **--verbose** <flag>

Produces output (on stderr) while the database is read or written. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-c <mode>, **--color** <mode>

Should the output (when the *-v* option is used) be colored? If `auto` is specified (the default) then the output is colored if stderr is a terminal. Valid modes are `yes`, `no` or `auto`.

-x <flag>, **--execute** <flag>

When the *-x* argument is given the SQL script isn't printed on stdout, but is executed directly in the schema specified via the *connectstring* option. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-r <flag>, **--rebuild** <flag>

If given, the script uses `ALTER INDEX ... REBUILD` to rebuild indexes instead of dropping and recreating them. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

--format <format>

If *--execute* is not given, this determines the output format: Plain SQL (format `sql`), or PySQL (format `pysql`) which can be piped into *ll.pysql*.

`oracycles` – Finding cyclic foreign keys

Purpose

oracycles checks the foreign key references in an Oracle database for cyclic references, either direct ones (i.e. foreign keys that reference the same table) or indirect ones.

Options

oracycles supports the following options:

connectstring

An Oracle connectstring.

-v <flag>, **--verbose** <flag>

Produces output (on stderr) while the database is read or foreign keys are checked for cycles. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-c <mode>, **--color** <mode>

Should the output (when the *-v* option is used) be colored? If `auto` is specified (the default) then the output is colored if it goes to a terminal. Valid modes are `yes`, `no` or `auto`.

6.12 ll.nightshade – Using Oracle with CherryPy

This module provides a class *Call* that allows you to use Oracle PL/SQL procedures/functions as CherryPy response handlers. A *Call* objects wraps a *ll.orasql.Procedure* or *ll.orasql.Function* object from the *ll.orasql* module.

For example, you might have the following PL/SQL function:

```
create or replace function helloworld
(
   who varchar2
)
return varchar2
as
begin
   return '<html><head><h>Hello ' || who || '</h></head><body><h1>Hello, ' || who ||
↪'!</h1></body></html>';
end;
```

Using this function as a CherryPy response handler can be done like this:

```
import cherrypy

from ll import orasql, nightshade

proc = nightshade.Call(orasql.Function("helloworld"), connectstring="user/pwd")

class HelloWorld:
   @cherrypy.expose
   def default(self, who="World"):
      cherrypy.response.headers["Content-Type"] = "text/html"
      return proc(who=who)

cherrypy.quickstart(HelloWorld())
```

class ll.nightshade.**UTC**

Bases: tzinfo

Timezone object for UTC

ll.nightshade.**getnow**()

Get the current date and time as a datetime.datetime object in UTC with timezone info.

ll.nightshade.**httpdate**(*dt*)

Return a string suitable for a "Last-Modified" and "Expires" header.

dt is a datetime.datetime object. If dt.tzinfo is None dt is assumed to be in the local timezone (using the current UTC offset which might be different from the one used by dt).

class ll.nightshade.**Connect**

Bases: object

Connect objects can be used as decorators that wraps a function that needs a database connection.

If calling the wrapped function results in a database exception that has been caused by a lost connection to the database or similar problems, the function is retried with a new database connection.

__init__(*connectstring=None*, *pool=None*, *retry=3*, ***kwargs*)

Create a new parameterized *Connect* decorator. Either connectstring or pool (a database pool object) must be specified. retry specifies how often to retry calling the wrapped function after a database exception. kwargs will be passed on to the connect() call.

class `ll.nightshade.`**`Call`**

Bases: `object`

Wrap an Oracle procedure or function in a CherryPy handler.

A *`Call`* object wraps a procedure or function object from *`ll.orasql`* and makes it callable just like a CherryPy handler.

`__init__`(*callable*, *connection*)

Create a *`Call`* object wrapping the function or procedure `callable`.

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

Call the procedure/function with the arguments `args` and `kwargs` mapping Python function arguments to Oracle procedure/function arguments. On return from the procedure the `c_out` parameter is mapped to the CherryPy response body, and the parameters `p_expires` (the number of days from now), `p_lastmodified` (a date in UTC), `p_mimetype` (a string), `p_encoding` (a string), `p_etag` (a string) and `p_cachecontrol` (a string) are mapped to the appropriate CherryPy response headers. If `p_etag` is not specified a value is calculated.

If the procedure/function raised a PL/SQL exception with a code between 20200 and 20599, 20000 will be subtracted from this value and the resulting value will be used as the HTTP response code, i.e. 20404 will give a "Not Found" response.

6.13 `ll.scripts` – UL4 templates and URLs

This package contains the following scripts:

`rul4`

`rul4` renders an UL4 template. The available template variables allow system commands and database access. Supported databases are Oracle, SQLite, MySQL and Redis.

`uls`

`uls` is an URL-enabled version of the **`ls`** command for listing directory content. It uses *`ll.url`* and supports `ssh` and `oracle` URLs (via *`ll.orasql`*).

`ucp`

`ucp` is an URL-enabled version of the **`cp`** command for copying files (and file-like objects). It supports `ssh` and `oracle` URLs for both source and target files.

`ucat`

`ucat` is an URL-enabled version of the **`cat`** command for printing files (and file-like objects).

`udiff`

`udiff` is an URL-enabled version of the **`diff`** command for showing differences between two files or directories.

These scripts can either be called via Pythons `-m` option:

```
$ python -mll.scripts.rul4 --help
```

or as a simple script installed in the search path:

```
$ rul4 --help
```

6.13.1 `rul4` – Rendering UL4 templates

Purpose

rul4 is a script that can be used to render an UL4 template.

The `globals` object

Inside the template the object `globals` (an instance of the class *Globals*) will be available to make database connections, load and save files, compile templates, access environment variables and parameters etc. However access to those features can be switched off via command line options.

Options

rul4 supports the following options:

templates

One or more template files. A file named `-` will be treated as standard input. The first file in the list is the main template, i.e. the one that gets rendered. All templates will be available in the main template as the `globals.templates` dictionary. The keys are the base names of the files (i.e. `foo.ul4` will be `globals.templates.foo`; stdin will be `globals.templates.stdin`).

--oracle <flag>

Provide the method *Globals.oracle()* (as `globals.oracle`) to the template? If switched off `globals.oracle` will be `None`.

(Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`; the default is `true`)

--sqlite <flag>

Provide the method *Globals.sqlite()* (as `globals.sqlite`) to the template? If switched off `globals.sqlite` will be `None`.

(Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`; the default is `true`)

--mysql <flag>

Provide the method *Globals.mysql()* (as `globals.mysql`) to the template? If switched off `globals.mysql` will be `None`.

(Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`; the default is `true`)

--redis <flag>

Provide the method *Globals.redis()* (as `globals.redis`) to the template? If switched off `globals.redis` will be `None`.

(Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`; the default is `true`)

--system <flag>

Provide the method *Globals.system()* (as `globals.system`) to the template? If switched off `globals.system` will be `None`.

(Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`; the default is `true`)

--load <flag>

Provide the method *Globals.load()* (as `globals.load`) to the template? If switched off `globals.load` will be `None`.

(Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`; the default is `true`)

--save <flag>

Provide the method *Globals.save()* (as `globals.save`) to the template? If switched off `globals.save` will be `None`.

(Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`; the default is `true`)

--compile <flag>

Provide the method *Globals.compile()* (as `globals.compile`) to the template? If switched off `globals.compile` will be `None`.

(Allowed values are `false`, `no`, `0`, `true`, `yes` or `1`; the default is `true`)

-e <encoding>, **--encoding** <encoding>

The encoding of the templates files (default `utf-8`)

-w <value>, **--whitespace** <value>

Specifies how to handle whitespace in the template (Allowed values are `keep`, `strip`, or `smart`). This can of course be overwritten with the template tag `<?whitespace ...?>` in the template files.

-D, --define

Defines an additional variable that will be available inside the template (e.g. the variable `foo` will be available as `globals.vars.foo`). `-D` can be specified multiple times. The following formats are supported:

var
Defines `var` as an empty string;

var=value
Defines `var` as the string `value`;

var:type
Defines `var` as an empty variable of the type `type`;

var:type=value
Defines `var` as a variable of the type `type` with the value `value`.

`type` can be any of the following:

int
`value` is an integer value.

float
`value` is a float value.

bool
`value` is a boolean value. `0`, `no`, `false`, `False` or the empty string will be recognized as false and `1`, `yes`, `true` or `True` will be recognized as true.

str
`value` is a string.

oracle
`value` will be a connection to an Oracle database, e.g.:

```
-Ddb:oracle=user/password@database
```

sqlite
`value` is a connection to an SQLite database.

mysql
`value` is a connection to a MySQL database.

redis
`value` will be a connection to an Redis database, e.g.:

```
-Ddb:redis=192.168.123.1:6379/42
```

The port (i.e. the 6379 in the above value) is optional and defaults to 6379. The database number (i.e. the 42 in the above value) is also optional and defaults to 0.

Example

This example shows how to connect to an Oracle database and output the content of a `person` table into an XML file.

Suppose we have a database table that looks like this:

```
create table person
(
   id integer not null,
   firstname varchar2(200),
   lastname varchar2(200)
);
```

Then we can use the following template to output the table into an XML file:

```
<?xml version='1.0' encoding='utf-8'?>
<?code db = globals.oracle("user/password@database")?>
<persons>
   <?for p in db.query("select id, firstname, lastname from person order by 3, 2")?>
      <person id="<?printx p.id?>">
         <firstname><?printx p.firstname?></firstname>
         <lastname><?printx p.lastname?></lastname>
      </person>
   <?end for?>
</persons>
```

If we put the template into the file `person.ul4` we can call **`rul4`** like this:

```
$ rul4 person.ul4 >person.xml
```

We could also pass the connection to our database via the *-D* option and disallow the script to make any database connections itself or execute any system commands:

```
$ rul4 person.ul4 -Ddb:oracle=user/password@database --oracle=0 --sqlite=0 --mysql=0 -
↪-redis=0 --system=0 >person.xml
```

Then the template can use the Oracle connection object `db` directly.

API

class `ll.scripts.rul4.`**`Connection`**

Bases: `object`

A *`Connection`* object provides a database connection to an UL4 template.

To execute SQL the two methods *`query()`* and *`execute()`* are provided.

Calling functions or procedures with out parameters can be done with variable objects that can be created with the methods *`int()`*, *`number()`*, *`str()`*, *`clob()`* and *`date()`*. The resulting value of the out parameter is available from the `value` attribute of the variable object. The following example creates a function, calls it to get at the result and drops it again:

```
<?code db = oracle.connect('user/password@database')?>
<?code db.execute('''
   create or replace function ul4test(p_arg integer)
```

(continues on next page)

(continued from previous page)

```
   return integer
   as
   begin
      return 2*p_arg;
   end;
'''))?>
<?code vout = db.int()?>
<?code db.execute('begin ', vout, ' := ul4test(42); end;')?>
<?print vout.value?>
<?code db.execute('drop function ul4test')?>
```

A *Connection* object can be created with the methods *Globals.mysql()* or *Globals.sqlite()*.

query(**queryparts*)

Execute the query passed in and return an iterator over the resulting records.

At least one positional argument is required. Arguments alternate between fragments of the SQL query and parameters that will be embedded in the query. For example:

```
<?code db = globals.oracle("user/pwd@db")?>
<?code name = "Bob"?>
<ul>
   <?for p in db.query(
      "select * from person where firstname=",
      name,
      " or lastname=",
      name
   )?>
      <li><?print p.firstname?> <?print p.lastname?></li>
   <?end for?>
</ul>
```

The records returned from *query()* are dict-like objects mapping field names to field values.

queryone(**queryparts*)

Execute the query passed in and return the first result record (or `None` if the query didn't output any record). `queryparts` is handled the same way as *query()* does.

execute(**queryparts*)

Similar to *query()* and *queryone()*, but doesn't doesn't return a result. This can be used to call functions or procedures.

str(*value=None*)

Create a variable that can be used for OUT parameters of type `varchar`.

clob(*value=None*)

Create a variable that can be used for OUT parameters of type `clob`.

int(*value=None*)

Create a variable that can be used for OUT parameters of type `integer`.

number(*value=None*)

Create a variable that can be used for OUT parameters of type `number`.

date(*value=None*)

Create a variable that can be used for OUT parameters of type `date`.

class `ll.scripts.rul4.`**OracleConnection**

Bases: *Connection*

OracleConnection is a subclass of *Connection* that implements functionality that is specific to Oracle databases (e.g. support for variables). The inferface is the same as *Connection*s.

An *OracleConnection* object can be created with the method *Globals.oracle()*.

class ll.scripts.rul4.**RedisConnection**

Bases: object

A connection to a Redis database. A *RedisConnection* object provides the methods *get()* to read data from the database and *set()* to write data to the database.

Example:

```
<?code db = redis.connect("192.168.123.42/1")?>
<?code value = db.get("key")?>
<?if value is None?>
   <?code value = "foobar"?>
   <?code db.put("key", value, timedelta(seconds=10*60))?>
<?end if?>
```

get(*key*)

Return the value for the key `key` or `None` if the key doesn't exist.

set(*key*, *data*, *timeout=None*)

Store the string value `data` under the key `key`.

If `timeout` is `None` the value will be stored indefinitely. Otherwise it specifies when the value will expire. `timeout` can be an integer (the number of seconds) or a `timedelta` object.

class ll.scripts.rul4.**Globals**

Bases: object

An instance of the *Globals* class will be passed to the main template as the `globals` variable. The following attributes will be accessible to UL4 templates:

templates

[dictionary]
A dictionary containing the templates specified on the command line. This will include the main template.

vars

[dictionary]
A dictionary containing the variables that have been specified via the *-D*/*--define* option.

encoding

[string]
The encoding that will be used for output (this is the same as `sys.stdout.encoding`, so it can be set with the environment variable `PYTHONIOENCODING`).

env

[dictionary]
A reference to `os.environ`.

Furthermore the following methods can be called from UL4 templates: *error()*, *log()*, *oracle()*, *mysql()*, *sqlite()*, *redis()*, *system()*, *load()*, *save()* and *compile()*.

from_args(*args*)

Sets the attributes of `self` from the object `args` (which must be an instance of `argparse.Namespace`).

Returns the main template.

error(*message*, *ast=None*)

Can be called to output an error message and abort template execution. The signature is:

```
globals.error(message, ast=None)
```

`message` is the error message and `ast` can be an AST node from an UL4 template syntax tree to print an error message that originates from that node.

log(**args*, *sep=' '*, *end=\n'*, *flush=False*)

Logs `args` to `sys.stderr`.

The parameters `sep`, `end` and `flush` have the same meaning as for `print()`.

oracle(*connectstring*)

Return an `OracleConnection` object for the Oracle connect string passed in:

```
<?code db = globals.oracle("user/password@database")?>
<?for row in db.query("select sysdate as sd from dual")?>
   <?print row.sd?>
<?end for?>
```

mysql(*connectstring*)

Return a `Connection` object to a MySQL database for the connectstring passed in. The format of the connect string is:

```
user/password@host/database
```

sqlite(*connectstring*)

Return a `Connection` object to an SQLite database for the connectstring passed in. The connectstring will be passed directly to `sqlite3.connect()`.

redis(*connectstring*)

Return a `RedisConnection` object, which provides a connection to a Redis database. The connectstring has the format:

```
host:port/db
```

`port` is optional and defaults to 6379. `db` is optional too and defaults to 0.

system(*cmd*)

Execute the system command `cmd` and returns its output, e.g. the template:

```
<?print globals.system("whoami")?>
```

will output the user name.

load(*filename*, *encoding='utf-8'*)

Read a file from disk and returns the content. `filename` is the filename and `encoding` is the encoding of the file. The encoding parameter is optional and defaults to `"utf-8"`:

```
<?code data = globals.load("/home/user/data.txt", "iso-8859-1")?>
```

save(*filename*, *data*, *encoding='utf-8'*)

Save the string `data` to a file on disk. `filename` is the filename and `encoding` is the encoding of the file. The encoding parameter is optional and defaults to `"utf-8"`:

```
<?code globals.save("/home/user/data.txt", "foo\nbar\n", "iso-8859-1")?>
```

compile(*source*, *name=None*, *whitespace='keep'*, *signature=None*)

Compile the UL4 source `source` into a `Template` object and return it. All other parameters are passed to the `Template` constructor too.

6.13.2 `uls` – Listing directories

Purpose

uls is a script that lists the content of directories. It is an URL-enabled version of the **ls** system command. Via *ll.url* and *ll.orasql* **uls** supports `ssh` and `oracle` URLs.

Options

uls supports the following options:

urls

Zero or more URLs. If no URL is given the current directory is listed.

-c <mode>, **--color** <mode>

Should the output be colored? If `auto` is specified (the default) then the output is colored if stdout is a terminal. Valid modes are `yes`, `no` or `auto`

-1 <flag>, **--one** <flag>

Force output to be one URL per line. The default is to output URLs in multiple columns (as many as fit on the screen). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-l <flag>, **--long** <flag>

Output in long format: One URL per line containing the following information: file mode, owner name, group name, number of bytes in the file, number of links, URL. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-s <flag>, **--human-readable-sizes** <flag>

Output the file size in human readable form (e.g. `42M` for 42 megabytes). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-r <flag>, **--recursive** <flag>

List directories recursively. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-w <width>, **--spacing** <width>

The number of spaces (or padding characters) between columns (only relevant for multicolumn output, i.e. when neither *--long* nor *--one* is specified).

-P <string>, **--padding** <string>

The characters used for padding output in multicolumn or long format.

-i <pattern(s)>, **--include** <pattern(s)>

Only list files that match one of the specified patterns.

-e <pattern(s)>, **--exclude** <pattern(s)>

Don't list files that match one of the specified patterns.

--enterdir <pattern(s)>

Only enter directories that match one of the specified patterns.

--skipdir <pattern(s)>

Skip directories that match one of the specified patterns.

--ignorecase <flag>

Perform case-insensitive pattern matching. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

Examples

List the current directory:

```
$ uls
CREDITS.rst   installer.bmp   NEWS.rst           scripts/    test/
demos/        Makefile        OLDMIGRATION.rst   setup.cfg
docs/         MANIFEST.in     OLDNEWS.rst        setup.py
INSTALL.rst   MIGRATION.rst   README.rst         src/
```

List the current directory in long format with human readable file sizes:

```
$ uls -s -l
rw-r--r--  walter    staff      1114    1  2008-01-06 22:27:15  CREDITS.rst
rwxr-xr-x  walter    staff       170    5  2007-12-03 23:35:33  demos/
rwxr-xr-x  walter    staff       340   10  2010-12-08 16:48:53  docs/
rw-r--r--  walter    staff        2K    1  2010-12-08 16:48:53  INSTALL.rst
rw-r--r--  walter    staff       35K    1  2007-12-03 23:35:33  installer.bmp
rw-r--r--  walter    staff      1763    1  2011-01-21 17:22:32  Makefile
rw-r--r--  walter    staff       346    1  2011-02-25 11:13:18  MANIFEST.in
rw-r--r--  walter    staff       34K    1  2011-03-04 13:48:35  MIGRATION.rst
rw-r--r--  walter    staff      107K    1  2011-03-04 18:18:42  NEWS.rst
rw-r--r--  walter    staff        8K    1  2010-12-08 16:48:53  OLDMIGRATION.rst
rw-r--r--  walter    staff       75K    1  2010-12-08 16:48:53  OLDNEWS.rst
rw-r--r--  walter    staff        3K    1  2010-12-08 16:48:53  README.rst
rwxr-xr-x  walter    staff       578   17  2010-12-08 16:48:53  scripts/
rw-r--r--  walter    staff        39    1  2010-12-08 16:48:53  setup.cfg
rw-r--r--  walter    staff        7K    1  2011-03-03 13:33:21  setup.py
rwxr-xr-x  walter    staff       136    4  2007-12-04 01:43:13  src/
rwxr-xr-x  walter    staff        2K   68  2011-03-03 13:27:46  test/
```

Recursively list a remote directory:

```
$ uls ssh://user@www.example.org/~/dir/ -r
...
```

Recursively list the schema objects in an Oracle database:

```
$ uls oracle://user:pwd@oracle.example.org/ -r
...
```

6.13.3 ucp – Copying files/directories

Purpose

ucp is a script that copies files or directories. It is an URL-enabled version of the **cp** system command. Via *ll.url* and *ll.orasql* **ucp** supports `ssh` and `oracle` URLs.

Options

ucp supports the following options:

urls

Two or more URLs. If more than two URLs are given or the last URL refers to an existing directory, the last URL is the target directory. All other sources are copied into this target directory. Otherwise one file is copied to another file.

-v <flag>, **--verbose** <flag>

Give a report during the copy process about the files copied and their sizes? (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-c <mode>, **--color** <mode>

Should the output be colored? If `auto` is specified (the default) then the output is colored if stdout is a terminal. Valid modes are `yes`, `no` or `auto`.

-u <user>, **--user** <user>

A user id or name. If given **ucp** will change the owner of the target files.

-g <group>, **--group** <group>

A group id or name. If given **ucp** will change the group of the target files.

-r <flag>, **--recursive** <flag>

Copies files recursively. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-x <flag>, **--ignoreerrors** <flag>

Ignores errors occurring during the copy process (otherwise the copy process is aborted). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-i <pattern(s)>, **--include** <pattern(s)>

Only copy files that match one of the specified patterns.

-e <pattern(s)>, **--exclude** <pattern(s)>

Don't copy files that match one of the specified patterns.

--enterdir <pattern(s)>

Only enter directories that match one of the specified patterns.

--skipdir <pattern(s)>

Skip directories that match one of the specified patterns.

--ignorecase <flag>

Perform case-insensitive pattern matching? (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

Examples

Copy one file to another:

```
$ ucp foo.txt bar.txt
```

Copy a file into an existing directory:

```
$ ucp foo.txt dir/
```

Copy multiple files into a new or existing directory (and give a progress report):

```
$ ucp foo.txt bar.txt baz.txt dir/ -v
ucp: foo.txt -> dir/foo.txt (1,114 bytes)
ucp: bar.txt -> dir/bar.txt (2,916 bytes)
ucp: baz.txt -> dir/baz.txt (35,812 bytes)
```

Recursively copy the schema objects in an Oracle database to a local directory:

```
$ ucp oracle://user:pwd@oracle.example.org/ db/ -r
```

Recursively copy the schema objects in an Oracle database to a remote directory:

```
$ ucp oracle://user:pwd@oracle.example.org/ ssh://user@www.example.org/~/db/ -r
```

6.13.4 `ucat` – Printing files

Purpose

ucat is a script for printing files. It is an URL-enabled version of the **cat** system command. Via *ll.url* and *ll.orasql* **ucat** supports `ssh` and `oracle` URLs.

Options

ucat supports the following options:

urls
: One or more URLs to be printed.

-r <flag>, **--recursive** <flag>
: Prints directory content recursively. (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-x <flag>, **--ignoreerrors** <flag>
: Ignores file i/o errors occurring during the output process (otherwise the script will be aborted). (Valid flag values are `false`, `no`, `0`, `true`, `yes` or `1`)

-i <pattern(s)>, **--include** <pattern(s)>
: Only print files whose name matches one of the specified patterns.

-e <pattern(s)>, **--exclude** <pattern(s)>
: Don't print files whose name matches one of the specified patterns.

--enterdir <pattern(s)>
: Only enter directories whose name matches one of the specified patterns.

--skipdir <pattern(s)>
: Don't enter directories whose name matches one of the specified patterns.

Examples

Print a file:

```
$ ucat foo.txt
```

Print a remote file:

```
$ ucat ssh://user@www.example.org/~/foo.txt
```

Print the SQL source code of the procedure FOO in an Oracle database:

```
$ ucat oracle://user:pwd@oracle.example.org/procedure/FOO
```

6.13.5 udiff – Diffing files/directories

Purpose

udiff is a script that can be used to show the differences between files or directories. It is an URL-enabled version of the **diff** system command. Via *ll.url* and *ll.orasql* **udiff** supports ssh and oracle URLs.

Options

udiff supports the following options:

url1

The first URL to be compared (Note that a trailing / is required for directories).

url2

The second URL to be compared (Note that a trailing / is required for directories).

--encoding <encodingname>

The encoding name to use for decoding files (default utf-8).

--error <errorhandlingname>

Encoding error handling to use for reading text files (e.g. strict, replace, ignore, xmlcharrefreplace or backslashreplace; default replace).

-c <mode>, **--color** <mode>

Should the output of **udiff** be colored? The default auto uses coloring if stdout supports it. Valid modes are yes, no or auto.

-v <flag>, **--verbose** <flag>

Prints which files are compared before the comparison. When false **udiff** will be silent as long as no differences are detected. (Valid flag values are false, no, 0, true, yes or 1)

-r <flag>, **--recursive** <flag>

Compare directories recursively. (Valid flag values are false, no, 0, true, yes or 1)

-i <pattern(s)>, **--include** <pattern(s)>

Only compares files whose name matches one of the specified patterns.

-e <pattern(s)>, **--exclude** <pattern(s)>

Don't compares files whose name matches one of the specified patterns.

--enterdir <pattern(s)>

Only enter directories whose name matches one of the specified patterns.

--skipdir <pattern(s)>

Don't enter directories whose name matches one of the specified patterns.

-n <integer>, **--context** <integer>

How many lines of copied context to show (default 2).

-b <mode>, **--blank** <mode>

How to compare whitespace within lines. `literal` compares whitespace literally. `trail` ignores differences in trailing whitespace, `lead` ignores differences in leading whitespace, `both` ignores both leading and trailing whitespace and `collapse` collapses whitespace into a single space before comparing lines.

Examples

Compare two files:

```
$ udiff foo.txt bar.txt
```

Recursively compare two directories, but skip the `.git` directory:

```
$ udiff foo/ bar/ -r --skipdir=.git
```

Recursively compare two Oracle schemas:

```
$ udiff oracle://user1:pwd@database1  oracle://user2:pwd@database2 -r
```

6.14 Installation

6.14.1 Requirements

To use XIST you need the following software packages:

- Python 3.6;
- cssutils;
- Pillow (if you want to use automatic image size calculation);
- lxml (if you want to parse "broken" HTML; at least version 3.0);
- cx_Oracle (if you want to use *ll.orasql*);
- pytest (if you want to run the test suite)
- execnet (if you want to use ssh URLs)
- and a C compiler supported by pip, if you want to install the source distribution.

6.14.2 Installation

pip is used for installation so you can install this package with the following command:

```
$ pip install ll-xist
```

For some versions a Windows distribution is provided. To install it, double click it, and follow the instructions.

6.15 Downloads

You can download XIST from the Cheeseshop, go directly to the HTTP download directory or choose one of the following archives:

6.15.1 5.75.1 (released 04/11/2024)

File	Type	Size
ll-xist-5.75.1.tar.gz	Source	742K
ll_xist-5.75.1-cp310-cp310-win32.whl	Windows wheel (Python 3.10)	559K
ll_xist-5.75.1-cp310-cp310-win_amd64.whl	Windows wheel (Python 3.10)	561K
ll_xist-5.75.1-cp311-cp311-win32.whl	Windows wheel (Python 3.11)	558K
ll_xist-5.75.1-cp311-cp311-win_amd64.whl	Windows wheel (Python 3.11)	561K
ll_xist-5.75.1-cp312-cp312-win32.whl	Windows wheel (Python 3.12)	559K
ll_xist-5.75.1-cp312-cp312-win_amd64.whl	Windows wheel (Python 3.12)	561K
ll_xist-5.75.1-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	560K
ll_xist-5.75.1-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	562K
ll_xist-5.75.1-cp39-cp39-macosx_10_9_x86_64.whl	Mac wheel (Python 3.9)	557K
ll_xist-5.75.1-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	560K
ll_xist-5.75.1-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	562K

6.15.2 5.75 (released 11/21/2023)

File	Type	Size
ll-xist-5.75.tar.gz	Source	743K
ll_xist-5.75-cp39-cp39-macosx_10_9_x86_64.whl	Mac wheel (Python 3.9)	558K

6.15.3 5.74 (released 03/01/2023)

File	Type	Size
ll-xist-5.74.tar.gz	Source	740K
ll-xist-5.74.win-amd64-py3.10.msi	Windows installer (Python 3.10)	1336K
ll-xist-5.74.win32-py3.10.msi	Windows installer (Python 3.10)	1284K
ll_xist-5.74-cp310-cp310-win32.whl	Windows wheel (Python 3.10)	557K
ll_xist-5.74-cp310-cp310-win_amd64.whl	Windows wheel (Python 3.10)	560K
ll_xist-5.74-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	557K
ll_xist-5.74-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	560K
ll_xist-5.74-cp39-cp39-macosx_10_9_x86_64.whl	Mac wheel (Python 3.9)	559K
ll_xist-5.74-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	557K
ll_xist-5.74-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	560K

6.15.4 5.73.2 (released 08/16/2022)

File	Type	Size
ll-xist-5.73.2.tar.gz	Source	739K
ll-xist-5.73.2.win-amd64-py3.10.msi	Windows installer (Python 3.10)	1336K
ll-xist-5.73.2.win32-py3.10.msi	Windows installer (Python 3.10)	1284K
ll_xist-5.73.2-cp310-cp310-win32.whl	Windows wheel (Python 3.10)	557K
ll_xist-5.73.2-cp310-cp310-win_amd64.whl	Windows wheel (Python 3.10)	559K
ll_xist-5.73.2-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	557K
ll_xist-5.73.2-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	560K
ll_xist-5.73.2-cp39-cp39-macosx_10_9_x86_64.whl	Mac wheel (Python 3.9)	559K
ll_xist-5.73.2-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	557K
ll_xist-5.73.2-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	559K

6.15.5 5.73.1 (released 08/10/2022)

File	Type	Size
ll-xist-5.73.1.tar.gz	Source	738K
ll_xist-5.73.1-cp39-cp39-macosx_10_9_x86_64.whl	Mac wheel (Python 3.9)	559K

6.15.6 5.73 (released 08/10/2022)

File	Type	Size
ll-xist-5.73.tar.gz	Source	739K
ll_xist-5.73-cp39-cp39-macosx_10_9_x86_64.whl	Mac wheel (Python 3.9)	560K

6.15.7 5.72 (released 08/04/2022)

File	Type	Size
ll-xist-5.72.tar.gz	Source	735K
ll_xist-5.72-cp39-cp39-macosx_10_9_x86_64.whl	Mac wheel (Python 3.9)	556K

6.15.8 5.71 (released 07/08/2022)

File	Type	Size
ll-xist-5.71.tar.gz	Source	735K
ll_xist-5.71-cp39-cp39-macosx_10_9_x86_64.whl	Mac wheel (Python 3.9)	556K

6.15.9 5.70 (released 03/11/2022)

File	Type	Size
ll-xist-5.70.tar.gz	Source	733K
ll_xist-5.70-cp310-cp310-macosx_12_0_x86_64.whl	Mac wheel (Python 3.10)	554K

6.15.10 5.69.1 (released 12/13/2021)

(no files for this version)

6.15.11 5.69 (released 11/17/2021)

File	Type	Size
ll-xist-5.69.tar.gz	Source	731K
ll-xist-5.69.win-amd64-py3.10.msi	Windows installer (Python 3.10)	1332K
ll-xist-5.69.win32-py3.10.msi	Windows installer (Python 3.10)	1280K
ll_xist-5.69-cp310-cp310-macosx_11_0_x86_64.whl	Mac wheel (Python 3.10)	554K
ll_xist-5.69-cp310-cp310-win32.whl	Windows wheel (Python 3.10)	553K
ll_xist-5.69-cp310-cp310-win_amd64.whl	Windows wheel (Python 3.10)	556K
ll_xist-5.69-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	553K
ll_xist-5.69-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	556K
ll_xist-5.69-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	553K
ll_xist-5.69-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	556K

6.15.12 5.68.1 (released 09/23/2021)

File	Type	Size
ll-xist-5.68.1.tar.gz	Source	730K
ll-xist-5.68.1.win-amd64-py3.10.msi	Windows installer (Python 3.10)	1332K
ll-xist-5.68.1.win32-py3.10.msi	Windows installer (Python 3.10)	1280K
ll_xist-5.68.1-cp310-cp310-win32.whl	Windows wheel (Python 3.10)	553K
ll_xist-5.68.1-cp310-cp310-win_amd64.whl	Windows wheel (Python 3.10)	555K
ll_xist-5.68.1-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	556K
ll_xist-5.68.1-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	559K
ll_xist-5.68.1-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	553K
ll_xist-5.68.1-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	556K
ll_xist-5.68.1-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	553K
ll_xist-5.68.1-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	556K
ll_xist-5.68.1-cp39-cp39-macosx_11_0_x86_64.whl	Mac wheel (Python 3.9)	554K
ll_xist-5.68.1-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	553K
ll_xist-5.68.1-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	555K

6.15.13 5.68 (released 08/04/2021)

File	Type	Size
ll-xist-5.68.tar.gz	Source	730K
ll_xist-5.68-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	556K
ll_xist-5.68-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	559K
ll_xist-5.68-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	553K
ll_xist-5.68-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	556K
ll_xist-5.68-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	553K
ll_xist-5.68-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	556K
ll_xist-5.68-cp39-cp39-macosx_11_0_x86_64.whl	Mac wheel (Python 3.9)	554K
ll_xist-5.68-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	553K
ll_xist-5.68-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	555K

6.15.14 5.67.2 (released 06/30/2021)

File	Type	Size
ll-xist-5.67.2.tar.gz	Source	734K
ll_xist-5.67.2-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	562K
ll_xist-5.67.2-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	565K
ll_xist-5.67.2-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	559K
ll_xist-5.67.2-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	562K
ll_xist-5.67.2-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	559K
ll_xist-5.67.2-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	562K
ll_xist-5.67.2-cp39-cp39-macosx_10_15_x86_64.whl	Mac wheel (Python 3.9)	561K
ll_xist-5.67.2-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	559K
ll_xist-5.67.2-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	562K

6.15.15 5.67.1 (released 06/28/2021)

File	Type	Size
ll-xist-5.67.1.tar.gz	Source	734K
ll_xist-5.67.1-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	562K
ll_xist-5.67.1-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	565K
ll_xist-5.67.1-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	559K
ll_xist-5.67.1-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	562K
ll_xist-5.67.1-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	559K
ll_xist-5.67.1-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	562K
ll_xist-5.67.1-cp39-cp39-macosx_10_15_x86_64.whl	Mac wheel (Python 3.9)	561K
ll_xist-5.67.1-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	559K
ll_xist-5.67.1-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	562K

6.15.16 5.67 (released 06/25/2021)

File	Type	Size
ll-xist-5.67.tar.gz	Source	735K
ll_xist-5.67-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	562K
ll_xist-5.67-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	565K
ll_xist-5.67-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	559K
ll_xist-5.67-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	562K
ll_xist-5.67-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	559K
ll_xist-5.67-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	562K
ll_xist-5.67-cp39-cp39-macosx_10_15_x86_64.whl	Mac wheel (Python 3.9)	561K
ll_xist-5.67-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	559K
ll_xist-5.67-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	562K

6.15.17 5.66.1 (released 06/24/2021)

File	Type	Size
ll-xist-5.66.1.tar.gz	Source	733K
ll_xist-5.66.1-cp39-cp39-macosx_10_15_x86_64.whl	Mac wheel (Python 3.9)	562K

6.15.18 5.66 (released 06/15/2021)

File	Type	Size
ll-xist-5.66.tar.gz	Source	736K
ll_xist-5.66-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	564K
ll_xist-5.66-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	567K
ll_xist-5.66-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	560K
ll_xist-5.66-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	563K
ll_xist-5.66-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	560K
ll_xist-5.66-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	562K
ll_xist-5.66-cp39-cp39-macosx_10_15_x86_64.whl	Mac wheel (Python 3.9)	564K
ll_xist-5.66-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	559K
ll_xist-5.66-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	562K

6.15.19 5.65 (released 01/13/2021)

File	Type	Size
ll-xist-5.65.tar.gz	Source	713K
ll_xist-5.65-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	555K
ll_xist-5.65-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	558K
ll_xist-5.65-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	552K
ll_xist-5.65-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	555K
ll_xist-5.65-cp38-cp38-win32.whl	Windows wheel (Python 3.8)	552K
ll_xist-5.65-cp38-cp38-win_amd64.whl	Windows wheel (Python 3.8)	554K
ll_xist-5.65-cp39-cp39-macosx_10_15_x86_64.whl	Mac wheel (Python 3.9)	556K
ll_xist-5.65-cp39-cp39-win32.whl	Windows wheel (Python 3.9)	551K
ll_xist-5.65-cp39-cp39-win_amd64.whl	Windows wheel (Python 3.9)	554K

6.15.20 5.64 (released 10/30/2020)

File	Type	Size
ll-xist-5.64.tar.gz	Source	709K
ll_xist-5.64-cp38-cp38-macosx_10_15_x86_64.whl	Mac wheel (Python 3.8)	546K

6.15.21 5.63.1 (released 10/26/2020)

File	Type	Size
ll-xist-5.63.1.tar.gz	Source	715K
ll_xist-5.63.1-cp38-cp38-macosx_10_15_x86_64.whl	Mac wheel (Python 3.8)	545K

6.15.22 5.63 (released 09/08/2020)

File	Type	Size
ll-xist-5.63.tar.gz	Source	708K
ll_xist-5.63-cp38-cp38-macosx_10_15_x86_64.whl	Mac wheel (Python 3.8)	538K

6.15.23 5.62 (released 07/13/2020)

(no files for this version)

6.15.24 5.61.2 (released 07/09/2020)

File	Type	Size
ll-xist-5.61.2.tar.gz	Source	699K
ll_xist-5.61.2-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	549K
ll_xist-5.61.2-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	553K
ll_xist-5.61.2-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	537K
ll_xist-5.61.2-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	546K
ll_xist-5.61.2-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	549K

6.15.25 5.61.1 (released 07/09/2020)

File	Type	Size
ll-xist-5.61.1.tar.gz	Source	699K
ll_xist-5.61.1-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	537K

6.15.26 5.61 (released 07/07/2020)

File	Type	Size
ll-xist-5.61.tar.gz	Source	699K
ll_xist-5.61-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	537K

6.15.27 5.60 (released 07/03/2020)

File	Type	Size
ll-xist-5.60.tar.gz	Source	699K
ll_xist-5.60-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	550K
ll_xist-5.60-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	553K
ll_xist-5.60-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	537K
ll_xist-5.60-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	546K
ll_xist-5.60-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	549K

6.15.28 5.59 (released 06/30/2020)

File	Type	Size
ll-xist-5.59.tar.gz	Source	698K
ll_xist-5.59-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	550K
ll_xist-5.59-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	553K
ll_xist-5.59-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	536K
ll_xist-5.59-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	546K
ll_xist-5.59-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	549K

6.15.29 5.58 (released 06/12/2020)

File	Type	Size
ll-xist-5.58.tar.gz	Source	698K
ll_xist-5.58-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	537K

6.15.30 5.57 (released 04/14/2020)

File	Type	Size
ll-xist-5.57.tar.gz	Source	695K
ll_xist-5.57-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	535K

6.15.31 5.56 (released 12/12/2019)

File	Type	Size
ll-xist-5.56.tar.gz	Source	692K
ll_xist-5.56-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	546K
ll_xist-5.56-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	549K
ll_xist-5.56-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	533K
ll_xist-5.56-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	542K
ll_xist-5.56-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	545K

6.15.32 5.55 (released 11/11/2019)

File	Type	Size
ll-xist-5.55.tar.gz	Source	691K
ll_xist-5.55-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	545K
ll_xist-5.55-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	549K
ll_xist-5.55-cp37-cp37m-macosx_10_15_x86_64.whl	Mac wheel (Python 3.7)	533K
ll_xist-5.55-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	542K
ll_xist-5.55-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	545K

6.15.33 5.54.1 (released 10/24/2019)

File	Type	Size
ll-xist-5.54.1.tar.gz	Source	690K
ll_xist-5.54.1-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	544K
ll_xist-5.54.1-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	547K
ll_xist-5.54.1-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	532K
ll_xist-5.54.1-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	541K
ll_xist-5.54.1-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	544K

6.15.34 5.54 (released 10/24/2019)

File	Type	Size
ll-xist-5.54.tar.gz	Source	690K
ll_xist-5.54-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	545K
ll_xist-5.54-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	548K
ll_xist-5.54-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	532K
ll_xist-5.54-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	541K
ll_xist-5.54-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	544K

6.15.35 5.53 (released 09/30/2019)

File	Type	Size
ll-xist-5.53.tar.gz	Source	688K
ll_xist-5.53-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	543K
ll_xist-5.53-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	546K
ll_xist-5.53-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	531K
ll_xist-5.53-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	540K
ll_xist-5.53-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	543K

6.15.36 5.52.1 (released 09/05/2019)

File	Type	Size
ll-xist-5.52.1.tar.gz	Source	687K
ll_xist-5.52.1-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	531K

6.15.37 5.52 (released 07/29/2019)

File	Type	Size
ll-xist-5.52.tar.gz	Source	688K
ll_xist-5.52-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	544K
ll_xist-5.52-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	547K
ll_xist-5.52-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	531K
ll_xist-5.52-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	540K
ll_xist-5.52-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	543K

6.15.38 5.51 (released 07/26/2019)

File	Type	Size
ll-xist-5.51.tar.gz	Source	687K
ll_xist-5.51-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	543K
ll_xist-5.51-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	546K
ll_xist-5.51-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	530K
ll_xist-5.51-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	540K
ll_xist-5.51-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	543K

6.15.39 5.50 (released 07/16/2019)

File	Type	Size
ll-xist-5.50.tar.gz	Source	686K
ll_xist-5.50-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	543K
ll_xist-5.50-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	546K
ll_xist-5.50-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	530K
ll_xist-5.50-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	540K
ll_xist-5.50-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	543K

6.15.40 5.49 (released 07/04/2019)

File	Type	Size
ll-xist-5.49.tar.gz	Source	685K
ll_xist-5.49-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	542K
ll_xist-5.49-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	545K
ll_xist-5.49-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	530K
ll_xist-5.49-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	539K
ll_xist-5.49-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	542K

6.15.41 5.48 (released 07/03/2019)

File	Type	Size
ll-xist-5.48.tar.gz	Source	685K
ll_xist-5.48-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	542K
ll_xist-5.48-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	545K
ll_xist-5.48-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	530K
ll_xist-5.48-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	539K
ll_xist-5.48-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	542K

6.15.42 5.47 (released 07/01/2019)

File	Type	Size
ll-xist-5.47.tar.gz	Source	685K
ll_xist-5.47-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	542K
ll_xist-5.47-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	545K
ll_xist-5.47-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	529K
ll_xist-5.47-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	539K
ll_xist-5.47-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	542K

6.15.43 5.46 (released 06/26/2019)

File	Type	Size
ll-xist-5.46.tar.gz	Source	686K
ll_xist-5.46-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	543K
ll_xist-5.46-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	546K
ll_xist-5.46-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	530K
ll_xist-5.46-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	539K
ll_xist-5.46-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	542K

6.15.44 5.45 (released 06/24/2019)

File	Type	Size
ll-xist-5.45.tar.gz	Source	685K
ll_xist-5.45-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	542K
ll_xist-5.45-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	545K
ll_xist-5.45-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	529K
ll_xist-5.45-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	539K
ll_xist-5.45-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	542K

6.15.45 5.44 (released 06/07/2019)

File	Type	Size
ll-xist-5.44.tar.gz	Source	685K
ll_xist-5.44-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	543K
ll_xist-5.44-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	546K
ll_xist-5.44-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	530K
ll_xist-5.44-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	539K
ll_xist-5.44-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	542K

6.15.46 5.43 (released 05/07/2019)

File	Type	Size
ll-xist-5.43.tar.gz	Source	677K
ll_xist-5.43-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	537K
ll_xist-5.43-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	540K
ll_xist-5.43-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	524K
ll_xist-5.43-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	534K
ll_xist-5.43-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	537K

6.15.47 5.42.1 (released 04/29/2019)

File	Type	Size
ll-xist-5.42.1.tar.gz	Source	675K
ll_xist-5.42.1-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	535K
ll_xist-5.42.1-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	538K
ll_xist-5.42.1-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	522K
ll_xist-5.42.1-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	532K
ll_xist-5.42.1-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	535K

6.15.48 5.42 (released 04/26/2019)

File	Type	Size
ll-xist-5.42.tar.gz	Source	675K
ll_xist-5.42-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	535K
ll_xist-5.42-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	538K
ll_xist-5.42-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	522K
ll_xist-5.42-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	532K
ll_xist-5.42-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	535K

6.15.49 5.41 (released 03/29/2019)

File	Type	Size
ll-xist-5.41.tar.gz	Source	673K
ll_xist-5.41-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	533K
ll_xist-5.41-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	536K
ll_xist-5.41-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	521K
ll_xist-5.41-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	530K
ll_xist-5.41-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	533K

6.15.50 5.40.2 (released 03/26/2019)

File	Type	Size
ll-xist-5.40.2.tar.gz	Source	672K
ll_xist-5.40.2-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	531K
ll_xist-5.40.2-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	534K
ll_xist-5.40.2-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	518K
ll_xist-5.40.2-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	528K
ll_xist-5.40.2-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	531K

6.15.51 5.40.1 (released 03/25/2019)

File	Type	Size
ll-xist-5.40.1.tar.gz	Source	672K
ll_xist-5.40.1-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	531K
ll_xist-5.40.1-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	534K
ll_xist-5.40.1-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	519K
ll_xist-5.40.1-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	528K
ll_xist-5.40.1-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	531K

6.15.52 5.40 (released 03/25/2019)

File	Type	Size
ll-xist-5.40.tar.gz	Source	672K
ll_xist-5.40-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	531K
ll_xist-5.40-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	534K
ll_xist-5.40-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	518K
ll_xist-5.40-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	528K
ll_xist-5.40-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	531K

6.15.53 5.39 (released 01/30/2019)

File	Type	Size
ll-xist-5.39.tar.gz	Source	671K
ll_xist-5.39-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	531K
ll_xist-5.39-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	534K
ll_xist-5.39-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	518K
ll_xist-5.39-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	527K
ll_xist-5.39-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	530K

6.15.54 5.38 (released 11/15/2018)

File	Type	Size
ll-xist-5.38.tar.gz	Source	671K
ll_xist-5.38-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	531K
ll_xist-5.38-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	534K
ll_xist-5.38-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	518K
ll_xist-5.38-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	527K
ll_xist-5.38-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	530K

6.15.55 5.37.1 (released 11/13/2018)

File	Type	Size
ll-xist-5.37.1.tar.gz	Source	670K
ll_xist-5.37.1-cp37-cp37m-macosx_10_14_x86_64.whl	Mac wheel (Python 3.7)	518K

6.15.56 5.37 (released 11/08/2018)

File	Type	Size
ll-xist-5.37.tar.gz	Source	671K
ll_xist-5.37-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	531K
ll_xist-5.37-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	534K
ll_xist-5.37-cp37-cp37m-macosx_10_13_x86_64.whl	Mac wheel (Python 3.7)	2355K
ll_xist-5.37-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	527K
ll_xist-5.37-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	530K

6.15.57 5.36 (released 10/31/2018)

File	Type	Size
ll-xist-5.36.tar.gz	Source	669K
ll_xist-5.36-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	530K
ll_xist-5.36-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	533K
ll_xist-5.36-cp37-cp37m-macosx_10_13_x86_64.whl	Mac wheel (Python 3.7)	2355K
ll_xist-5.36-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	527K
ll_xist-5.36-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	530K

6.15.58 5.35 (released 09/14/2018)

File	Type	Size
ll-xist-5.35.tar.gz	Source	669K
ll_xist-5.35-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	530K
ll_xist-5.35-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	533K
ll_xist-5.35-cp37-cp37m-macosx_10_13_x86_64.whl	Mac wheel (Python 3.7)	2355K
ll_xist-5.35-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	527K
ll_xist-5.35-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	530K

6.15.59 5.34 (released 06/03/2018)

File	Type	Size
ll-xist-5.34.tar.bz2	Source	551K
ll-xist-5.34.tar.gz	Source	662K
ll-xist-5.34.zip	Source	778K
ll_xist-5.34-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	528K
ll_xist-5.34-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	531K
ll_xist-5.34-cp37-cp37m-macosx_10_13_x86_64.whl	Mac wheel (Python 3.7)	515K
ll_xist-5.34-cp37-cp37m-win32.whl	Windows wheel (Python 3.7)	525K
ll_xist-5.34-cp37-cp37m-win_amd64.whl	Windows wheel (Python 3.7)	528K

6.15.60 5.33 (released 05/15/2018)

File	Type	Size
ll-xist-5.33.tar.bz2	Source	552K
ll-xist-5.33.tar.gz	Source	667K
ll-xist-5.33.zip	Source	778K
ll_xist-5.33-cp36-cp36m-macosx_10_13_x86_64.whl	Mac wheel (Python 3.6)	515K
ll_xist-5.33-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	529K
ll_xist-5.33-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	532K

6.15.61 5.32 (released 02/20/2018)

File	Type	Size
ll-xist-5.32.tar.bz2	Source	554K
ll-xist-5.32.tar.gz	Source	666K
ll-xist-5.32.zip	Source	783K
ll_xist-5.32-cp36-cp36m-macosx_10_13_x86_64.whl	Mac wheel (Python 3.6)	524K
ll_xist-5.32-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	528K
ll_xist-5.32-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	531K

6.15.62 5.31 (released 01/29/2018)

File	Type	Size
ll-xist-5.31.tar.bz2	Source	553K
ll-xist-5.31.tar.gz	Source	666K
ll-xist-5.31.zip	Source	782K
ll_xist-5.31-cp36-cp36m-macosx_10_13_x86_64.whl	Mac wheel (Python 3.6)	524K
ll_xist-5.31-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	528K
ll_xist-5.31-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	531K

6.15.63 5.30 (released 01/17/2018)

File	Type	Size
ll-xist-5.30.tar.bz2	Source	554K
ll-xist-5.30.tar.gz	Source	666K
ll-xist-5.30.zip	Source	782K
ll_xist-5.30-cp36-cp36m-macosx_10_13_x86_64.whl	Mac wheel (Python 3.6)	524K
ll_xist-5.30-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	528K
ll_xist-5.30-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	531K

6.15.64 5.29 (released 11/29/2017)

File	Type	Size
ll-xist-5.29.tar.bz2	Source	551K
ll-xist-5.29.tar.gz	Source	662K
ll-xist-5.29.zip	Source	779K
ll_xist-5.29-cp36-cp36m-macosx_10_12_x86_64.whl	Mac wheel (Python 3.6)	523K
ll_xist-5.29-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	527K
ll_xist-5.29-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	530K

6.15.65 5.28.2 (released 08/03/2017)

File	Type	Size
ll-xist-5.28.2.tar.bz2	Source	550K
ll-xist-5.28.2.tar.gz	Source	662K
ll-xist-5.28.2.zip	Source	779K
ll_xist-5.28.2-cp36-cp36m-macosx_10_12_x86_64.whl	Mac wheel (Python 3.6)	522K
ll_xist-5.28.2-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	527K
ll_xist-5.28.2-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	530K

6.15.66 5.28.1 (released 08/02/2017)

File	Type	Size
ll-xist-5.28.1.tar.bz2	Source	550K
ll-xist-5.28.1.tar.gz	Source	662K
ll-xist-5.28.1.zip	Source	779K
ll_xist-5.28.1-cp36-cp36m-macosx_10_12_x86_64.whl	Mac wheel (Python 3.6)	522K
ll_xist-5.28.1-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	527K
ll_xist-5.28.1-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	530K

6.15.67 5.28 (released 08/01/2017)

File	Type	Size
ll-xist-5.28.tar.bz2	Source	550K
ll-xist-5.28.tar.gz	Source	662K
ll-xist-5.28.zip	Source	778K
ll_xist-5.28-cp36-cp36m-macosx_10_12_x86_64.whl	Mac wheel (Python 3.6)	522K

6.15.68 5.27 (released 03/21/2017)

File	Type	Size
ll-xist-5.27.tar.gz	Source	660K
ll_xist-5.27-cp35-none-win32.whl	Windows wheel (Python 3.5)	527K
ll_xist-5.27-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	530K
ll_xist-5.27-cp36-cp36m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.6)	523K
ll_xist-5.27-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	527K
ll_xist-5.27-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	530K

6.15.69 5.26.1 (released 03/03/2017)

File	Type	Size
ll-xist-5.26.1.tar.bz2	Source	548K
ll-xist-5.26.1.tar.gz	Source	660K
ll-xist-5.26.1.zip	Source	777K
ll_xist-5.26.1-cp35-none-win32.whl	Windows wheel (Python 3.5)	527K
ll_xist-5.26.1-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	530K
ll_xist-5.26.1-cp36-cp36m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.6)	522K
ll_xist-5.26.1-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	527K
ll_xist-5.26.1-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	530K

6.15.70 5.26 (released 02/28/2017)

File	Type	Size
ll-xist-5.26.tar.bz2	Source	547K
ll-xist-5.26.tar.gz	Source	660K
ll-xist-5.26.zip	Source	776K
ll_xist-5.26-cp35-none-win32.whl	Windows wheel (Python 3.5)	527K
ll_xist-5.26-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	530K
ll_xist-5.26-cp36-cp36m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.6)	522K
ll_xist-5.26-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	527K
ll_xist-5.26-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	530K

6.15.71 5.25.1 (released 02/15/2017)

File	Type	Size
ll-xist-5.25.1.tar.bz2	Source	547K
ll-xist-5.25.1.tar.gz	Source	658K
ll-xist-5.25.1.zip	Source	776K
ll_xist-5.25.1-cp35-none-win32.whl	Windows wheel (Python 3.5)	527K
ll_xist-5.25.1-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	530K
ll_xist-5.25.1-cp36-cp36m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.6)	522K
ll_xist-5.25.1-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	527K
ll_xist-5.25.1-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	530K

6.15.72 5.25 (released 02/13/2017)

File	Type	Size
ll-xist-5.25.tar.bz2	Source	547K
ll-xist-5.25.tar.gz	Source	658K
ll-xist-5.25.zip	Source	775K
ll_xist-5.25-cp35-none-win32.whl	Windows wheel (Python 3.5)	527K
ll_xist-5.25-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	530K
ll_xist-5.25-cp36-cp36m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.6)	522K
ll_xist-5.25-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	527K
ll_xist-5.25-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	530K

6.15.73 5.24 (released 02/12/2017)

File	Type	Size
ll-xist-5.24.tar.bz2	Source	546K
ll-xist-5.24.tar.gz	Source	657K
ll-xist-5.24.zip	Source	774K
ll_xist-5.24-cp35-none-win32.whl	Windows wheel (Python 3.5)	526K
ll_xist-5.24-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	529K
ll_xist-5.24-cp36-cp36m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.6)	522K
ll_xist-5.24-cp36-cp36m-win32.whl	Windows wheel (Python 3.6)	526K
ll_xist-5.24-cp36-cp36m-win_amd64.whl	Windows wheel (Python 3.6)	529K

6.15.74 5.23 (released 12/16/2016)

File	Type	Size
ll-xist-5.23.tar.bz2	Source	542K
ll-xist-5.23.tar.gz	Source	653K
ll-xist-5.23.zip	Source	769K
ll_xist-5.23-cp34-none-win32.whl	Windows wheel (Python 3.4)	529K
ll_xist-5.23-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	528K
ll_xist-5.23-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	517K
ll_xist-5.23-cp35-none-win32.whl	Windows wheel (Python 3.5)	527K
ll_xist-5.23-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	530K

6.15.75 5.22.1 (released 11/02/2016)

File	Type	Size
ll-xist-5.22.1.tar.bz2	Source	542K
ll-xist-5.22.1.tar.gz	Source	653K
ll-xist-5.22.1.zip	Source	770K
ll_xist-5.22.1-cp34-none-win32.whl	Windows wheel (Python 3.4)	529K
ll_xist-5.22.1-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	528K
ll_xist-5.22.1-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	517K
ll_xist-5.22.1-cp35-none-win32.whl	Windows wheel (Python 3.5)	527K
ll_xist-5.22.1-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	530K

6.15.76 5.22 (released 10/18/2016)

File	Type	Size
ll-xist-5.22.tar.bz2	Source	543K
ll-xist-5.22.tar.gz	Source	653K
ll-xist-5.22.zip	Source	769K
ll_xist-5.22-cp34-none-win32.whl	Windows wheel (Python 3.4)	529K
ll_xist-5.22-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	529K
ll_xist-5.22-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	517K
ll_xist-5.22-cp35-none-win32.whl	Windows wheel (Python 3.5)	527K
ll_xist-5.22-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	530K

6.15.77 5.21 (released 09/19/2016)

File	Type	Size
ll-xist-5.21.tar.bz2	Source	541K
ll-xist-5.21.tar.gz	Source	651K
ll-xist-5.21.zip	Source	767K
ll_xist-5.21-cp34-none-win32.whl	Windows wheel (Python 3.4)	527K
ll_xist-5.21-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	527K
ll_xist-5.21-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	515K
ll_xist-5.21-cp35-none-win32.whl	Windows wheel (Python 3.5)	525K
ll_xist-5.21-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	528K

6.15.78 5.20.1 (released 08/04/2016)

File	Type	Size
ll-xist-5.20.1.tar.bz2	Source	539K
ll-xist-5.20.1.tar.gz	Source	649K
ll-xist-5.20.1.zip	Source	766K
ll_xist-5.20.1-cp34-none-win32.whl	Windows wheel (Python 3.4)	526K
ll_xist-5.20.1-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	526K
ll_xist-5.20.1-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	513K
ll_xist-5.20.1-cp35-none-win32.whl	Windows wheel (Python 3.5)	525K
ll_xist-5.20.1-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	528K

6.15.79 5.20 (released 07/29/2016)

File	Type	Size
ll-xist-5.20.tar.bz2	Source	539K
ll-xist-5.20.tar.gz	Source	649K
ll-xist-5.20.zip	Source	765K
ll_xist-5.20-cp34-none-win32.whl	Windows wheel (Python 3.4)	526K
ll_xist-5.20-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	526K
ll_xist-5.20-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	513K
ll_xist-5.20-cp35-none-win32.whl	Windows wheel (Python 3.5)	525K
ll_xist-5.20-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	528K

6.15.80 5.19.4 (released 06/30/2016)

File	Type	Size
ll-xist-5.19.4.tar.bz2	Source	537K
ll-xist-5.19.4.tar.gz	Source	648K
ll-xist-5.19.4.zip	Source	764K
ll_xist-5.19.4-cp34-none-win32.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19.4-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19.4-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	512K
ll_xist-5.19.4-cp35-none-win32.whl	Windows wheel (Python 3.5)	524K
ll_xist-5.19.4-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	527K

6.15.81 5.19.3 (released 06/29/2016)

File	Type	Size
ll-xist-5.19.3.tar.bz2	Source	536K
ll-xist-5.19.3.tar.gz	Source	647K
ll-xist-5.19.3.zip	Source	763K
ll_xist-5.19.3-cp34-none-win32.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19.3-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19.3-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	512K
ll_xist-5.19.3-cp35-none-win32.whl	Windows wheel (Python 3.5)	523K
ll_xist-5.19.3-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	527K

6.15.82 5.19.2 (released 06/21/2016)

File	Type	Size
ll-xist-5.19.2.tar.bz2	Source	536K
ll-xist-5.19.2.tar.gz	Source	646K
ll-xist-5.19.2.zip	Source	763K
ll_xist-5.19.2-cp34-none-win32.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19.2-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	537K
ll_xist-5.19.2-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	512K
ll_xist-5.19.2-cp35-none-win32.whl	Windows wheel (Python 3.5)	523K
ll_xist-5.19.2-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	526K

6.15.83 5.19.1 (released 06/20/2016)

File	Type	Size
ll-xist-5.19.1.tar.bz2	Source	535K
ll-xist-5.19.1.tar.gz	Source	646K
ll-xist-5.19.1.zip	Source	763K
ll_xist-5.19.1-cp34-none-win32.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19.1-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19.1-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	512K
ll_xist-5.19.1-cp35-none-win32.whl	Windows wheel (Python 3.5)	523K
ll_xist-5.19.1-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	527K

6.15.84 5.19 (released 06/14/2016)

File	Type	Size
ll-xist-5.19.tar.bz2	Source	534K
ll-xist-5.19.tar.gz	Source	646K
ll-xist-5.19.zip	Source	762K
ll_xist-5.19-cp34-none-win32.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.19-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	511K
ll_xist-5.19-cp35-none-win32.whl	Windows wheel (Python 3.5)	523K
ll_xist-5.19-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	527K

6.15.85 5.18 (released 05/17/2016)

File	Type	Size
ll-xist-5.18.tar.bz2	Source	525K
ll-xist-5.18.tar.gz	Source	630K
ll-xist-5.18.zip	Source	734K
ll_xist-5.18-cp34-none-win32.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.18-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.18-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	510K
ll_xist-5.18-cp35-none-win32.whl	Windows wheel (Python 3.5)	523K
ll_xist-5.18-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	527K

6.15.86 5.17.1 (released 05/10/2016)

File	Type	Size
ll-xist-5.17.1.tar.bz2	Source	523K
ll-xist-5.17.1.tar.gz	Source	628K
ll-xist-5.17.1.win-amd64-py3.3.exe	Windows installer (Python 3.3)	771K
ll-xist-5.17.1.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1470K
ll-xist-5.17.1.win-amd64-py3.5.exe	Windows installer (Python 3.5)	1388K
ll-xist-5.17.1.win32-py3.3.exe	Windows installer (Python 3.3)	741K
ll-xist-5.17.1.win32-py3.4.exe	Windows installer (Python 3.4)	1391K
ll-xist-5.17.1.win32-py3.5.exe	Windows installer (Python 3.5)	1329K
ll-xist-5.17.1.zip	Source	733K
ll_xist-5.17.1-cp34-none-win32.whl	Windows wheel (Python 3.4)	524K
ll_xist-5.17.1-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	524K
ll_xist-5.17.1-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	510K
ll_xist-5.17.1-cp35-none-win32.whl	Windows wheel (Python 3.5)	523K
ll_xist-5.17.1-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	526K

6.15.87 5.17 (released 05/04/2016)

File	Type	Size
ll-xist-5.17.tar.bz2	Source	523K
ll-xist-5.17.tar.gz	Source	628K
ll-xist-5.17.win-amd64-py3.3.exe	Windows installer (Python 3.3)	772K
ll-xist-5.17.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1471K
ll-xist-5.17.win-amd64-py3.5.exe	Windows installer (Python 3.5)	1389K
ll-xist-5.17.win32-py3.3.exe	Windows installer (Python 3.3)	742K
ll-xist-5.17.win32-py3.4.exe	Windows installer (Python 3.4)	1392K
ll-xist-5.17.win32-py3.5.exe	Windows installer (Python 3.5)	1330K
ll-xist-5.17.zip	Source	732K
ll_xist-5.17-cp34-none-win32.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.17-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.17-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	510K
ll_xist-5.17-cp35-none-win32.whl	Windows wheel (Python 3.5)	523K
ll_xist-5.17-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	540K

6.15.88 5.16 (released 04/13/2016)

File	Type	Size
ll-xist-5.16.tar.bz2	Source	523K
ll-xist-5.16.tar.gz	Source	628K
ll-xist-5.16.win-amd64-py3.3.exe	Windows installer (Python 3.3)	773K
ll-xist-5.16.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1472K
ll-xist-5.16.win-amd64-py3.5.exe	Windows installer (Python 3.5)	1390K
ll-xist-5.16.win32-py3.3.exe	Windows installer (Python 3.3)	743K
ll-xist-5.16.win32-py3.4.exe	Windows installer (Python 3.4)	1393K
ll-xist-5.16.win32-py3.5.exe	Windows installer (Python 3.5)	1331K
ll-xist-5.16.zip	Source	732K
ll_xist-5.16-cp34-none-win32.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.16-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	525K
ll_xist-5.16-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	510K
ll_xist-5.16-cp35-none-win32.whl	Windows wheel (Python 3.5)	524K
ll_xist-5.16-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	527K

6.15.89 5.15.1 (released 03/21/2016)

File	Type	Size
ll-xist-5.15.1.tar.bz2	Source	521K
ll-xist-5.15.1.tar.gz	Source	625K
ll-xist-5.15.1.win-amd64-py3.3.exe	Windows installer (Python 3.3)	770K
ll-xist-5.15.1.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1469K
ll-xist-5.15.1.win-amd64-py3.5.exe	Windows installer (Python 3.5)	1386K
ll-xist-5.15.1.win32-py3.3.exe	Windows installer (Python 3.3)	740K
ll-xist-5.15.1.win32-py3.4.exe	Windows installer (Python 3.4)	1390K
ll-xist-5.15.1.win32-py3.5.exe	Windows installer (Python 3.5)	1328K
ll-xist-5.15.1.zip	Source	730K
ll_xist-5.15.1-cp34-none-win32.whl	Windows wheel (Python 3.4)	523K
ll_xist-5.15.1-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	542K
ll_xist-5.15.1-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	507K
ll_xist-5.15.1-cp35-none-win32.whl	Windows wheel (Python 3.5)	521K
ll_xist-5.15.1-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	524K

6.15.90 5.15 (released 03/18/2016)

File	Type	Size
ll-xist-5.15.tar.bz2	Source	521K
ll-xist-5.15.tar.gz	Source	626K
ll-xist-5.15.win-amd64-py3.3.exe	Windows installer (Python 3.3)	771K
ll-xist-5.15.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1470K
ll-xist-5.15.win-amd64-py3.5.exe	Windows installer (Python 3.5)	1388K
ll-xist-5.15.win32-py3.3.exe	Windows installer (Python 3.3)	741K
ll-xist-5.15.win32-py3.4.exe	Windows installer (Python 3.4)	1391K
ll-xist-5.15.win32-py3.5.exe	Windows installer (Python 3.5)	1329K
ll-xist-5.15.zip	Source	729K
ll_xist-5.15-cp34-none-win32.whl	Windows wheel (Python 3.4)	523K
ll_xist-5.15-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	523K
ll_xist-5.15-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	508K
ll_xist-5.15-cp35-none-win32.whl	Windows wheel (Python 3.5)	522K
ll_xist-5.15-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	525K

6.15.91 5.14.2 (released 03/02/2016)

File	Type	Size
ll-xist-5.14.2.tar.bz2	Source	520K
ll-xist-5.14.2.tar.gz	Source	625K
ll-xist-5.14.2.win-amd64-py3.3.exe	Windows installer (Python 3.3)	766K
ll-xist-5.14.2.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1465K
ll-xist-5.14.2.win-amd64-py3.5.exe	Windows installer (Python 3.5)	1383K
ll-xist-5.14.2.win32-py3.3.exe	Windows installer (Python 3.3)	736K
ll-xist-5.14.2.win32-py3.4.exe	Windows installer (Python 3.4)	1386K
ll-xist-5.14.2.win32-py3.5.exe	Windows installer (Python 3.5)	1324K
ll-xist-5.14.2.zip	Source	729K
ll_xist-5.14.2-cp34-none-win32.whl	Windows wheel (Python 3.4)	519K
ll_xist-5.14.2-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	519K
ll_xist-5.14.2-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	508K
ll_xist-5.14.2-cp35-none-win32.whl	Windows wheel (Python 3.5)	518K
ll_xist-5.14.2-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	521K

6.15.92 5.14.1 (released 12/04/2015)

File	Type	Size
ll-xist-5.14.1.tar.bz2	Source	515K
ll-xist-5.14.1.tar.gz	Source	619K
ll-xist-5.14.1.win-amd64-py3.3.exe	Windows installer (Python 3.3)	766K
ll-xist-5.14.1.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1465K
ll-xist-5.14.1.win-amd64-py3.5.exe	Windows installer (Python 3.5)	1382K
ll-xist-5.14.1.win32-py3.3.exe	Windows installer (Python 3.3)	736K
ll-xist-5.14.1.win32-py3.4.exe	Windows installer (Python 3.4)	1386K
ll-xist-5.14.1.win32-py3.5.exe	Windows installer (Python 3.5)	1324K
ll-xist-5.14.1.zip	Source	723K
ll_xist-5.14.1-cp34-none-win32.whl	Windows wheel (Python 3.4)	519K
ll_xist-5.14.1-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	519K
ll_xist-5.14.1-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	504K
ll_xist-5.14.1-cp35-none-win32.whl	Windows wheel (Python 3.5)	517K
ll_xist-5.14.1-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	520K

6.15.93 5.14 (released 12/02/2015)

File	Type	Size
ll-xist-5.14.tar.bz2	Source	518K
ll-xist-5.14.tar.gz	Source	623K
ll-xist-5.14.win-amd64–py3.5.exe	Windows installer (Python 3.5)	1388K
ll-xist-5.14.win-amd64-py3.3.exe	Windows installer (Python 3.3)	772K
ll-xist-5.14.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1471K
ll-xist-5.14.win32-py3.3.exe	Windows installer (Python 3.3)	742K
ll-xist-5.14.win32-py3.4.exe	Windows installer (Python 3.4)	1391K
ll-xist-5.14.win32-py3.5.exe	Windows installer (Python 3.5)	1330K
ll-xist-5.14.zip	Source	726K
ll_xist-5.14-cp34-none-win32.whl	Windows wheel (Python 3.4)	522K
ll_xist-5.14-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	522K
ll_xist-5.14-cp35-cp35m-macosx_10_9_x86_64.whl	Mac wheel (Python 3.5)	507K
ll_xist-5.14-cp35-none-win32.whl	Windows wheel (Python 3.5)	520K
ll_xist-5.14-cp35-none-win_amd64.whl	Windows wheel (Python 3.5)	524K

6.15.94 5.13.1 (released 06/12/2015)

File	Type	Size
ll-xist-5.13.1.tar.bz2	Source	721K
ll-xist-5.13.1.tar.gz	Source	931K
ll-xist-5.13.1.win-amd64-py3.3.exe	Windows installer (Python 3.3)	751K
ll-xist-5.13.1.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1449K
ll-xist-5.13.1.win32-py3.3.exe	Windows installer (Python 3.3)	720K
ll-xist-5.13.1.win32-py3.4.exe	Windows installer (Python 3.4)	1370K
ll-xist-5.13.1.zip	Source	1054K
ll_xist-5.13.1-cp34-none-win32.whl	Windows wheel (Python 3.4)	503K
ll_xist-5.13.1-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	503K

6.15.95 5.13 (released 12/18/2014)

File	Type	Size
ll-xist-5.13.tar.bz2	Source	507K
ll-xist-5.13.tar.gz	Source	604K
ll-xist-5.13.win-amd64-py3.3.exe	Windows installer (Python 3.3)	751K
ll-xist-5.13.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1450K
ll-xist-5.13.win32-py3.3.exe	Windows installer (Python 3.3)	723K
ll-xist-5.13.win32-py3.4.exe	Windows installer (Python 3.4)	1373K
ll-xist-5.13.zip	Source	708K
ll_xist-5.13-cp34-none-win32.whl	Windows wheel (Python 3.4)	506K
ll_xist-5.13-cp34-none-win_amd64.whl	Windows wheel (Python 3.4)	503K

6.15.96 5.12.1 (released 12/09/2014)

File	Type	Size
ll-xist-5.12.1.tar.bz2	Source	504K
ll-xist-5.12.1.tar.gz	Source	599K
ll-xist-5.12.1.win-amd64-py3.3.exe	Windows installer (Python 3.3)	750K
ll-xist-5.12.1.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1449K
ll-xist-5.12.1.win32-py3.3.exe	Windows installer (Python 3.3)	722K
ll-xist-5.12.1.win32-py3.4.exe	Windows installer (Python 3.4)	1372K
ll-xist-5.12.1.zip	Source	703K

6.15.97 5.12 (released 11/07/2014)

File	Type	Size
ll-xist-5.12.tar.bz2	Source	504K
ll-xist-5.12.tar.gz	Source	599K
ll-xist-5.12.win-amd64-py3.3.exe	Windows installer (Python 3.3)	746K
ll-xist-5.12.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1444K
ll-xist-5.12.win32-py3.3.exe	Windows installer (Python 3.3)	718K
ll-xist-5.12.win32-py3.4.exe	Windows installer (Python 3.4)	1368K
ll-xist-5.12.zip	Source	703K

6.15.98 5.11 (released 10/29/2014)

File	Type	Size
ll-xist-5.11.tar.bz2	Source	502K
ll-xist-5.11.tar.gz	Source	597K
ll-xist-5.11.win-amd64-py3.3.exe	Windows installer (Python 3.3)	746K
ll-xist-5.11.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1444K
ll-xist-5.11.win32-py3.3.exe	Windows installer (Python 3.3)	718K
ll-xist-5.11.win32-py3.4.exe	Windows installer (Python 3.4)	1368K
ll-xist-5.11.zip	Source	701K

6.15.99 5.10 (released 10/09/2014)

File	Type	Size
ll-xist-5.10.tar.bz2	Source	500K
ll-xist-5.10.tar.gz	Source	594K
ll-xist-5.10.win-amd64-py3.3.exe	Windows installer (Python 3.3)	743K
ll-xist-5.10.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1441K
ll-xist-5.10.win32-py3.3.exe	Windows installer (Python 3.3)	715K
ll-xist-5.10.win32-py3.4.exe	Windows installer (Python 3.4)	1365K
ll-xist-5.10.zip	Source	697K

6.15.100 5.9.1 (released 09/29/2014)

File	Type	Size
ll-xist-5.9.1.tar.bz2	Source	499K
ll-xist-5.9.1.tar.gz	Source	593K
ll-xist-5.9.1.win-amd64-py3.3.exe	Windows installer (Python 3.3)	742K
ll-xist-5.9.1.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1441K
ll-xist-5.9.1.win32-py3.3.exe	Windows installer (Python 3.3)	714K
ll-xist-5.9.1.win32-py3.4.exe	Windows installer (Python 3.4)	1364K
ll-xist-5.9.1.zip	Source	697K

6.15.101 5.9 (released 09/22/2014)

File	Type	Size
ll-xist-5.9.tar.bz2	Source	500K
ll-xist-5.9.tar.gz	Source	595K
ll-xist-5.9.win-amd64-py3.3.exe	Windows installer (Python 3.3)	746K
ll-xist-5.9.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1445K
ll-xist-5.9.win32-py3.3.exe	Windows installer (Python 3.3)	719K
ll-xist-5.9.win32-py3.4.exe	Windows installer (Python 3.4)	1369K
ll-xist-5.9.zip	Source	698K

6.15.102 5.8.1 (released 06/18/2014)

File	Type	Size
ll-xist-5.8.1.tar.bz2	Source	489K
ll-xist-5.8.1.tar.gz	Source	581K
ll-xist-5.8.1.win-amd64-py3.3.exe	Windows installer (Python 3.3)	732K
ll-xist-5.8.1.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1353K
ll-xist-5.8.1.win32-py3.3.exe	Windows installer (Python 3.3)	702K
ll-xist-5.8.1.win32-py3.4.exe	Windows installer (Python 3.4)	1280K
ll-xist-5.8.1.zip	Source	682K

6.15.103 5.8 (released 05/05/2014)

File	Type	Size
ll-xist-5.8.tar.bz2	Source	489K
ll-xist-5.8.tar.gz	Source	582K
ll-xist-5.8.win-amd64-py3.3.exe	Windows installer (Python 3.3)	733K
ll-xist-5.8.win-amd64-py3.4.exe	Windows installer (Python 3.4)	1354K
ll-xist-5.8.win32-py3.3.exe	Windows installer (Python 3.3)	703K
ll-xist-5.8.win32-py3.4.exe	Windows installer (Python 3.4)	1281K
ll-xist-5.8.zip	Source	682K

6.15.104 5.7.1 (released 02/13/2014)

File	Type	Size
ll-xist-5.7.1.tar.bz2	Source	488K
ll-xist-5.7.1.tar.gz	Source	580K
ll-xist-5.7.1.zip	Source	681K

6.15.105 5.7 (released 01/30/2014)

File	Type	Size
ll-xist-5.7.tar.bz2	Source	488K
ll-xist-5.7.tar.gz	Source	580K
ll-xist-5.7.zip	Source	680K

6.15.106 5.6 (released 01/28/2014)

File	Type	Size
ll-xist-5.6.tar.bz2	Source	487K
ll-xist-5.6.tar.gz	Source	578K
ll-xist-5.6.zip	Source	676K

6.15.107 5.5.1 (released 01/27/2014)

File	Type	Size
ll-xist-5.5.1.tar.bz2	Source	487K
ll-xist-5.5.1.tar.gz	Source	578K
ll-xist-5.5.1.zip	Source	676K

6.15.108 5.5 (released 01/23/2014)

File	Type	Size
ll-xist-5.5.tar.bz2	Source	487K
ll-xist-5.5.tar.gz	Source	578K
ll-xist-5.5.zip	Source	675K

6.15.109 5.4.1 (released 12/18/2013)

File	Type	Size
ll-xist-5.4.1.tar.bz2	Source	484K
ll-xist-5.4.1.tar.gz	Source	576K
ll-xist-5.4.1.zip	Source	675K

6.15.110 5.4 (released 11/29/2013)

File	Type	Size
ll-xist-5.4.tar.bz2	Source	483K
ll-xist-5.4.tar.gz	Source	576K
ll-xist-5.4.zip	Source	674K

6.15.111 5.3 (released 10/28/2013)

File	Type	Size
ll-xist-5.3.tar.bz2	Source	483K
ll-xist-5.3.tar.gz	Source	575K
ll-xist-5.3.zip	Source	674K

6.15.112 5.2.7 (released 10/15/2013)

File	Type	Size
ll-xist-5.2.7.tar.bz2	Source	483K
ll-xist-5.2.7.tar.gz	Source	575K
ll-xist-5.2.7.zip	Source	674K

6.15.113 5.2.6 (released 10/15/2013)

File	Type	Size
ll-xist-5.2.6.tar.bz2	Source	483K
ll-xist-5.2.6.tar.gz	Source	575K
ll-xist-5.2.6.zip	Source	674K

6.15.114 5.2.5 (released 10/09/2013)

File	Type	Size
ll-xist-5.2.5.tar.bz2	Source	483K
ll-xist-5.2.5.tar.gz	Source	575K
ll-xist-5.2.5.zip	Source	674K

6.15.115 5.2.4 (released 10/09/2013)

(no files for this version)

6.15.116 5.2.3 (released 10/09/2013)

File	Type	Size
ll-xist-5.2.3.tar.bz2	Source	483K
ll-xist-5.2.3.tar.gz	Source	575K
ll-xist-5.2.3.zip	Source	674K

6.15.117 5.2.2 (released 10/07/2013)

File	Type	Size
ll-xist-5.2.2.tar.bz2	Source	483K
ll-xist-5.2.2.tar.gz	Source	575K
ll-xist-5.2.2.zip	Source	674K

6.15.118 5.2.1 (released 10/02/2013)

File	Type	Size
ll-xist-5.2.1.tar.bz2	Source	483K
ll-xist-5.2.1.tar.gz	Source	575K
ll-xist-5.2.1.zip	Source	674K

6.15.119 5.2 (released 10/01/2013)

File	Type	Size
ll-xist-5.2.tar.bz2	Source	483K
ll-xist-5.2.tar.gz	Source	576K
ll-xist-5.2.zip	Source	674K

6.15.120 5.1 (released 08/02/2013)

File	Type	Size
ll-xist-5.1.tar.bz2	Source	478K
ll-xist-5.1.tar.gz	Source	567K
ll-xist-5.1.zip	Source	666K

6.15.121 5.0 (released 06/04/2013)

File	Type	Size
ll-xist-5.0.tar.bz2	Source	477K
ll-xist-5.0.tar.gz	Source	567K
ll-xist-5.0.zip	Source	666K

6.15.122 4.10 (released 03/04/2013)

File	Type	Size
ll-xist-4.10.tar.bz2	Source	460K
ll-xist-4.10.tar.gz	Source	549K
ll-xist-4.10.zip	Source	646K

6.15.123 4.9.1 (released 01/17/2013)

File	Type	Size
ll-xist-4.9.1.tar.bz2	Source	460K
ll-xist-4.9.1.tar.gz	Source	548K
ll-xist-4.9.1.win32-py3.3.exe	Windows installer (Python 3.3)	689K
ll-xist-4.9.1.zip	Source	646K

6.15.124 4.9 (released 01/17/2013)

File	Type	Size
ll-xist-4.9.tar.bz2	Source	461K
ll-xist-4.9.tar.gz	Source	548K
ll-xist-4.9.win32-py3.3.exe	Windows installer (Python 3.3)	689K
ll-xist-4.9.zip	Source	645K

6.15.125 4.8 (released 01/15/2013)

File	Type	Size
ll-xist-4.8.tar.bz2	Source	459K
ll-xist-4.8.tar.gz	Source	547K
ll-xist-4.8.win32-py3.3.exe	Windows installer (Python 3.3)	688K
ll-xist-4.8.zip	Source	644K

6.15.126 4.7 (released 01/11/2013)

File	Type	Size
ll-xist-4.7.tar.bz2	Source	459K
ll-xist-4.7.tar.gz	Source	547K
ll-xist-4.7.win32-py3.3.exe	Windows installer (Python 3.3)	689K
ll-xist-4.7.zip	Source	644K

6.15.127 4.6 (released 12/18/2012)

File	Type	Size
ll-xist-4.6.tar.bz2	Source	457K
ll-xist-4.6.tar.gz	Source	545K
ll-xist-4.6.win32-py3.3.exe	Windows installer (Python 3.3)	689K
ll-xist-4.6.zip	Source	641K

6.15.128 4.5 (released 11/29/2012)

File	Type	Size
ll-xist-4.5.tar.bz2	Source	454K
ll-xist-4.5.tar.gz	Source	538K
ll-xist-4.5.zip	Source	635K

6.15.129 4.4 (released 11/08/2012)

File	Type	Size
ll-xist-4.4.tar.bz2	Source	453K
ll-xist-4.4.tar.gz	Source	537K
ll-xist-4.4.zip	Source	634K

6.15.130 4.3.1 (released 11/06/2012)

File	Type	Size
ll-xist-4.3.1.tar.bz2	Source	456K
ll-xist-4.3.1.tar.gz	Source	541K
ll-xist-4.3.1.zip	Source	641K

6.15.131 4.3 (released 11/02/2012)

File	Type	Size
ll-xist-4.3.tar.bz2	Source	457K
ll-xist-4.3.tar.gz	Source	542K
ll-xist-4.3.zip	Source	641K

6.15.132 4.2 (released 10/22/2012)

File	Type	Size
ll-xist-4.2.tar.bz2	Source	387K
ll-xist-4.2.tar.gz	Source	455K
ll-xist-4.2.zip	Source	539K

6.15.133 4.1.1 (released 10/04/2012)

File	Type	Size
ll-xist-4.1.1.tar.bz2	Source	381K
ll-xist-4.1.1.tar.gz	Source	449K
ll-xist-4.1.1.zip	Source	534K

6.15.134 4.1 (released 10/02/2012)

File	Type	Size
ll-xist-4.1.tar.bz2	Source	381K
ll-xist-4.1.tar.gz	Source	449K
ll-xist-4.1.zip	Source	534K

6.15.135 4.0 (released 08/08/2012)

File	Type	Size
ll-xist-4.0.tar.bz2	Source	377K
ll-xist-4.0.tar.gz	Source	446K
ll-xist-4.0.zip	Source	528K

6.15.136 3.25 (released 08/12/2011)

File	Type	Size
ll-xist-3.25.tar.bz2	Source	379K
ll-xist-3.25.tar.gz	Source	446K
ll-xist-3.25.zip	Source	528K

6.15.137 3.24.1 (released 08/10/2011)

File	Type	Size
ll-xist-3.24.1.tar.bz2	Source	378K
ll-xist-3.24.1.tar.gz	Source	444K
ll-xist-3.24.1.zip	Source	526K

6.15.138 3.24 (released 08/09/2011)

File	Type	Size
ll-xist-3.24.tar.bz2	Source	378K
ll-xist-3.24.tar.gz	Source	444K
ll-xist-3.24.zip	Source	526K

6.15.139 3.23.1 (released 07/28/2011)

File	Type	Size
ll-xist-3.23.1.tar.bz2	Source	378K
ll-xist-3.23.1.tar.gz	Source	444K
ll-xist-3.23.1.win32-py2.7.exe	Windows installer (Python 2.7)	642K
ll-xist-3.23.1.zip	Source	526K

6.15.140 3.23 (released 07/20/2011)

File	Type	Size
ll-xist-3.23.tar.bz2	Source	379K
ll-xist-3.23.tar.gz	Source	444K
ll-xist-3.23.zip	Source	526K

6.15.141 3.22 (released 07/14/2011)

File	Type	Size
ll-xist-3.22.tar.bz2	Source	378K
ll-xist-3.22.tar.gz	Source	443K
ll-xist-3.22.zip	Source	524K

6.15.142 3.21 (released 06/03/2011)

File	Type	Size
ll-xist-3.21.tar.bz2	Source	378K
ll-xist-3.21.tar.gz	Source	443K
ll-xist-3.21.zip	Source	524K

6.15.143 3.20.2 (released 05/23/2011)

File	Type	Size
ll-xist-3.20.2.tar.bz2	Source	377K
ll-xist-3.20.2.tar.gz	Source	443K
ll-xist-3.20.2.zip	Source	524K

6.15.144 3.20.1 (released 05/18/2011)

File	Type	Size
ll-xist-3.20.1.tar.bz2	Source	377K
ll-xist-3.20.1.tar.gz	Source	443K
ll-xist-3.20.1.zip	Source	524K

6.15.145 3.20 (released 05/05/2011)

File	Type	Size
ll-xist-3.20.tar.bz2	Source	378K
ll-xist-3.20.tar.gz	Source	444K
ll-xist-3.20.zip	Source	524K

6.15.146 3.19 (released 04/26/2011)

File	Type	Size
ll-xist-3.19.tar.bz2	Source	377K
ll-xist-3.19.tar.gz	Source	442K
ll-xist-3.19.zip	Source	522K

6.15.147 3.18.1 (released 04/13/2011)

File	Type	Size
ll-xist-3.18.1.tar.bz2	Source	375K
ll-xist-3.18.1.tar.gz	Source	441K
ll-xist-3.18.1.zip	Source	522K

6.15.148 3.18 (released 04/08/2011)

File	Type	Size
ll-xist-3.18.tar.bz2	Source	376K
ll-xist-3.18.tar.gz	Source	442K
ll-xist-3.18.zip	Source	523K

6.15.149 3.17.3 (released 03/02/2011)

File	Type	Size
ll-xist-3.17.3.tar.bz2	Source	367K
ll-xist-3.17.3.tar.gz	Source	431K
ll-xist-3.17.3.zip	Source	506K

6.15.150 3.17.2 (released 02/25/2011)

File	Type	Size
ll-xist-3.17.2.tar.bz2	Source	367K
ll-xist-3.17.2.tar.gz	Source	431K
ll-xist-3.17.2.zip	Source	506K

6.15.151 3.17.1 (released 02/25/2011)

File	Type	Size
ll-xist-3.17.1.tar.bz2	Source	367K
ll-xist-3.17.1.tar.gz	Source	431K
ll-xist-3.17.1.zip	Source	506K

6.15.152 3.17 (released 02/24/2011)

File	Type	Size
ll-xist-3.17.tar.bz2	Source	360K
ll-xist-3.17.tar.gz	Source	424K
ll-xist-3.17.win32-py2.7.exe	Windows installer (Python 2.7)	618K
ll-xist-3.17.zip	Source	498K

6.15.153 3.16 (released 01/21/2011)

File	Type	Size
ll-xist-3.16.tar.bz2	Source	364K
ll-xist-3.16.tar.gz	Source	428K
ll-xist-3.16.zip	Source	502K

6.15.154 3.15.3 (released 11/26/2010)

File	Type	Size
ll-xist-3.15.3.tar.bz2	Source	361K
ll-xist-3.15.3.tar.gz	Source	425K
ll-xist-3.15.3.zip	Source	498K

6.15.155 3.15.2 (released 11/25/2010)

File	Type	Size
ll-xist-3.15.2.tar.bz2	Source	361K
ll-xist-3.15.2.tar.gz	Source	424K
ll-xist-3.15.2.zip	Source	498K

6.15.156 3.15.1 (released 11/24/2010)

File	Type	Size
ll-xist-3.15.1.tar.bz2	Source	361K
ll-xist-3.15.1.tar.gz	Source	424K
ll-xist-3.15.1.zip	Source	498K

6.15.157 3.15 (released 11/09/2010)

File	Type	Size
ll-xist-3.15.tar.bz2	Source	361K
ll-xist-3.15.tar.gz	Source	424K
ll-xist-3.15.zip	Source	497K

6.15.158 3.14 (released 11/05/2010)

File	Type	Size
ll-xist-3.14.tar.bz2	Source	362K
ll-xist-3.14.tar.gz	Source	425K
ll-xist-3.14.zip	Source	498K

6.15.159 3.13 (released 10/22/2010)

File	Type	Size
ll-xist-3.13.tar.bz2	Source	350K
ll-xist-3.13.tar.gz	Source	414K
ll-xist-3.13.zip	Source	487K

6.15.160 3.12.1 (released 10/21/2010)

File	Type	Size
ll-xist-3.12.1.tar.bz2	Source	349K
ll-xist-3.12.1.tar.gz	Source	413K
ll-xist-3.12.1.zip	Source	487K

6.15.161 3.12 (released 10/21/2010)

File	Type	Size
ll-xist-3.12.tar.bz2	Source	350K
ll-xist-3.12.tar.gz	Source	414K
ll-xist-3.12.zip	Source	487K

6.15.162 3.11.1 (released 10/18/2010)

File	Type	Size
ll-xist-3.11.1.tar.bz2	Source	348K
ll-xist-3.11.1.tar.gz	Source	412K
ll-xist-3.11.1.zip	Source	487K

6.15.163 3.11 (released 10/15/2010)

File	Type	Size
ll-xist-3.11.tar.bz2	Source	348K
ll-xist-3.11.tar.gz	Source	412K
ll-xist-3.11.zip	Source	486K

6.15.164 3.10.1 (released 10/13/2010)

File	Type	Size
ll-xist-3.10.1.tar.bz2	Source	345K
ll-xist-3.10.1.tar.gz	Source	409K
ll-xist-3.10.1.zip	Source	483K

6.15.165 3.10 (released 09/24/2010)

File	Type	Size
ll-xist-3.10.tar.bz2	Source	345K
ll-xist-3.10.tar.gz	Source	409K
ll-xist-3.10.zip	Source	483K

6.15.166 3.9 (released 08/04/2010)

File	Type	Size
ll-xist-3.9.tar.bz2	Source	345K
ll-xist-3.9.tar.gz	Source	407K
ll-xist-3.9.zip	Source	482K

6.15.167 3.8.3 (released 07/29/2010)

File	Type	Size
ll-xist-3.8.3.tar.bz2	Source	344K
ll-xist-3.8.3.tar.gz	Source	406K
ll-xist-3.8.3.zip	Source	481K

6.15.168 3.8.2 (released 06/21/2010)

File	Type	Size
ll-xist-3.8.2.tar.bz2	Source	350K
ll-xist-3.8.2.tar.gz	Source	413K
ll-xist-3.8.2.win32-py2.6.exe	Windows installer (Python 2.6)	637K
ll-xist-3.8.2.zip	Source	482K
ll_xist-3.8.2-py2.6-win32.egg	Windows egg (Python 2.6)	870K

6.15.169 3.8.1 (released 06/17/2010)

File	Type	Size
ll-xist-3.8.1.tar.bz2	Source	350K
ll-xist-3.8.1.tar.gz	Source	413K
ll-xist-3.8.1.win32-py2.6.exe	Windows installer (Python 2.6)	637K
ll-xist-3.8.1.zip	Source	482K
ll_xist-3.8.1-py2.6-win32.egg	Windows egg (Python 2.6)	868K

6.15.170 3.8 (released 06/15/2010)

File	Type	Size
ll-xist-3.8.tar.bz2	Source	351K
ll-xist-3.8.tar.gz	Source	415K
ll-xist-3.8.win32-py2.6.exe	Windows installer (Python 2.6)	641K
ll-xist-3.8.zip	Source	483K
ll_xist-3.8-py2.6-win32.egg	Windows egg (Python 2.6)	870K

6.15.171 3.7.6 (released 05/14/2010)

File	Type	Size
ll-xist-3.7.6.tar.bz2	Source	342K
ll-xist-3.7.6.tar.gz	Source	400K
ll-xist-3.7.6.zip	Source	477K

6.15.172 3.7.5 (released 04/19/2010)

File	Type	Size
ll-xist-3.7.5.tar.bz2	Source	345K
ll-xist-3.7.5.tar.gz	Source	405K
ll-xist-3.7.5.zip	Source	481K
ll_xist-3.7.5-py2.6-macosx-10.4-i386.egg	Egg (Python 2.6)	809K

6.15.173 3.7.4 (released 03/25/2010)

File	Type	Size
ll-xist-3.7.4.tar.bz2	Source	346K
ll-xist-3.7.4.tar.gz	Source	404K
ll-xist-3.7.4.win32-py2.5.exe	Windows installer (Python 2.5)	473K
ll-xist-3.7.4.zip	Source	481K

6.15.174 3.7.3 (released 02/27/2010)

File	Type	Size
ll-xist-3.7.3.tar.bz2	Source	341K
ll-xist-3.7.3.tar.gz	Source	400K
ll-xist-3.7.3.zip	Source	477K

6.15.175 3.7.2 (released 02/26/2010)

File	Type	Size
ll-xist-3.7.2.tar.bz2	Source	347K
ll-xist-3.7.2.tar.gz	Source	408K
ll-xist-3.7.2.zip	Source	482K

6.15.176 3.7.1 (released 02/08/2010)

File	Type	Size
ll-xist-3.7.1.tar.bz2	Source	341K
ll-xist-3.7.1.tar.gz	Source	400K
ll-xist-3.7.1.zip	Source	477K
ll_xist-3.7.1-py2.6-macosx-10.4-i386.egg	Egg (Python 2.6)	808K

6.15.177 3.7 (released 09/10/2009)

File	Type	Size
ll-xist-3.7.tar.bz2	Source	347K
ll-xist-3.7.tar.gz	Source	411K
ll-xist-3.7.zip	Source	482K

6.15.178 3.6.6 (released 07/09/2009)

File	Type	Size
ll-xist-3.6.6.tar.bz2	Source	320K
ll-xist-3.6.6.tar.gz	Source	378K
ll-xist-3.6.6.zip	Source	436K

6.15.179 3.6.5 (released 06/02/2009)

File	Type	Size
ll-xist-3.6.5.tar.bz2	Source	312K
ll-xist-3.6.5.tar.gz	Source	368K
ll-xist-3.6.5.zip	Source	432K

6.15.180 3.6.4 (released 03/19/2009)

File	Type	Size
ll-xist-3.6.4.tar.bz2	Source	322K
ll-xist-3.6.4.tar.gz	Source	382K
ll-xist-3.6.4.win32-py2.5.exe	Windows installer (Python 2.5)	407K
ll-xist-3.6.4.zip	Source	438K
ll_xist-3.6.4-py2.5-win32.egg	Windows egg (Python 2.5)	731K

6.15.181 3.6.3 (released 03/02/2009)

File	Type	Size
ll-xist-3.6.3.tar.bz2	Source	321K
ll-xist-3.6.3.tar.gz	Source	381K
ll-xist-3.6.3.win32-py2.5.exe	Windows installer (Python 2.5)	407K
ll-xist-3.6.3.zip	Source	437K
ll_xist-3.6.3-py2.5-win32.egg	Windows egg (Python 2.5)	731K

6.15.182 3.6.2 (released 02/16/2009)

File	Type	Size
ll-xist-3.6.2.tar.bz2	Source	321K
ll-xist-3.6.2.tar.gz	Source	381K
ll-xist-3.6.2.win32-py2.5.exe	Windows installer (Python 2.5)	407K
ll-xist-3.6.2.zip	Source	436K
ll_xist-3.6.2-py2.5-win32.egg	Windows egg (Python 2.5)	730K

6.15.183 3.6.1 (released 01/27/2009)

File	Type	Size
ll-xist-3.6.1.tar.bz2	Source	320K
ll-xist-3.6.1.tar.gz	Source	380K
ll-xist-3.6.1.win32-py2.5.exe	Windows installer (Python 2.5)	407K
ll-xist-3.6.1.zip	Source	436K
ll_xist-3.6.1-py2.5-win32.egg	Windows egg (Python 2.5)	729K

6.15.184 3.6 (released 12/31/2008)

File	Type	Size
ll-xist-3.6.tar.bz2	Source	305K
ll-xist-3.6.tar.gz	Source	363K
ll-xist-3.6.zip	Source	427K

6.15.185 3.5 (released 12/05/2008)

File	Type	Size
ll-xist-3.5.tar.bz2	Source	314K
ll-xist-3.5.tar.gz	Source	371K
ll-xist-3.5.win32-py2.5.exe	Windows installer (Python 2.5)	405K
ll-xist-3.5.zip	Source	434K
ll_xist-3.5-py2.5-win32.egg	Windows egg (Python 2.5)	721K

6.15.186 3.4.4 (released 09/16/2008)

File	Type	Size
ll-xist-3.4.4.tar.bz2	Source	309K
ll-xist-3.4.4.tar.gz	Source	364K
ll-xist-3.4.4.win32-py2.5.exe	Windows installer (Python 2.5)	402K
ll-xist-3.4.4.zip	Source	428K
ll_xist-3.4.4-py2.5-win32.egg	Windows egg (Python 2.5)	716K

6.15.187 3.4.3 (released 09/09/2008)

File	Type	Size
ll-xist-3.4.3.tar.bz2	Source	309K
ll-xist-3.4.3.tar.gz	Source	364K
ll-xist-3.4.3.win32-py2.5.exe	Windows installer (Python 2.5)	402K
ll-xist-3.4.3.zip	Source	428K
ll_xist-3.4.3-py2.5-win32.egg	Windows egg (Python 2.5)	716K

6.15.188 3.4.2 (released 09/03/2008)

File	Type	Size
ll-xist-3.4.2.tar.bz2	Source	308K
ll-xist-3.4.2.tar.gz	Source	364K
ll-xist-3.4.2.win32-py2.5.exe	Windows installer (Python 2.5)	402K
ll-xist-3.4.2.zip	Source	428K
ll_xist-3.4.2-py2.5-win32.egg	Windows egg (Python 2.5)	716K

6.15.189 3.4.1 (released 08/29/2008)

File	Type	Size
ll-xist-3.4.1.tar.bz2	Source	308K
ll-xist-3.4.1.tar.gz	Source	363K
ll-xist-3.4.1.win32-py2.5.exe	Windows installer (Python 2.5)	401K
ll-xist-3.4.1.zip	Source	428K
ll_xist-3.4.1-py2.5-win32.egg	Windows egg (Python 2.5)	715K

6.15.190 3.4 (released 08/19/2008)

File	Type	Size
ll-xist-3.4.tar.bz2	Source	309K
ll-xist-3.4.tar.gz	Source	364K
ll-xist-3.4.win32-py2.5.exe	Windows installer (Python 2.5)	406K
ll-xist-3.4.zip	Source	429K
ll_xist-3.4-py2.5-win32.egg	Windows egg (Python 2.5)	716K

6.15.191 3.3.2 (released 07/15/2008)

File	Type	Size
ll-xist-3.3.2.tar.bz2	Source	302K
ll-xist-3.3.2.tar.gz	Source	355K
ll-xist-3.3.2.win32-py2.5.exe	Windows installer (Python 2.5)	395K
ll-xist-3.3.2.zip	Source	421K
ll_xist-3.3.2-py2.5-win32.egg	Windows egg (Python 2.5)	705K

6.15.192 3.3.1 (released 07/14/2008)

File	Type	Size
ll-xist-3.3.1.tar.bz2	Source	302K
ll-xist-3.3.1.tar.gz	Source	355K
ll-xist-3.3.1.win32-py2.5.exe	Windows installer (Python 2.5)	394K
ll-xist-3.3.1.zip	Source	420K
ll_xist-3.3.1-py2.5-win32.egg	Windows egg (Python 2.5)	704K

6.15.193 3.3 (released 07/11/2008)

File	Type	Size
ll-xist-3.3.tar.bz2	Source	303K
ll-xist-3.3.tar.gz	Source	356K
ll-xist-3.3.win32-py2.5.exe	Windows installer (Python 2.5)	397K
ll-xist-3.3.zip	Source	421K
ll_xist-3.3-py2.5-win32.egg	Windows egg (Python 2.5)	705K

6.15.194 3.2.7 (released 05/16/2008)

File	Type	Size
ll-xist-3.2.7.tar.bz2	Source	264K
ll-xist-3.2.7.tar.gz	Source	316K
ll-xist-3.2.7.zip	Source	376K

6.15.195 3.2.6 (released 05/07/2008)

File	Type	Size
ll-xist-3.2.6.tar.bz2	Source	273K
ll-xist-3.2.6.tar.gz	Source	325K
ll-xist-3.2.6.zip	Source	379K

6.15.196 3.2.5 (released 04/11/2008)

File	Type	Size
ll-xist-3.2.5.tar.bz2	Source	272K
ll-xist-3.2.5.tar.gz	Source	325K
ll-xist-3.2.5.win32-py2.5.exe	Windows installer (Python 2.5)	366K
ll-xist-3.2.5.zip	Source	380K
ll_xist-3.2.5-py2.5-win32.egg	Windows egg (Python 2.5)	638K

6.15.197 3.2.4 (released 04/02/2008)

File	Type	Size
ll-xist-3.2.4.tar.bz2	Source	273K
ll-xist-3.2.4.tar.gz	Source	326K
ll-xist-3.2.4.win32-py2.5.exe	Windows installer (Python 2.5)	364K
ll-xist-3.2.4.zip	Source	381K
ll_xist-3.2.4-py2.5-win32.egg	Windows egg (Python 2.5)	635K

6.15.198 3.2.3 (released 03/04/2008)

File	Type	Size
ll-xist-3.2.3.tar.bz2	Source	272K
ll-xist-3.2.3.tar.gz	Source	324K
ll-xist-3.2.3.win32-py2.5.exe	Windows installer (Python 2.5)	362K
ll-xist-3.2.3.zip	Source	379K
ll_xist-3.2.3-py2.5-win32.egg	Windows egg (Python 2.5)	632K

6.15.199 3.2.2 (released 02/25/2008)

File	Type	Size
ll-xist-3.2.2.tar.bz2	Source	271K
ll-xist-3.2.2.tar.gz	Source	324K
ll-xist-3.2.2.win32-py2.5.exe	Windows installer (Python 2.5)	362K
ll-xist-3.2.2.zip	Source	379K
ll_xist-3.2.2-py2.5-win32.egg	Windows egg (Python 2.5)	632K

6.15.200 3.2.1 (released 02/05/2008)

File	Type	Size
ll-xist-3.2.1.tar.bz2	Source	263K
ll-xist-3.2.1.tar.gz	Source	315K
ll-xist-3.2.1.win32-py2.5.exe	Windows installer (Python 2.5)	362K
ll-xist-3.2.1.zip	Source	374K
ll_xist-3.2.1-py2.5-win32.egg	Windows egg (Python 2.5)	632K

6.15.201 3.2 (released 02/01/2008)

File	Type	Size
ll-xist-3.2.tar.bz2	Source	271K
ll-xist-3.2.tar.gz	Source	324K
ll-xist-3.2.win32-py2.5.exe	Windows installer (Python 2.5)	363K
ll-xist-3.2.zip	Source	378K
ll_xist-3.2-py2.5-win32.egg	Windows egg (Python 2.5)	631K

6.15.202 3.1 (released 01/18/2008)

File	Type	Size
ll-xist-3.1.tar.bz2	Source	208K
ll-xist-3.1.tar.gz	Source	256K
ll-xist-3.1.win32-py2.5.exe	Windows installer (Python 2.5)	289K
ll-xist-3.1.zip	Source	305K
ll_xist-3.1-py2.5-win32.egg	Windows egg (Python 2.5)	464K

6.15.203 3.0 (released 01/07/2008)

File	Type	Size
ll-xist-3.0.tar.bz2	Source	205K
ll-xist-3.0.tar.gz	Source	253K
ll-xist-3.0.win32-py2.5.exe	Windows installer (Python 2.5)	296K
ll-xist-3.0.zip	Source	303K

6.15.204 2.15.5 (released 07/17/2007)

File	Type	Size
ll-xist-2.15.5.tar.bz2	Source	238K
ll-xist-2.15.5.tar.gz	Source	292K
ll-xist-2.15.5.zip	Source	346K
ll_xist-2.15.5-py2.5-linux-i686.egg	Linux egg (Python 2.5)	548K

6.15.205 2.15.4 (released 07/16/2007)

File	Type	Size
ll-xist-2.15.4.tar.bz2	Source	286K
ll-xist-2.15.4.tar.gz	Source	375K
ll-xist-2.15.4.zip	Source	429K
ll_xist-2.15.4-py2.5-linux-i686.egg	Linux egg (Python 2.5)	548K

6.15.206 2.15.3 (released 07/16/2007)

File	Type	Size
ll-xist-2.15.3.tar.bz2	Source	282K
ll-xist-2.15.3.tar.gz	Source	370K
ll-xist-2.15.3.win32-py2.5.exe	Windows installer (Python 2.5)	282K
ll-xist-2.15.3.zip	Source	423K
ll_xist-2.15.3-py2.5-linux-i686.egg	Linux egg (Python 2.5)	548K

6.15.207 2.15.2 (released 01/24/2007)

File	Type	Size
ll-xist-2.15.2.tar.bz2	Source	282K
ll-xist-2.15.2.tar.gz	Source	370K
ll-xist-2.15.2.win32-py2.4.exe	Windows installer (Python 2.4)	294K
ll-xist-2.15.2.win32-py2.5.exe	Windows installer (Python 2.5)	282K
ll-xist-2.15.2.zip	Source	423K
ll_xist-2.15.2-py2.4-win32.egg	Windows egg (Python 2.4)	477K
ll_xist-2.15.2-py2.5-linux-i686.egg	Linux egg (Python 2.5)	548K

6.15.208 2.15.1 (released 09/25/2006)

File	Type	Size
ll-xist-2.15.1.tar.bz2	Source	282K
ll-xist-2.15.1.tar.gz	Source	369K
ll-xist-2.15.1.win32-py2.4.exe	Windows installer (Python 2.4)	294K
ll-xist-2.15.1.zip	Source	422K
ll_xist-2.15.1-py2.4-linux-i686.egg	Linux egg (Python 2.4)	548K
ll_xist-2.15.1-py2.4-win32.egg	Windows egg (Python 2.4)	477K

6.15.209 2.15 (released 09/24/2006)

File	Type	Size
ll-xist-2.15.tar.bz2	Source	282K
ll-xist-2.15.tar.gz	Source	369K
ll-xist-2.15.win32-py2.4.exe	Windows installer (Python 2.4)	294K
ll-xist-2.15.zip	Source	422K
ll_xist-2.15-py2.4-linux-i686.egg	Linux egg (Python 2.4)	548K
ll_xist-2.15-py2.4-win32.egg	Windows egg (Python 2.4)	477K
ll_xist-2.15-py2.5-linux-i686.egg	Linux egg (Python 2.5)	4122K

6.15.210 2.14.2 (released 07/04/2006)

File	Type	Size
ll-xist-2.14.2.tar.bz2	Source	272K
ll-xist-2.14.2.tar.gz	Source	360K
ll-xist-2.14.2.win32-py2.4.exe	Windows installer (Python 2.4)	286K
ll-xist-2.14.2.zip	Source	443K
ll_xist-2.14.2-py2.4-linux-i686.egg	Linux egg (Python 2.4)	515K
ll_xist-2.14.2-py2.4-win32.egg	Windows egg (Python 2.4)	467K
ll_xist-2.14.2-py2.5-linux-i686.egg	Linux egg (Python 2.5)	4120K

6.15.211 2.14.1 (released 06/29/2006)

File	Type	Size
ll-xist-2.14.1.tar.bz2	Source	272K
ll-xist-2.14.1.tar.gz	Source	360K
ll-xist-2.14.1.win32-py2.4.exe	Windows installer (Python 2.4)	284K
ll-xist-2.14.1.zip	Source	443K
ll_xist-2.14.1-py2.4-linux-i686.egg	Linux egg (Python 2.4)	515K
ll_xist-2.14.1-py2.4-win32.egg	Windows egg (Python 2.4)	466K

6.15.212 2.14 (released 06/28/2006)

File	Type	Size
ll-xist-2.14.tar.bz2	Source	272K
ll-xist-2.14.tar.gz	Source	361K
ll-xist-2.14.win32-py2.4.exe	Windows installer (Python 2.4)	284K
ll-xist-2.14.zip	Source	442K
ll_xist-2.14-py2.4-linux-i686.egg	Linux egg (Python 2.4)	508K
ll_xist-2.14-py2.4-win32.egg	Windows egg (Python 2.4)	466K

6.15.213 2.13 (released 10/31/2005)

File	Type	Size
ll-xist-2.13.tar.bz2	Source	256K
ll-xist-2.13.tar.gz	Source	323K
ll-xist-2.13.win32-py2.4.exe	Windows installer (Python 2.4)	261K
ll-xist-2.13.zip	Source	404K

6.15.214 2.12 (released 10/13/2005)

File	Type	Size
ll-xist-2.12.tar.bz2	Source	256K
ll-xist-2.12.tar.gz	Source	323K
ll-xist-2.12.win32-py2.4.exe	Windows installer (Python 2.4)	290K
ll-xist-2.12.zip	Source	409K

6.15.215 2.11 (released 07/29/2005)

File	Type	Size
ll-xist-2.11.tar.bz2	Source	258K
ll-xist-2.11.tar.gz	Source	323K
ll-xist-2.11.win32-py2.4.exe	Windows installer (Python 2.4)	284K
ll-xist-2.11.zip	Source	399K

6.15.216 2.10 (released 05/20/2005)

File	Type	Size
ll-xist-2.10.tar.bz2	Source	253K
ll-xist-2.10.tar.gz	Source	312K
ll-xist-2.10.win32-py2.4.exe	Windows installer (Python 2.4)	278K
ll-xist-2.10.zip	Source	390K

6.15.217 2.9 (released 04/21/2005)

File	Type	Size
ll-xist-2.9.tar.bz2	Source	249K
ll-xist-2.9.tar.gz	Source	331K
ll-xist-2.9.win32-py2.4.exe	Windows installer (Python 2.4)	278K
ll-xist-2.9.zip	Source	403K

6.15.218 2.8.1 (released 03/22/2005)

File	Type	Size
ll-xist-2.8.1.tar.bz2	Source	246K
ll-xist-2.8.1.tar.gz	Source	327K
ll-xist-2.8.1.win32-py2.4.exe	Windows installer (Python 2.4)	278K
ll-xist-2.8.1.zip	Source	401K

6.15.219 2.8 (released 01/03/2005)

File	Type	Size
ll-xist-2.8.tar.bz2	Source	246K
ll-xist-2.8.tar.gz	Source	327K
ll-xist-2.8.win32-py2.4.exe	Windows installer (Python 2.4)	278K
ll-xist-2.8.zip	Source	398K

6.15.220 2.7 (released 11/24/2004)

File	Type	Size
ll-xist-2.7.tar.bz2	Source	246K
ll-xist-2.7.tar.gz	Source	326K
ll-xist-2.7.win32-py2.3.exe	Windows installer (Python 2.3)	283K
ll-xist-2.7.win32-py2.4.exe	Windows installer (Python 2.4)	279K
ll-xist-2.7.zip	Source	398K

6.15.221 2.6.2 (released 06/06/2005)

File	Type	Size
ll-xist-2.6.2.tar.bz2	Source	245K
ll-xist-2.6.2.tar.gz	Source	326K
ll-xist-2.6.2.zip	Source	400K

6.15.222 2.6.1 (released 11/02/2004)

File	Type	Size
ll-xist-2.6.1.tar.bz2	Source	245K
ll-xist-2.6.1.tar.gz	Source	326K
ll-xist-2.6.1.win32-py2.3.exe	Windows installer (Python 2.3)	283K
ll-xist-2.6.1.zip	Source	399K

6.15.223 2.6 (released 10/26/2004)

File	Type	Size
ll-xist-2.6.tar.bz2	Source	245K
ll-xist-2.6.tar.gz	Source	325K
ll-xist-2.6.win32-py2.3.exe	Windows installer (Python 2.3)	284K
ll-xist-2.6.zip	Source	397K

6.15.224 2.5 (released 06/30/2004)

File	Type	Size
ll-xist-2.5.tar.bz2	Source	241K
ll-xist-2.5.tar.gz	Source	316K
ll-xist-2.5.win32-py2.3.exe	Windows installer (Python 2.3)	244K
ll-xist-2.5.zip	Source	386K

6.15.225 2.4.1 (released 01/05/2004)

File	Type	Size
ll-xist-2.4.1.tar.bz2	Source	223K
ll-xist-2.4.1.tar.gz	Source	286K
ll-xist-2.4.1.win32-py2.3.exe	Windows installer (Python 2.3)	231K
ll-xist-2.4.1.zip	Source	357K

6.15.226 2.4 (released 01/02/2004)

File	Type	Size
ll-xist-2.4.tar.bz2	Source	223K
ll-xist-2.4.tar.gz	Source	286K
ll-xist-2.4.win32-py2.3.exe	Windows installer (Python 2.3)	231K
ll-xist-2.4.zip	Source	355K

6.15.227 2.3 (released 12/08/2003)

File	Type	Size
ll-xist-2.3.tar.bz2	Source	222K
ll-xist-2.3.tar.gz	Source	287K
ll-xist-2.3.win32-py2.3.exe	Windows installer (Python 2.3)	239K
ll-xist-2.3.zip	Source	356K

6.15.228 2.2 (released 07/31/2003)

File	Type	Size
ll-xist-2.2.tar.bz2	Source	224K
ll-xist-2.2.tar.gz	Source	288K
ll-xist-2.2.win32-py2.3.exe	Windows installer (Python 2.3)	235K
ll-xist-2.2.zip	Source	354K

6.15.229 2.1.4 (released 06/13/2003)

File	Type	Size
XIST-2.1.4.tar.bz2	Source	197K
XIST-2.1.4.tar.gz	Source	256K
XIST-2.1.4.win32-py2.2.exe	Windows installer (Python 2.2)	234K
XIST-2.1.4.zip	Source	339K

6.15.230 2.1.3 (released 05/07/2003)

File	Type	Size
XIST-2.1.3.tar.bz2	Source	196K
XIST-2.1.3.tar.gz	Source	256K
XIST-2.1.3.win32-py2.2.exe	Windows installer (Python 2.2)	234K
XIST-2.1.3.zip	Source	338K

6.15.231 2.1.2 (released 02/27/2003)

File	Type	Size
XIST-2.1.2.tar.bz2	Source	196K
XIST-2.1.2.tar.gz	Source	255K
XIST-2.1.2.win32-py2.2.exe	Windows installer (Python 2.2)	234K
XIST-2.1.2.zip	Source	315K

6.15.232 2.1.1 (released 02/11/2003)

File	Type	Size
XIST-2.1.1.tar.bz2	Source	196K
XIST-2.1.1.tar.gz	Source	253K
XIST-2.1.1.win32-py2.2.exe	Windows installer (Python 2.2)	234K
XIST-2.1.1.zip	Source	338K

6.15.233 2.1 (released 12/09/2002)

File	Type	Size
XIST-2.1.tar.bz2	Source	195K
XIST-2.1.tar.gz	Source	252K
XIST-2.1.win32-py2.2.exe	Windows installer (Python 2.2)	234K
XIST-2.1.zip	Source	337K

6.15.234 2.0.8 (released 11/20/2002)

File	Type	Size
XIST-2.0.8.tar.bz2	Source	196K
XIST-2.0.8.tar.gz	Source	248K
XIST-2.0.8.win32-py2.2.exe	Windows installer (Python 2.2)	192K
XIST-2.0.8.zip	Source	329K

6.15.235 2.0.7 (released 11/12/2002)

File	Type	Size
XIST-2.0.7.tar.bz2	Source	196K
XIST-2.0.7.tar.gz	Source	248K
XIST-2.0.7.win32-py2.2.exe	Windows installer (Python 2.2)	192K
XIST-2.0.7.zip	Source	329K

6.15.236 2.0.6 (released 11/11/2002)

File	Type	Size
XIST-2.0.6.tar.bz2	Source	196K
XIST-2.0.6.tar.gz	Source	248K
XIST-2.0.6.win32-py2.2.exe	Windows installer (Python 2.2)	192K
XIST-2.0.6.zip	Source	329K

6.15.237 2.0.5 (released 11/11/2002)

File	Type	Size
XIST-2.0.5.tar.bz2	Source	196K
XIST-2.0.5.tar.gz	Source	247K
XIST-2.0.5.win32-py2.2.exe	Windows installer (Python 2.2)	192K
XIST-2.0.5.zip	Source	329K

6.15.238 2.0.4 (released 11/08/2002)

File	Type	Size
XIST-2.0.4.tar.bz2	Source	192K
XIST-2.0.4.tar.gz	Source	251K
XIST-2.0.4.win32-py2.2.exe	Windows installer (Python 2.2)	192K
XIST-2.0.4.zip	Source	329K

6.15.239 2.0.3 (released 10/30/2002)

File	Type	Size
XIST-2.0.3.tar.bz2	Source	196K
XIST-2.0.3.tar.gz	Source	247K
XIST-2.0.3.win32-py2.2.exe	Windows installer (Python 2.2)	192K
XIST-2.0.3.zip	Source	329K

6.15.240 2.0.2 (released 10/21/2002)

File	Type	Size
XIST-2.0.2.tar.bz2	Source	194K
XIST-2.0.2.tar.gz	Source	244K
XIST-2.0.2.win32-py2.2.exe	Windows installer (Python 2.2)	191K
XIST-2.0.2.zip	Source	324K

6.15.241 2.0.1 (released 10/17/2002)

File	Type	Size
XIST-2.0.1.tar.bz2	Source	193K
XIST-2.0.1.tar.gz	Source	243K
XIST-2.0.1.win32-py2.2.exe	Windows installer (Python 2.2)	191K
XIST-2.0.1.zip	Source	323K

6.15.242 2.0 (released 10/16/2002)

File	Type	Size
XIST-2.0.tar.bz2	Source	193K
XIST-2.0.tar.gz	Source	243K
XIST-2.0.win32-py2.2.exe	Windows installer (Python 2.2)	191K
XIST-2.0.zip	Source	322K

6.15.243 1.6.1 (released 08/25/2003)

File	Type	Size
XIST-1.6.1.tar.bz2	Source	161K
XIST-1.6.1.tar.gz	Source	206K
XIST-1.6.1.win32-py2.3.exe	Windows installer (Python 2.3)	203K
XIST-1.6.1.zip	Source	288K

6.15.244 1.6 (released 07/02/2003)

File	Type	Size
XIST-1.6.tar.bz2	Source	159K
XIST-1.6.tar.gz	Source	203K
XIST-1.6.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.6.zip	Source	284K

6.15.245 1.5.13 (released 07/01/2003)

File	Type	Size
XIST-1.5.13.tar.bz2	Source	160K
XIST-1.5.13.tar.gz	Source	203K
XIST-1.5.13.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.5.13.zip	Source	286K

6.15.246 1.5.12 (released 06/17/2003)

File	Type	Size
XIST-1.5.12.tar.bz2	Source	160K
XIST-1.5.12.tar.gz	Source	203K
XIST-1.5.12.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.5.12.zip	Source	286K

6.15.247 1.5.11 (released 06/13/2003)

File	Type	Size
XIST-1.5.11.tar.bz2	Source	160K
XIST-1.5.11.tar.gz	Source	203K
XIST-1.5.11.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.5.11.zip	Source	286K

6.15.248 1.5.10 (released 06/13/2003)

File	Type	Size
XIST-1.5.10.tar.bz2	Source	159K
XIST-1.5.10.tar.gz	Source	202K
XIST-1.5.10.win32-py2.2.exe	Windows installer (Python 2.2)	189K
XIST-1.5.10.zip	Source	248K

6.15.249 1.5.9 (released 04/30/2003)

File	Type	Size
XIST-1.5.9.tar.bz2	Source	159K
XIST-1.5.9.tar.gz	Source	202K
XIST-1.5.9.win32-py2.2.exe	Windows installer (Python 2.2)	189K
XIST-1.5.9.zip	Source	248K

6.15.250 1.5.8 (released 02/27/2003)

File	Type	Size
XIST-1.5.8.tar.bz2	Source	159K
XIST-1.5.8.tar.gz	Source	202K
XIST-1.5.8.win32-py2.2.exe	Windows installer (Python 2.2)	189K
XIST-1.5.8.zip	Source	247K

6.15.251 1.5.7 (released 11/12/2002)

File	Type	Size
XIST-1.5.7.tar.bz2	Source	159K
XIST-1.5.7.tar.gz	Source	202K
XIST-1.5.7.win32-py2.2.exe	Windows installer (Python 2.2)	182K
XIST-1.5.7.zip	Source	247K

6.15.252 1.5.6 (released 11/11/2002)

File	Type	Size
XIST-1.5.6.tar.bz2	Source	159K
XIST-1.5.6.tar.gz	Source	202K
XIST-1.5.6.win32-py2.2.exe	Windows installer (Python 2.2)	182K
XIST-1.5.6.zip	Source	247K

6.15.253 1.5.5 (released 11/11/2002)

File	Type	Size
XIST-1.5.5.tar.bz2	Source	159K
XIST-1.5.5.tar.gz	Source	202K
XIST-1.5.5.win32-py2.2.exe	Windows installer (Python 2.2)	182K
XIST-1.5.5.zip	Source	247K

6.15.254 1.5.4 (released 09/30/2002)

File	Type	Size
XIST-1.5.4.tar.bz2	Source	159K
XIST-1.5.4.tar.gz	Source	202K
XIST-1.5.4.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.5.4.zip	Source	284K

6.15.255 1.5.3 (released 09/25/2002)

File	Type	Size
XIST-1.5.3.tar.bz2	Source	159K
XIST-1.5.3.tar.gz	Source	201K
XIST-1.5.3.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.5.3.zip	Source	284K

6.15.256 1.5.2 (released 09/19/2002)

File	Type	Size
XIST-1.5.2.tar.bz2	Source	159K
XIST-1.5.2.tar.gz	Source	201K
XIST-1.5.2.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.5.2.zip	Source	284K

6.15.257 1.5.1 (released 09/17/2002)

File	Type	Size
XIST-1.5.1.tar.bz2	Source	159K
XIST-1.5.1.tar.gz	Source	201K
XIST-1.5.1.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.5.1.zip	Source	284K

6.15.258 1.5 (released 08/27/2002)

File	Type	Size
XIST-1.5.tar.bz2	Source	159K
XIST-1.5.tar.gz	Source	200K
XIST-1.5.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.5.zip	Source	283K

6.15.259 1.4.5 (released 06/18/2002)

File	Type	Size
XIST-1.4.5.tar.bz2	Source	159K
XIST-1.4.5.tar.gz	Source	200K
XIST-1.4.5.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.4.5.zip	Source	283K

6.15.260 1.4.4 (released 06/16/2002)

File	Type	Size
XIST-1.4.4.tar.bz2	Source	158K
XIST-1.4.4.tar.gz	Source	200K
XIST-1.4.4.win32-py2.2.exe	Windows installer (Python 2.2)	155K
XIST-1.4.4.zip	Source	283K

6.15.261 1.4.3 (released 04/29/2002)

File	Type	Size
XIST-1.4.3.tar.bz2	Source	158K
XIST-1.4.3.tar.gz	Source	200K
XIST-1.4.3.win32-py2.2.exe	Windows installer (Python 2.2)	154K
XIST-1.4.3.zip	Source	283K

6.15.262 1.4.2 (released 03/22/2002)

File	Type	Size
XIST-1.4.2.tar.bz2	Source	157K
XIST-1.4.2.tar.gz	Source	199K
XIST-1.4.2.win32-py2.2.exe	Windows installer (Python 2.2)	152K
XIST-1.4.2.zip	Source	280K

6.15.263 1.4.1 (released 03/21/2002)

File	Type	Size
XIST-1.4.1.tar.bz2	Source	157K
XIST-1.4.1.tar.gz	Source	199K
XIST-1.4.1.win32-py2.2.exe	Windows installer (Python 2.2)	152K
XIST-1.4.1.zip	Source	280K

6.15.264 1.4 (released 03/18/2002)

File	Type	Size
XIST-1.4.tar.bz2	Source	157K
XIST-1.4.tar.gz	Source	198K
XIST-1.4.win32-py2.2.exe	Windows installer (Python 2.2)	158K
XIST-1.4.zip	Source	279K

6.15.265 1.3.1 (released 03/14/2002)

File	Type	Size
XIST-1.3.1.tar.bz2	Source	161K
XIST-1.3.1.tar.gz	Source	203K
XIST-1.3.1.win32-py2.2.exe	Windows installer (Python 2.2)	158K
XIST-1.3.1.zip	Source	285K

6.15.266 1.3 (released 02/12/2002)

File	Type	Size
XIST-1.3.tar.bz2	Source	161K
XIST-1.3.tar.gz	Source	203K
XIST-1.3.win32-py2.2.exe	Windows installer (Python 2.2)	157K
XIST-1.3.zip	Source	283K

6.15.267 1.2.5 (released 12/03/2001)

File	Type	Size
XIST-1.2.5.tar.bz2	Source	123K
XIST-1.2.5.tar.gz	Source	161K
XIST-1.2.5.win32-py2.1.exe	Windows installer (Python 2.1)	123K
XIST-1.2.5.zip	Source	232K

6.15.268 1.2.4 (released 11/23/2001)

File	Type	Size
XIST-1.2.4.tar.bz2	Source	123K
XIST-1.2.4.tar.gz	Source	161K
XIST-1.2.4.win32-py2.1.exe	Windows installer (Python 2.1)	122K
XIST-1.2.4.zip	Source	192K

6.15.269 1.2.3 (released 11/22/2001)

File	Type	Size
XIST-1.2.3.tar.bz2	Source	123K
XIST-1.2.3.tar.gz	Source	161K
XIST-1.2.3.win32-py2.1.exe	Windows installer (Python 2.1)	122K
XIST-1.2.3.zip	Source	191K

6.15.270 1.2.2 (released 11/16/2001)

File	Type	Size
XIST-1.2.2.tar.bz2	Source	123K
XIST-1.2.2.tar.gz	Source	160K
XIST-1.2.2.win32-py2.1.exe	Windows installer (Python 2.1)	122K
XIST-1.2.2.zip	Source	191K

6.15.271 1.2.1 (released 10/08/2001)

File	Type	Size
XIST-1.2.1.tar.bz2	Source	120K
XIST-1.2.1.tar.gz	Source	154K
XIST-1.2.1.win32-py2.1.exe	Windows installer (Python 2.1)	121K
XIST-1.2.1.zip	Source	190K

6.15.272 1.2 (released 10/03/2001)

File	Type	Size
XIST-1.2.tar.bz2	Source	120K
XIST-1.2.tar.gz	Source	155K
XIST-1.2.win32-py2.1.exe	Windows installer (Python 2.1)	121K
XIST-1.2.zip	Source	190K

6.15.273 1.1.3 (released 09/17/2001)

File	Type	Size
XIST-1.1.3.tar.bz2	Source	119K
XIST-1.1.3.tar.gz	Source	156K
XIST-1.1.3.win32-py2.1.exe	Windows installer (Python 2.1)	113K
XIST-1.1.3.zip	Source	181K

6.15.274 1.1.2 (released 08/21/2001)

File	Type	Size
XIST-1.1.2.tar.bz2	Source	120K
XIST-1.1.2.tar.gz	Source	157K
XIST-1.1.2.win32-py2.1.exe	Windows installer (Python 2.1)	110K
XIST-1.1.2.zip	Source	185K

6.15.275 1.1.1 (released 08/01/2001)

File	Type	Size
XIST-1.1.1.tar.bz2	Source	123K
XIST-1.1.1.tar.gz	Source	160K
XIST-1.1.1.win32-py2.1.exe	Windows installer (Python 2.1)	110K
XIST-1.1.1.zip	Source	184K

6.15.276 1.1 (released 07/19/2001)

File	Type	Size
XIST-1.1.tar.bz2	Source	121K
XIST-1.1.tar.gz	Source	158K
XIST-1.1.win32-py2.1.exe	Windows installer (Python 2.1)	110K
XIST-1.1.zip	Source	184K

6.15.277 1.0 (released 06/18/2001)

File	Type	Size
XIST-1.0.tar.bz2	Source	118K
XIST-1.0.tar.gz	Source	155K
XIST-1.0.win32-py2.1.exe	Windows installer (Python 2.1)	107K
XIST-1.0.zip	Source	181K

6.15.278 0.4.7 (released 11/24/2000)

File	Type	Size
xist-0.4.7.tar.bz2	Source	81K
xist-0.4.7.tar.gz	Source	107K

6.15.279 0.4.6 (released 11/03/2000)

File	Type	Size
xist-0.4.6.tar.bz2	Source	78K
xist-0.4.6.tar.gz	Source	104K

6.15.280 0.4.5 (released 11/01/2000)

File	Type	Size
xist-0.4.5.tar.bz2	Source	78K
xist-0.4.5.tar.gz	Source	104K

6.15.281 0.4.4 (released 10/27/2000)

File	Type	Size
xist-0.4.4.tar.bz2	Source	78K
xist-0.4.4.tar.gz	Source	103K

6.15.282 0.4.3 (released 10/19/2000)

File	Type	Size
xist-0.4.3.tar.bz2	Source	78K
xist-0.4.3.tar.gz	Source	103K

6.15.283 0.4.2 (released 09/24/2000)

File	Type	Size
xist-0.4.2.tar.bz2	Source	77K
xist-0.4.2.tar.gz	Source	102K

6.15.284 0.4.1 (released 09/21/2000)

File	Type	Size
xist-0.4.1.tar.bz2	Source	77K
xist-0.4.1.tar.gz	Source	102K

6.15.285 0.4 (released 09/19/2000)

File	Type	Size
xist-0.4.tar.bz2	Source	76K
xist-0.4.tar.gz	Source	101K

6.15.286 0.3.9 (released 08/10/2000)

File	Type	Size
xist-0.3.9.tar.bz2	Source	71K
xist-0.3.9.tar.gz	Source	94K

6.15.287 0.3.8 (released 07/14/2000)

File	Type	Size
xist-0.3.8.tar.gz	Source	93K

6.15.288 0.3.7 (released 07/06/2000)

File	Type	Size
xist-0.3.7.tar.gz	Source	93K

6.15.289 0.3.6 (released 07/04/2000)

File	Type	Size
xist-0.3.6.tar.gz	Source	93K

6.15.290 0.3.5 (released 07/02/2000)

File	Type	Size
xist-0.3.5.tar.gz	Source	93K

6.15.291 0.3.4 (released 05/31/2000)

File	Type	Size
xist-0.3.4.tar.gz	Source	84K

6.15.292 0.3.3 (released 05/30/2000)

File	Type	Size
xist-0.3.3.tar.gz	Source	84K

6.15.293 0.3.2 (released 05/26/2000)

File	Type	Size
xist-0.3.2.tar.gz	Source	83K

6.15.294 0.3.1 (released 05/25/2000)

File	Type	Size
xist-0.3.1.tar.gz	Source	83K

6.15.295 0.3 (released 05/25/2000)

File	Type	Size
xist-0.3.tar.gz	Source	83K

6.15.296 0.2 (released 05/17/2000)

File	Type	Size
xist-0.2.tar.gz	Source	78K

6.15.297 0.1.1 (released 05/16/2000)

File	Type	Size
xist-0.1.1.tar.gz	Source	78K

6.15.298 0.1 (released 05/15/2000)

File	Type	Size
xist-0.1.tar.gz	Source	78K

6.15.299 Older packages

Older versions of some modules were distributed seperately before XIST 3.2 (when the `ll.core` package was merged into XIST) and XIST 3.7 (when *`ll.orasql`* and *`ll.nightshade`* were merged into XIST). To download those packages you have to hunt around in the HTTP download directory.

6.16 History

This document describes the new features, bug fixes and changes in each version of XIST. For a description of how to update your code to each versions of XIST see *Migration*.

6.16.1 Changes in 5.75.1 (released 2024-04-11)

- Added support for the HTML pseudo class `:disabled` to *`ll.xist.css.applystylesheets()`*. Now the following works:

```
>>> from ll.xist.ns import html
>>> from ll.xist import css
>>> e = html.html(
...     html.style('''
...        button {color: blue;}
...        button:disabled {color: red;}
...        '''),
...     html.button('gurk', disabled=True)
...   )
...
>>> css.applystylesheets(e)
>>> e.string()
'<html><button disabled="disabled" style="color: red">gurk</button></html>'
```

6.16.2 Changes in 5.75 (released 2023-11-21)

- In UL4 <?doc?> and <?note?> can now be nested, and otherwise can contain anything up to the matching <?end doc?> or <?end note?> tag, but only if the initial <?doc?> or <?note?> tag contains only whitespace (e.g. <?doc ?> etc.). This makes it possible to have source code examples in <?doc?> or <?note?> tags.
- Added a new class `BoundTemplate` to *ll.ul4c*: This is an UL4 template bound to an object.

  Calling or rendering a `BoundTemplate` instance passes the object to which the template is bound as the first positional argument.
- Add attribute `namespace` to UL4 templates.
- Add method `timestamp` to UL4 type `datetime`.
- The UL4 function `getattr` now returns an `UndefinedKey` object when accessing an undefined attribute and no default value is passed.
- The result of calling `dir()` on an object in UL4 can now be customized by implementing `ul4_dir()`.
- Updated UL4 and UL4ON test infrastructure to work around problems with gradle.
- Replace a check for the ElementTree method `getchildren()` with a check for `findall()` in `xml2xsc.iterpath()` since `getchildren()` has been removed in Python 3.9.
- Copied the implementation of `cgi.parse_header()` to *ll.misc* since `cgi.parse_header()` will be deprecated in Python 3.13. (`cgi.parse_header()` is used by *ll.url*, *ll.xist.ns.html* and *ll.xist.ns.jsp*).

6.16.3 Changes in 5.74 (released 2023-03-01)

- Fixed handling of the `cond` attribute in the PySQL command `commit` and `rollback`.
- Bumped required `cssutils` version to at least 2.6.0.
- *ll.xist.css* now ignores the CSS pseudo classes `:valid` and `:invalid`.
- Since the Java and Javascript versions of UL4 now supports `format(float)` this is now checked by the tests.
- The UL4 function `format(float)` now honors the specified language and uses the correct decimal separator.
- Running the UL4 tests for Java now is done via Gradle, so the tests no longer require Java UL4 and all its dependencies on the `CLASSPATH`. (Instead a full Java UL4 checkout is required in `~/checkouts/LivingLogic.Java.ul4`.)

6.16.4 Changes in 5.73.2 (released 2022-08-16)

- Fixed handing of the CSS selectors `:nth-child(odd)` and `:nth-child(even)` in *ll.xist.css.selector()*.

6.16.5 Changes in 5.73.1 (released 2022-08-10)

- Fixed handling of the PySQL variable `connection` on `connect` and `disconnect` calls.

6.16.6 Changes in 5.73 (released 2022-08-10)

- *ll.pysql* now supports Postgres. To connect to a Postgres database pass a connectstring to `connect` starting with `postgres:`, for example:

  ```
  connect("postgres:host=localhost dbname=test user=me password=secret")
  ```

  This will create a Postgres database connection via:

  ```
  psycopg.connect(
     "host=localhost dbname=test user=me password=secret",
     cursor_factory=extras.DictCursor
  )
  ```

  All other connectstrings will be interpreted as Oracle connectstrings. An Oracle connectstring may start with the prefix `oracle:` which will be stripped off, before passing it to `cx_Oracle.connect()` or *ll.orasql.connect()*.

  Note that Postgres currently doesn't support the `drop_types` command.

- Some PySQL commands have been renamed: `resetsequence` to `reset_sequence`, `checkerrors` to `check_errors`, `raiseexceptions` to `raise_exceptions`, `pushraiseexceptions` to `push_raise_exceptions` and `popraiseexceptions` to `pop_raise_exceptions`.
- The PySQL commands `procedure` and `sql` have an additional argument `argtypes` that can be used to add casts to the parameter values in the call to convert the value to the proper Postgres datatype (to guide Postgres to find the correct overloaded version of the procedure).
- When a `var` object is passed a second time in PySQL, now instead of the variable's value a proper variable object will be passed to the `procedure` or `sql` call. This means if the variable gets changed by the call, the new value will be picked up by the local variable.

  If you want to pass the variable's value instead as a simple IN parameter, simply pass the local variable instead.

- The argument `raiseexceptions` to various PySQL commands has been renamed to `raise_exceptions`.

6.16.7 Changes in 5.72 (released 2022-08-04)

- *ll.ul4c.TextAST* objects now always store the final text string instead of slicing it from the source code on every call.
- UL4 AST classes store source offsets in the UL4ON dump now as two `int`s instead of a `slice` object. The Java and Javascript implementations directly use these two ints instead of converting them to a `slice` object. This reduces memory usage in the Java version. However the Python version still stores `slice` objects internally.

6.16.8 Changes in 5.71 (released 2022-07-08)

- Fixed *ll.xist.xfind.selector()* when running under Python 3.9.
- Restored support for `ll.xist.xsc._Node_Meta.__or__()` for XIST classes under Python 3.9.
- *ll.xist.xfind.Selector* now implements the reflected operators too. This reenables certain argument combinations as `ll.xist.xsc._Node_Meta.__or__()` isn't available on Python 3.10.
- *ll.xist.xfind.NotCombinator* now converts its argument to a *Selector* object, instead of expecting it to already be one.
- Publishing of `<script>` elements now properly outputs the elements content as `CDATA` unless in XML mode. I.e.

```
html.script("&").string()
```

will now output <script>&</script> and

```
html.script("&").string(xhtml=2)
```

outputs <script>&</script> as before.

6.16.9 Changes in 5.70 (released 2022-03-11)

- Fixed a bug in the PySQL command *commit*: Removed the useless argument `sql`.
- Fixed a bug in the method `ll.orasql.Connection.getobject()` (which we only keep for backwards compatibility).
- Fixed a bug in parsing UL4 source code: UL4 would treat a tag name that starts with a valid prefix as a valid tag name, although it shouldn't (i.e. it treated <?printe?> as <?print e?>).
- Added new UL4 AST classes: *RenderOrPrintAST*, *RenderOrPrintXAST*, *RenderXOrPrintAST* and *RenderXOrPrintXAST*.

6.16.10 Changes in 5.69 (released 2021-11-17)

- The UL4 function `urlquote()` now uses `urllib.parse.quote()` instead of `urllib.parse.quote_plus()`. This means that the space character will get encoded as `%20` instead of +.
- `ll.xist.xsc._Node_Meta` no longer overwrites `__or__()` (since in Python 3.10 this now returns a union type). Instead *ll.xist.xfind.selector()* converts a union type into a *ll.xist.xfind.IsInstanceSelector*.
- Added the method `keys` to UL4 `dict` objects (The method was already documented).

6.16.11 Changes in 5.68.1 (released 2021-09-23)

- Fixed a bug in *ll.pysql* that resulted in the local variable `connection` being wrapped in a tuple.

6.16.12 Changes in 5.68 (released 2021-08-04)

- UL4 templates now support the <?ignore?> tag. It can be used to "comment out" template code. <?ignore?>/<?end ignore?> tags nest.
- Added the following three methods to *ll.orasql.Table*:
  - *ll.orasql.Table.fks()* returns all foreign keys for the table;
  - *ll.orasql.Table.uniques()* returns all unique constraints for the table;
  - *ll.orasql.Table.checks()* returns all check constraints for the table.
- Fixed scoping problems in literal Python blocks in PySQL scripts: List comprehension were not able to access local variables.
- Removed the ancient XIST namespace modules `ll.xist.ns.kid` and `ll.xist.ns.ihtml`.

6.16.13 Changes in 5.67.2 (released 2021-06-30)

- Fixed handling of values for the `--define`/`--D` option of *ll.pysql*.

6.16.14 Changes in 5.67.1 (released 2021-06-28)

- Fixed a typo in the tag `external` when logging exceptions in Sisyphus jobs.

6.16.15 Changes in 5.67 (released 2021-06-25)

- Added the options `--sentry_environment`, `--sentry_release` and `--sentry_debug` to Sisyphus jobs.
- Sisyphus jobs now warning when the modulde `sentry_sdk` can not be imported.
- For Sisyphus jobs it's now possible to log to the `ll.sisyphus.EMailLogger`, *ll.sisyphus.MattermostLogger* and *ll.sisyphus.SentryLogger* by using the tag `external`.
- The Sispyhus log message for setting up the Sentry logging is now a delayed message.
- The method `ll.orasql.Connection.getobject()` has been renamed to *ll.orasql.Connection.object_named()*.
- The method `ll.orasql.Connection.objects_name()` has been added that returns all database objects with the specified name.

6.16.16 Changes in 5.66.1 (released 2021-06-24)

- When a sisyphus job logs to Sentry, flush messages after logging them to make sure that they arrive even in forking mode.

6.16.17 Changes in 5.66 (released 2021-06-15)

- UL4 now use functions and methods with positional-only parameters, so Python 3.8 is required now.
- UL4 functions and methods have been updated to use positional-only or keyword-only arguments to match the signature of the corresponding Python function/method.
- Some functions now use positional-only arguments:
  - `ll.misc.item(iterable, index, /, default=None)`
  - `ll.misc.first(iterable, /, default=None)`
  - `ll.misc.last(iterable, /, default=None)`
  - `ll.misc.count(iterable, /)`
  - `ll.misc.isfirst(iterable, /)`
  - `ll.misc.islast(iterable, /)`
  - `ll.misc.isfirstlast(iterable, /)`
  - `ll.misc.monthdelta.__init__(self, months=0, /)`
  - `ll.ul4on.dumps(obj, /, indent)`
  - `ll.ul4on.dump(obj, /, stream, indent)`
  - `ll.ul4on.load(stream, /, registry=None)`
  - `ll.ul4on.loads(dump, /, registry=None)`
  - `ll.ul4on.loadclob(clob, /, bufsize=1024*1024, registry=None)`

as well as the UL4 function `enumfl(iterable, /)`.

- Subclasses of *ll.ul4c.AST* have been renamed so that their name matches the name of the corresponding class in the Java implementation. (For example `ll.ul4c.Add` has been renamed to *ll.ul4c.AddAST*).
- The UL4 function `type()` now returns "type objects" instead of simple strings. A type object can be used for checking whether an object is an instance of a certain type (via the newly introduced function `isinstance()`). Some type objects can also be used to create new instances of that type (by calling the type object).
- The following builtin UL4 functions are now callable type objects: `bool`, `int`, `float`, `str`, `date`, `datetime`, `timedelta`, `monthdelta`, `list`, `set`, `dict` and `color.Color`.
- The following modules have been added to the builtin UL4 objects: `ul4` contains all UL4 AST classes (`ul4.Template` is callable to create an UL4 template from source), `ul4on` contains the functions `dumps()` and `loads()` and the types `Encoder` and `Decoder`, `operator` contains the type `attrgetter` and `math` contains the constants `e`, `pi` and `tau` as well as the functions `cos()`, `sin()`, `tan()`, `sqrt()` and `isclose()`. `color` contains the type `Color` and the functions `css` and `mix`.
- Naming of attributes that are used by UL4 to implement UL4 functionality has been made more uniform. This affects the following attributes: The methods `ul4_getattr()`, `ul4_setattr()` and `ul4_hasattr()` for implementing object attribute access from UL4; the methods `ul4_call()`, `ul4_render()` and `ul4_renders()` for making objects callable or renderable from UL4; the class attribute `ul4_attrs` for exposing a number of readonly attributes to UL4 and the attribute `ul4_context` that is used for marking a callable as needing the context as an argument in the call.
- Now the `ul4_attr` attribute of objects gets honored in the implementation of the `dir()` function in UL4.
- Support for using custom tag delimiters for UL4 templates has been removed, i.e. now the tag delimiters will always be <? and ?>.
- *ll.ul4on.Decoder* gained a new method *forget_persistent_object()*.
- *ll.sisyphus.Job*s can now log to Sentry.
- *ll.sisyphus.Task*s constructor and the method *ll.sisyphus.Job.task* now take arbitrary additional keyword arguments. Those will be passed as additional breadcrumb data when logging to Sentry.
- *ll.sisyphus.Job.tasks()* now takes an additional argument `data` that is responsible for returning additional data for the task.
- The following methods of *ll.color.Color* have been renamed: `abslum()` to `abslight()` and `rellum()` to `rellight()`.
- The following methods have been added to *ll.color.Color*: *hue()*, *light()*, *sat()*, *withhue()*, *withsat()*, *withlum()*, *abslum()*, *rellum()* and *invert()*. They have also been made available to UL4. The color method *combine()* and the functions *ll.color.css()* and *ll.color.mix()* are now also available to UL4.

6.16.18 Changes in 5.65 (released 2021-01-13)

- *ll.ul4on* now supports "persistent" objects, i.e. objects that can be uniquely identified across several unrelated decoding calls. Loading a persistent object that has been loaded before will reinitialize the existing object instead of creating a new one.
- In sisyphus jobs, when the value of the `nextrun` parameter was a `datetime.timedelta` object, it was interpreted relative to the start of the run, not relative to now. This has been fixed.
- The modules `ll._xml_codec` and `ll.xist.sgmlop` have been updated to directly support and produce **PEP 393** style unicode strings. (`ll._misc` already supported them).
- A method `date` has been added to `date` and `datetime` objects in UL4 templates.

6.16.19 Changes in 5.64 (released 2020-10-30)

- It is now possible to specify a custom port for `ssh` URLs.
- A second URL scheme `ssh-nocheck` has been added for `ssh` URLs. Using `ssh-nocheck` disables the host key check when establishing the `ssh` connection.
- With these changes it is now possible to do the following:

```
>>> from ll import url
>>> u = url.URL("ssh-nocheck://www.example.org:2222/~/foo.txt")
>>> with url.Context():
...   data = u.open("rb").read()
```

6.16.20 Changes in 5.63.1 (released 2020-10-26)

- Add a workaround for Python issue #41889 (https://bugs.python.org/issue41889) to *Enum* and *IntEnum*.

6.16.21 Changes in 5.63 (released 2020-09-08)

- All PySQL commands now support the argument `cond`.
- Added a PySQL command `constraint_exists` that checks the existance of database constraints.

6.16.22 Changes in 5.62 (released 2020-07-13)

- Update HTML for the documentation to use our own (RTD based) theme instead of using and overwriting the RTD theme.

6.16.23 Changes in 5.61.2 (released 2020-07-09)

- Fixed a regression in *ll.sisyphus* from 5.61.1: Stay in delayed logs mode after a successful job run.

6.16.24 Changes in 5.61.1 (released 2020-07-09)

- Fixed handling of the `--exit_on_error` option in *ll.sisyphus*.
- Rewrote handling of delayed logs in *ll.sisyphus* to work better in forking mode.

6.16.25 Changes in 5.61 (released 2020-07-07)

- Updated handling of custom data types when calling functions or procedures in *ll.orasql* to work with `cx_Oracle` 8.0.
- Reordered table comments, column comments and the primary key of a table in the output of *ll.orasql.Connection.objects()* so that these objects have the same order as in `ll.orasql.Table.referencedby()`.

6.16.26 Changes in 5.60 (released 2020-07-03)

- The handling of delayed logs and uneventful runs in *ll.sisyphus* has been changed: "Delayed logs" mode is now always active. If only delayed log messages are output they will never be written to the logfiles. If a job run is uneventful (i.e. *execute()* returns `None`) no log messages will be written. If the job run is successful only the job result will be written.

  The option and the class/instance attribute `delaylogs` no longer exist.

- Added a new option and class/instance attribute `exit_on_error` which will stop job execution in repeat mode when an exception occurs.

6.16.27 Changes in 5.59 (released 2020-06-30)

- *ll.orasql* now requires `cx_Oracle` 8.0.
- *ll.orasql.Connection.objects()* now outputs `TableComment` objects too.
- Fixed *ll.orasql.Table.comment()* when the table was owned by a different user.

6.16.28 Changes in 5.58 (released 2020-06-12)

- For running healthchecks for sisyphus jobs it's no longer neccessary (or even allowed) to implement the `healthcheck()` method. Instead the job writes a healthfile at the end of each run, and the age and content of this file will be used to determine the health of the job. The option `--maxhealthcheckage` can be used to configure the maximum allowed age.
- Logging to emails was broken when sisyphus jobs were running in fork mode (the default): The child process was collecting log messages for the email, but the parent process didn't, so it never sent an email. This has been fixed now: Both processes write log messages to files, and those will be used after the job run to create the email.
- Now links will be created for every possible result status of a job run. So it's immediate clear when the last successful job run was, when the last job run failed with an exception, was canceled or timed out.
- The filenames for log files can no longer be changed via options or job attributes, instead one of the following methods must be overwritten:
  - `basedir()`
  - `logfilename()`
  - `currentloglinkname()`
  - `lastsuccessfulloglinkname()`
  - `lastfailedloglinkname()`
  - `lastinterruptedloglinkname()`
  - `lasttimeoutloglinkname()`
  - `healthfilename()`
  - `emailfilename()`

  Those methods must return an absolute path as a `pathlib.Path` object.

6.16.29 Changes in 5.57 (released 2020-04-14)

- Added a "delayed logs" mode to *ll.sisyphus*. This makes it possible to delay output of any log messages until something interesting happens. When nothing interesting happens, log messages will be thrown away.
- Use `pathlib` internally for handling log file names in *ll.sisyphus*.
- When a *ll.sisyphus* job compresses log files the compressed log file now retains the modification timestamp of the original log file.
- The API for *ll.ul4on.Encoder* and *ll.ul4on.Decoder* has been updated to support multiple calls for encoding/decoding an UL4ON dump to multiple strings or streams that nonetheless keep the state for the encoding/decoding machinery.
- UL4ON functionality is now available to UL4 templates in a `ul4on` "module". This module provides the functions/types `loads`, `dumps`, `Decoder` and `Encoder`.
- The parameter `string` for the UL4 function `fromul4on` has been renamed to `dump`.

6.16.30 Changes in 5.56 (released 2019-12-12)

- `ll.orasql.Comment` has been renamed to *ll.orasql.ColumnComment*.
- Added a class *ll.orasql.TableComment* for table comments.
- Added a method *ll.orasql.Table.comment()* that returns the *ll.orasql.TableComment* object for this table.
- UL4 templates now support global variables. To be able to pass global variables to UL4 templates the following methods have been added to *ll.ul4c.Template*: `render_with_globals()`, `renders_with_globals()` and `call_with_globals()`.

6.16.31 Changes in 5.55 (released 2019-11-11)

- Added an option `--ignoreerrors` to `orareindex`.
- UL4 dictionaries now have a method `pop()`.
- Added an UL4 function `scrypt` implementing the scrypt hashing function (see https://en.wikipedia.org/wiki/Scrypt).
- Added a new method *ll.orasql.Table.compression()* that returns the table compression (`None`, `"BASIC"` or `"ADVANCED"`).
- Added a new method *ll.orasql.Column.compression()* that returns the column compression for LOB columns (`None`, `"LOW"`, `"MEDIUM"` or `"HIGH"`).
- `ll.orasql.Table.createsql()` now can handle table and LOB column compression.
- Added a method `loadcontentitems()` to the class *ll.ul4on.Decoder* which can be used to load the content of an object as `(key, value)` pairs.

6.16.32 Changes in 5.54.1 (released 2019-10-24)

- Fixed wrong HTTP header when posting sisyphus log entries to Mattermost.

6.16.33 Changes in 5.54 (released 2019-10-24)

- The tab width used by *ll.xist.ns.html.astext()* is now configurable and long words will no longer be broken across multiple lines. This should prevent long URLs from being broken.
- `ll.misc.sysinfo` now exposes its attributes to UL4.
- *ll.sisyphus* log entries can now be logged to a Mattermost chat channel.

6.16.34 Changes in 5.53 (released 2019-09-30)

- Fixed a bug in the handling of users and job classes (i.e. objects that don't have an owner) in `ll.orasql.OracleURLConnection.walk()`.
- Added an option `--healthcheck` to `ll.sispyhus` jobs: Starting a job with this option runs a separate method `healthcheck()` that is used to check that the job is doing what it's supposed to be doing.

6.16.35 Changes in 5.52.1 (released 2019-09-05)

- Fixed handling of indentation when an UL4 `<?renderblock?>` contains a `<?render?>` call.

6.16.36 Changes in 5.52 (released 2019-07-29)

- The method `getobject()` for *ll.orasql.Synonym* has been renamed to `object()`.
- A new method *ll.orasql.Privilege.object()* has been added. This method returns the object for which the *ll.orasql.Privilege* grants a privilege. E.g. if the *Privilege* `p` grant the `SELECT` privilege on a table, `p.object()` will return that table.
- A new method *ll.orasql.OwnedSchemaObject.synonyms()* has been added. This generater yields all the synonyms for the object it is called on.
- A new method *ll.orasql.OwnedSchemaObject.privileges()* has been added. This generater yields all the privileges for the object it is called on.
- A new method *ll.orasql.Connection.synonyms()* has been added.
- `ll.orasql.Synonym.createsql()` now omits the schema name from the name for the object if it's the current user.

6.16.37 Changes in 5.51 (released 2019-07-26)

- *ll.orasql.Synonym.names()* and *ll.orasql.Synonym.objects()* can now filter on the owner of the object (i.e. the object for which the *Synonym* is a synonym) via the new parameter `object_owner`.
- Fixed the `repr` output of UL4 dictionary comprehensions.

6.16.38 Changes in 5.50 (released 2019-07-16)

- There's a new option `-a/--ascii` for running PySQL scripts: With this PySQL will not use unicode characters for drawing fancy boxes.
- Fixed a bug in the filename handling of PySQL, so that showing source in stacktraces works again: as we're changing directories now, using relative paths no longer worked.
- PySQL no longer uses exception chaining for displaying the location and the include chain of an error, as this is now part of the normal stacktrace anyway.
- Fixed logic for showing line numbers for locations in PySQL scripts.
- Fixed a bug in the PySQL command `resetsequence`.
- Added classes *`ll.misc.Enum`* and *`ll.misc.IntEnum`* that are subclasses of `enum.Enum` and `enum.IntEnum`, but show the module and fully qualified class name in the `repr()` output for classes and instances.

6.16.39 Changes in 5.49 (released 2019-07-04)

- Privileges returned by *`ll.orasql.Privilege.objects()`* will now have a stable sort order.

6.16.40 Changes in 5.48 (released 2019-07-03)

- Filename printed by a PySQL script will now always be relative to the current directory at the start of the script.
- Fixed a bug in the filename handling of the PySQL command `file`.

6.16.41 Changes in 5.47 (released 2019-07-01)

- Include commands in PySQL scripts now actually change the current directory so that literal Python blocks execute with the current directory set to the directory of the PySQL file containing the Python block.
- The PySQL commands `procedure` and `sql` now report new variable values.

6.16.42 Changes in 5.46 (released 2019-06-26)

- The method *`ll.scripts.rul4.Globals.log()`* now supports the keywords arguments `sep`, `end` and `flush` with the same meaning as for `print()`.
- Exception chaining in *`ll.scripts.rul4.Globals.error()`* has been fixed.
- For *`ll.xist.ns.html.astext()`* default styles have been added for `em`, `strong` and `i`.
- *`ll.xist.ns.html.astext()`* now honors all styles passed as keyword arguments not just those for `h1`, `h2`, `h3`, `h4`, `h5`, `h6`, `dl`, `dt`, `dd`, `ol`, `ol_li`, `ul`, `ul_li`, `pre`, `blockquote`, `div`, `p`, `hr`, `address`, `th`, `td`, `b`, `u` and `code`.
- *`ll.xist.ns.html.astext()`* now supports callables for the `prefix` and `suffix` style parameter. If a callable is passed it will be called with the node in question and the resulting string is used as the prefix/suffix. For example it's possible to output links in Markdown syntax like this:

```
>>> from ll.xist.ns import html
>>> e = html.p(
...   "We ",
...   html.em("love"),
...   " ",
...   html.a("Python", href="http://www.python.org/"),
...   "!"
```

(continues on next page)

(continued from previous page)

```
... )
>>> html.astext(
...   e,
...   a=dict(
...      display="inline",
...      prefix="(",
...      suffix=lambda n: f")[{n.attrs.href}]"
...   )
... )
'We *love* (Python)[http://www.python.org/]!'
```

6.16.43 Changes in 5.45 (released 2019-06-24)

- UL4 AST nodes for blocks now have additional attributes `startpos` and `stoppos`. `startpos` is the position of the start tag and `stoppos` is the position of the end tag. The attributes `line` and `col` have been renamed to `startline` and `startcol` and attributes `stopline` and `stopcol` have been added.

  Furthermore two attributes `startsource` and `stopsource` have been added. They return the source code of the start tag and the end tag. So for example for the loop `<?for i in range(10)?><?print i?><?end for?>` the `startsource` is `<?for i in range(10)?>` and the `stopsource` is `<?end for?>` (and `source` is `<?for i in range(10)?><?print i?><?end for?>`).

  Additionally attributes `startsourceprefix`, `startsourcesuffix`, `stopsourceprefix` and `stopsourcesuffix` have been added.

  In exception messages `startsource` is now used as the exception location. This means when a for loop iterates over something that is not iteratable the location marked will now be the loop start tag instead of the complete loop.

  (For non-block nodes `startpos` is the same as `pos`, `startsource` is the same as `source`, `startsourceprefix` is the same as `sourceprefix` and `startsourcesuffix` is the same as `sourcesuffix`.)

6.16.44 Changes in 5.44 (released 2019-06-07)

- *`ll.orasql.Connection.objects()`* now outputs `Job` objects too. Since Oracle provides no dependency information about jobs, `Job` objects will always be output last (in "create" mode; in "drop" mode they will be output first).
- `ll.orasql.Job.references()` will now yield the appropriate *`ll.orasql.JobClass`* object (if the job class isn't a system job class).
- `ll.orasql.JobClass.referencedby()` will now yield all *`ll.orasql.Job`* objects that use this job class.
- The `owner` argument for various *`ll.orasql`* methods now supports passing a set or tuple of owner names.
- PySQL scripts now can contains PySQL commands in "function call form", i.e. `checkerrors()` instead of `{'type': 'checkerrors'}`.
- PySQL scripts can now contains literal Python source code (between lines with #>>> and #<<<, e.g.:

```
#>>>
``cursor`` .cursor()
cursor.execute("drop user foo cascade")
#<<<
```

- Comments in PySQL scripts are supported now (via lines starting with #).

- Since PySQL scripts can open their own database connections the `connectstring` argument for the `pysql` script is now optional.
- The PySQL command `compileall` has been removed. This same effect can simply be achieved by calling `utl_recomp.recomp_parallel()` or `dbms_utility.compile_schema()`.
- Added several new PySQL commands: *commit* and *rollback*, *drop_types*, *user_exists*, *object_exists* and *env*.
- The `--commit` argument for the pysql script (with the options `record`, `once` and `never`) has been replaced with a flag option `--rollback`. Automatically committing after every record is no longer available.
- The PySQL terminator comment (`-- @@@`) can now no longer be specified via a command line option.
- The `-v`/`--verbose` option for *ll.pysql* now supports new output modes (`file` and `log`) and full mode now outputs much more information.
- The *URL* methods `owner()` and `group()` now will return the `uid` or `gid` respectively when the user or group name can't be determined instead of raising a `KeyError`.
- Fixed SQL statement for dropping *ll.orasql.Job* objects.

6.16.45 Changes in 5.43 (released 2019-05-07)

- The functions *ll.xist.css.iterrules()* and *ll.xist.css.applystylesheets()* now treat `<style>` and `<link rel="stylesheet">` elements without a `type` attribute as containing/linking to CSS.
- *ll.sisyphus.Job* now provides a repeat mode. With this the Python script can function as its own minimal cron daemon.

6.16.46 Changes in 5.42.1 (released 2019-04-29)

- Fixed a bug in `ll.orasql.OracleURLConnection._walk()` to support orasql objects that don't have an owner (i.e. *User* and *JobClass*).
- Simplified clean up logic for sisyphus jobs (which makes the new "delete log files for uneventful runs" logic work on Windows).

6.16.47 Changes in 5.42 (released 2019-04-26)

- By returning `None` from the method *ll.sisyphus.Job.execute()* a sisyphus job can now report that the job run was "uneventful" (i.e. the job had nothing to do) and that the log file can be deleted immediately.

6.16.48 Changes in 5.41 (released 2019-03-29)

- Added a script **oracycles** that can find cyclic foreign key references in an Oracle database schema.

6.16.49 Changes in 5.40.2 (released 2019-03-26)

- The jobname for *ll.orasql.Job* objects now no longer includes the owner name.

6.16.50 Changes in 5.40.1 (released 2019-03-25)

- Fixed a bug in *ll.orasql* for `Comment` objects.

6.16.51 Changes in 5.40 (released 2019-03-25)

- *ll.orasql* now supports jobs and job classes via the new classes `Job` and `JobClass`.
- The UL4 functions `isfirst`, `islast` and `isfirstlast` are now available as standalone Python functions in the *ll.misc* module.

6.16.52 Changes in 5.39 (released 2019-01-30)

- *ll.misc.SysInfo* now uses `platform.uname()` instead of `os.uname()` for its host information, so this will work on Windows too. This also means *ll.misc.SysInfo* gained a new attribute `host_processor` which is provided by `platform.uname()`. Furthermore the user information on Windows now supports `user_name` and `user_dir` (all other user attribute are `None`).
- *ll.orasql* objects that have source code (like `Type`, `Procedure` etc.) should now be able to better handle any duplicate spaces in its source code header introduced by Oracle 18.

6.16.53 Changes in 5.38 (released 2018-11-15)

- Added the following attributes to the UL4 class `AST`: `line` (the line number in the template source code), `col` (the column number in the template source code), `sourceprefix` (a part of the templates source code before the source code of the `AST` node) and `sourcesuffix` (a part of the templates source code after the `AST`s source code).

  These attributes are also accessible to UL4 templates.

6.16.54 Changes in 5.37.1 (released 2018-11-13)

- Each UL4 AST node now has an attribute `fullsource` which is the full source of the outermost template.
- Fixed `source` attribute of empty UL4 templates.

6.16.55 Changes in 5.37 (released 2018-11-08)

- The chaining of UL4 exceptions has been inverted. This means that the exception that will get raised from the UL4 template is the original innermost exception. `LocationError` instances will be added to the `__cause__` attribute to specify the exact location in the UL4 source. Note that this means that those `LocationError` instances won't have a traceback added, as they will never be raised.
- The structure of compiled UL4 templates has been simplified internally: Each `AST` instance has attributes `template` and `pos` that directly reference the template and the source code location of the `AST` node. The `Tag` objects are gone (they will only be used internally during compilation). Also `AST` nodes have gained a `source` property which returns the source code of the node itself.

6.16.56 Changes in 5.36 (released 2018-10-31)

- As `cx_Oracle` provides its own class `Object` *`ll.orasql`*s class `Object` has been renamed to `SchemaObject`.
- `sisyphus` jobs can now run even if `os.fork()` and `fcntl` are not available or `signal.signal()` doesn't support `SIGALRM` (i.e. on Windows). In this case various features will be missing.
- It is now possible to pass instances of `cx_Oracle.Object` as arguments when calling function or procedures in `orasql`.

6.16.57 Changes in 5.35 (released 2018-09-14)

- UL4 now support both `datetime` and `date` objects. A function `today` has been added that returns the current day as a `date` object.
- The UL4 method `week` for date objects now by default returns the ISO calendar week number. The day that should be considered the first day of the week and how many days the first week of the year has to have can be passed as parameters.
- A new UL4 method `calendar` has been added that returns the ISO calendar year, the ISO calendar week and the weekday. Similar to `week` the day that should be considered the first day of the week and how many days the first week of the year has to have can be passed as parameters.
- Speed up deserializing strings from UL4ON dumps.
- *`ll.sisyphus`* now uses `psutil` to terminate all child processes when the maximum runtime is exceeded. If `psutil` is not available only the forked child process itself will be terminated as before.
- Fixed a bug in *`ll.orasql.ForeignKey.refconstraint()`* for foreign keys that reference a table in another schema.

6.16.58 Changes in 5.34 (released 2018-06-03)

- Renamed the class `ll.xist.ns.html.script.Attrs.async` because `async` is a keyword in Python 3.7.
- XIST is Python 3.7 compatible now.

6.16.59 Changes in 5.33 (released 2018-05-15)

- *`ll.orasql.PrimaryKey.columns()`* and *`ll.orasql.ForeignKey.columns()`* now yield *`ll.orasql.Column`* objects that link back to the db schema from which they originated (so it is possible to call methods on them without passing the database connection).
- *`ll.orasql.Column`* has two new attributes: `tablename` is the name of the table that the column belongs to and `columnname` is the name of the column (without the table name).
- Exceptions from `ssh` URLs are no longer checked whether they are from the correct module. Instead they're always sent across the `execnet` channel, so that the receiving side has to deal with them.
- Since UL4 relies on ordered dictionaries (i.e. ordinary dictionaries in Python 3.6) and Javascript doesn't guarantee iteration order of objects and Spidermonkey doesn't support sets and maps, testing UL4 with Spidermonkey has been dropped.

6.16.60 Changes in 5.32 (released 2018-02-20)

- *ll.orasql.Connection.objects()* now makes sure that no objects from other schemas are returned when a specific schema owner is requested.
- The default value for the `owner` parameter in various *ll.orasql* methods has changed from `ALL` to `None` (i.e. it now returns the objects from the current schema instead of all schemas).

6.16.61 Changes in 5.31 (released 2018-01-29)

- The UL4ON decoder now has a new method `loadcontent()` that can be used to iteratively load the content of an object. This makes it possible to handle object dumps where the writing side dumped a different number of items than the reading side expects.

6.16.62 Changes in 5.30 (released 2018-01-17)

- The new UL4 tag `<?renderx?>` works like `<?render?>`, but the output from the template will be XML escaped.
- The new tag `<?renderblocks?>` is syntactic sugar for rendering a template and passing various other templates as keyword arguments, i.e. if we have a template:

```
<?def page(head, body)?>
   <html>
      <head>
         <?render head()?>
      </head>
      <body>
         <?render body()?>
      </body>
   </html>
<?end def?>
```

then:

```
<?renderblocks page()?>
   <?def head?>
      <title>Foo</title>
   <?end def?>
   <?def body?>
      <h1>Bar!</h1>
   <?end def?>
<?end renderblocks?>
```

is syntactic sugar for:

```
<?def head?>
   <title>Foo</title>
<?end def?>
<?def body?>
   <h1>Bar!</h1>
<?end def?>

<?render page(head=head, body=body)?>
```

except that with `<?renderblocks?>` the templates `head` and `body` will not leak into the surrounding namespace.

- The new tag `<?renderblock?>` is similar to `<?renderblocks?>`. However the complete content of the `<?renderblock?>` call will be passed as the `content` keyword argument to the render call. I.e.:

```
<?renderblock foo()?>
   bar
<?end renderblock?>
```

is syntactic sugar for:

```
<?def content?>
   bar
<?end def?>
<?render foo(content=content)?>
```

6.16.63 Changes in 5.29 (released 2017-11-29)

- When an exception happens during decoding of an UL4ON stream the stack of types that is currently being decoded is included in the exception message now.
- The Javascript implementations of UL4 and UL4ON are now tested against Node.js (in addition to V8 and Spidermonkey).
- The UL4 string methods `startswith` and `endswith` now support lists of strings as an argument.

6.16.64 Changes in 5.28.2 (released 2017-08-03)

- The character < is now escaped as `\x3c` in UL4ON dumps to help XSS prevention.

6.16.65 Changes in 5.28.1 (released 2017-08-02)

- Fixed a bug in `ll.sisyphus.Task.__str__()`.
- The UL4 function `asjson` now escapes < as `\u003c` to help XSS prevention.

6.16.66 Changes in 5.28 (released 2017-08-01)

- XIST requires Python 3.6 now.
- As dicts are ordered in Python 3.6 the `Attrs` attribute `xmlorder` is gone. Attributes will always be serialized in the same order they have been parsed/created.
- UL4 no longer tries to disguise objects as dictionaries. I.e. for objects with an `ul4attrs` class attribute the methods `items`, `keys`, `values` and `get` are no longer synthesized. This also means that `len`, `list`, item access and containment tests no longer work on objects.
- New UL4 functions `getattr`, `setattr`, `hasattr` and `dir` have been added to work with attributes of objects.
- Fixed an UL4ON bug: Strings containing line feeds can now be deserialized properly.

6.16.67 Changes in 5.27 (released 2017-03-21)

- When deserializing UL4ON dumps it is now possible to pass in a custom type registry dictionary. This can be used to customize which objects get created for which type.

6.16.68 Changes in 5.26.1 (released 2017-03-03)

- The fields of a unique constraint are now output in the correct order by `ll.orasql.UniqueConstraint.createsql()`.

6.16.69 Changes in 5.26 (released 2017-02-28)

- UL4 templates now support a new tag: `<?doc?>` may contain the documentation for the template and will by accessible in UL4 templates via the attribute `doc`.
- The signature of UL4 templates is now available to UL4 templates as the `signature` attribute:

```
<?def f(x=17, y=23)?>
   <?print x+y?>
<?end def?>
<?print f.signature?>
```

will output

```
(x=17, y=23)
```

6.16.70 Changes in 5.25.1 (released 2017-02-15)

- Fixed a problem with the `renders` method of local UL4 templates. The local template didn't see the variables from the surrounding scope.

6.16.71 Changes in 5.25 (released 2017-02-13)

- UL4ON dumps can now contain UL4 templates in "source" format, i.e. the template will be compiled by the UL4ON decoder. This makes it possible to dump UL4 templates via PL/SQL code (see the LivingLogic.Oracle.ul4 project on GitHub for more info).

6.16.72 Changes in 5.24 (released 2017-02-12)

- Dictionary literals or dictionary comprehensions in UL4 templates now always create ordered dictionaries (i.e. `collections.OrderedDict` objects on Python 3.5 and normal `dict` objects on Python 3.6).

6.16.73 Changes in 5.23 (released 2016-12-16)

- UL4ON now supports ordered maps.

6.16.74 Changes in 5.22.1 (released 2016-11-02)

- Fixed the default value for the `pysql` option `--commit`.

6.16.75 Changes in 5.22 (released 2016-10-18)

- PySQL now supports connections to multiple databases via the new `pushconnection` and `popconnection` commands (and the `connectname` key for the PySQL commands `procedure`, `sql`, `checkerrors`, `compileall` and `resetsequence`).
- All PySQL commands now support comments via the `"comment"` key.
- The values for the *`ll.pysql`* option `-v/--verbose` have changed: `-v1` now is `-vdot`, `-v2` is `-vtype` and `-v3` is `-vfull`.

6.16.76 Changes in 5.21 (released 2016-09-19)

- Added a function `md5` to UL4.
- If constant folding doesn't work for unary or binary operators in UL4, compiling the template no longer fails. Instead the original AST will be used unchanged (and executing the template will then fail).
- The method `ll.color.Color.__add__()` has been removed, i.e. color can no longer be added.
- The method `ll.orasql.ForeignKey.pk()` has been renamed to `refconstraint()` and supports foreign keys that reference a unique constraint now.

6.16.77 Changes in 5.20.1 (released 2016-08-04)

- Fixed a bug in **`rul4`** when database connections are specified on the command line.

6.16.78 Changes in 5.20 (released 2016-07-29)

- Dictionaries and sets in UL4 now support the `clear()` method.
- **`rul4`** now supports saving files to disk, making log calls that print messages to `stderr` and accessing environment variables. All variables passed to the templates have been moved into an object named `globals`.

6.16.79 Changes in 5.19.4 (released 2016-06-30)

- *`ll.orasql`* now honors the logging mode of a table or index when creating SQL for it. A new method *`ll.orasql.Table.logging()`* has been added to table objects. It returns whether loggging is enabled for this table.
- *`ll.orasql.Connection`* and *`ll.orasql.connect()`* now support a new argument `decimal`. If this argument is true, `NUMBER`s will be returned as `decimal.Decimal` objects (otherwise as `float`s).

6.16.80 Changes in 5.19.3 (released 2016-06-29)

- Added a new method *ll.orasql.Record.replace()*.

6.16.81 Changes in 5.19.2 (released 2016-06-21)

- Fixed a bug in `ll.orasql.Constraint.names()`.

6.16.82 Changes in 5.19.1 (released 2016-06-20)

- The field `USER_TAB_COLUMNS.DATA_DEFAULT` is different in Oracle 11 and Oracle 12 installations. *ll.orasql* has been updated to handle the difference.

6.16.83 Changes in 5.19 (released 2016-06-14)

- The documentation has been ported to Sphinx.
- *ll.pysql* now supports *var* objects with `None` as the key. Those variables will be used as OUT parameter, but will be thrown away after the procedure call.

6.16.84 Changes in 5.18 (released 2016-05-17)

- Added a function `isexception()` to UL4 that returns `True` if the argument is an exception object.
- The UL4 exception `ll.ul4c.Error` has been renamed to `LocationError`.
- The `__cause__` or `__context__` attribute of exception objects now gets exposed to UL4 templates as the `cause` attribute. In addition to that for the UL4 exception *ll.ul4c.LocationError* the following attributes get exposed: `location` (which is the AST node or tag where the error occured), `template` (the innermost template where the exception occurred), `outerpos` (the position of the tag where the error occurred) and `innerpos` (the position of the AST node where the error occurred).
- The UL4 function `type` now returns the Python class name for date, color, template exception objects.

6.16.85 Changes in 5.17.1 (released 2016-05-10)

- Fixed a bug in the query done by `orasql.Connection.getobject()`.

6.16.86 Changes in 5.17 (released 2016-05-04)

- The internal structure of UL4 templates has changed to simplify exception handling (Text nodes and tags reference the template now instead of only the template source).
- The **rul4** function `import` has been dropped. Instead two functions `load` (for reading the content of a file) and `compile` (for compiling a string into an UL4 template) have been added.
- A function `error` for outputting an error message and aborting template rendering have been added to **rul4**.

6.16.87 Changes in 5.16 (released 2016-04-13)

- `orasql` now supports check constraints.
- `orasql` now handles inline primary key constraints properly.
- The scripts **`oracreate`**, **`oradelete`**, **`oradrop`**, **`oragrant`**, **`orareindex`** and **`oradiff`** have a new option `--format`. The option value `pysql` switches the output format to PySQL.
- Procedure and function source code created by *`ll.orasql`* will now no longer have a linefeed introduced before the parameter list.
- `orasql.Comment` objects now work even if the comment contains linefeeds.
- `orasql.Comment` objects now have a method `table()` that returns the table to which the comment belongs.
- Some methods in `orasql` have been renamed: Iterating methods no longer have `iter` in their name (e.g. `itertables()` is now simply called `tables()`). The `ddl` part of some method names has been changed to `sql` (e.g. `createddl()` is now called `createsql()`).
- Importing `pysql` now doesn't fail if the module `pwd` or `grp` doesn't exist (e.g. on Windows). (However the PySQL `file` command will still fail if a user/group name is given.)

6.16.88 Changes in 5.15.1 (released 2016-03-21)

- Fixed some Python 3 compatibility problems in the module *`ll.daemon`* and updated the command line argument parsing to make it extensible.

6.16.89 Changes in 5.15 (released 2016-03-18)

- Calls to UL4 functions and templates now support specifying a * or ** argument multiple times (similar to Python's **PEP 448**).
- Also * and ** expressions are now allowed in list, set and dict "literals".
- The UL4 function `sorted` now supports a `key` and `reverse` argument.
- Strings in UL4 now support the `splitlines` method.
- An UL4 function `ascii` has been added.
- PySQL no longer supports the `-- !!!` command terminator. Use the `raiseexceptions` command instead to specify error handling.
- The PySQL command `setvar` now uses the `name` key as the variable name instead of the `var` key.
- A new PySQL command `unsetvar` has been added for deleting an existing variable.
- PySQL variables can now be used in expressions, e.g.:

```
var('foo_10', 'str').upper()
```

- The PySQL function `load` has been replaced by two functions `loadstr` for loading strings and `loadbytes` for loading bytes.
- `orasql.Index` now has a method `itercolumns()` for iterating through the columns of the index.

6.16.90 Changes in 5.14.2 (released 2016-03-02)

- Fixed color blending in `color.Color` to use "premultiplied alpha" values. With this change blending colors gives the same result as CSS.

6.16.91 Changes in 5.14.1 (released 2015-12-04)

- Fixed a bug in *`ll.make`* so that `Project` objects can now be used as arguments in `CallAction` objects.

6.16.92 Changes in 5.14 (released 2015-12-02)

- Whitespace handling for UL4 templates has been extended. There are three possible whitespace handling modes now (specified via the new `whitespace` parameter): `"keep"` (the old `keepws=True`) `"strip"` (the old `keepws=False`) and the new `"smart"`.

  In smart mode if a line contains only indentation and one tag that doesn't produce output, the indentation and the linefeed after the tag will be stripped from the text. Furthermore the additional indentation that might be introduced by a `for`, `if`, `elif`, `else` or `def` block will be ignored.

  Rendering a template from within another template will reindent the output of the inner template to the indentation of the outer template.

- Rendering an UL4 template from inside a UL4 template is now again done via the `<?render?>` tag. To update your code replace `<?code r.render()?>` with `<?render r()?>`.
- Whitespace handling mode for UL4 templates can now be specified in the template source itself via the `<?whitepace?>` tag:

```
<?whitespace smart?>
```

- The name and signature of an UL4 template can now be specified in the template source too like this:

```
<?ul4 name(x, y, *args, **kwargs)?>
```

- Closures in UL4 templates no longer see the state of the variables at the time when the local template was defined, but at the time when it is called. This is similar to most other languages that support closures.
- In UL4 tags whitespace is allowed now before the tag name, i.e.:

```
<? for i in range(10) ?>
   <? print i ?>
<? end for ?>
```

- Exposing attributes of objects to UL4 templates can now be customized via the methods `ul4getattr()` and `ul4setattr()`. Support for making attributes writable or exposing them under a different name via `ul4attrs` has been removed.
- An object can now be made renderable by UL4 templates by implementing the method `ul4render()`.
- An object can now be made callable by UL4 templates by implementing the method `ul4call()` (`__call__()` is still supported).
- Stacktraces produced by UL4 templates now include less chained exceptions and are much more informative.
- The **`rul4`** option `--keepws` has been renamed to `--whitespace` and defaults to `smart` now.
- **`rul4`** got a new option `--stacktrace`: `full` displays the full Python stack trace, `short` (the new default) only displays the exception chain without displaying any Python source.
- Templates used in **`rul4`** have access to a new function: `import`, which can be used to load templates from any file.
- UL4 got two new comparison operators: `is` and `is not` for checking for object identity.

- `oradd` has been renamed to `pysql`. The commands are now no longer limited to being on one line. Normal SQL commands are now also supported. Normal SQL commands must be terminated with a comment line starting with `-- @@@` and PySQL commands must be either on one line, or start with a line containing only `{` and end with a line containing only `}`.

  Three new commands have been added: `include` includes another `pysql` file. `compileall` recompiles all objects in the schema and `checkerrors` raises an exception if there are objects with compilation errors in the schema.

  Also `str`/`bytes` values can be loaded from external files via the `load` class.
- If an identifier is given when invoking a `sisyphus` job it will be included in the log file name now by default.
- Three new helper functions were added to *ll.misc*: `format_class()` formats the name of a class object (e.g. `ValueError` or `http.client.HTTPException`). `format_exception()` formats an exception:

```
>>> misc.format_exception(ValueError("bad value"))
'ValueError: bad value'
```

  `exception_chain()` traverses the chain of exceptions (via the `__cause__` and `__context__` attributes).
- + in the path part of URLs are now considered safe characters. Spaces will be escaped as `%20` and no longer as +.
- `ll.orasql.Comment` has a new method `comment()` that returns the text of the column comment itself.
- The database objects output by `ll.orasql.Object.iterreferences()` and `ll.orasql.Oracle.iterreferencedby()` are now sorted by name to get a stable order of dependencies.
- *ll.misc* has two new functions: `notifystart()` and `notifyfinish()`. The can be used for issuing Mac OS X notifications.

6.16.93 Changes in 5.13.1 (released 2015-06-12)

- *ll.url.URL.relative* can now produce "scheme relative" URLs if requested via the parameter `allowschemerel`, i.e.:

```
>>> u1 = url.URL("http://www.example.org/about/index.html")
>>> u2 = url.URL("http://www.example.com/images/logo.png")
>>> u2.relative(u1, allowschemerel=True)
URL('//www.example.com/images/logo.png')
```

- The XIST publishing methods now have an additional parameter `allowschemerelurls`. When `allowschemerelurls` is true, scheme relative urls are allowed in the output:

```
>>> node = html.a(href="http://www.example.org/index.html")
>>> node.bytes(base="http://www.example.com", allowschemerelurls=True)
b'<a href="//www.example.org/index.html"></a>'
```

6.16.94 Changes in 5.13 (released 2014-12-18)

- UL4 templates now support signatures. Signatures can be used for top level templates and for subtemplates. This makes it possible to define default values for template variables and to call templates with positional arguments.
- The option `setproctitle` for sisyphus jobs has been renamed to `proctitle`. The new method `setproctitle()` sets the process title and can be overwritten to customize setting the process title.
- Locally defined UL4 templates no longer see themselves among the variables of the parent template. This avoids cycles in the object graph.

- The default for the name parameter in `tasks()` for sisyphus jobs has changed from `str` to `None`, i.e. it defaults to unnamed tasks now.
- `misc.Const` now allows to specify a module name.

6.16.95 Changes in 5.12.1 (released 2014-12-09)

- Fixed a bug in `ll.oradd`: Printing the final report failed when no commands where executed.

6.16.96 Changes in 5.12 (released 2014-11-07)

- Fixed bugs in Oracle URLs: the types `comment` and `column` are now skipped when iterating a user "directory". Content in the `user` directory now works correctly.
- UL4ON has been reimplemented to be human-readable and more robust against modification of the dumped data. For generating UL4ON dumps with Oracle a PL/SQL package is available at https://github.com/LivingLogic/LivingLogic.Oracle.ul4

6.16.97 Changes in 5.11 (released 2014-10-29)

- UL4 now supports sets, set literals and set comprehensions.
- `misc.javaexpr()` now supports sets.
- Sisyphus jobs have a new method `tasks()` that loops over an iterable and calls `task()` for each item:

```
items = sys.modules.items()
for (name, module) in self.tasks(items, "module", lambda kv: kv[0]):
   self.log(f"module is {module}")
```

- An option `--maxemailerrors` has been added to sisyphus jobs: This options limits the number of exceptions and errors messages that will get attached to the failure email.
- An option `--setproctitle` has been added to sisyphus jobs: When this options is specified, the process title will be modified during execution of the job, so that the `ps` command shows what the processes are doing. (This requires that the module `setproctitle` is installed.)

6.16.98 Changes in 5.10 (released 2014-10-09)

- Old sisyphus logfiles can now be compressed automatically via `gzip`, `bzip2` or `lzma`.
- The functions `misc.gzip()` and `misc.gunzip()` have been removed as Python 3.2 has the functions: `gzip.compress()` and `gzip.decompress()` which work the same.

6.16.99 Changes in 5.9.1 (released 2014-09-29)

- Fixed the precedence of the boolean `not` operator in UL4: Now it has a lower precedence than the comparison operators. i.e. `not x in y` is parsed as `not (x in y)`.

6.16.100 Changes in 5.9 (released 2014-09-22)

- A script **udiff** has been added for doing line by line comparisons of two files or directories. **udiff** supports all URLs that *ll.url* supports (e.g. `ssh` and `oracle` URLs).
- The script **db2ul4** has been renamed to **rul4**. The following new features have been added: Additional variables can be passed to the UL4 template via the `-D/--define` option. Access to Oracle, SQLite and MySQL databases can be disallowed with the options `--oracle`, `--sqlite` and `--mysql`. Executing system commands can be disallowed with the option `--system`. SQL code that doesn't return results can be executed with the new `Connection` method `execute()`. "out" parameters can now be used via variable objects that can be created with the `int()`, `number()`, `str()`, `clob()` and `date()` methods.
- A new script **orareindex** has been added that can be used to rebuild/recreate all indexes and unique constraints in an Oracle database.
- All objects in *ll.orasql* that represent objects in the database now have a method `exists()` that returns whether the object exists in the target database.
- *ll.orasql.Index* has a new method `rebuildddl()` that returns SQL for rebuilding the index.
- URLs have a new `walk()` method that works similar to the `walk()` method for XIST trees: `walk()` is a generator that returns a `Cursor` object that contains information about the state of the directory traversal and can be used to influence which parts of the directory hierarchy are traversed.
- The URL methods `listdir()`, `files()`, `dirs()` are generators now.
- The old URL method `walk()` has been renamed to `walkall()` and `listdir()`, `files()`, `dirs()`, `walkall()`, `walkfiles()` and `walkdirs()` have been enhanced:

  `listdir()`, `files()` and `dirs()` now have arguments `include` and `exclude` instead of `pattern` (which worked like `include` does now). Also patterns can now be lists of strings.

  `walkall()`, `walkfiles()` and `walkdirs()` gained the same arguments. Additionally the arguments `enterdir` and `skipdir` can be used to skip directories during traversal.
- Oracle URLs now support the methods `walk()`, `walkall()`, `walkfiles()` and `walkdirs()` (with the new arguments `include`, `exclude`, `enterdir` and `skipdir`). The methods `listdir()`, `files()` and `dirs()` support the arguments `include` and `exclude`.
- The various directory traversal methods in *ll.url.URL* will now output URLs in sorted order.
- `URL.open()` for Oracle URLs now supports the `encoding` and `errors` parameter.
- URLs no longer forward attribute access to unknown attributes to the connection to avoid problems with code that uses `hasattr()` to check for the presence of an attribute.
- Fixed handling of the current directory in `url.Dir()`: `url.Dir("")` now returns `URL('file:./')`.
- `misc.SysInfo` has a new attribute: `script_url` returns the name of the running script as an `ssh` URL (e.g. `ssh://user@www.example.org/~/project/script.py`)
- The evaluation order of keyword arguments in calls to UL4 functions/templates has been fixed.
- The test suite for UL4 now runs the Javascript versions of the templates not only on V8 but on Spidermonkey too.

6.16.101 Changes in 5.8.1 (released 2014-06-18)

- The UL4 function `repr` now handles recursive lists/dicts similar to Python `repr` (i.e. it doesn't raise an exception for infinite recursion).
- `url.URL` now handles filenames containing spaces correctly when converting between URLs and filenames.

6.16.102 Changes in 5.8 (released 2014-05-05)

- UL4 supports `while` loops now.
- `misc.item()` now supports index sequences, which will be applied recursively, so `item(["foo", "bar"], (1, -1))` returns `'r'`.
- A new context manager `misc.timeout()` has been added, that uses `signal.alarm()` to limit the runtime of the body of the `with` block.
- Updated the required version of `cssutils` to 1.0.
- Fixed the `oraddresetsequence` command to really reset the sequence. The parameters `minvalue` and `increment` are now optional. If missing, they will be taken from the existing sequence.
- Passing the `clientinfo` parameter to `cx_Oracle.connect()` doesn't work with Oracle 11.2.0.4.0 (leading to an `ORA-03113: end-of-file on communication channel` error). The method `orasql.connect()` has been changed to set the `clientinfo` parameter after the connection has been established.
- Fixed cloning of plain XML attributes.
- Fixed a bug in the C source code that broke compiling with Visual C. From now on we will have Windows installation packages again.

6.16.103 Changes in 5.7.1 (released 2014-02-13)

- Fixed a bug in the script **oradiff** that resulting in the wrong order of the output.
- The **oradd** command `file` will now create directories if they don't exist.

6.16.104 Changes in 5.7 (released 2014-01-30)

- The `ll.oradd` command `file` has been renamed to `scp`.
- The new `ll.oradd` command `file` will now save the file directly from Python. A file mode, owner and group can be set.
- The JSON payload of the *`ll.sisyphus`* failure email will now be encoded in base64 format to work around a bug in the quoted-printable encoder.
- To conform to the Python 3 dictionary interface `ll.orasql.Record.iterkeys()` has been renamed to *`ll.orasql.Record.keys()`* and `ll.orasql.Record.itervalues()` has been renamed to `ll.orasql.Record.values()`. The original methods *`ll.orasql.Record.keys()`* and `ll.orasql.Record.values()` have been dropped.

6.16.105 Changes in 5.6 (released 2014-01-28)

- `ll.oradd` has been updated to support variables and literal SQL in a more direct way. However the old method (via `"keys"` and `"sql"`) is still supported, but will be removed in one of the next versions.
- The key `"args"` is now optional for the **oradd** commands `procedure` and `sql`.
- Support for oradd dumps in UL4ON format has been removed from `ll.oradd`.
- Lines in an **oradd** dump starting with # will now be ignored.

6.16.106 Changes in 5.5.1 (released 2014-01-27)

- *`ll.orasql`* now unterstands type bodies (so the script **oracreate** will output them).

6.16.107 Changes in 5.5 (released 2014-01-23)

- If expressions (i.e. `code if cond else code`) have been added to UL4.
- The bitwise operators &, |, ^, ~, << and >> (and their augmented assigment counterparts &=, |=, ^=, <<= and >>=) have been added to UL4.
- UL4ON now supports `slice` objects.
- The **oradd** script has a new option `-d/--directory` that is the base directory for file copy actions.
- **oradd** now supports executing SQL directly.
- The project repository is hosted on GitHub now.

6.16.108 Changes in 5.4.1 (released 2013-12-18)

- Use quoted printable encoding for the JSON attachment in the sisyphus failure email.

6.16.109 Changes in 5.4 (released 2013-11-29)

- `ssh` URLs now can handle any version of Python on the remote end. The `remotepython` parameter has been renamed to `python`.
- The default Python version for `ssh` URLs can now be specified with the environment variable `LL_URL_SSH_PYTHON`.

6.16.110 Changes in 5.3 (released 2013-10-28)

- `xist.parse.Tidy` can now pass the XML declaration and the doctype to the application (however internal DTD subsets will be ignored).

6.16.111 Changes in 5.2.7 (released 2013-10-15)

- `orasql.Record` objects are now instances of `collections.Mapping` and are handled correctly by UL4 now.

6.16.112 Changes in 5.2.6 (released 2013-10-15)

- Attribute access has been fixed in UL4: For objects that supported the dictionary interface without being a dict, a `KeyError` was raised before instead of returning an "undefined" object.

6.16.113 Changes in 5.2.5 (released 2013-10-09)

- `starttime` and `endtime` are now included in the JSON data sent with the `sisyphus` failure report email too.

6.16.114 Changes in 5.2.4 (released 2013-10-09)

- `python_executable` and `python_version` are now included in the JSON data sent with the `sisyphus` failure report email.

6.16.115 Changes in 5.2.3 (released 2013-10-09)

- The `task()` context manager function in `sisyphus` now allows any UL4 compatible object for the task type and name not just strings or `None`.

6.16.116 Changes in 5.2.2 (released 2013-10-07)

- `sisyphus` now doesn't reraise the exception if it was handled via email. This means that you will only get one email: either from `sisyphus` or from your cron daemon. Exceptions that are not instances of `Exception` will not be handled by `sisyphus` (i.e. you won't get an email when you press CTRL-C, but a normal stack trace).
- In case of a parse error in UL4 templates an exception will now be raised.

6.16.117 Changes in 5.2.1 (released 2013-10-02)

- Fixed a bug in one of the UL4 templates for `sisyphus`.

6.16.118 Changes in 5.2 (released 2013-10-01)

- Added support for bound methods to UL4 templates. This means that methods that should be callable must be included in `ul4attrs`.
- UL4 templates now support attribute, item and slice assignment, i.e. the following code will work:

```
<?code d = {}?><?code d.foo = 'bar'?>
<?code d = {}?><?code d['foo'] = 'bar'?>
<?code d = [17]?><?code d[0] = 23?>
<?code d = [1, 7, 4]?><?code d[1:2] = [2, 3]?>
```

- For objects with attributes exposed to UL4, attributes can be specified as being writable by prepending the name with a + in `ul4attrs`.
- Added UL4 functions `first` and `last` that return the first or last item produced by an iterable.
- The default argument for the functions `misc.first()` and `misc.last()` now defaults to `None`. I.e. for empty iterators the default value will always be returned instead of generating an exception.
- `ll.sispyhus` can now send an email itself in case of a failure. This email includes information about the failure in plain text, HTML and JSON format.

- `ll.sispyhus` now supports subtasks via the method `task()`. This replaces the `prefix()` method.
- `ll.sispyhus` now creates a relative symbolic link for the current logfile instead of an absolute one.
- **oradd** now outputs the keys in its logging output.
- **oradd** can now be used to reset sequences.
- Committing the transactions in **oradd** can now be done after each record with the new option `--commit`. `--rollback` has been removed.
- Renamed the attributes `scriptname` and `shortscriptname` of the `misc.sysinfo` object to `script_name` and `short_script_name`.
- Fixed the user related attributes of `misc.sysinfo`.

6.16.119 Changes in 5.1 (released 2013-08-02)

- The HTML namespace (*`ll.xist.ns.html`*) now supports microdata attributes.
- Added support for triple quoted strings to UL4 templates.
- Added an UL4 function `sum` that works like the Python function `sum`.
- Variables assigned in the body of a <?for?> loop in UL4 now survive the end of the loop. As a consequence of this, loop variables now leak into the surrounding scope (but not the loop variables for list/dictionary comprehensions or generator expressions).
- Made checking for recoverable Oracle exceptions in *`ll.nightshade`* more robust.
- Added missing processing instruction class `ll.xist.ns.ul4.note`.
- `ll.oradd` now prints the data object before trying to call the procedure and can handle foreign keys that are `NULL`.
- The methods `abslum()` and `rellum()` of `Color` objects are now exposed to UL4 templates.
- The **oradd** script has a new option `--dry-run` to roll back all database changes instead of committing them. This can be used to test whether an **oradd** dump will work.
- **oradd** can now copy files via `scp`. Parts of the file names used may depend on key values.
- **oradd** now supports other out types than integers.
- The `query` method for database connections in **db2ul4** scripts has changed: Instead of a query and a parameter dictionary, you have to pass in positional arguments that alternate between fragments of the SQL query and parameters. I.e.:

  ```
  db.query("select * from table where x=:x and y=:y", x=23, y=42)
  ```

  becomes:

  ```
  db.query("select * from table where x=", 23, " and y=", 42)
  ```

  This makes **db2ul4** independent from the parameter format of the database driver.

6.16.120 Changes in 5.0 (released 2013-06-04)

- The HTML namespace (*ll.xist.ns.html*) has been updated to support the current HTML5 definition.

  However old elements/attributes from the previous HTML namespace are still supported.

- XIST now allows arbitrary elements and attributes. *ll.xist.parse* will parse any XML file, even if the pool object doesn't contain an element for the element name, and even if an attribute name isn't declared for an element.

  Undeclared elements will be "plain" instances of *ll.xist.xsc.Element* (i.e. not instances of a subclass of *ll.xist.xsc.Element*) with the attributes `xmlns` and `xmlname` set accordingly and undeclared attributes will be "plain" instances of *ll.xist.xsc.Attr* (with proper `xmlns` and `xmlname` attributes).

  This new feature requires several API changes which will be described below.

- Validation is now off by default, to turn it on pass `validate=True` to `parse.tree()` or `parse.itertree()` for parsing, or the publisher object or the `bytes()`, `iterbytes()`, `string()` or `iterstring()` methods for publishing.

- Accessing an attribute via `__getattr__()` (i.e. `htmlelement.attrs.class_`) only works for attributes that are declared for the class, all other attributes must be accessed via `__getitem__()` (i.e. `htmlelement.attrs["class"]`). `__getitem__()` always requires the XML name of the attribute. `__getitem__()` also allows an attribute name for a global attribute in Clark notation (i.e. `htmlelement.attrs["{http://www.w3.org/XML/1998/namespace}lang"]`). A global attribute can also be accessed via a (namespace name, attribute name) tuple (i.e. `htmlelement.attrs[("http://www.w3.org/XML/1998/namespace", "lang")]`). Using an attribute class or attribute object is also possible (i.e. `htmlelement.attrs[xml.Attr.lang]` or `htmlelement.attrs[xml.Attr.lang('de')]`).

- Using `__setattr__()` to set attributes only works for declared attributes too. Using `__setitem__()` to set attributes supports the same kind of arguments as `__getitem__()` does. For declared attributes the resulting attribute object will always be an instance of the declared attribute class. For all other attributes it will be an instance of *ll.xist.xsc.Attr* except when an attribute class or instance is used as the key. In this case the attribute will be an instance of that class.

- The methods `convert()`, `clone()`, `__copy__()`, `__deepcopy__()`, `compacted()`, `withsep()`, `reversed()`, `filtered()`, `shuffled()`, `mapped()` and `normalized()` make sure that plain nodes are copied properly, i.e. they retain their custom `xmlns` and `xmlname` attributes.

- The keys in an attribute dictionary (i.e. an *ll.xist.xsc.Attrs* object) are no longer the attribute classes, but the (namespace name, attribute name) tuples:

```
>>> node = html.div({xml.Attrs.lang: 'de'}, id='id42', class_='foo')
>>> list(node.attrs.keys())
[('http://www.w3.org/XML/1998/namespace', 'lang'),
 (None, 'class'),
 (None, 'id')]
```

- For all methods that existed in Python/XML pairs (e.g. `withnames()` and `withnames_xml()` in `xsc.Attrs` or `elementclass()` and `elementclass_xml()` in `xsc.Pool` etc.) there is only one version now: A method without the `_xml` suffix that accepts the XML version of the name.

- The method `checkvalid()` has been renamed to `validate()`. It no longer calls `warnings.warn()` itself, but is a generator that returns the warning objects. Furthermore the `model` objects now get passed the complete path instead of only the target node (this is used to implement HTML5's transparent content model).

- Validating whether an attribute is allowed is now done in `Attrs.validateattr()`. The default implementation yields warnings about undeclared local attributes. The HTML5 namespace extends this to also accept any attribute whose name starts with `data-` or `aria-`.

- Node comparison now ignores the classes for elements, entities and processsing instructions, so that plain nodes compare equal to instances of `Element`, `Entity` or `ProcInst` subclasses as long as the name and content of the node matches.

- *ll.xist.parse.Tidy* no longer has a `skipbad` argument.

- Converter contexts now support string as keys (which must be hierarchical dot-separated names similar to Java package names (e.g. `"org.example.project.handler"`) to avoid name collisions).
- The `docbook` module has been updated to support DocBook 5.0.
- `URL` objects are pickable now.
- When whitespace is removed in the literal text of UL4 templates (via the `keepws` parameter), any initial spaces (before the first line feed) are now no longer removed.
- If you have Cython installed and the environment variable `LL_USE_CYTHON` set, several modules will now be compiled into extension modules.
- It's now possible to expose attributes and methods of objects to UL4 templates. Exposing attributes can be done by setting a class or instance attribute `ul4attrs` to a sequence of attribute names. Exposing methods can be done with the decorators `ul4c.expose_method()` and `ul4c.expose_generatormethod()`.
- A new UL4 function `list` has been added. This function works like the Python function `list`, creating a copy of a sequence or materialzing an iterator.
- A new UL4 function `slice` has been added. It works like `itertools.slice()`, i.e. returning a slice from an iterator.
- The function `html.astext()` that converts an XIST tree containing HTML to plain text is now implemented in plain Python so it no longer requires a text mode browser. The function also got more configurable.
- The objects available to **db2ul4** scripts have been changed: `oracle`, `sqlite` and `mysql` are now objects with a `connect` method that returns a connection object. A connection object now has a method `query` that executes the query and returns an iterator over the results. Furthermore `query` supports keyword arguments for parameterized queries, i.e. you can now do:

```
<?code db = oracle.connect("user/pwd@db")?>
<?for row in db.query("select * from foo where bar = :bar", bar=42)?>
   <?print row?>
<?end for?>
```

  The `system` object now has an `execute` method that executes the system command.
- Fixed a bug in `orasql.OracleFileResource.close()` that surfaced when writing to an Oracle object.

6.16.121 Changes in 4.10 (released 2013-03-04)

- It's now possible to use UL4 templates as functions by using the `<?return?>` tag:

```
>>> from ll import ul4c
>>> f = ul4c.Template("<?return 2*x?>")
>>> f(x=42)
84
```

  It's also possible to call a template as a function inside another template:

```
>>> from ll import ul4c
>>> t = ul4c.Template("<?def x?><?return 42?><?end def?><?print x()?>")
>>> t.renders()
'42'
```

  Normal output of the template will be ignored if it is used as a function.

  If the template runs through to the end without encountering a `<?return?>` tag, `None` will be returned if the template is used as a function.

  If the template is used as a template and a `<?return?>` tag is encountered executing the template will be stopped (the return value will be ignored).

- The UL4 tag <?code?> may now contain not only variable assigments, but any other expression. Of course this makes only sense for expressions that have side effects (e.g. a call to the `render` method).
- The tag <?render?> has been removed. To update your code replace <?render r()?> with <?code r.render()?>.
- UL4 functions `print` and `printx` have been added. They behave like the respective tags <?print?> and <?printx?>, but can output an arbitrary number of arguments.
- The builtin UL4 functions are now real objects that can be passed to templates as arguments.
- The UL4 methods `days`, `seconds`, `microseconds` and `months` have been added for `timedelta`/`monthdelta` objects.
- Lists in UL4 now support the methods `append`, `insert` and `pop`.
- Dictionaries in UL4 now support the method `update`.
- The **db2ul4** script now supports a `-w`/`--keepws` argument.
- The UL4 functions `vars` and `get` have been removed.
- The `**` syntax has been removed for UL4 dict literals.
- The automatic UL4 variable `stack` has been removed too.

6.16.122 Changes in 4.9.1 (released 2013-01-17)

- Fixed a bug the `printx` tag for UL4 templates.

6.16.123 Changes in 4.9 (released 2013-01-17)

- Fixed a bug in UL4 templates when a template called a top-level template which in turn called its own subtemplate.
- Fixed and enhanced `repr` output of UL4 templates and added support for IPythons pretty printing framework.

6.16.124 Changes in 4.8 (released 2013-01-15)

- Linefeeds and indentation in the literal text for UL4 templates can now be ignored by specifying `keepws=False` in the template constructor.

6.16.125 Changes in 4.7 (released 2013-01-11)

- A variable `stack` is now automatically defined in all UL4 templates. This list contains a stack of the currently executing UL4 templates. `stack[-1]` is the current template.
- UL4 templates now support lexical scopes. A locally defined subtemplate can access all local variables of the template in which it is defined.
- UL4 functions and methods now support keyword arguments, e.g. `format(now(), fmt="%Y-%m-%d", lang="en")`.
- UL4 templates can no longer be converted to Java `CompiledTemplate` objects. (However converting it to an `InterpretedTemplate` is of course still supported).
- If the view `ctx_preferences` doesn't exist `orasql.Preference.itername()` now will simply return an empty iterator instead of failing with an Oracle exception `ORA-00942: table or view does not exist`.
- For sisyphus jobs, the class attribute `maxtime` can now be set to a `datetime.timedelta` object.

6.16.126 Changes in 4.6 (released 2012-12-18)

- The *walk()* method in XIST has been changed: The return value is a cursor object that provides information about the path and can be used to skip subtrees in the traversal. Filters (which are called selectors now) can no longer influence which parts of the trees are traversed, only whether a node is returned by the iterator or not.
- *itertree()* now supports the same interface is the *walk()* method.
- A new function *filter()* has been added that filters the output of `walk()` or *itertree()* against a *Selector* object.
- The XIST parse events have been renamed: The `"start*"` events to `"enter*"` and the `"end*"` events to `"leave*"`.
- Slicing XIST elements now returns a sliced element, instead of a slice from the content *Frag*:

```
>>> from ll.xist.ns import html
>>> html.ul(html.li(i) for i in range(5))[1:3].string()
'<ul><li>1</li><li>2</li></ul>'
```

- Functions with keyword only arguments are now supported in *ll.xist.ns.doc.explain()*.
- *monthdelta* now supports the `abs()` function (i.e. `abs(monthdelta(-1))` returns `monthdelta(1)`.)

6.16.127 Changes in 4.5 (released 2012-11-29)

- Added UL4 functions `any` and `all`.
- To improve UL4 exception messages there are now several undefined objects, which give information about which key/name/index resulted in the undefined object being created.
- UL4ON can no longer read or write undefined values.
- The UL4 function `format` now swallows all exceptions produced by `locale`.
- Oracle URLs now support reading and writing bytes.
- Because of problems with distribute/pip and pytest `ll/__init__.py` has been reintroduced.

6.16.128 Changes in 4.4 (released 2012-11-08)

- Python 3.3 is required now (as the code uses `yield from` and `__qualname__`).
- `ll/__init__.py` has been removed, i.e. XIST is now a PEP 420 compatible namespace package.
- Fixed tab/space mix in `antlr3/debug.py`.

6.16.129 Changes in 4.3.1 (released 2012-11-06)

- Added a method `values` to UL4 for dictionaries.
- Fixed a bug in `ll.misc.SysInfo.user_shell`.
- Fixed function *ll.xist.ns.doc.explain()* for methods.

6.16.130 Changes in 4.3 (released 2012-11-02)

- UL4 now uses a parser generated by ANTLR instead of using spark. This means that the Python parser can now use the same grammar as the Java parser. (A Python 3 port of the ANTLR runtime is included).
- Accessing nonexistent variables in UL4 templates now no longer raises an exception but returns the special object `Undefined`. The same is true for accessing nonexistent dictionary keys or list/string indexes that are out of range.

  In a boolean context `Undefined` is treated as false and `str(Undefined)` returns the empty string.
- Two new UL4 functions have been added: `isundefined` returns whether the argument is the `Undefined` object or not. `isdefined` has the inverted logic, i.e. it returns `True` if the argument is *not* the `Undefined` object.
- The characters CR and LF are no longer allowed in UL4 string constants. Furthermore the escape sequence `\e` is no longer supported.
- All AST nodes for loading constants have been merged into a single class `Const`.
- *`ll.ul4on`* can now read and write `datetime.timedelta` and `misc.monthdelta` objects as well as the new `Undefined` object from UL4 (`ll.ul4c.Undefined`).

6.16.131 Changes in 4.2 (released 2012-10-22)

- UL4 templates now support list and dictionary comprehension as well as generator expressions.
- A new UL4 function `date` has been added.
- The UL4 method `join` no longer calls `str` on the items in the argument list.
- The UL4 function format now supports a third argument: the language for formatting dates. So `format(date(2012, 10, 10), '%A', 'de')` outputs `Mittwoch`.
- UL4 date objects now have a new `week` method. This method returns the week number of the year. It supports one argument: the weekday number (0 for Monday, ... 6 for Sunday) that should be considered the start day of the week. All days in a new year preceding the first week start day are considered to be in week 0. The week start day defaults to 0 (Monday).
- `datetime.timedelta` objects are now completely supported in UL4 templates: They can be created with the `timedelta` function and can be type tested for with `istimedelta`.
- Added a new class *`ll.misc.monthdelta`*. `monthdelta` objects can be used to add months/years to a `datetime.datetime` or `datetime.date` object. If the resulting day falls out of the range of valid days for the target month, the last day for the target month will be used instead.
- `monthdelta` objects are now supported in UL4 templates: They can be created with the `monthdelta` function and can be type tested for with `ismonthdelta`.

6.16.132 Changes in 4.1.1 (released 2012-10-04)

- Fixed a bug in the UL4 handling of slices. (`('0' + str(x))[-2:]` didn't work correctly.)

6.16.133 Changes in 4.1 (released 2012-10-02)

- Loop variable unpacking in UL4 now allows arbitrary nesting.
- Variable assignment in UL4 now allows variable unpacking too, i.e. `(a, b) = [17, 23]`.
- The support for Growl notifications in *`ll.make`* on the Mac has been replaced by support for Mountain Lions Notification Center via terminal-notifier.
- `sispyhus` jobs now support notifications too.
- Java conversion of `ll.ul4c.And` has been fixed to evaluate the second operand only when the result isn't clear from the first.
- *`ll.ul4on.Decoder`* now raises an `EOFError` when reading from an empty stream.
- A new script has been added: **`oradd`** can be used for importing data into an Oracle database (via procedure calls).

6.16.134 Changes in 4.0 (released 2012-08-08)

- The source has been ported to Python 3. From now on XIST is a Python 3 only project. A big thanks goes to Martin v. Löwis, who got this conversion started at PyCon DE 2011. He did the basic `2to3` conversion and updated the C source to work on Python 3. Without Martin, XIST wouldn't have made the leap to Python 3 for several years.
- As there's no Python 3 port of libxml2s Python wrapper, XIST now uses lxml for HTML parsing.

  This change shouldn't have any visible consequences.
- UL4 templates are no longer compiled to byte code, instead the AST is evaluated or converted to the target sourcecode directly.
- Generating the final Javascript source code for UL4 templates is now done in Javascript itself.
- A new module `ul4on` has been added. This module provides functions for encoding and decoding a lightweight extensible machine-readable text format for serializing the object types supported by UL4.
- The following new functions have been added to UL4: `isfirst`, `islast`, `isfirstlast`, `enumfl`. They are variants of `enumerate` that give information about whether the item is the first and/or last item.
- The following new functions have been added to UL4: `urlquote` and `urlunquote`. They encode/decode the `%`-escaped form of URL query parameters.
- The UL4 function `json` has been renamed to `asjson` and the following new UL4 functions have been added: `fromjson`, `asul4on` and `fromul4on`.
- The UL4 function `enumerate` now supports 1 or 2 arguments (the second argument being the start value).
- The UL4 functions `str`, `bool`, `int` and `float` now support being called without arguments (just like in Python).
- Date constants in UL4 have changed again. They are now written like this: `@(2012-04-12)` or `@(2012-04-12T12:34:56)`.
- The `<?render?>` tag in UL4 now looks like a method call instead of a function call. I.e. `<?render t(a=17, b=23)?>` has changed to `<?render t.render(a=17, b=23)?>`.
- UL4 stacktraces now use exception chaining to report the exception location in nested templates.
- The UL4 methods `find` and `rfind` now support lists and tuples.
- Two new UL4 functions have been added: `min` and `max`.
- The sort order for attributes when publishing XIST trees can be overwritten by setting the `xmlorder` class attribute to a string. This string will be used for sorting the attribute. Attributes that have `xmlorder` set will always be published before those that don't.

- Support for the old `ipipe` infrastructure has been removed. Support for IPythons new pretty printing infrastructure has been added. Output looks like this:

```
In [1]: from ll.xist.ns import xml, html
In [2]: html.a(
   ...:         'gurk',
   ...:         xml.Attrs(lang='de'),
   ...:         class_='link',
   ...:         href='http://www.example.org/',
   ...:         id='dings42',
   ...: )
Out[2]:
ll.xist.ns.html.a(
   'gurk',
   ll.xist.ns.xml.Attrs.lang='de',
   class_='link',
   href='http://www.example.org/',
   id='dings42')
```

- Added the attributes `allowfullscreen` and `flashvars` to *ll.xist.ns.html.embed*.
- Added the attribute `allowfullscreen` to *ll.xist.ns.html.iframe*.
- The `isdir()` method now always returns `False` for real (i.e. non-file or ssh) URLs. This allows stuff like:

```
ucp http://www.python.org/ftp/python/2.7.2/Python-2.7.2.tar.bz2 \
    ssh://user@www.example.org/~/src/
```

- *ll.orasql.Index* now uses the `*_INDEXES` views to get a list of all indexes and `LOB` indexes are filtered out, since they will be recreated with the LOB itself. The method `table()` has been fixed for indexes that belong to a different user than the index.
- *ll.orasql.LOBStream* has a new method `seek()`.
- *ll.make.FileAction* supports encoding/decoding when writing/reading the file. For this use the `encoding` and `errors` arguments.
- The XIST node method `sorted()` has been removed, as it no longer makes sense, because with Python 3 nodes might be uncomparable.
- Th support for `%u` escapes in URLs has been removed.
- The function `html.astext()` now uses the newer links 2.
- The scripts **oracreate**, **oradrop**, **oradelete**, **oradiff**, **oramerge**, **oragrant**, **orafind** and **uhpp** no longer have an `-e/--encoding` option. They always use Pythons output encoding.
- The options `-i/--inputencoding` and `-o/--outputencoding` of the script **db2ul4** have been replaced with an option `-e/--encoding` for the encoding of the template files. For printing the result Pythons output encoding is used.
- The options `--inputencoding/--inputerrors` and `--outputencoding/--outputerrors` of *ll.sisyphus.Job* have been replaced with option `--encoding/--errors` for the encoding of the log files.
- **oradiff** now iterates through the object in correct order, so if you're running **oradiff** with `-mfull` the output shouldn't produce any errors when executed.
- *ll.orasql.Index* can now handle domain indexes.
- *ll.orasql.Preference* has been added.
- *ll.orasql* now ignores indexes of type `IOT - TOP`.
- *ll.orasql* can now handle primary keys where the underlying index has a different name.
- *ll.orasql* now ignores tables with names starting with `DR$` (i.e. those created by Oracle Text Search).

- Attributes of *ll.misc.SysInfo* instances are now calculated on demand. With this change only one instance of *ll.misc.SysInfo* is required. This instance is `ll.misc.sysinfo`.
- When connecting to the database *ll.orasql* sets the `client_info` attribute to the name of the running script (unless `clientinfo=None` is passed to the `connect()` call).
- `ll.xist.ns.specials.loremipsum` now repeats the text if the specified `len` attribute is greater that the length of the lorem ipsum text.

6.16.135 Changes in 3.25 (released 2011-08-12)

- *ll.xist.parse.Tidy* will now output the attribute events in sorted order. Publishing an XIST node will output the attributes in sorted order too.
- The `compact()` method has been renamed to `compacted()` to avoid collisions with the `compact` attribute in HTML elements.
- A new script **uhpp** has been added, that can be used for pretty printing HTML. As the attributes are output in alphabetical order it can also be used as a tool for comparing HTML files.

6.16.136 Changes in 3.24.1 (released 2011-08-10)

- Fixed a bug in the new *ll.xist.xsc.AttrElement* class that surfaced in the context of boolean attributes.

6.16.137 Changes in 3.24 (released 2011-08-09)

- The `ProcInst` subclass `ll.xist.xsc.AttrProcInst` has been replaced with an `Element` subclass *ll.xist.xsc.AttrElement*. Conditional handling of the attribute will be used, if the `AttrElement` instance is the only child of the attribute. Outside of attributes the `AttrElement` instance will be published normally (via `publish()`, which must be implemented).
- `ll.xist.ns.ul4.attr_if` is an *ll.xist.xsc.AttrElement* subclass now. The condition is in the `cond` attribute and the attribute content is inside the element. Outside of an attribute `attr_if` will put a normal UL4 `if` condition around its content.
- `ll.xist.ns.ul4.attr_ifnn` has been removed.

6.16.138 Changes in 3.23.1 (released 2011-07-28)

- Fixed a bug in *ll.sisyphus*: The code wasn't updated to use *ll.ul4c.Template* instead of `ll.ul4c.compile()`.

6.16.139 Changes in 3.23 (released 2011-07-20)

- UL4 template objects now have a name. This name will be displayed in exception messages. Nested templates display their own name in the exception message.
- The module global functions `ll.ul4c.compile()`, `ll.ul4c.load()` and `ll.ul4c.loads()` have been removed. Instead of them the `Template` constructor and the class methods `load()` and `loads()` can be used.
- The script **oradelete** now supports the options `--include`, `--exclude` and `--keepjunk` too.

6.16.140 Changes in 3.22 (released 2011-07-14)

- The scripts **oracreate**, **oradrop** and **oragrant** have new options `--include` and `--exclude` that can be used to filter the objects that will be output.

6.16.141 Changes in 3.21 (released 2011-06-03)

- Oracle 10 doesn't have a `DBA_ARGUMENTS` view. Fixed *ll.orasql.Function* and *ll.orasql.Procedure* accordingly.
- The `type` attribute for the `input` element now supports the new input types from HTML5.
- The form elements `input`, `select` and `textarea` gained the additional attributes from HTML5.

6.16.142 Changes in 3.20.2 (released 2011-05-23)

- Unicode parameters in `execute()` and `executemany()` in `ll.xist.orasql` now get encoded to the Oracle client character set.

6.16.143 Changes in 3.20.1 (released 2011-05-18)

- Fixed a bug in the Java code generation for UL4 templates: When the template source code contained C-style comments (i.e. `/* foo */`) invalid Java source code was produced.

6.16.144 Changes in 3.20 (released 2011-05-05)

- It's now possible to specify the connection mode (i.e. `SYSDBA` and `SYSOPER`) in `oracle` URLs like this:

  ```
  $ uls oracle://sys:pwd:sysdba@dsn/
  ```

  Supported are the values `normal` (the default), `sysdba` and `sysoper`.
- The `schema` argument used by various methods in *ll.orasql* has been replaced by a `owner` argument that can be `None` (for the current user), the constant `ALL` for all users (which uses the `DBA_*` variant of various meta data views if possible or the `ALL_*` variants otherwise) and a specific user name.

  These views are also used if possible in all spots where the `ALL_` views where used before.
- It's now possible to list all users in the database with the class methods `User.iternames()` and `User.iterobjects()` and with `Connection.iterusers()`.
- Oracle `Column` objects have a new method `table()` that returns the table the column belongs to.
- Oracle URLs now support the directory `user/` which contains all users, i.e. `oracle://user:pwd@db/user/` lists all users and `oracle://user:pwd@db/user/foo/` lists the same stuff as `oracle://foo:pwd@db/`. This directory however will *not* be listed in the root directory `oracle://user:pwd@db/`.
- *ll.orasql* now supports tables without columns.
- *ll.orasql.Table* has a new method `pk()` that returns the primary key constraint (or `None` if the table has now primary key constraint).
- A bug in the queries for `Index` objects in *ll.orasql* has been fixed.
- `ipipe` support has been removed from *ll.orasql*.
- Fixed a bug in *ll.xist.xsc.Pool*: Registered elements/entities etc. now show up as attributes of the pool object.

6.16.145 Changes in 3.19 (released 2011-04-26)

- *ll.orasql* now requires cx_Oracle 5.1.
- If the `readlobs` option is false for *ll.orasql* cursors, the CLOBs/BLOBs returned will be wrapped into something that behaves like a Python file.

6.16.146 Changes in 3.18.1 (released 2011-04-13)

- The methods `elements()`, `procinsts()`, `entities()` and `charrefs()` of *ll.xist.xsc.Pool* now handle base pools properly.

6.16.147 Changes in 3.18 (released 2011-04-08)

- Fixed a regression in `ll.orasql.OracleConnection`.
- Fixed `ZeroDivisionError` in script **uls** for empty directories.
- Added a class method `ll.orasql.Constraint.iternames()` and a class method `ll.orasql.Index.iternames()` that skips those indexes that are generated by constraints. With this addition **uls**/**ucp** now list/copy constraints and indexes properly. All `iternames` methods now skip objects whose name starts with `BIN$`.
- The scripts **uls**, **ucp** and **ucat** have new options `--include` and `--exclude` for including/excluding URLs that match a regular expression. They also have an new option `--all` to include/exclude dot files (i.e. files/directories whose name starts with a dot).
- **ucp** now supports to new options `--padding` and `--separator` which are used for column output.
- Two unused options were removed: `--verbose` from the script **ucat** and `--defaults` from the script **tld2xsc**.
- `ucp -x` now prints exception details.
- The variables available in UL4 templates used by **db2ul4** have changed. Instead of a `connect` object, there are now three objects for each supported database (i.e. `oracle`, `sqlite` and `mysql`)
- The script **doc2txt** now reads from `stdin` and writes to `stdout` instead of requiring file names on the command line.
- If the scripts **xml2xsc** or **dtd2xsc** are called without arguments `stdin` is read.
- `ll.xist.ns.rest` now handles option lists.
- The Oracle URLs provided by *ll.orasql* now have a `.sql` extension for all schema objects. On writing a `.sql` extension will be stripped to get the name of the schema object.
- Oracle URLs now should support schema objects with fancy names (i.e. ones that contain accented characters).
- *ll.orasql.Table* has a new method `organization()` that returns `"heap"` or normal tables and `"index"` for index organized tables.
- Pretty printing of XIST trees can now be customized with the class attributes `prettyindentbefore` and `prettyindentafter`. The values will be added to the current indentation level before and after the node in question.
- All scripts that are part of XIST (**uls**, **ucp**, **ucat**, **db2ul4**, **dtd2xsc**, **tld2xsc**, **doc2txt**, **xml2xsc**, **oracreate**, **oradrop**, **oradelete**, **oradiff**, **oramerge**, **oragrant** and **orafind**) are now properly documented on the webpages.

6.16.148 Changes in 3.17.3 (released 2011-03-02)

- Enhanced support for table and column names containing non-ASCII characters in *ll.orasql*.
- Fixed a bug in the **uls** script: In long recursive mode files were printed twice.

6.16.149 Changes in 3.17.2 (released 2011-02-25)

- Fixed `setup.py` so that the spacer GIF and the UL4 Javascript support library *really* get installed.

6.16.150 Changes in 3.17.1 (released 2011-02-25)

- Due to a bug in `MANIFEST.in` the spacer GIF and the UL4 Javascript support library where not included in the distirbution package. This has been fixed.

6.16.151 Changes in 3.17 (released 2011-02-24)

- The UL4 function `repr` now handles all instances of `collections.Mapping` and `collections.Sequence` too.
- The spacer pixel `px/spc.gif` and the UL4 Javascript support library `ul4.js` will now be installed alongside the Python modules (in `ll.xist.data`).
- The Java source code produced by `ll.ul.Template.javasource()` will now contain register declarations only for the registers that are actually used.
- `misc.javastring()` has been renamed to `misc.javaexpr()` can can now produce the Java sourcecode for more types.
- The UL4 method `isoformat` now omits the time part if it is `00:00:00`.
- The UL4 function `repr` now produces a valid UL4 date literal for date objects.
- The UL4 method `format` is now a function instead.
- The tests for UL4 now test the Java implementation too.

6.16.152 Changes in 3.16 (released 2011-01-21)

- The UL4 functions `json`, `type`, `islist` and `isdict` can now handle all instances of `collections.Mapping` and `collections.Sequence` not just `tuple`, `list` and `dict`.
- *ll.sisyphus* logging of exceptions and tracebacks should be more robust against encoding problems.
- The `cssutils` version has been bumped to 0.9.7.
- **dtd2xsc** can now combine the content of more than one DTD into a namespace. Handling of duplicate elements can be specified with a new `duplicates` option.
- **xml2xsc** can now collect the XML info from multiple XML files.
- Fixed a bug in the command line argument handling of **dtd2xsc**.
- **dtd2xsc** can now handle undefined entities.
- The help message for all scripts in XIST now show the default for all options.
- Replaced the function `misc.flag()` with a class `misc.FlagAction` that can be used as the action in `argparse.ArgumentParser.add_argument()` calls.
- Command line options for all scripts have been enhanced: Flags without a `yes/no` value now toggle the default (using the new `misc.FlagAction`).

- The script **xml2xsc** has a new option `--defaultxmlns` for setting a namespace name for elements without a namespace.
- `ll.xist.xnd` and the related scripts have seen some refactoring.

6.16.153 Changes in 3.15.3 (released 2010-11-26)

- *ll.sisyphus* now supports a non-forking mode (`--fork=no`). In this mode executing the job and monitoring the maximum runtime is done by the same (single) process.

6.16.154 Changes in 3.15.2 (released 2010-11-25)

- Publishing an *ll.xist.ns.xml.XML* object will now always put the correct encoding into the XML declaration, no matter where in the XML tree the `xml.XML` object sits.

6.16.155 Changes in 3.15.1 (released 2010-11-24)

- Fixed a bug in the error handling code of the UL4 compiler when an unknown function or method was encountered.
- Fixed str/unicode problems with the search string in **orafind**.

6.16.156 Changes in 3.15 (released 2010-11-09)

- It's now possible to create Java source code from UL4 templates with the method *ll.ul4c.Template.javasource()*.
- Creating source code (in Python, Javascript and Java) from UL4 templates has been moved out of *ll.ul4c.Template* into separate classes.
- The function `ll.xist.ns.fromul4()` now uses the new method *ll.ul4c.Template.javasource()* for generating JSP.
- The binary format for UL4 templates has changed to enhance readability.
- `ll.xist.ns.jsp.javastring()` has been moved to *ll.misc*.

6.16.157 Changes in 3.14 (released 2010-11-05)

- UL4 templates now have a method `jssource()` that returns Javascript source code. This means that now UL4 templates can be converted to: Python source code, JSP source code and Javascript source code.
- Date constants in UL4 have changed. They are now written like this: `@2010-11-05T`.
- `ul4c.Template.pythonsource()` no longer accepts `None` as the function name. The output will always be a full function.

6.16.158 Changes in 3.13 (released 2010-10-22)

- sisyphus jobs now have a new method `prefix()`. This method is a context manager. For the duration of the `with` block, the passed in prefix will be prepended to all log lines.
- *ll.sisyphus* job can now log to `stdout` and `stderr` with the new options `-o/--log2stdout` and `-e/--log2stderr`.
- The tags that *ll.sisyphus* itself uses for logging have changed slightly. For more info see the module documentation.
- The option `-l` for sisyphus jobs has been renamed to `-f`.

6.16.159 Changes in 3.12.1 (released 2010-10-21)

- Fixed a bug in *ll.sisyphus* when logging exceptions.

6.16.160 Changes in 3.12 (released 2010-10-21)

- The way that *ll.sisyphus* handles running jobs has changed. Jobs no longer create a pid file. Avoiding duplicate running jobs is done with a file lock on the script file and limiting the maximum runtime is done by forking the process and monitoring the runtime in the parent process. This means that a job that is past its maximum allowed runtime will not be killed by the next job invocation. Instead the job will kill itself.
- A new class *ll.misc.SysInfo* has been added that provides host/user/python/script information. *ll.sisyphus* uses this new class.
- Changed the default output of tags in *ll.sisyphus* log files from:

  ```
  [tag1, tag2, tag3]
  ```

  to:

  ```
  [tag1][tag2][tag3]
  ```

- The default location for *ll.sisyphus* log files has changed to `~/ll.sisyphus/projectname/jobname/`.
- *ll.orasql.ForeignKey* has a new method `itercolumns()` for iterating over the columns the foreign key consists of.
- Fixed a bug in the **uls** script: For remote URLs uid and gid must be resolved on the remote host.

6.16.161 Changes in 3.11.1 (released 2010-10-18)

- Fixed two bugs in the error handling for unknown XML parsing events in *ll.xist.parse.Expat.__call__()* and *ll.xist.parse.SGMLOP.__call__()* (exceptions were yielded instead of raised).
- *ll.sisyphus* jobs now don't break if they can't find the script source.

6.16.162 Changes in 3.11 (released 2010-10-15)

- *ll.sisyphus* has been rewritten. The new version supports: One log file per job invocation; enhanced configuration for logging; command line arguments.
- Various attributes of UL4 templates are exposed to UL4 itself.
- Fixed a bug in `ll.url.LocalConnection.rename()`.

6.16.163 Changes in 3.10.1 (released 2010-10-13)

- Fixed bugs in the handling of the `def` and `enddef` opcodes in *ll.xist.ns.jsp.fromul4()*.
- Fixed a bug in the handling of the `render` method in *ll.xist.ns.jsp.fromul4()*.

6.16.164 Changes in 3.10 (released 2010-09-24)

- Python 2.7 is required now as XIST now uses set literals, set and dict comprehension, the new `argparse` module and various other new features of Python 2.7.
- All scripts and *ll.make* have been ported to use `argparse`.
- Fixed a bug in *ll.nightshade*. If the function/procedure didn't set an encoding, the handling of the response body was totally broken (which resulted in a ISO-8859-1 encoded output).
- *ll.xist.parse.Tidy* now supports an additional parameter: If `skipbad` is true, unknown elements and attributes will be skipped.
- The random number functions `random`, `randrange` and `randchoice` have been added to UL4.
- A new function *ll.misc.prettycsv()* has been added. It can be used to pretty print the data produced by the `csv` module.

6.16.165 Changes in 3.9 (released 2010-08-04)

- *ll.xist.ns.html.html* will no longer change the `lang` and `xml:lang` attributes. This functionality has been moved to the new element *ll.xist.ns.htmlspecials.html*. Furthermore this new element won't change existing attributes.
- *ll.xist.ns.html.title* no longer does any manipulation of its content.
- The Java string literal formatting function in *ll.xist.ns.jsp* has been exposed as `javastring()`.
- Fixed a bug in **`oracreate`**: If the source of procedures and functions didn't have whitespace between the name and the `(` the `(` was missing from the output.

6.16.166 Changes in 3.8.3 (released 2010-07-29)

- `str` arguments are now always treated as `BLOB`s in *ll.orasql* functions and procedures.

6.16.167 Changes in 3.8.2 (released 2010-06-21)

- Fixed a bug in the logging methods of *ll.sisyphus.Job*: Logging unicode strings didn't work. Now all strings are promoted to unicode.
- The default encoding for *ll.sisyphus* log files has changed to UTF-8. This can be changed by setting the class attribute `encoding` in the class derived from *ll.sisyphus.Job*.

6.16.168 Changes in 3.8.1 (released 2010-06-17)

- The method *ll.url.URL.import_()* that had been dropped in version 3.8 has been reintroduced. However internally `misc.module()` is used for creating the module object. A side effect of this is that importing from non-local URLs now works:

```
>>> from ll import url
>>> u = url.URL("http://www.livinglogic.de/Python/misc/index_module.py")
>>> m = u.import_()
>>> m.last("gurk")
"k"
```

6.16.169 Changes in 3.8 (released 2010-06-15)

- The parsing infrastructure has been completely rewritten to be more modular and to support iterative parsing (similar to ElementTree).

  Now parsing XML is done in a pipelined approach that looks like this:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import html
>>> doc = parse.tree(
...   parse.String("<a href='http://www.python.org/'>Python</a>")
...   parse.Expat()
...   parse.NS(html)
...   parse.Node(pool=xsc.Pool(html))
... )
>>> doc.bytes()
'<a href="http://www.python.org/">Python</a>'
```

  Iterative parsing looks like this:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import xml, html, chars
>>> for (evtype, path) in parse.itertree(
...   parse.URL("http://www.python.org/"),
...   parse.Expat(ns=True),
...   parse.Node(pool=xsc.Pool(xml, html, chars)),
...   filter=html.a/html.img
... ):
...   print path[-1].attrs.src, "-->", path[-2].attrs.href
http://www.python.org/images/python-logo.gif --> http://www.python.org/
http://www.python.org/images/trans.gif --> http://www.python.org/#left%2Dhand
↪%2Dnavigation
http://www.python.org/images/trans.gif --> http://www.python.org/#content%2Dbody
http://www.python.org/images/donate.png --> http://www.python.org/psf/donations/
http://www.python.org/images/worldmap.jpg --> http://wiki.python.org/moin/
↪Languages
```

(continues on next page)

(continued from previous page)

```
http://www.python.org/images/success/tribon.jpg --> http://www.python.org/about/
↪success/tribon/
```

- The XIST element `ll.xist.ns.specials.z` has been moved to the *ll.xist.ns.doc* module.
- The function `ll.xist.xsc.docprefixes` has been dropped. A new function *ll.xist.xsc.docpool* has been added.
- The module `ll.xist.parsers` has been renamed to `parse`.
- The module `ll.xist.presenters` has been renamed to `present`.
- The classes `ll.xist.converters.Converter` and `ll.xist.publishers.Publisher` has been moved to *ll.xist.xsc*. The modules `ll.xist.converters` and `ll.xist.publishers` no longer exist.
- The walk methods `walknode()` and `walkpath()` have been renamed to `walknodes()` and `walkpaths()` and the implemention has been moved from the nodes classes into `WalkFilter`. `WalkFilter` has been moved to *ll.xist.xfind*.
- A new selector has been added to *ll.xist.xfind*: `AnySelector` outputs all nodes.
- Added a new function `misc.module()` that creates a module from source code.
- `ll.url.Path` has been simplified: Path segments are strings instead of tuples now.
- The old `URL` method `import_()` has been removed. The new function `misc.module()` can now be used for that.
- The two classes `ll.make.PoolAction` and `ll.make.XISTPoolAction` have been dropped. You can use `make.ObjectAction(misc.Pool).call()` and `make.ObjectAction(xsc.Pool).call()` for that.
- The class `XISTParseAction` has been removed. This action can be replaced by a combination of `ObjectAction`, `CallAction` and `CallAttrAction`.
- Two new UL4 functions `abs` and `utcnow` have been added.
- A few methods have been added to UL4 date objects: `mimeformat`, `day`, `month`, `year`, `hour`, `minute`, `second`, `microsecond`, `weekday` and `yearday`.
- Use autoboxing in the Java code generated by *ll.xist.ns.jsp.fromul4*.
- All code has been switched to using the `format()` method instead of using the `%` operator.
- ssh URLs in *ll.url* now use the standalone `execnet` package.
- ssh URLs now support a `nice` argument instead of `ssh_config`.

6.16.170 Changes in 3.7.6 (released 2010-05-14)

- Fixed a bug in *ll.xist.ns.htmlspecials.autopixel*.

6.16.171 Changes in 3.7.5 (released 2010-04-19)

- *ll.orasql.PrimaryKey* has a new method `itercolumns()` that returns an iterator over the columns this primary key consists of.

6.16.172 Changes in 3.7.4 (released 2010-03-25)

- Fixed a bug in *ll.xist.ns.rss20.guid*. The `isPermaLink` attribute was a `URLAttr`, but must be a `TextAttr`.

6.16.173 Changes in 3.7.3 (released 2010-02-27)

- Fixed a bug in the generated JSP code for the `def` opcode in *ll.xist.ns.jsp.fromul4()*.

6.16.174 Changes in 3.7.2 (released 2010-02-26)

- Fixed two bugs in the XML codecs:
  - An externally specified encoding wasn't honored in the incremental decoder.
  - Fixed reset() for incremental codecs: If encoding has been changed during parsing in the incremental codecs it now gets reset to its proper initial value.
- Fixed a bug in the handling of the UL4 opcode `addlist` in *ll.xist.ns.jsp.fromul4()*.
- Added missing processing instruction class for the UL4 `def` tag to the `ll.xist.ns.ul4` namespace module.
- The generated JSP code for the `loadvar` opcode now uses the Java method `Utils.getItem`, so that non-existent variables no longer get treated as `None`/`null`.

6.16.175 Changes in 3.7.1 (released 2010-02-08)

- *ll.xist.ns.jsp.fromul4()* now outputs the correct code for calling the `format` method on date objects (This requires version exp-22 of the UL4 Java package).

6.16.176 Changes in 3.7 (released 2009-09-10)

- In UL4 templates it's now possible to define locale templates via `<?def tmpl?>templatecode<?end def?>`.
- Python 2.6 is required now.
- *ll.orasql* and *ll.nightshade* are now part of the distribution.
- *ll.make* has a new Action class: `ObjectAction` simply returns an existing object.
- The following classes have been removed from *ll.make*: `EncodeAction`, `DecodeAction`, `EvalAction`, `GZipAction`, `GUnzipAction`, `JavascriptMinifyAction`, `XISTBytesAction`, `XISTStringAction`, `JoinAction`, `UnpickleAction`, `PickleAction`, `TOXICAction`, `TOXICPrettifyAction`, `SplatAction`, `UL4CompileAction`, `UL4RenderAction`, `UL4DumpAction`, `UL4LoadAction`, `XISTTextAction` and `XISTConvertAction`. All of these actions can be executed by using `CallAction` or `CallAttrAction`.
- *ll.make.PipeAction* has been renamed to `TransformAction`.
- The new *ll.make.PipeAction* pipes the input through an external command.
- *ll.make.FileAction* now automatically wraps the `key` argument into an `URL` object.
- *ll.make.FileAction* has two new methods `chmod()` and `chown()` that return a `ModeAction` and `OwnerAction` for modifying the file created by the `FileAction`.
- *ll.make.Action* has three new methods: `call()`, `getattr()` and `callattr()` create a `CallAction`, `GetAttrAction` or `CallAttrAction` object respectively.
- The division operator is no longer implemented for `Action` objects in *ll.make*.
- Two new UL4 functions have been added: `float` and `iscolor`.

- Two new scripts have been added: **uls** can be used to list any directory given as an URL. **ucat** can be used to output any file or directory.
- The script **ucp** now changes the user and group only if a user or group is given.
- A bug in the 64-bit support for `sgmlop` has been fixed.
- Fixed a bug in the remote `stat()` method for ssh URLs (it seems that the `posix.stat_result` tuple objects can no longer be pickled).
- There's a new function `misc.itersplitat()` for splitting a string at specified positions.
- For ssh URLs a keyword argument `ssh_config` is supported now instead of `identity` (This mirrors the corresponding change in the py library)

6.16.177 Changes in 3.6.6 (released 2009-07-09)

- Fixed handling of empty pid files in *ll.sisyphus* (Fixes issue #11 reported by Jarek Zgoda).

6.16.178 Changes in 3.6.5 (released 2009-06-02)

- Fix UL4 templates that produce no output: As the generated Python sourcecode didn't contain any `yield` statements, the resulting function was an ordinary function instead of a generator.

6.16.179 Changes in 3.6.4 (released 2009-03-19)

- A new UL4 method `join` has been added. It works like the Python string method join.
- *ll.misc* has three new functions: `gzip()` und `gunzip()` can be used for compressing and uncompressing byte strings with gzip. `jsmin()` can be used to minify Javascript source.
- Parsing an empty string with `tidy=True` in `ll.xist.parsers.parsestring()` now works again.

6.16.180 Changes in 3.6.3 (released 2009-03-02)

- The xfind operators `attrhasvalue`, `attrhasvalue_xml`, `attrcontains`, `attrcontains_xml`, `attrstartswith`, `attrstartswith_xml`, `attrendswith`, `attrendswith_xml`, `hasid` and `hasclass` now support multiple values. The operator matches the node if it matches with any of the given values.
- A new function `reversed` is now available in UL4 templates. It returns an iterator that will output the items of any sequence in reverse order.

6.16.181 Changes in 3.6.2 (released 2009-02-16)

- Inside UL4 templates rendering other templates can now be done with the new `render` method. This method returns the template output as a string. Passing parameters can be done via keyword arguments or with the `**` syntax like when using the `render` tag.
- A new version of the `int` function has been added to UL4: When called with two arguments, the first must be a string, and the second is treated as the base for the conversion.

6.16.182 Changes in 3.6.1 (released 2009-01-27)

- Generating the Python source from an UL4 template is now 20-25% faster.
- Fixed a buffer overrun in the C portions of the url module.
- Added a class `addattr` to *ll.xist.xsc*. This can be used to extend XML attributes via `with` blocks.
- Added the function *ll.xist.ns.jsp.fromul4()* which can turn an UL4 template into JSP source code.

6.16.183 Changes in 3.6 (released 2008-12-31)

- The following `Color` class methods have been dropped: `fromrgba`, `fromrgba4`, `fromrgba8`, `fromint4`, `fromint8`.
- The following `Color` properties have been dropped: `r4`, `g4`, `b4`, `a4`, `r8`, `g8`, `b8`, `a8`, `r`, `g`, `b`, `a` `int4`, `int8`, `rgb4`, `rgba4`, `rgb8`, and `rgba8`. The new methods `r`, `g`, `b` and `a` return the 8 bit component values.
- The class methods `fromhsva` and `fromhlsa` have been renamed to `fromhsv` and `fromhls`.
- The property `css` has been dropped. Instead the CSS string is returned by `__str__`.
- Dividing colors now does a scalar division. Blending colors is now done with the modulo operator.
- Support for color objects has been added to UL4.
- The XPIT templating language and `ll.make.XPITAction` have been removed.
- Fixed a bug in `ll.make.CacheAction.get()`: The action must return real data when called with `bigbang` as the timestamp.
- `ll.make.UL4RenderAction` has been fixed.

6.16.184 Changes in 3.5 (released 2008-12-05)

- A new function `json` has been added to UL4: This function returns a JSON dump of the object passed in (this requires either `simplejson` or Python 2.6).
- The UL4 function `csvescape` has been renamed to `csv`.
- A new option `--showregistration`/`-r` has been added to make scripts.
- *ll.make* now supports Growl notifications on Mac OS X. To activate it set the `LL_MAKE_GROWL` environment variable to 1 or use the `-g` or `--growl` options.
- *ll.make* has a new action class `JavascriptMinifyAction` for minimizing Javascript source.
- *ll.color.Color* has been rewritten to create immutable objects with the components being 8 bit values (i.e. 0-255) instead of floating point values between 0 and 1. An alpha component has been added.
- A `strong` element has been added to the *ll.xist.ns.doc* namespace.

6.16.185 Changes in 3.4.4 (released 2008-09-16)

- Fixed a bug in `ll.make.JoinAction.execute()`.

6.16.186 Changes in 3.4.3 (released 2008-09-09)

- `css.applystylesheets()` could no longer handle style declarations containing comments. This has been fixed now.

6.16.187 Changes in 3.4.2 (released 2008-09-03)

- Parsing didn't work when `tidy` was set to true and a `base` argument was given. This has been fixed now.

6.16.188 Changes in 3.4.1 (released 2008-08-29)

- Bugs with thread local storage have been fixed so using `xsc.Pool`, `xsc.build` and URL contexts in `with` blocks in multithreaded applications should work now.

6.16.189 Changes in 3.4 (released 2008-08-19)

- Templates can no longer be passed as a separate dictionary to UL4 templates but are passed as variables like other data objects too.
- Strings in UL4 have gained a new method `capitalize`.
- Printing XML escaped strings in UL4 has now gained its own tag and opcode. `<?printx foo?>` is equivalent to `<?print xmlescape(foo)?>`.
- Exception handling in UL4 has been rewritten to allow proper error reporting when calling nested templates.
- UL4 has gained a new function `zip`. It can be called with two or three arguments and does what `itertools.zip()` does.
- UL4 has gained another new function: `type` returns the type of its argument as a string.
- UL4 now supports tuple unpacking in `<?for?>` tags with three variables.
- UL4 has a new tag for comments: `<?note This is comment?>`.
- A new script `db2ul4.py` has been added that can render UL4 templates with database content.
- In UL4s `<?render?>` tags it's now possible to pass along a complete argument dictionary via the `**arg` syntax just like in Python. This syntax can even be used multiple times in the call. This syntax is available in dictionary literals too, i.e. `{1:2, 3:4}` and `{**{1:2}, **{3:4}}` are equivalent.
- A new UL4 function `get` has been added that works similar to the dictionary method `get`, but works with global variables.
- The missing processing instruction `render` has been added to `ll.xist.ns.ul4`.
- `xml_codec` now partially works, even if the C module is missing. As long as you explicitly specify an encoding on parsing and publishing it should work.
- A new processing instruction class `ll.xist.AttrProcInst` has been introduced. When an `AttrProcInst` node is the first node in an attribute, it takes over publishing of the attribute. In all other cases the processing instruction disappears completely. UL4 uses this to implement "conditional attributes" (via the new classes `attr_if` and `attr_ifnn`).
- Building trees with `with` blocks has changed slightly. Nodes used in `with` blocks and with + are now passed to a `with` handler instead of building the tree directly. This fixes a problem when nested `convert()` calls use `with` blocks.
- The element `ll.xist.ns.form.memo` has been renamed to `textarea` and `ll.xist.ns.form.edit` has been renamed to `text`. Classes `ll.xist.ns.form.button` and `ll.xist.ns.form.file` have been added.

- Iterating through the inputs in *ll.make* actions has been fixed (i.e. the additional inputs will be output too). *ll.make.Project.findpaths()* has been fixed to work with non-*ll.make.Action* inputs. (This means that now you *have* to pass a real registered target action to `findpaths()` not just its key).
- *ll.make* has gained a new action: `XISTStringAction` publishes an XIST node as a unicode string. `XISTPublishAction` has been renamed to `XISTBytesAction`.
- Fixed a bug in the caching logic in *ll.make.CacheAction()*.
- `ll.make.CallMethAction` has been renamed to `CallAttrAction` because it can be used to e.g. call functions in a module too.
- The properties `showaction`, `showstep` and `shownote` of *ll.make.Project* object can now be assigned booleans as well (which results in *all* or *no* actions being shown.
- The version number for `cssutils` has been bumped to 0.9.5.1.

6.16.190 Changes in 3.3.2 (released 2008-07-15)

- Dictionaries in UL4 have gained a new method `get`.
- The version number for `cssutils` has been bumped again (to 0.9.5rc2 or a later 0.9.5 version).
- Fixed a bug in the parsing of slice expressions in UL4.
- *ll.make* has gained a new `UL4RenderAction` action.
- Fixed a bug in the formatting for the `getslice2` opcode for UL4.

6.16.191 Changes in 3.3.1 (released 2008-07-14)

- Fixed a bug in the implementation of the "not" operator in UL4.
- When the UL4 compiler encounters unclosed blocks, it will now include the start location of the block in the error message.

6.16.192 Changes in 3.3 (released 2008-07-11)

- XIST has gained its fourth templating language: `UL4` the "Universal Layout Language". This templating language is similar in capabilities to Djangos templating language. However `UL4` templates are compiled to a bytecode format, which makes it possible to implement template renderers in other languages and makes the template code "secure" (i.e.template code can't open or delete files).
- *ll.make* has gained new actions: `GZipAction`, `GUnzipAction`, `CallFuncAction`, `CallMethAction`, `UL4CompileAction`, `UL4DumpAction` and `UL4LoadAction`.
- The version number for `cssutils` has been bumped to 0.9.5rc1.
- Nodes of type *ll.xist.xsc.Comment* and *ll.xist.xsc.DocType* inside of attributes are now simply ignored when publishing instead of generating an exception.
- All actions in *ll.make* no longer check whether their inputs are action objects. Non-action objects are simply treated as ancient input data. This also means that most action classes have an `input` parameter in their constructor again, as this input could now be a constant.
- Most attributes of action objects in *ll.make* can now be action objects themselves, so for example the name of the encoding to be used in an `EncodeAction` can be the output of another action.
- `ll.make.ImportAction` has been dropped as now the module object can be used directly (e.g. as the input for an `XISTPoolAction` object).
- `ll.misc.xmlescape()` now escapes `'` as `'` for IE compatibility.
- Functions `ll.misc.xmlescape_text()` and `ll.misc.xmlescape_attr()` have been added that implement the functionality from XIST 3.2.5 and earlier.

- The default parser for XIST is expat now. To switch back to sgmlop simply pass an `SGMLOPParser` object to the parsing functions:

```
>>> from ll.xist import parsers
>>> node = parsers.parsestring("<a>", parser=parsers.SGMLOPParser())
```

- TOXIC has been split into a compiler module `ll.toxicc` and an XIST namespace *ll.xist.ns.toxic*. TOXIC now supports output for SQL Server. The function `xml2ora()` as been renamed to `compile()` (and has a new `mode` argument for specifying the database type).
- The `targetroot` parameter for `ll.make.XISTConvertAction.__init__()` has been renamed to `root`.

6.16.193 Changes in 3.2.7 (released 2008-05-16)

- Added the missing file `_misc_include.c` to the distribution archives.

6.16.194 Changes in 3.2.6 (released 2008-05-07)

- A new action class `EvalAction` has been added to *ll.make*.
- `ll.xist.helpers.escapetext()` and `ll.xist.helpers.escapeattr()` have been merged into one function that escapes all special characters (including `'` and `"`) and has been renamed/moved to `ll.misc.xmlescape()`.
- Python versions of all the functions in the module *ll.misc* have been added. Those versions will be used in case the C module is not available.

6.16.195 Changes in 3.2.5 (released 2008-04-11)

- A refcounting bug in the attribute parsing code of `sgmlop` has been fixed.
- The helper function `cssescapereplace()` has been removed, as it's no longer needed.
- Pure Python versions of `helpers.excapetext()` and `helpers.escapeattr()` have been added, in case the C module is not available.

6.16.196 Changes in 3.2.4 (released 2008-04-02)

- The following functions have been added to *ll.xist.css*: `parsestring()`, `parsestream()`, `parsefile()`, `parseurl()` and `write()`. They parse CSS resources and are similar to the XML/HTML parsing functions in that they apply the specified base URL to all URLs in the style sheet.
- `cssutils` 0.9.5b2 is required now.
- *ll.xist.css.iterrules()* and *ll.xist.css.applystylesheets()* now support specifying whether the preferred stylesheets or an alternate stylesheet group should be used.
- `ll.xist.xsc.ProcInst.__mul__()` and `ll.xist.xsc.ProcInst.__rmul__()` now return a fragment containing the node repeated a number of times instead of one processing instruction node containing repeated content.
- The constructor for `ll.xist.parsers.ExpatParser` now takes two additional arguments:

xmldecl
: If this is true the XML declaration will appear in the resulting XIST tree.

doctype
: If this is true the doctype declaration will appear in the resulting XIST tree (however any internal DTD subset will be dropped).

6.16.197 Changes in 3.2.3 (released 2008-03-04)

- `cssutils` 0.9.5 is used now. This simplifies the implementation of `css.selector()`.
- A function *`ll.xist.css.geturls()`* has been added. This returns a list of all the URLs in a `cssutils` stylesheet.
- `toxic.xml2ora()` now treats unknown processing instructions as text. This makes it possible to e.g. output an XML header via toxic.
- The pseudo-elements in *`ll.xist.ns.jsp`* are no longer in a namespace, so they will always be published without any prefixes.

6.16.198 Changes in 3.2.2 (released 2008-02-25)

- A new method `replaceurls()` has been added to *`ll.xist.xsc.StyleAttr`*. With this method all URLs in a `style` attribute can be replaced.
- Fixed a bug in `ll.xist.parsers.SGMLOPParser.begin()`: The encoding wasn't passed properly to the XML decoder.
- `ll.xist.xsc.ProcInst.publish()` now calls the `checkvalid()` method too.

6.16.199 Changes in 3.2.1 (released 2008-02-05)

- It's now possible to force the publisher to output certain `xmlns` attributes via the `showxmlns` argument to the `Publisher` constructor.

6.16.200 Changes in 3.2 (released 2008-02-01)

- The core package has been moved into XIST, installing XIST now only requires *one* package.
- `ll.toxic` has been moved into XIST and is now available as *`ll.xist.ns.toxic`*.
- When a `ll.make.XISTParseAction` object is executed the content of the pool will now be extended by the content of the pool from the `XISTPoolAction` instead of being replaced.
- `ll.make.Pool` and *`ll.xist.xsc.Pool`* no longer use a `WeakValueDictionary`, but a simple `dict`. This means they can now store *any* object. A method `clear()` has been added, which removes all registered objects.
- Fixed a bug in *`ll.xist.css.iterrules()`* that surfaced when a `base` argument was given.
- Fixed a second bug in *`ll.xist.css.iterrules()`* where the `href` of a `link` element wasn't applied to the URLs in the stylesheet.

6.16.201 Changes in 3.1 (released 2008-01-18)

- Fixed the problem that the source distibution didn't include header files.
- If an `URLAttr` attribute contains a processing instruction XIST will no longer transform the URL in any way.
- Fixed a parser bug where attributes were dropped when the attribute value was empty.
- Putting a module into a `Pool` object now copies the `xmlns` attribute too. This makes it possible to use `Pool` objects as conversion targets.

6.16.202 Changes in 3.0 (released 2008-01-07)

- Namespaces have been greatly simplified. There are no namespace modules any longer. An element class can be assigned a namespace by setting the `xmlns` class attribute to the namespace name. Global attributes can be assigned a namespace by setting the `xmlns` attribute on the attribute class itself (*not* on the `Attrs` class). The classes `Prefixes` and `NSPool` are gone too. Instead a new class `Pool` is used to specify which classes should be used for parsing.
- Dependency on PyXML has finally been dropped. XIST now uses its own XML parsing API. Two parsers are available: One based on expat and one based on a custom version of sgmlop.
- Tree traversal has been rewritten again. XFind expressions involving multiple uses of `//` now work correctly. The method `walk()` now doesn't yield `Cursor` objects, but simple path lists (actually it's always the same list, if you want distinct lists use `walkpath()`). Applying XFind expressions to nodes directly is no longer supported, you have to call `walk()`, `walknode()` or `walkpath()` with the XFind expression instead. Many XFind operators have been renamed and/or reimplemented (see the documentation for the `xfind` module for more information).
- The methods `__getitem__()`, `__setitem__()` and `__delitem__()` for `Frag` and `Element` now support the new walk filters, so you can do:
  - `del node[html.p]` to delete all `html.p` child elements of `node`;
  - `del node[html.p[2]]` to delete only the third `html.p`;
  - `node[xfind.hasclass("note")] = html.p("There was a note here!")` to replace several child nodes with a new one;
  - `for c in node[xfind.empty]: print c.bytes()` to print all empty (element) children of `node`;
  - `del node[node[0]]` to delete the first child node (which is silly, but illustrates that you can pass a node to get/replace/delete that node);
- A new module *`ll.xist.css`* has been added which contains CSS related functionality: The generator function `iterrules()` can be passed an XIST tree and it will produce all CSS rules defined in any `html.link` or `html.style` elements or imported by them (via the CSS rule `@import`). This requires the `cssutils` package.
- The function `applystylesheets()` modifies the XIST tree passed in by removing all CSS (from `html.link` and `html.style` elements and their `@imported` stylesheets) and putting the styles into `style` attributes of the affected elements instead.
- The function `selector()` returns a tree walk filter from a CSS selector passed in as a string.
- Constructing trees can now be done with `with` blocks. Code looks like this:

```
with xsc.Frag() as node:
   +xml.XML()
   +html.DocTypeXHTML10transitional()
   with html.html():
      with html.head():
         +meta.contenttype()
         +html.title("Example page")
      with html.body():
         +html.h1("Welcome to the example page")
         with html.p():
            +xsc.Text("This example page has a link to the ")
            +html.a("Python home page", href="http://www.python.org/")
            +xsc.Text(".")

print node.conv().bytes(encoding="us-ascii")
```

Also the function `xsc.append()` has been renamed to `add()` and supports `with` blocks now instead of XPython.

- A subset of ReST is supported now for docstrings when using the *ll.xist.ns.doc* module. The module attribute `__docformat__` is now honored (Set it to `"xist"` to get XIST docstrings).
- Many classes in the *ll.xist.ns.doc* have been renamed to more familiar names (from HTML, XHTML 2 or ReST).
- The `media` attribute of `html.link` and `html.style` now has a method `hasmedia()`.
- The node method `asBytes()` has been renamed to `bytes()` and `bytes()` has been renamed to `iterbytes()`.
- The node method `asString()` has been renamed to `string()` and a new method `iterstring()` has been added.
- `ll.xist.ns.xml.XML10` is gone now. Use *ll.xist.ns.xml.XML* instead.
- `xsc.tonode()` now will raise an exception when it can't handle an argument instead of issuing a warning.
- A class attribute `empty` inside element classes will now no longer get converted into `model`.
- *ll.xist.ns.doc.pyref* now copes better with decorated methods.
- The deprecated `Element` methods `hasAttr()`, `hasattr()`, `isallowedattr()`, `getAttr()`, `getattr()`, `setDefaultAttr()`, `setdefaultattr()`, `attrkeys()`, `attrvalues()`, `attritems()`, `iterattrkeys()`, `iterattrvalues()`, `iterattritems()`, `allowedattrkeys()`, `allowedattrvalues()`, `allowedattritems()`, `iterallowedattrkeys()`, `iterallowedattrvalues()`, `iterallowedattritems()` and `copyDefaultAttrs()` have been removed. The deprecated `Attrs` method `copydefaults()` has been removed too.
- The namespace module `ll.xist.ns.cond` has been removed.
- When calling the function `ll.xist.parsers.parseURL()` the arguments `headers` and `data` are now passed along to the parser's method only if they are specified. This makes it possible to pass ssh URLs to `ll.xist.parsers.parseURL()`.
- The methods `withnames()` and `withoutnames()` have been split into two that take Python names and two that take XML names. Multiple arguments are used now (instead of one argument that must be a sequence). Passing a namespace to remove all attributes from the namespace is no longer supported.
- The `Attrs` methods `updatenew()` and `updatexisting()` have been removed.

6.16.203 Changes in 2.15.5 (released 2007-07-17)

- The Python quotes example no longer contains the XML source or the generated HTML.

6.16.204 Changes in 2.15.4 (released 2007-07-16)

- The Python quotes example now always parses the file from the original URL.
- The Python quotes and the media example now print the result to `stdout`.

6.16.205 Changes in 2.15.3 (released 2007-07-16)

- Use a consistent license (MIT) everywhere. This should make XIST Debian compatible.
- Change the Python quotes example, so that it works even if there's no `python-quotes.xml` in the current directory.

6.16.206 Changes in 2.15.2 (released 2007-01-24)

- Fixed a bug in `presenters.CodePresenter.__str__()`.
- Fixed base URL handling for tidy parsing.
- Updated examples.
- Updated `xiter()` and `xattrs()` implementations for `Node` and `Namespace` to conform to the newest version of IPython.

6.16.207 Changes in 2.15.1 (released 2006-09-25)

- Fixed a few bugs in the `sgmlop` function declarations.
- Readded the spacer pixel.

6.16.208 Changes in 2.15 (released 2006-09-24)

- XIST has been made compatible with Python 2.5: Code has been updated to use the proper C API for memory management and **PEP 353** support has been added. XIST now includes its own fixed version of `sgmlop`.
- The *`ll.xist.xsc.Attrs`* methods `with()` and `without()` have been renamed to `withnames()` and `withoutnames()` for Python 2.5 compatibility.
- *`ll.xist.ns.htmlspecials.pixel`* no longer handles colors via different GIFs. It uses the `background-color` in the `style` attribute instead. The same change has been implemented for *`ll.xist.ns.htmlspecials.autopixel`*. It's now possible to overwrite the default `src` attribute value of `root:px/spc.gif` either via the XML attribute or via the converter context.
- The node method `asText()` has been made a function, moved into the `html` namespace and renamed to `astext()`. Furthermore elinks is used for plain text formatting now instead of w3m.

6.16.209 Changes in 2.14.2 (released 2006-07-04)

- Fixed a bug in the `presentAttr()` method of `ll.xist.presenters.TreePresenter`.

6.16.210 Changes in 2.14.1 (released 2006-06-29)

- Fixed a bug in the `presentEntity()` method of `ll.xist.presenters.CodePresenter`.
- Updated installation instructions.

6.16.211 Changes in 2.14 (released 2006-06-28)

- Namespaces for RSS 0.91, RSS 2.0 and Atom 1.0 have been added.
- A new namespace *ll.xist.ns.detox* has been added that is similar to `ll.toxic` but can be used to generate Python code instead of PL/SQL code. Using `detox` templates is about 50 times faster than using XIST trees directly and about 10 times faster than Kid.
- Presenters are now compatible to IPython `ipipe` module. This means that you can browse XIST trees interactively if you have IPython installed. `NormalPresenter` and the `Node` methods `repr()` and `asrepr()` have been removed.
- A new processing instruction *ll.xist.ns.specials.url* has been added that does the same URL transformation as *ll.xist.xsc.URLAttr* does.
- On publishing *ll.xist.ns.html.html* now only adds a `lang` and `xml:lang` attribute, if neither of them exists.
- `setuptools` is now supported for installation.

6.16.212 Changes in 2.13 (released 2005-10-31)

- `ll.xist.xsc.Namespace.tokenize()` requires a `unicode` object as input now. This makes it possible to use encodings that are not ASCII compatible (such as UTF-16). The `encoding` argument is gone.
- `ll.xist.xsc.Node.asString()` uses the `encoding` argument to determine which characters have to be output as character references now. (You'll still get a unicode object as the result.)
- A new processing instruction class *ll.xist.ns.specials.literal* has been added, that will output its content literally when published. This can be used for embedding preformatted XML (e.g. from a database) into an XIST tree.

6.16.213 Changes in 2.12 (released 2005-10-13)

- Namespaces for Relax NG and Kid have been added.
- XIST requires version 1.0 of the core package now.
- The class name for the DocBook DTD class has been fixed.

6.16.214 Changes in 2.11 (released 2005-07-29)

- A script `xml2xsc.py` has been added, that can be used to parse an XML file and generate a rudimentary XIST namespace from it.
- A `DocType` for XHTML 1.1 has been added (suggested by Elvelind Grandin).
- Line number information is now added when parsing HTML.
- The `sorted()` method now supports the same arguments (`cmp`, `key` and `reverse`) as `list.sort()` and `sorted()` in Python 2.4.
- The `walk()` doesn't yield the node directly, but yields a `Cursor` object now, with has several ways of referencing the node.
- New methods `walknode()`, `walkpath()` and `walkindex()` have been added.
- Presenters use an iterator API instead of a stream API now. Dumping an XML tree presentation to the terminal can now start immediately instead of having to wait for the complete string to be formatted.
- Fixed a bug with element/attribute names that contained a `.` character. (This broke `ll.xist.ns.fo`.)
- Fixed a bug with `xmlns` attributes in nested elements. When an element ended the parser restored the wrong prefix mapping.

- The `python-quotes` demo has been updated to use the current version of AMK's XML file.
- Removed iterator stuff from *ll.xist.xfind*, as this is now part of the `ll` package/module.
- The function `ToNode()` has been renamed to `tonode()`.
- `ll.xist.Context` no longer subclasses `list`.
- *ll.xist.ns.doc.explain* will now try to output the objects in the order in which they appear in the Python source.
- The node methods `find()` and `findfirst()` have been removed.
- `ll.xist.ns.cond` now uses a sandbox dictionary in a converter context for evaluating expression.

6.16.215 Changes in 2.10 (released 2005-05-20)

- The content of the processing instruction *ll.xist.ns.code.pyexec* will not be executed at construction time, but at conversion time. The code in *ll.xist.ns.code.pyexec* or *ll.xist.ns.code.pyeval* will no longer be executed in the `ll.xist.sandbox` module (which has been removed), but in a sandbox dictionary in the converter context of the *ll.xist.ns.code* namespace.
- The tests have been ported to py.test.
- The method `mapped()` is now callable without arguments. In this case a converter will be created on the fly. You can pass constructor arguments for this converter to `mapped()` as keyword arguments.
- The publishing API has changed again: `ll.xist.publishers.Publisher.publish()` no longer accepts an argument `stream` to which the byte strings are written, but it is a generator now. The publisher methods `write()` and `writetext()` have been renamed to `encode()` and `encodetext()` and return the encoded byte string, instead of writing it directly to the stream. There's a new generator method `bytes()` for nodes now, which can be passed the same arguments as `asBytes()`. These changes should help when using XIST in WSGI applications.
- The iterator returned from `Element.__getitem__()`, `Frag.__getitem__()` and the `walk()` method now supports `__getitem__()` itself, so you can write `table[html.tr][0]` to get the first row from a table or `page.walk(xsc.FindTypeAll(html.td))[-1]` to get the last table cell from a complete HTML page.
- Several bugs in the namespaces *ll.xist.ns.meta*, *ll.xist.ns.form* and *ll.xist.ns.specials* have been fixed.
- The namespace modules `ll.xist.ns.css` and `ll.xist.ns.cssspecials` have been removed.

6.16.216 Changes in 2.9 (released 2005-04-21)

- XIST trees can now be pickled. The only restriction is that global attributes must come from a namespace that has been turned into a module via `makemod()`, so that this module can be imported on unpickling.
- Two arguments of the `walk()` method have been renamed: `filtermode` has been renamed to `inmode` and `walkmode` has been renamed to `outmode`. For these modes two new values are supported:

  `ll.xist.xsc.walkindex`
  The value passed to the filter function or yielded from the iterator is a list containing child indizes and attribute names that specify the path to the node in question.

  `ll.xist.xsc.walkrootindex`
  The filter function will be called with two arguments: The first is the root node of the tree (i.e. the node for which `walk()` has been called), the second one is an index path (just like for `ll.xist.xsc.walkindex`). If used as an `outmode` a tuple with these two values will be yielded.

- **Attribute mappings now support `__getitem__()`, `__setitem__()` and `__delitem__()`** with list arguments, i.e. you can do:

```
>>> from ll.xist.ns import html
>>> e = html.a("gurk", href=("hinz", "kunz"))
>>> print e.attrs[["href", 0]]
hinz
>>> e.attrs[["href", 0]] = "hurz"
>>> print e["href"]
hurzkunz
>>> del e.attrs[["href", 0]]
>>> print e["href"]
kunz
```

XML attributes can now be accessed as Python attributes, i.e.:

```
>>> from ll.xist.ns import html
>>> e = html.a("spam", href="eggs")
>>> print e.attrs.href
eggs
```

(Don't confuse this with `e.Attrs.href` which is the attribute class.)

- `Frag` and `Element` now support `Node` subclasses as arguments to their `__getitem__()` method: An iterator for all children of the specified type will be returned.
- The encoding used for parsing now defaults to `None`. When reading from an URL and no default encoding has been specified the one from the `Content-Type` header will be used. If this still doesn't result in a usable encoding, `"utf-8"` will be used when parsing XML and `"iso-8859-1"` will be used when parsing broken HTML.
- All error and warning classes from `ll.xist.errors` have been merged into *`ll.xist.xsc`*. This avoids import problems with circular imports.
- The attributes `showLocation` and `showPath` of `ll.xist.presenters.TreePresenter` have been lowercased and presenters are properly reset after they've done their job.
- The class attribute `xmlname` will no longer be turned into a list containing the Python and the XML name, but will be the XML name only. You can get the Python name of `foo` from `foo.__class__.__name__`.
- `DeprecationWarning`s for `name` and `attrHandlers` have finally been removed.
- Instances of *`ll.xist.xsc.Entity`* subclasses can now be compared. `__eq__()` simply checks if the objects are instances of the same class.

6.16.217 Changes in 2.8.1 (released 2005-03-22)

- Added a note about the package init file to the installation documentation.

6.16.218 Changes in 2.8 (released 2005-01-03)

- XIST requires Python 2.4 now.
- `ll.xist.ns.specials.x` has been renamed to *`ll.xist.ns.specials.ignore`*.
- `ll.xist.utils.findAttr()` has been renamed to `ll.xist.utils.findattr()`.
- `ll.xist.xfind.item` no longer handles slices.
- XFind has been enhanced to support item and slice operators, i.e. if `foo` is an XFind operator, `foo[0]` is an operator that will produce the first node from `foo` (if there is one). Negative values and slices are supported too.
- Operators can be chained via division: `html.a/html.b` is an operator that can be passed around and applied to a node.

- XIST requires the new core module and makes use of the new "cooperative displayhook" functionality defined there: If you install the displayhook you can tweak or replace `ll.xist.presenters.hookpresenter` to change the output.

6.16.219 Changes in 2.7 (released 2004-11-24)

- The transparent pixel used by *ll.xist.ns.htmlspecials.pixel* has been renamed to `spc.gif` to avoid problems with IE.
- Removed a debug print in `ll.xist.xfind.Finder.__getitem__`.
- *ll.xist.xfind* now has a new function `item()`, that can be used to get a certain item or slice from an iterator. `xfind.first()` and `xfind.last()` have been changed to use `xfind.item()`, so you now have to pass a default value to get the old behaviour.
- Obsolete options in `ll.xist.options` have been removed (and `reprEncoding` has been renamed to `reprencoding`).

6.16.220 Changes in 2.6.2 (released 2005-06-06)

- Fixed a bug in `ll.xist.parsers.Parser.parse()`.

6.16.221 Changes in 2.6.1 (released 2004-11-02)

- Fixed a bug in `ll.xist.xfind.Finder.__floordiv__()`.
- Restricted characters as defined in XML 1.1 will now be published as character references.

6.16.222 Changes in 2.6 (released 2004-10-26)

- `ToNode()` now tries iterating through the value passed in, so it's now possible to pass iterators and generators (and generator expressions in Python 2.4) to `Frag` and `Element` constructors.
- A new API named XFind has been added for iterating through XML trees. XFind expressions look somewhat like XPath expressions but are pure Python expressions. For example finding all images inside links in an HTML page can be done like this:

```
from ll.xist import parsers, xfind
from ll.xist.ns import html
node = parsers.parseURL("http://www.python.org/", tidy=True)
for img in node//html.a/html.img:
   print img["src"]
```

- The module *ll.xist.xfind* contains several operators that can be used in XFind expressions.
- Parsing broken HTML is now done with the HTML parser from libxml2. The parsing functions no longer accept options for tidy, only the boolean value of the `tidy` argument is used.
- The publishing API has been simplified: Publication can now be done with a call to `ll.xist.publishers.Publisher.publish()`, passing in a *ll.xist.xsc.Node*. Writing strings to the publisher output is now done with `ll.xist.publishers.Publisher.write()`. The methods `beginPublication()` and `endPublication()` have been removed.
- The presentation API has been simplified in the same way: You'll get a presentation by calling: `string = presenter.present(node)`. The methods `beginPresentation()` and `endPresentation()` have been removed.
- The parser now has the option to ignore illegal elements, attributes, processing instructions and entities. The default behaviour is to raise an exception, but this can now be reconfigured via Python's warning framework.

- The classmethod `tokenize()` from `ll.toxic` has been moved to `ll.xist.xsc.Namespace`, so it's now possible to tokenize an XML string for other processing instructions as well.
- A new class `ll.xist.xsc.NSPool` has been added. An `NSPool` contains a pool of namespaces from which the parser selects the appropriate namespace once an `xmlns` attribute is encountered.</item>
- The script `xscmake.py` (which has been unmaintained for a while now) has been removed.</item>
- Elements `hostname`, `tty`, `prompt` and `input` were added to *ll.xist.ns.doc*.
- The method `ll.xist.xsc.Attrs.set()` now returns the new attribute object.
- The `visit()` method has been removed.
- `ll.xist.xsc.FindOld()` has been removed.
- `ll.xist.ns.xml.header` has been renamed to *ll.xist.ns.xml.declaration*.

6.16.223 Changes in 2.5 (released 2004-06-30)

- Specifying content models for elements has seen major enhancements. The boolean class attribute `empty` has been replaced by an object `model` whose `checkvalid()` method will be called for validating the element content.
- A new module *ll.xist.sims* has been added that provides a simple schema validation. Schema violations will be reported via Pythons warning framework.
- All namespace modules have been updated to use `sims` information. The SVG module has been updated to SVG 1.1. The docbook module has been updated to DocBook 4.3.
- It's possible to switch off validation during parsing and publishing.
- *ll.xist.xsc.Frag* and *ll.xist.xsc.Element* both have a `__call__()` method with the same arguments as their constructors. Those methods will append content nodes (and set attributes for *ll.xist.xsc.Element*) and return `self`, so they can be used when creating an object tree. This makes it possible to put the attributes close to the tag name, instead of putting them at the end after the content.

  Instead of:

  ```
  node = html.table(
     html.tr(
        html.td("foo"),
        html.td("bar"),
     ),
     html.tr(
        html.td("spam"),
        html.td("eggs")
     ),
     class_="example"
  ```

  you can now use the following:

  ```
  node = html.table(class_="example")(
     html.tr(
        html.td("foo"),
        html.td("bar"),
     ),
     html.tr(
        html.td("spam"),
        html.td("eggs")
     )
  )
  ```

- Experimental support for Holger Krekel's XPython has been added. Code might look like this:

```
from ll.xist import xsc, converters
from ll.xist.ns import html, meta

import random

c = converters.Converter()
<c>:
   <html.html()>:
      <html.head()>:
         <meta.contenttype()>: pass
         <html.title()>:
            xsc.append("The title")
      <html.body(class_="foo")>:
         <html.h1()>:
            flag = random.choice((0, 1))
            if flag:
               xsc.append("The foo page", class_="foo")
            else:
               xsc.append("The bar page", class_="bar")
         <html.p()>:
            if flag:
               xsc.append("The foo content")
            else:
               xsc.append("The bar content")

print c.lastnode.asBytes()
```

- Creating global attributes has been simplified. Passing an instance of `ll.xist.xsc.Namespace.Attrs` to an `Element` constructor now does the right thing:

```
from ll.xist.ns import html, xml
node = html.html(
   html.head(),
   xml.Attrs(lang="de"),
   lang="en",
)
```

- Creating skeleton implementations of XIST namespaces is no longer done via XML conversion (i.e. the namespace module `ll.xist.ns.xndl`), but through the new module `ll.xist.xnd`. The script `dtdxsc.py` will automatically generate `sims` information.
- *`ll.xist.xsc.CharRef`* now inherits from *`ll.xist.xsc.Text`* too, so you don't have to special case `CharRef`s any more. When publishing, `CharRef`s will be handled like `Text` nodes.
- *`ll.xist.ns.meta.contenttype`* now has an attribute `mimetype` (defaulting to `"text/html"`) for specifying the MIME type.
- `ll.xist.ns.htmlspecials.caps` has been removed.
- Registering elements in namespace classes has been rewritten to use a cache now.
- Pretty printing has been changed: Whitespace will only be added now if there are no text nodes in element content.
- Two mailing lists are now available: One for discussion about XIST and one for XIST announcements.

6.16.224 Changes in 2.4.1 (released 2004-01-05)

- Changed the xmlname of `ll.xist.ns.jsp.directive_page` back again (it's `directive.page` only for the XML form, which we don't use anyway.)
- Drop the default value for `ll.xist.ns.jsp.directive_page.Attrs.language`, as this attribute can only be used once.
- If an `ll.xist.xsc.Prefixes` object has a prefix mapping for a namespace it will return this prefix too, if asked for a prefix for a subclass of this namespace.

6.16.225 Changes in 2.4 (released 2004-01-02)

- The class `ll.xist.parsers.Handler` has been renamed to `Parser` and has been made reusable, i.e. it is possible to instantiate a parser once and use it multiple times for parsing. All the classes derived from `xml.sax.xmlreader.InputSource` have been dropped and the methods for parsing strings, URLs and files have been implemented as methods of the parser. Most of the arguments that had to be passed to the various parsing functions are passed to the parser constructor now. The basic parsing functionality is implemented by parsing streams instead of `InputSource` objects.
- Similar to the changes for parsing, publishers have been changed to be reusable and most arguments to the publishing functions are available as arguments to the publisher constructor.
- Now converter contexts are no longer bound to an element class, but to the context class defined by the element class, i.e. the attribute `Context` of the argument for `Converter.__getitem__()` will be used as the dictionary key. This makes it possible to use a class and it subclasses interchangeably (as long as the base class defines its own `Context` class and the subclasses don't overwrite it).
- Added a find functor `FindTypeAllAttrs` that searches content and attributes.
- Fixed the XML name for `ll.xist.ns.jsp.directive_page`.
- All character references in `ll.xist.ns.ihtml` that exist in `ll.xist.ns.chars` too have been removed.

6.16.226 Changes in 2.3 (released 2003-12-08)

- It's now possible to parse XML without generating location information for each node, by passing `loc=False` to the constructor of the `Handler`.
- The `HTMLParser` no longer complains about global attributes or `xmlns`.
- XIST now supports uTidylib in addition to mxTidy. uTidylib is found it is preferred over mxTidy.
- It's possible now to pass arguments to tidy simple by passing an argument dictionary for the `tidy` argument in the parsing functions.
- The methods `parsed()` and `checkvalid()` have been separated.
- *`ll.xist.ns.htmlspecials.pixel`* and *`ll.xist.ns.htmlspecials.autopixel`* now check whether their `color` attribute is ok.
- The base URL is now set correctly when parsing from an URL even if the original URL does a redirect. (This requires *`ll.url`* version 0.11.3).
- Namespace handling has been rewritten again, to be more standards compliant: Now there is no prefixes for entities and processing instructions any longer. Prefix mappings can be created much simpler, and they no longer contain any namespace stack for parsing, as this is now done by the parser itself. `xsc.NamespaceAttrMixIn` is gone too.
- The processing instructions `exec_` and `eval_` from *`ll.xist.ns.code`* have been renamed to `pyexec` and `pyeval` and `import_` has been removed.
- `CharRefs` from *`ll.xist.ns.html`* have been moved to a new module named `ll.xist.ns.chars`.

- The method names `beginPublication()`, `endPublication()` and `doPublication()` have been lowercased.

6.16.227 Changes in 2.2 (released 2003-07-31)

- Namespace handling has been completely rewritten. Namespaces are now classes derived from `ll.xist.xsc.Namespace`. Defining element classes can be done inside or outside the namespace class. If the element classes are defined outside the namespace class, they can be moved inside the namespace with a simple attribute assignment:

```
class foo(xsc.Element):
   empty = False

class xmlns(xsc.Namespace):
   xmlname = "foo"
   xmlurl = "http://www.foo.com/ns/foo"

xmlns.foo = foo
```

- The methods `elementkeys()`, `iterelementkeys()`, `elementvalues()`, `iterelementvalues()`, `elementitems()` and `iterelementitems()` can be used for iterating through the element classes and their names. You can use the method `element()` to get an element class with a certain name:

```
>>> from ll.xist.ns import html
>>> html.element("div")
<element class ll.xist.ns.html/div at 0x824363c>
```

- For processing instructions, entities and character references similar methods are available.
- The method `update()` can be used to add many element classes to a namespace at once, simply by passing a dictionary with those classes (use `vars()` to add everything that's defined inside your module). The method `updatenew()` does the same, but copies only those attributes that don't exist in the namespace, `updateexisting()` copies only those that do exist. You can turn a namespace into a module with `makemod()`:

```
from ll.xist import xsc

class foo(xsc.Element):
   empty = False

class xmlns(xsc.Namespace):
   xmlname = "foo"
   xmlurl = "http://www.foo.com/ns/foo"
xmlns.makemod(vars())
```

- Put the above code into `foo.py` and you can do the following:

```
>>> import foo
>>> foo
<namespace foo/xmlns name=u'foo' url=u'http://www.foo.com/ns/foo' with 1␣
↪elements from 'foo.py' at 0x81bfc14>
```

- `getns()` has been dropped, so you always have to pass in a `Namespace` class where a namespace is required.
- For the `ll.xist.ns.jsp.directive_page` element automatic generation of the correct `charset` option in the `contentType` attribute is only done when there is a `contentType` attribute, as `contentType` is optional.
- The converter has a new property `node()`. `node` can't be passed to `conv()` but will be set to `self` by `conv()` automatically. This makes it possible to access the "document root" during conversion.

- *ll.xist.ns.htmlspecials.autoimg* no longer touches existing width and height attributes. This means that %-formatting of the existing attributes is no longer done.
- Added a new class *ll.xist.ns.htmlspecials.autopixel* that works like *ll.xist.ns.htmlspecials.pixel* but inherits the size for the image specified via the `src` attribute.
- `Frag` and `Element` now support extended slices.
- `Frag` and `Element` now support the methods `extend()` and `__iadd__()`.
- For walking the tree the method `walk()` has been completely rewritten and a new method `visit()` has been added. For more info see the docstrings.
- `Node` now has two new methods `copy()` and `deepcopy()` and supports the `copy` module from the Python standard library.
- Calling `mapped()` through `conv()` has been removed. You again have to call `mapped()` directly and pass a node and a converter.
- The HTML handling of the `HTMLParser` has been improved (it now uses code from `xml.sax.drivers2.drv_sgmlop_html` (which is part of PyXML.
- The core functionality found in the script `dtd2xsc.py` has been moved to a class method `ll.xist.ns.xndl.fromdtd()` in the `ll.xist.ns.xndl` namespace.
- `ll.xist.parsers.ExpatParser` is now a real subclass instead of an alias for `xml.sax.expatreader.ExpatParser` It reports unknown entity references to the application (if loading of external entities is switched off, which is done by `ll.xist.parsers.Handler` and only outside of attributes).
- Namespaces have been added for Zope's TAL and METAL specifications.
- A namespace has been added for XSL-FO.

6.16.228 Changes in 2.1.4 (released 2003-06-13)

- Remove the checks for attributes in attributes and moved the publication code for the full element into a separate method. This allows JSP tag library namespaces to simply overwrite `publish()` to publish the element even inside attributes. (This is the same fix as in release 1.5.10).

6.16.229 Changes in 2.1.3 (released 2003-05-07)

- The methods `sorted()`, `reversed()` and `shuffled()` have been rewritten so they no longer use `sys.maxint`. This change fixes those methods for 64 bit platforms (reported by Giles Frances Hall)

6.16.230 Changes in 2.1.2 (released 2003-02-27)

- `ll.xist.ns.struts_config11.plug_in` now allows content (as the DTD states). (This is the same fix as in release 1.5.8.)

6.16.231 Changes in 2.1.1 (released 2003-02-11)

- Added a few elements and attributes to *ll.xist.ns.doc*: `username`, which is used for the name of a user account, `xref`, which is used for internal cross references and the attribute `id` for `section`, which specifies the target for an `xref`.

6.16.232 Changes in 2.1 (released 2002-12-09)

- Added a new namespace module `ll.xist.ns.xndl` that contains the "XIST namespace definition language", i.e. elements that describe an XIST namespace and can be used by various scripts to generate skeleton namespace modules. The first of these script is the DTD to namespace converter `dtd2xsc.py`.
- Added a new namespace module `ll.xist.ns.tld` that contains the definition for Java Server Pages Tag Library descriptors and a script `tld2xsc.py` that uses this namespace to generate namespace modules from `tld` files.
- `Attr` now supports the method `filtered()`. This is used by `without()` now. The arguments for `without()` have changed, because handling global attributes was too "magic". A new method `with()` has been added, with does the opposite of `without()`, i.e. it removes all attributes that are not specified as parameters.
- The Python name of each `Node` subclass is now available as the class attribute `pyname`.
- To continue the great renaming `withSep()` has been renamed to `withsep()`.
- The namespace name for the *`ll.xist.ns.struts_html`* module has been fixed.
- The argument `defaultEncoding` for the various parsing functions has been renamed to `encoding`.

6.16.233 Changes in 2.0.8 (released 2002-11-20)

- `ll.xist.ns.doc.getDoc()` has been renamed to `getdoc()`.
- The CSS parser was dropping the `%` from percentage values. This has been fixed.

Changes in 2.0.7 (released 2002-11-12)

- `xsc.Element.__nonzero__()` can no longer fall back to `xsc.Frag.__nonzero__()`. (this is the same fix as in 1.5.7).

6.16.234 Changes in 2.0.6 (released 2002-11-11)

- Performance optimizations.

6.16.235 Changes in 2.0.5 (released 2002-11-11)

- Fixed a bug in `ll.xist.ns.specials.autoimg`: Attributes were not converted before the size check was done (this is the same fix as in 1.5.5).

6.16.236 Changes in 2.0.4 (released 2002-11-08)

- Fixed a regression bug in `ll.xist.ns.jsp.directive` and several documentation issues.

6.16.237 Changes in 2.0.3 (released 2002-10-30)

- Fixed a few bugs in `HTMLParser`.
- Added DocBook conversion for several elements in *`ll.xist.ns.doc`*.
- Now the `__init__.py` file for the `ll` package is included.

6.16.238 Changes in 2.0.2 (released 2002-10-21)

- Fixed a bug in `Frag.__rmul__()` (by reusing `__mul__()`).
- Fixed a bug with the backwards compatible prefix mapping: Defining element classes in `exec` processing instructions didn't work, because the prefixes object used for parsing wouldn't be updated when the namespace object is defined inside the processing instruction. Now using the default for the `prefixes` argument in calls to the parsing functions uses one global shared `Prefixes` instances where all the namespaces that are newly defined will be registered too.

6.16.239 Changes in 2.0.1 (released 2002-10-17)

- Fixed `xscmake.py` by removing the prefix handling. `OldPrefixes` will always be used for parsing now.

6.16.240 Changes in 2.0 (released 2002-10-16)

- XIST now requires at least Python 2.2.1.
- Attribute handling has been largely rewritten. Instead of a class attribute `attrHandlers`, the attributes are now defined through a nested class named `Attrs` inside the element. This class must be derived from `ll.xist.Element.Attrs` (or one of its subclasses if you want to inherit attributes from this class). Defining attributes is done through classes nested inside this attributes class and derived from any of the known attribute classes (like `TextAttr`, `URLAttr` etc.). The class name will be the attribute name (and can be overwritten with a class attribute `xmlname`. This makes it possible to have docstrings for attributes. Furthermore it's possible to define an attribute default value via the class attribute `default`, allowed values for the attribute via `values`, which is a list of allowed values, and whether the attribute is required or not via `required`.
- XIST now has real namespace support. The new class `ll.xist.xsc.Prefixes` can be used to define a mapping between prefixes and namespace names. This can be used for parsing and publishing. Namespace support is even available for entities and processing instruction.
- Global attributes are supported now. Namespace modules for the `xml` and `xlink` namespaces have been added (and `ll.xist.xsc.XML` was moved to *`ll.xist.ns.xml`*).
- A new namespace module for SVG 1.0 has been added: *`ll.xist.ns.svg`*.
- The HTML specific parts of *`ll.xist.ns.specials`* have been split off into a separate module *`ll.xist.ns.htmlspecials`*.
- Comparison of attributes with strings has been removed. You have to use `__unicode__()` or `__str__()` now before comparing.
- The `HTMLParser` now removes unknown attributes instead of complaining.
- There is a new parser class `BadEntityParser`, which is a SAX2 parser that recognizes the character entities defined in HTML and tries to pass on unknown or malformed entities to the handler literally.
- To give all nodes a chance to do something after they have been parsed (e.g. to prepend the base URL for `URLAttr` nodes), the parser now calls the method `parsed()` immediately after node creation. This is used for the new class `StyleAttr`, which uses the `CSSTokenizer`, to prepend the base URL to all URLs found in a style attribute.
- The pixel images have been moved to the directory `px` to make image URLs shorter.

6.16.241 Changes in 1.6.1 (released 2003-08-25)

- Updated to work with newer versions of `ll.ansistyle`.
- Updated the namespaces *ll.xist.ns.struts_html* and `ll.xist.ns.struts_config11` to the state of Struts 1.1 final.

6.16.242 Changes in 1.6 (released 2003-07-02)

- Removed the default value for the `className` attribute in `ll.xist.ns.struts_config11.action`.
- Added an attribute `type` to `ll.xist.ns.struts_config11.action_mapping`.

6.16.243 Changes in 1.5.13 (released 2003-07-01)

- Implemented `ll.xist.xsc.Namespace.__eq__()`, so that replacing a namespace in the registry really works.
- Added an attribute `target` to *ll.xist.ns.html.area*.

6.16.244 Changes in 1.5.12 (released 2003-06-17)

- Fixed a bug in the new *ll.xist.ns.jsp*.

6.16.245 Changes in 1.5.11 (released 2003-06-13)

- Updated *ll.xist.ns.jsp* to JSP 1.2.

6.16.246 Changes in 1.5.10 (released 2003-06-13)

- Remove the checks for attributes in attributes and moved the publication code for the full element into a separate method. This allows JSP tag library namespaces to simply overwrite `publish()` to publish the element even inside attributes.

6.16.247 Changes in 1.5.9 (released 2003-04-30)

- Reregistering a namespace now properly overwrites the old version in `xsc.namespaceRegistry`.

6.16.248 Changes in 1.5.8 (released 2003-02-27)

- `ll.xist.ns.struts_config11.plug_in` now allows content (as the DTD states).

6.16.249 Changes in 1.5.7 (released 2002-11-12)

- `xsc.Element.__nonzero__()` can no longer fall back to `xsc.Frag.__nonzero__()`.

6.16.250 Changes in 1.5.6 (released 2002-11-11)

- Performance optimizations.

6.16.251 Changes in 1.5.5 (released 2002-11-11)

- Fixed a bug in `ll.xist.ns.specials.autoimg`: Attributes were not converted before the size check was done.

6.16.252 Changes in 1.5.4 (released 2002-09-30)

- `xscmake.py` now tries to strip off a trailing `xsc` from the filename before it falls back to the extension `html` (The builtin extension mapping is still tried first).

6.16.253 Changes in 1.5.3 (released 2002-09-25)

- Added new processing instruction class `ll.xist.ns.php.expression` that generates a PHP `print` statement from its content.

6.16.254 Changes in 1.5.2 (released 2002-09-19)

- Removed the `value` magic from `ll.xist.ns.form.checkbox` as this conflicted with dynamic `value` values.

6.16.255 Changes in 1.5.1 (released 2002-09-17)

- Comparison of attributes with strings has been removed. You have to use `__unicode__()` or `__str__()` instead.
- The `HTMLParser` now removes unknown attributes instead of complaining.
- There is a new parser class `BadEntityParser`, which is a SAX2 parser that recognizes the character entities defined in HTML and tries to pass on unknown or malformed entities to the handler literally.
- To give all nodes a chance to do something after they have been parsed (e.g. to prepend the base URL for `URLAttr` nodes), the parser now calls the method `parsed()` immediately after node creation. This is used for the new class `StyleAttr`, which uses the `CSSTokenizer`, to prepend the base url to all urls found in a style attribute.
- The `HTMLParser` now removes unknown attributes instead of complaining.
- There is a new parser class `BadEntityParser`, which is a SAX2 parser that recognizes the character entities defined in HTML and tries to pass on unknown or malformed entities to the handler literally.
- To give all nodes a chance to do something after they have been parsed (e.g. to prepend the base URL for `URLAttr` nodes), the parser now calls the method `parsed()` immediately after node creation. This is used for the new class `StyleAttr`, which uses the `CSSTokenizer`, to prepend to base URL to all URLs found in a style attribute.

6.16.256 Changes in 1.4.3 (released 2002-04-29)

- New namespace module `xist.ns.struts_config11` allows to parse and modify Struts configuration files conforming to the Struts 1.1 DTD.

6.16.257 Changes in 1.4.2 (released 2002-03-22)

- Updated `xscmake.py` to be compatible with the new `url` module.
- `xist.ns.jsp.directive_page` now automatically sets the `contentType` on publishing.

6.16.258 Changes in 1.4.1 (released 2002-03-21)

- Removed `TidyURLInputSource`. Now it's possible to pass a `tidy` flag to the remaining functions `parseString()`, `parseFile()` and `parseURL()` to specify whether the source should be tidied.
- To prevent an element from being registered in a `Namespace` the class attribute `register` can be used now. This makes it possible to have a name for the element even when it's not registered.
- `xist.ns.form` elements now have all the attributes that the corresponding elements from `xist.ns.html` have.
- Removed the old `xist.url` from the Windows distribution.

6.16.259 Changes in 1.4 (released 2002-03-18)

- Reimplemented URL handling again. Now the new global module `url` is used for that.

6.16.260 Changes in 1.3.1 (released 2002-03-14)

- Added a method `pretty()` to `Node` for generating a pretty printable version of the node.
- `xsc.Node.name` no longer is a class method, but a class attribute, that will be set at class instantiation time by the meta class.

6.16.261 Changes in 1.3 (released 2002-02-12)

- Ported to Python 2.2. `Node` is now derived from `object`, `Frag` from `list` and there's a new class `Attrs` which is derived from `dict` for the attribute mappings. All presenters have been adapted to work with `Attrs`. In addition to the usual dictionary methods and operators `Attrs` has a method `without()` that returns a copy of the `Attrs` instance with some specified attributes removed.
- All the node classes now have a new method `walk()` that generates all nodes in the tree using the new generator feature of Python 2.2.
- Also a new method `walkPath()` has been added that works the same as `walk()` but yields the complete path to each node as a list.
- Added a class `block` to `xist.ns.jsp`. The content of the `block` instance will simply be enclosed in a `{}` block. `xist.ns.php` got such a class too.
- Added a new module `xist.ns.ihtml` for i-mode HTML.
- Added new modules `xist.ns.css` and `xist.ns.cssspecials` for generating CSS.
- Now the various attributes of the `Converter` object are collected in a `ConverterState` object and it's possible to push and pop those states, i.e. it's now easy to temporarily modify a converter object during a `convert()` call and revert back to a previous state afterwards.

- `parseURL()` and `parseTidyURL()` now have an additional parameter `headers` which is a list of string pairs specifying additional headers to be passed in with the request.
- `parseString()` has an additional parameter `systemId` which will be the system id of the `InputSource`.
- The distribution now includes the makefile and the XML source files so now the distribution can rebuild ifself.
- Various other small bugfixes and enhancements.

6.16.262 Changes in 1.2.5 (released 2001-12-03)

- Added a new element `contentscripttype` to `xist.ns.meta` that generates a `<meta http-equiv="Content-Script-Type" ...>` element.
- `xist.ns.doc.explain()` now generates anchor elements for the class, function and method description, so now the links on the XIST webpages work.
- Docstrings and documentation has been reworked. Now `xist.ns.doc.pyref` no longer implies a font change. Use the classes `xist.ns.doc.module`, `xist.ns.doc.class`, `xist.ns.doc.method`, `xist.ns.doc.function` and `xist.ns.doc.arg` to mark up your Python identifiers.
- Added the attributes `type` and `key` to `xist.ns.struts_config.data_source`.

6.16.263 Changes in 1.2.4 (released 2001-11-23)

- Added the deprecated attributes `start` to `xist.ns.html.ol` and `value` to `xist.ns.html.li`.

6.16.264 Changes in 1.2.3 (released 2001-11-22)

- Added missing `asPlainString()` methods to `Comment` and `DocType`.

6.16.265 Changes in 1.2.2 (released 2001-11-16)

- `xist.url.URL.fileSize()` and `xist.url.URL.imageSize()` now use the warning framework to report errors.
- There is a new presenter named `CodePresenter` that dumps the tree as Python source code.
- The filenames of the pixel images used by `xist.ns.specials.pixel` have changed. These images are now included.

6.16.266 Changes in 1.2.1 (released 2001-10-08)

- URLs that are completely dynamic will now be left in peace when parsing or publishing.

6.16.267 Changes in 1.2 (released 2001-10-03)

- `xist.ns.meta.keywords` and `xist.ns.meta.description` no longer call `asPlainString()` on their content. This makes it possible to e.g. generate the keywords via JSP:

```
>>> from xist import parsers
>>> from xist.ns import meta, jsp
>>> s = '<keywords>' + \
...     '<?jsp:expression "foo"?>' + \
...     '</keywords>'
```

(continues on next page)

(continued from previous page)

```
>>> e = parsers.parseString(s)
>>> print e.conv().asBytes()
<meta name="keywords" content="<%= "foo" %>" />
```

- When an element occurs inside an attribute during publishing, there won't be an exception raised any more. Instead the content of the element will be published. This fixes problems with abbreviation entities inside attributes.
- `xist.parsers.TidyURLInputSource` now uses the new experimental eGenix mx Extension package, which includes a Python port of tidy.
- `__repr__()` now uses the new class `presenters.PlainPresenter` which gives a little more info than the default `__repr__()`.
- URL handling has been changed again. Upto now, `URLAttr` had an additional instance attribute `base`, which was the "base" file/URL from which the attribute was parsed. Now the base URL will be directly incorporated into the URL. You can pass the base URL to all the parsing functions. Similar to that when publishing you can specify a base URL. All URLs in the tree will be output relative to this base URL. Joining URLs is now done via `__div__()` and no longer via `__add__()`. This makes it more consistent with `fileutils`. The plan is to make URLs string like immutable objects and to merge them with `fileutils.Filename`.
- `xist.ns.specials.php` has been moved to its own module (`xist.ns.php`). This module provided additional convenience processing instructions (just like `xist.ns.jsp` does).

6.16.268 Changes in 1.1.3 (released 2001-09-17)

- The global namespace registry now keeps a sequential list of all registered namespaces, which will be used by the parser when searching for names. This gives a predictable search order even without using `Namespaces` and its `pushNamespace()` method: modules imported last will be searched first.
- Processing instructions are now allowed inside attributes when publishing.
- `xist.ns.docbooklite` has been renamed to `xist.ns.doc`. It can now generate HTML and Docbook output and has improved a lot. The XIST web pages now use this for automatic documentation generation. The doc example has been removed.
- `xist.url.URL` now has a new method `info()` that returns the headers for the file/URL.
- `xist.url.URL` now has a methods `fileSize()` and `imageSize()` too.
- `xist.ns.jsp.directive_page` now has new attribute `session`.

6.16.269 Changes in 1.1.2 (released 2001-08-21)

- `__repr__()` now uses the new class `presenters.PlainPresenter` which gives a little more info than the default `__repr__()`.

6.16.270 Changes in 1.1.1 (released 2001-08-01)

- Small bugfix in `presenters.strProcInst()`.
- Fixed `xist.ns.struts_html.option` to allow content.

6.16.271 Changes in 1.1 (released 2001-07-19)

- Sequences in constructor arguments for `Frag` and `Element` are again expanded and it's again possible to pass dictionaries in an `Element` constructor to specify attributes. As sequences are always unpacked, the method `extend()` is gone. This works for `append()` and `insert()` too.
- `Node` and `Frag` implement `__mul__()` and `__rmul__()`, so you can do stuff like:

  ```
  html.br()*5
  ```

  This returns a `Frag` with five times to same node.
- Arguments for the converter constructor can be passed to `xist.xsc.Node.conv()` now, so it's possible to do stuff like this:

  ```
  from xist.ns import code
  print code.Eval("return converter.lang").conv(lang="en").asBytes()
  ```

  which will print `en`.
- The option `XHTML` for the publishers has been changed to lowercase.
- `xist.ns.html.html` will automatically generate a `lang` and `xml:lang` attribute when the converter has a language set.

6.16.272 Changes in 1.0 (released 2001-06-18)

- New module for WML 1.3.
- The publishing interface has changed internally and publishing should be faster now.
- Publishers now support a new parameter: `usePrefix`, which specifies if namespace prefixes should be output for the element names.
- Part of the implementation of the publishing stuff has been moved to C, so now you'll need a C compiler to install XIST.
- When publishing `"`, it will now only be replaced with `"` inside attributes.
- All the `asHTML()` methods now have an additional argument `converter`. This makes it possible to implement different processing modes or stages for new elements. All currently implemented elements and entities ignore this argument, but pass it on in the call to their childrens' `asHTML()` method. As the name `asHTML()` no longer makes sense, `asHTML()` has been renamed to `convert()`.
- There is now a tool `dtd2xsc.py` in the `scripts` directory that creates a skeleton XIST module from a DTD (this requires xmlproc from the PyXML package).
- New preliminary module for DocBook 4.12. (Incomplete: `convert()` methods and Unicode character entities are missing; any volunteers for implementing 375 classes?)
- New module `ruby.py` that implements the W3C Ruby draft.
- `sql.py` has been removed from XIST, but is available as a separate module.
- The parsing interface has been changed. Parsing is now done with the functions `parseFile()`, `parseString()`, `parseURL()` and `parseTidyURL()` in the module `parsers`. It's now possible to specify which parser should be used for parsing by passing a SAX2 parser instance to any of these functions. XIST now includes a rudimentary SAX2 driver for `sgmlop` and a rudimentary HTML parser that emits SAX2 events.
- The python-quotes example has been updated to work with expat.
- Added a new example: media.
- All abbreviation entities have been moved to a new module `abbr.py`.
- All the modules that provide new elements and entitites have been moved to a subpackage `ns`.

- `Frag` and `Element` now have new methods `sorted()`, `reversed()`, `filtered()` and `shuffled()` that return sorted, reversed, filtered and shuffled versions of the `Frag`/`Element` object.
- New namespace modules `ns/jsp.py` and `ns/struts_html.py` have been added that allow you to use JSP and Struts tags with XIST.
- A new method `asText()` was added, that returns the node as a formatted plain ASCII text (this requires that w3m is installed.)
- `make.py` has been renamed to `xscmake.py` and moved to the `scripts` directory, it will be installed as a callable script with `python setup.py install_scripts`.
- `xscmake.py` has a new option `--files`/`-f`. The argument is a file containing a list of filenames (one name per line) that should be converted.
- `xscmake.py` has a new option `--parser`/`-r` for specifying which parser to use. Allowed values are `sgmlop` and `expat`.
- `xscmake.py` has a new option `--namespace`/`-n` that can be used for appending `Namespace` objects to the `Namespaces` object used by `xscmake.py`:

  ```
  xscmake.py -n html -n spam eggs.xsc
  ```

  With this call the parser will find element classes from the module with the prefix name `spam` before those from `html` and those before anything else.
- `xist.url.URL` no longer has an attribute `ext`. `file` and `ext` are merged.
- The special treatment of sequences as constructor arguments to `Frag` and `Element` has been removed, so XIST will no longer remove one level of nesting. If you still want that, use a * argument.
- `Frag` and `Element` now have a new method `mapped()`, that recursively maps the nodes through a function. This is like `convert()` but via an external function.
- Attribute handling has been improved thanks to a suggestion by Hartmut Goebel: `Element.__getitem__()` now always works as long as the attribute name is legal. If the attribute is not set, an empty attribute will be returned. All empty attributes will be considered as being not set and so `hasAttr()` returns `0` for them, and `publish()` doesn't publish them. This simplifies several very common cases:
  - Copying an attribute from one element to another works regardless of whether the attribute is set or not;
  - Testing for an attributes presence can now be done much simpler: `if element["attrname"]` instead of `if element.hasAttr("attrname")` (which still works, and should be a little faster);
  - When you construct an XIST tree and the presence or absence of an attribute is tied to a condition, you can construct the attribute in advance and use it afterwards in the tree construction:

    ```
    if condition:
       align = "right"
    else:
       align = None
    node = html.div("spam", align=align)
    ```

    So, when the `condition` is false, the node will not have the attribute `align` set.
- `xist.ns.cond.If` (and `xist.ns.cond.ElIf`) can now be used to test for attributes of the converter. I.e. it's possible to write the following XML:

  ```
  <if lang="en">Title
  <elif lang="de">Überschrift
  </if>
  ```

- URL handling has be completely changed and is much, much simpler now. There are no more path markers. To specify an URL that is relative to the current directory use the scheme `root` (e.g. `root:main.css`).

6.16.273 Changes in 0.4.7 (released 2000-11-24)

- Fixed a bug in the entity handling.
- Added a few deprecated elements and attributes to the `html` module.
- Improved the publishing of attributes. Now all attribute values will be published. For boolean attributes no value will be published for `XHTML==0` and the attribute name will be used for `XHTML==1` or `XHTML==2`.
- `Element.compact()` now works (better) ;).
- Incorparated many bug fixes from Hartmut Goebel.
- Implemented `xsc.Element.copyDefaultAttrs()`, which copies unset attributes over from a dictionary (simplifies implementing `specials.plaintable` and `specials.plainbody`).
- `providers.Provider.pushNamespace()` now handles multiple arguments which may be `Namespace` objects or modules (in which case, `module.namespace` will be pushed).
- `providers.Providers.popNamespace()` can now pop multiple namespaces at once.
- `providers.TidyURIProvider` now uses `os.popen3()` for piping the file through tidy, so now there will be no more temporary files. The call to tidy now includes options that hopefully make the output more suited to XIST.
- Incorparated a new `url.py` by Hartmut Goebel, that fixes many problem (e.g. optimizing `http://server/foo/bar/../../baz.gif` now works.)
- `make.py` includes a new option `--path` for adding directories to `sys.path`.

6.16.274 Changes in 0.4.6 (released 2000-11-03)

- Now uses `sgmlop.XMLParser` instead of `sgmlop.SGMLParser`, so case is preserved.
- Fixed another regression from the URL to string conversion change.

6.16.275 Changes in 0.4.5 (released 2000-11-01)

- Converting URLs to nodes is now done in `ToNode()`, so `URL` objects can be used everywhere.
- Fixed a few bugs in `Text._strtext()` and `URLAttr._str()`.

6.16.276 Changes in 0.4.4 (releases 10/27/2000)

- Now testing if characters can be encoded with the specified encoding is done directy. This means, that escaping unencodable characters now works even with exotic encodings (tested with JapaneseCodecs 1.0.1.
- The `URLAttr` constructor now can handle a single parameter of the type `URL`.
- The URL to string conversion function have changed: `URL.asString()` returns the URL with path markers, `URL.asPlainString()` returns the URL without path markers.
- Added the `i18n` attribute to the `font` element.
- Fixed the clashes between the class names for the elements and entities `sub` and `sup` in `html.py`.
- Several small enhancements and bug fixes contributed by Hartmut Goebel.

6.16.277 Changes in 0.4.3 (released 2000-10-19)

- Now processing instruction classes are registered in the same way as elements and entities are.
- The leaf nodes (`Text`, `Comment`, `ProcInst`) are now considered immutable. This means that their `asHTML()` method can simply return `self`, because now those nodes can be shared between trees. Functionality for manipulation the objects is provided by a mixin class very similar to `UserString`. All this results in a speedup of about 10% for the python-quotes example.
- Small optimizations in the `asHTML()` methods of `Element` and `Frag` optimized away many calls to `append()`, `extend()` and `ToNode()` and result in a speedup of about 30% for the python-quotes example. One consequence of this is that `Null` objects will no longer be ignored.

6.16.278 Changes in 0.4.2 (released 2000-09-24)

- New elements `keywords` and `description` in `meta.py`.
- Fixed a bug in `Namespace.register()`, now setting `name=None` to prevent an element from being registered works again.

6.16.279 Changes in 0.4.1 (released 2000-09-21)

- A new module named `meta.py` has been created, that simplifies generating meta tags.
- Various small bugfixes.

6.16.280 Changes in 0.4 (released 2000-09-19)

- XIST now requires at least Python 2.0b1.
- A new bugfixed version of the sgmlop source is available from the FTP site.
- XIST now completely supports Unicode. For output any encoding known to Python can be used, so now you can output your HTML in ASCII, Latin-1, UTF-8, UTF-16, ...
- All publishers have been updated to support Unicode. The publishing interface has been streamlined (`encoding` and `XHTML` parameters are now attributes of the publisher).
- `asString()` will now always return a Unicode string. If you want a byte string use `asBytes()` instead, where the encoding can be specified as an argument.
- There an additional publisher class `FilePublisher`, which can be used for publishing to a file (or anything else that has a `write()` and a `writelines()` method, and is supported by the stream writer available through `codecs.lookup()`).
- Element and attribute names are no longer converted to lowercase. If you have an attribute name which clashes with a Python keyword (e.g. `class`) append an underscore (_), which will be removed before accessing the attribute. This is the "official" Python method for handling these cases.
- Elements and entities are no longer registered one by one. Now you can build `Namespace` objects which are used for searching and there are `pushNamespace()` and `popNamespace()` functions in `XSC.xsc`. For more info, see the source.
- Image size calculation has been removed from `html.img` and `html.input`. Use `specials.autoimg` and `specials.autoinput` for that.
- `__getitem__()`, `__setitem__()` and `__delitem()` of `Frag` and `Element` now accepts a list as an argument. The method will be applied recursively, i.e. `e[[0, 1, "foo", 2]` is the same as `e[0][1]["foo"][2]`.
- The deprecated module `db.py` no longer exists. Useful functions and elements from `db.py` have been moved to `sql.py` and `form.py` respectively.

- When using `xsc.make()` the encoding and XHTML parameters to use can now be specified on the command line (e.g. `--encoding utf-8 --xhtml 2`)
- Handling of multiline `<?xsc-eval?>` and `<?xsc-exec?>` has been enhanced, although XIST will not be able to guess the correct indentation in all cases. As a workarround simply add a Python comment to the beginning. So the following won't work:

```
<?xsc-exec
   for i in xrange(10):
      do(i)
?>
```

But this will:

```
<?xsc-exec
   #
   for i in xrange(10):
      do(i)
?>
```

- Make functionality has been moved to `make.py`, as certain modules can't be used as the main script, because reimporting them in processing instructions won't work. Now you can simply call:

```
make.py --import xist.html --import spam eggs.xsc
```

- There is a new module `cond.py`, that contains elements that can be used for conditionals:

```
<?xsc-exec a=42?>
<if cond="a==21">
   <b>foo</b>
<elif cond="a==42"/>
   <i>bar</i>
<else/>
   baz
</if>
```

6.16.281 Changes in 0.3.9 (released 2000-08-10)

- sgmlop will now be found either via `import sgmlop` or via `from xml.parsers import sgmlop`.

6.16.282 Changes in 0.3.8 (released 2000-07-14)

- Fixed a bug in `URLAttr.publish()`, which prevented `URLAttr` from working at all.

6.16.283 Changes in 0.3.7 (released 2000-07-06)

- Fixed a bug in `html.img` and `html.input`. Now image size calculation works again.
- Changes in 0.3.6 (released 2000-07-04)
- Fixed a bug in `Node._matches()`, which resulted in a non working `find()`.

6.16.284 Changes in 0.3.5 (released 2000-07-02)

- The documentation example has been enhanced. Now documenting methods works.
- When the member `elementname`: in the element class is set before calling `registerElement()`, this element name will be used for the element. This allows custom names even when using `registerAllElements()`.
- Comparison of scheme and server in URLs is done case insensitive (as **RFC 2068** requires.)
- Image size calculation is now done in `asString()` and not in `asHTML()`. This allows to write faster code. Old method:

```
e = html.div(html.img(...),gurk.hurz()).asHTML().asString()
```

  New method:

```
e = html.div(html.img(...),gurk.hurz().asHTML()).asString()
```

- Image size calculation is now done for `<input type="image">`. The `size` attribute is set to the image width.
- Manipulating the path in an URL is now done via the usual `__setitem__()`/`__getitem__()` stuff, which keeps the path in a consistent state:

```
>>> from xist.URL import URL
>>> u = URL("/foo/*/../bar/baz.gif")
>>> del u[1]
>>> u
URL(scheme='server', path=['bar'], file='baz', ext='gif')
```

- `findNodes()` (which has been shortened to `find()`) has an additional argument `test`, which can be a test function that will be called when the node passes all other tests.
- `asString()` no longer generates a string directly, but uses the new method `publish()`, which has an additional argument `publisher`, to which the strings to be output are passed.

6.16.285 Changes in 0.3.4 (released 2000-05-31)

- Location information is now copied over in `clone()`, `asHTML()` and `compact()` where appropriate, so you know even in the HTML tree where something came from.
- `xsc.repransi` can now have three values:

0
: coloring is off

1
: coloring is on for a dark background

2
: coloring is on for a light background

- All `repransi` variables are now arrays with two strings, the first for dark, the second for light.

6.16.286 Changes in 0.3.3 (released 2000-05-30)

- The workaround for the trailing CDATA bug in sgmlop has been removed, so now you'll need a newer version of sgmlop (included in PyXML 0.5.5.1).

6.16.287 Changes before 0.3.3

- These changes predate written history.

6.17 Migration

This document describes how to update your code to each version of XIST. Only incompatible changes are listed here. For a list of all changes see *History*.

6.17.1 Migrating to version 5.73

Changes to `ll.pysql`

- Some PySQL commands have been renamed: `resetsequence` to `reset_sequence`, `checkerrors` to `check_errors`, `raiseexceptions` to `raise_exceptions`, `pushraiseexceptions` to `push_raise_exceptions` and `popraiseexceptions` to `pop_raise_exceptions`.
- The argument `raiseexceptions` to various PySQL commands has been renamed to `raise_exceptions`.

6.17.2 Migrating to version 5.67

Changes to `ll.orasql`

- The method `ll.orasql.Connection.getobject()` has been renamed to *`ll.orasql.Connection.object_named()`*.

6.17.3 Migrating to version 5.66

Changes to `ll.color`

- The color method `abslum()` has been renamed to `abslight()` and the method `rellum()` has been renamed to `rellight()`.

Changes to UL4

- UL4 functions and methods have been updated to use positional-only or keyword-only arguments to match the signature of the corresponding Python function/method.
- Subclasses of *`ll.ul4c.AST`* have been renamed so that their name matches the name of the corresponding class in the Java implementation. (for example `ll.ul4c.Add` has been renamed to *`ll.ul4c.AddAST`*)
- The UL4 function `type()` now returns type objects instead of simple strings. To get the name of the type use the type objects `__name__` attribute, i.e. replace `type(foo)` with `type(foo).__name__`.
- Naming of attributes that are used to implement UL4 functionality is more uniform now. This affects the following attributes: The methods `ul4_getattr()`, `ul4_setattr()` and `ul4_hasattr()` for implementing object attribute access from UL4; the methods `ul4_call()`, `ul4_render()` and , `ul4_renders()` for making objects callable or renderable from UL4; the class attribute `ul4_attrs` for exposing a number of

readonly attributes to UL4; the attributes `ul4_context` that is used for marking a callable as needing the context as an argument in the call.

- Support for using custom tag delimiters for UL4 templates tag has been removed, i.e. now the tag delimiters will always be <? and ?>.
- The color method `abslum()` has been renamed to `abslight()` and the method `rellum()` has been renamed to `rellight()`.

Changes to `ll.misc` and `ll.ul4on`

- Some function now use positional-only arguments:
  - `ll.misc.item(iterable, index, /, default=None)`
  - `ll.misc.first(iterable, /, default=None)`
  - `ll.misc.last(iterable, /, default=None)`
  - `ll.misc.count(iterable, /)`
  - `ll.misc.isfirst(iterable, /)`
  - `ll.misc.islast(iterable, /)`
  - `ll.misc.isfirstlast(iterable, /)`
  - `ll.misc.monthdelta.__init__(self, months=0, /)`
  - `ll.ul4on.dumps(obj, /, indent)`
  - `ll.ul4on.dump(obj, /, stream, indent)`
  - `ll.ul4on.load(stream, /, registry=None)`
  - `ll.ul4on.loads(dump, /, registry=None)`
  - `ll.ul4on.loadclob(clob, /, bufsize=1024*1024, registry=None)`

  Update your calls accordingly.

6.17.4 Migrating to version 5.58

Changes to `ll.sisyphus`

- The method `healthcheck()` should no longer be implemented. Instead the option `--maxhealthcheckage` or the class/instance attribute `maxhealthcheckage` can be used to configure the maximum allowed age.
- The method `failed()` is no longer supported. If you need its functionality wrap the content of your `execute()` method in a `try` block and put the content of you `failed()` method into an `except` block (and reraise the exception at the end of the `except` block).
- The filenames for log files can no longer be changed via options or job attributes, instead one of the following methods must be overwritten:
  - `basedir()`
  - `logfilename()`
  - `currentloglinkname()`
  - `lastsuccessfulloglinkname()`
  - `lastfailedloglinkname()`
  - `lastinterruptedloglinkname()`
  - `lasttimeoutloglinkname()`

- `healthfilename()`
- `emailfilename()`

Those methods must return an absolute path as a `pathlib.Path` object.

6.17.5 Migrating to version 5.57

Changes to `ll.ul4on`

`ll.ul4on.Encoder` and *`ll.ul4on.Decoder`* now expect the stream to be passed in the call to `dump()` and `load()` instead of in the constructor. I.e. change:

```
Encoder(stream).dump(obj)
```

to:

```
Encoder().dump(obj, stream)
```

and:

```
obj = Decoder(stream).load()
```

to:

```
obj = Decoder().load(stream)
```

The parameter name for the UL4ON function `fromul4on()` has changed from `string` to `dump`.

6.17.6 Migrating to version 5.56

Changes to `ll.orasql`

`ll.orasql.Comment` has been renamed to *`ll.orasql.ColumnComment`*.

6.17.7 Migrating to version 5.52

Changes to `ll.orasql`

The method `getobject()` for *`ll.orasql.Synonym`* has been renamed to `object()`.

6.17.8 Migrating to version 5.45

Changes to UL4

The UL4 AST node attribute `line` and `col` have been renamed to `startline` and `startcol`.

6.17.9 Migrating to version 5.44

Changes to `ll.pysql`

The PySQL command `compileall` has been removed. This same effect can simply be achieved by calling `utl_recomp.recomp_parallel()` or `dbms_utility.compile_schema()`.

The PySQL terminator comment (`-- @@@`) can now no longer be specified via a command line option.

The `connectstring` argument for the `pysql` script is now optional, so it has to be specified via the optional argument `-d/--database`.

The `--commit` argument (with the options `record`, `once` and `never`) has been replaced with a flag option `--rollback`. Automatically committing after every record is no longer available. However manual committing is available via the *`commit`* command.

PySQL no longer supports multiple active database connections via the `connectname` key. When using literal SQL this couldn't be used anyway, so it has been dropped. If you really need this feature you can implement a workaround in literal Python blocks.

6.17.10 Migrating to version 5.42

Changes to `ll.sisyphus`

Returning `None` from *`ll.sisyphus.Job.execute()`* now has a special meaning: Delete the log file. If this isn't wanted a string (e.g. `"done"`) should be returned.

6.17.11 Migrating to version 5.40

Changes to `ll.orasql`

The class *`ll.orasql.SchemaObject`* has been renamed to *`ll.orasql.OwnedSchemaObject`*. (*`ll.orasql.SchemaObject`* now is for objects that don't have an owner (i.e. *`ll.orasql.User`* and *`ll.orasql.JobClass`*).)

6.17.12 Migrating to version 5.37

Changes to UL4

Exception chaining has changed, so the exception you get from calling/rendering an UL4 template now is the original exception. To get to the information about the location in the UL4 source code you have to iterate through the `__cause__` chain.

The interal structure of UL4 templates has been simplified, but that should only concern you if you've worked with the UL4 AST itself. Basically `Tag` objects are gone now, and instead each `AST` node has attributes `template` (referencing the outermost template) and `pos` (being the position of the nodes source code). This also means that blocks no longer have an `endtag` attribute.

6.17.13 Migrating to version 5.36

Changes to `ll.orasql`

As `cx_Oracle` provides its own `Object` `orasql.Object` has been renamed to `orasql.SchemaObject`.

6.17.14 Migrating to version 5.34

Changes to XIST

The class `ll.xist.ns.html.script.Attrs.async` has beend renamed to `async_`, because `async` is a keyword in Python 3.7.

6.17.15 Migrating to version 5.32

Changes to `ll.orasql`

The default value for the `owner` parameter in various *`ll.orasql`* methods has changed from `ALL` to `None` (i.e. it now returns the objects from the current schema instead of all schemas).

The changed methods are: `Connection.tables()`, `Connection.sequences()`, `Connection.fks()`, `Connection.privileges()`, `Connection.objects()`, `Object.names()`, `Object.objects()` (and all subclasses of `Object`) and `Privilege.objects()`.

To get the old behaviour back, simply pass `owner=orasql.ALL` to those methods.

6.17.16 Migrating to version 5.28

Changes to UL4

- UL4 now longer tries a disguise objects as dictionaries. I.e. for objects with an `ul4attrs` class attribute the methods `items`, `keys`, `values` and `get` are no longer synthesized. This also means that `len`, `list`, item access and containment tests no longer work on objects. However iterating over the attribute names of an object can now be done with the new function `dir`. To get, set and test attributes, the new functions `getattr`, `setattr` and `hasattr` can be used.

6.17.17 Migrating to version 5.22

Changes to `pysql`

- The values for the option `-v/--verbose` has changed: `-v1` now is `-vdot`, `-v2` is `-vtype` and `-v3` is `-vfull`.

6.17.18 Migrating to version 5.21

Changes to `ll.color`

- Colors can no longer be added. This was done with the formula:

```
0.5*(c1.r+c2.r), 0.5*(c1.g+c2.g), 0.5*(c1.b+c2.b), 255-(255-c1.a)*(255-c2.a)/255.
↪)
```

Changes to `ll.orasql`

- The method `ll.orasql.ForeignKey.pk()` has been renamed to `refconstraint()`.

6.17.19 Migrating to version 5.20

Changes to `ul4`

- The variables passed to UL4 templates in **`rul4`** have been moved into a `globals` objects. The following changes have to be made to the template source:
  - change `oracle.connect(...)` to `globals.oracle(...)`;
  - change `mysql.connect(...)` to `globals.mysql(...)`;
  - change `sqlite.connect(...)` to `globals.sqlite(...)`;
  - change `system.execute(...)` to `globals.system(...)`;
  - change `load(...)` to `globals.load(...)`;
  - change `error(...)` to `globals.error(...)`;
  - change `foo` to `globals.vars.foo` for a variable `foo` defined via `rul4 -D`.

6.17.20 Migrating to version 5.18

Changes to `ul4`

- The UL4 exception `ll.ul4c.Error` has been renamed to `LocationError`.
- The UL4 function `type` now returns the Python class name for date, color, template exception objects.

6.17.21 Migrating to version 5.17

Changes to `rul4`

The function `import` has been split into `load` for loading the content of a file and `compile` for compiling a string, so:

```
<?code template = import("/home/user/template/foo.ul4")?>
```

has to be replaced with:

```
<?code template = compile(load("/home/user/template/foo.ul4"))?>
```

6.17.22 Migrating to version 5.16

Changes to `orasql`

Some methods in `orasql` have been renamed: Iterating methods no longer have `iter` in their name (e.g. `itertables()` is now simply called `tables()`). The `ddl` part of some method names has been changed to `sql` (e.g. `createddl()` is now called `createsql()`).

6.17.23 Migrating to version 5.15

Changes to PySQL

- The function `load` has been replaced by two functions `loadstr` for loading strings and `loadbytes` for loading bytes, i.e. replace:

  ```
  load('foo.txt', 'utf-8', 'replace')
  ```

  with:

  ```
  loadstr('foo.txt', 'utf-8', 'replace')
  ```

  and:

  ```
  load('foo.png')
  ```

  with:

  ```
  loadbytes('foo.png')
  ```

- PySQL no longer supports the `-- !!!` command terminator. Use the `raiseexceptions` command instead to specify error handling.

6.17.24 Migrating to version 5.14

Changes to UL4

- The boolean parameter `keepws` for `ul4c.Template` has been renamed to `whitespace` and requires a string value now. Pass `whitespace="keep"` for the old `keepws=True` and `whitespace="strip"` for the old `keepws=False`.
- The `rul4` option `--keepws` has been renamed to `--whitespace` and defaults to `smart` now. So instead of the old `--keepws=1` pass `--whitespace=keep` and for `--keepws=0` pass `--whitespace=strip`.
- Rendering an UL4 template from inside a UL4 template is now again done via the `<?render?>` tag. So inside a template you have to replace the code:

  ```
  <?code template.render(foo, bar)?>
  ```

  with:

  ```
  <?render template(foo, bar)?>
  ```

- Closures in UL4 templates no longer see the state of the variables at the time when the local template was defined, but at the time when it is called. This is similar to most other languages that support closures.

  To emulate the old behaviour pass the variables you want to "freeze" to a locally defined template and define the original template there.

Changes to pysql

- SQL commands must be terminated with a `-- @@@` (or `-- !!!`) comment line now, i.e. now the comment *after* the command determines whether exceptions will be ignored, instead of the comment before the command.

6.17.25 Migrating to version 5.13

Changes to UL4

- Locally defined UL4 templates no longer see themselves among the variables of the parent template.

Changes to sisyphus

- The option `setproctitle` for sisyphus jobs has been renamed to `proctitle`.
- The default for the name parameter in `tasks()` for sisyphus jobs has changed from `str` to `None`, i.e. it defaults to unnamed tasks now.

6.17.26 Migrating to version 5.12

Changes to ul4on

- The UL4ON serialization format has been reimplemented to be more human-readable and robust. The new format is incompatible to the old. If you update your XIST installation to 5.12 you should update the corresponding UL4ON versions for Java/Javascript too.

6.17.27 Migrating to version 5.10

Changes to misc

- The functions `misc.gzip()` and `misc.gunzip()` have been removed as Python 3.2 has the functions `gzip.compress()` and `gzip.uncompress()`, which do the same.

6.17.28 Migrating to version 5.9

Changes to db2ul4

- The script `db2ul4` has been renamed to `rul4`.

Changes to ll.url

- The argument `pattern` of the URL methods `listdir()`, `files()`, `dirs()`, `walk()`, `walkfiles()` and `walkdirs()` has been renamed to `include`.
- The method `walk()` has been renamed to `walkall()`.

6.17.29 Migrating to version 5.7

Changes to `ll.oradd`

- The `file` command has been renamed to `scp`.

Changes to `ll.orasql`

- The methods *`ll.orasql.Record.keys()`* and `ll.orasql.Record.values()` return iterators now. `ll.orasql.Record.iterkeys()` and `ll.orasql.Record.itervalues()` have been removed.

6.17.30 Migrating to version 5.6

Changes to `ll.oradd`

- Support for `"keys"` and `"sqls"` has been removed from `ll.oradd`. So

```
{
   "type": "procedure",
   "name": "procname",
   "args": {
      "proc_id": "p_10",
      "proc_date": "sysdate",
      "keys": {"proc_id": "int"},
      "sqls": ["proc_date"]
   }
}
```

  has to be replaced with

```
{
   "type": "procedure",
   "name": "procname",
   "args": {
      "proc_id": var("p_10", int),
      "proc_date": sql("sysdate")
   }
}
```

- UL4ON dumps are no longer supported by `ll.oradd`. They must be reencoded as Python `repr` outputs, which can be done with code that looks like this:

```
import sys

from ll import ul4on

while True:
   try:
      print(repr(ul4on.load(sys.stdin)))
   except EOFError:
      break
```

6.17.31 Migrating to version 5.4

Changes to `ll.url`

- The `remotepython` parameter for `ssh` URLs has been renamed to `python`.

6.17.32 Migrating to version 5.2

Changes to `sisyphus`

- The method `prefix()` for `sisyphus` jobs has been replaced with `task()` which does something similar.

Changes to UL4

- The names of methods that should be callable for custom objects in UL4 templates must be added to the `ul4attrs` attributes.

Changes to `oradd`

- Committing the transactions in `oradd` can now be done after each record with the new option `--commit`. `--rollback` has been removed, so you have to replace `--rollback=1` with `--commit=never`.

Changes to `misc`

- The default argument for the functions `misc.first()` and `misc.last()` now defaults to `None`. I.e. for empty iterators the default value will always be returned instead of generating an exception. To simulate the old behaviour use a unique guard object as the default.
- Renamed the attributes `scriptname` and `shortscriptname` of the `misc.sysinfo` object to `script_name` and `short_script_name`.

6.17.33 Migrating to version 5.1

Changes to `db2ul4`

- The `query` method for database connections has changed: Instead of a query and a parameter dictionary, you have to pass in positional arguments that alternate between fragments of the SQL query and parameters. I.e.:

  ```
  db.query("select * from table where x=:x and y=:y", x=23, y=42)
  ```

  becomes:

  ```
  db.query("select * from table where x=", 23, " and y=", 42)
  ```

  This makes `db2ul4` independent from the parameter format of the database driver.

6.17.34 Migrating to version 5.0

Changes to XIST

- Accessing attributes via `__getattr__()`, `__setattr__()` and `__delattr__()` now requires the XML name of the attribute instead of the Python name. If you only have the Python name, you can convert it to the XML name with the method `Attrs._pyname2xmlname()`.
- For all methods that existed in Python/XML pairs (e.g. `withnames()` and `withnames_xml()` in `xsc.Attrs` or `elementclass()` and `elementclass_xml()` in `xsc.Pool` etc.) there is only one version now: A method without the `_xml` suffix in the name, that accepts the XML version of the name.
- Validation is now off by default, to turn it on pass `validate=True` to `parse.tree()` or `parse.itertree()` for parsing, or to the publisher object or the `bytes()`, `iterbytes()`, `string()` or `iterstring()` methods for publishing.

6.17.35 Migrating to version 4.10

Changes to UL4

- The UL4 tag `<?render?>` have been removed. To update your code replace `<?render r.render()?>` with `<?exe r.render()?>`.
- The UL4 functions `vars` and `get` have been removed.
- The automatic UL4 variable `stack` has been removed too.

6.17.36 Migrating to version 4.7

Changes to UL4

- Compiling a UL4 template to a Java `CompiledTemplate` is no longer supported (i.e. `template.javasource(interpreted=False)` no longer works. Use `template.javasource()` instead (which creates Java sourcecode for an `InterpretedTemplate`).

6.17.37 Migrating to version 4.6

Changes to `ll.xist`

- The `walk()` method has been changed to return a `Cursor` object instead of the path, so you have to replace:

```
for path in doc.walk(...):
   # use path
```

  with:

```
for cursor in doc.walk(...):
   # use cursor.path
```

- Furthermore walk filters have been removed. Determining whether an XIST tree is traversed top down or bottom up can instead by specified via distinct parameters to the `walk()` method. Replace:

```
for path in doc.walk((xfind.entercontent, xfind.enterattrs, True)):
   ...
```

  with:

```
for cursor in doc.walk(entercontent=True, enterattrs=True,␣
↪startelementnode=False, endelementnode=True):
   ...
```

If you want to enter an element only when a condition is true, you can do that by modifying the appropriate cursor attribute inside your loop:

```
for cursor in doc.walk(entercontent=True, enterattrs=True):
   if isinstance(cursor.node, html.script, html.textarea):
      cursor.entercontent = False
   ...
```

- *ll.xist.parse.itertree()* now returns `Cursor` objects too, instead of path lists.
- Slicing XIST elements now returns a sliced element, instead of a slice from the content `Frag`:

```
>>> from ll.xist.ns import html
>>> html.ul(html.li(i) for i in range(5))[1:3].string()
'<ul><li>1</li><li>2</li></ul>'
```

To get a slice from the content simply use:

```
>>> html.ul(html.li(i) for i in range(5)).content[1:3].string()
'<li>1</li><li>2</li>'
```

6.17.38 Migrating to version 4.4

Changes to the required Python version

Python 3.3 is required now.

6.17.39 Migrating to version 4.2

Changes to `ll.ul4c`

- The UL4 method `join` no longer calls `str` on the items in the argument list. Replace `sep.join(iterable)` with `sep.join(str(i) for i in iterable)` when you have an argument list that contains non-strings.

6.17.40 Migrating to version 4.1

Changes to `ll.make`

- The support for Growl notifications in *ll.make* on the Mac has been replaced by support for Mountain Lions Notification Center.

  The option has been renamed from `--growl` to `--notify`.

  For this to work you need to have terminal-notifier installed in its standard location (`/Applications/terminal-notifier.app`).

6.17.41 Migrating to version 4.0

Changes to the required Python version

Python 3.2 is required now.

Changes to UL4

- Date constants in UL4 have changed again. They are now written like this: `@(2012-04-12)` or `@(2012-04-12T12:34:56)`.
- The function `json` has been renamed to `asjson`.
- The `<?render?>` tag in UL4 now looks like a method call instead of a function call. I.e. `<?render t(a=17, b=23)?>` has changed to `<?render t.render(a=17, b=23)?>`.

Changes to scripts

- The scripts `oracreate`, `oradrop`, `oradelete`, `oradiff`, `oramerge`, `oragrant`, `orafind` and `uhpp` no longer have an `-e`/`--encoding` option. They always use Pythons output encoding.
- The options `-i`/`--inputencoding` and `-o`/`--outputencoding` of the script `db2ul4` have been replaced with an option `-e`/`--encoding` for the encoding of the template files. For printing the result Pythons output encoding is used.
- The options `--inputencoding`/`--inputerrors` and `--outputencoding`/`--outputerrors` of *`ll.sisyphus.Job`* have been replaced with option `--encoding`/`--errors` for the encoding of the log files.

6.17.42 Migrating to version 3.25

Changes to XIST

- The `compact()` method has been renamed to `compacted()` to avoid collisions with the `compact` attribute in HTML elements.

6.17.43 Migrating to version 3.24

Changes to `ll.xist.ns.ul4`

- `ll.xist.ns.ul4.attr_if` is now an *`ll.xist.xsc.AttrElement`* subclass. Change your code from:

  ```
  html.div(id=(ul4.attr_if("foo"), "bar"))
  ```

  to:

  ```
  html.div(id=ul4.attr_if("bar", cond="foo"))
  ```

- `ll.xist.ns.ul4.attr_ifnn` has been removed. Replace it with the equivalent `attr_if` call.

6.17.44 Migrating to version 3.23

Changes to `ll.ul4c`

- The module global functions `ll.ul4c.compile()`, `ll.ul4c.load()` and `ll.ul4c.loads()` have been removed. Instead of them the `Template` constructor and the class methods `load()` and `loads()` can be used.

6.17.45 Migrating to version 3.20

Changes to `ll.orasql`

- The `schema` argument used by various methods in *`ll.orasql`* has been replaced by a `owner` argument that can be `None` (for the current user), the constant `ALL` for all users (which uses the `DBA_*` variant of various meta data views if possible or the `ALL_*` variants otherwise) and a specific user name.

6.17.46 Migrating to version 3.19

Changes to `ll.orasql`

- *`ll.orasql`* now requires cx_Oracle 5.1 (i.e. `UNICODE` mode is no longer used).
- If the `readlobs` option is false for *`ll.orasql`* cursors, the CLOBs/BLOBs returned will be wrapped into something that behaves like a Python file. The original `LOB` object is available as the `value` attribute of the returned wrapper object:

```
db = orasql.connect("user/pwd@db")
c = db.cursor()
c.execute("select theclob from thetable")
row = c.fetchone()
print row[0].value.read()
```

6.17.47 Migrating to version 3.18

Changes to `db2ul4`

- The variables available in UL4 templates used by `db2ul4` have changed. Instead of a `connect` object, there are now three objects for each supported database (i.e. `oracle`, `sqlite` and `mysql`). To update your template replace:

```
connect["oracle:user/pwd@db"]
```

with:

```
oracle["user/pwd@db"]
```

Changes to scripts

- The script `doc2txt` now reads from `stdin` and writes to `stdout` instead of requiring file names on the command line.

6.17.48 Migrating to version 3.17

Changes to `ll.misc`

- `ll.misc.javastring()` has been renamed to *`ll.misc.javaexpr()`*.
- The UL4 method `format` is now a function instead.

6.17.49 Migrating to version 3.16

Changes to `ll.misc`

- `ll.misc.flag()` is gone. If the function is still required, here is the source:

```
def flag(value):
   if value in ("1", "true", "yes"):
      return True
   elif value in ("0", "false", "no"):
      return False
   raise ValueError("unknown flag value")
```

6.17.50 Migrating to version 3.15

Changes to `ll.xist.ns.jsp`

- `ll.xist.ns.jsp.javastring()` has been move to *`ll.misc`*.

6.17.51 Migrating to version 3.14

Changes to `ll.ul4c`

- Date constants now need a `@` as a prefix. I.e. chance `2010-11-03T` to `@2010-11-03T` etc.
- The `function` argument for `ul4c.Template.pythonsource()` is gone. The output will always be a full function.

6.17.52 Migrating to version 3.12

Changes to `ll.sisyphus`

- The maximum allowed runtime for jobs is now a hard limit. Previously a running job that exceeded the maximum allowed runtime would only be killed when the next job was started. Now the job will kill itself immediately after `maxtime` seconds. This means you *might* have to adjust your `maxtime` setting.
- The default location of log files has changed again. Now `~/ll.sisyphus/` is used as the base directory instead of `~/ll.sisyphus/log/`.

6.17.53 Migrating to version 3.11

Changes to `ll.sisyphus`

- The method `logLoop()` is gone. Replace:

```
self.logLoop("done")
```

with:

```
return "done"
```

- The method `logProgress()` is gone. Replace:

```
self.logProgress("parsing XML file")
```

with:

```
self.log("parsing XML file")
```

You might also add tags to the logging call via:

```
self.log.xml("parsing XML")
```

(This adds the tag `"xml"` to the log line.)

- The method `logError()` is gone. Replace:

```
self.logError("Can't parse XML file")
```

with:

```
self.log.error("Can't parse XML file")
```

If the object passed to `self.log` is an exception, the logging call will add the `exc` tag automatically.

- `sisyphus.Job` no longer has a constructor. Configuration is now done via class attributes. Replace:

```
class TransmogrifyStuff(sisyphus.Job):
   def __init__(self, connectstring):
      sisyphus.Job.__init__(self, 30, "ACME_TransmogrifyStuff", raiseerrors=True)
```

with:

```
class TransmogrifyStuff(sisyphus.Job):
   projectname = "ACME.MyProject"
   jobname = "TransmogrifyStuff"
   maxtime = 30
```

- The default location of run/log files has changed. Now `~/ll.sisyphus/log` is used for log files and `~/ll.sisyphus/run` is used for run files.

6.17.54 Migrating to version 3.10

Changes to the required Python version

Python 2.7 is required now.

Changes to `ll.make`

- *`ll.make`* uses `argparse` now.
- `ll.make.Project.optionparser()` has been renamed to `argparser()` and returns a `argparse.ArgumentParser` object now.
- `ll.make.Project.parseoptions()` has been renamed to `parseargs()` and returns a `argparse.Namespace` object now.

Changes to `ll.daemon`

- *`ll.daemon`* uses `argparse` now. `ll.daemon.Daemon.optionparser()` has been renamed to `argparser()`.

6.17.55 Migrating to version 3.9

Changes to `ll.xist.ns.html`

- *`ll.xist.ns.html.html`* will no longer change the `lang` and `xml:lang` attributes. This functionality has been moved to the new element *`ll.xist.ns.htmlspecials.html`*. Furthermore this new element will not change an attribute if this attribute has already been set.

  So if you need the functionality replace any use of *`ll.xist.ns.html.html`* with *`ll.xist.ns.htmlspecials.html`*.
- *`ll.xist.ns.html.title`* no longer does any manipulation of its content.

  If you needed this functionality, you can copy it from the old `title` element and put it into your own element class.

6.17.56 Migrating to version 3.8

Changes to parsing

- The parsing infrastructure has been completely rewritten to be more modular and to support iterative parsing (similar to ElementTree). Now parsing XML is done in a pipeline approach.

  Previously parsing a string looked like this:

```
>>> from ll.xist import xsc, parsers
>>> from ll.xist.ns import html
>>> source = "<a href='http://www.python.org/'>Python</a>"
>>> doc = parsers.parsestring(source, pool=xsc.Pool(html))
```

  Now this is done the following way:

```
>>> from ll.xist import xsc, parse
>>> from ll.xist.ns import html
>>> source = "<a href='http://www.python.org/'>Python</a>"
>>> doc = parse.tree(
```

(continues on next page)

(continued from previous page)

```
...     parse.String(source)
...     parse.Expat()
...     parse.NS(html)
...     parse.Node(pool=xsc.Pool(html))
... )
```

For more info see the module *ll.xist.parse*.

- Something that no longer works is parsing XML where elements from different namespaces use the same namespace prefix. You will either have to rewrite your XML or implement a new class for the parsing pipeline that handles namespaces prefixes *and* instantiating XIST classes (i.e. a combination of what *ll.xist.parse.NS* and *ll.xist.parse.Node* do).
- The module `ll.xist.parsers` has been renamed to `parse`.
- The module `ll.xist.presenters` has been renamed to `present`.
- The classes `ll.xist.converters.Converter` and `ll.xist.publishers.Publisher` have been moved to *ll.xist.xsc*. The modules `ll.xist.converters` and `ll.xist.publishers` no longer exist.

Changes to XISTs walk filters

- The walk methods `walknode()` and `walkpath()` have been renamed to `walknodes()` and `walkpaths()`. The class `WalkFilter` has been moved to *ll.xist.xfind*.

Changes to `ll.url`

- `ll.url.Path` has been simplified: Path segments are strings instead of tuples. If you need the path parameters (i.e. part after `;` in a path segment) you have to split the segment yourself.
- *ll.url.URL.import_()* is gone. As a replacement `misc.module()` can be used, i.e. replace:

```
>>> from ll import url
>>> u = url.File("foo.py")
>>> m = u.import_(mode="always")
```

with:

```
>>> from ll import url, misc
>>> u = url.File("foo.py")
>>> m = misc.module(u.openread().read(), u.local())
```

However, note that *ll.url.URL.import_()* has been reintroduced in 3.8.1 based on `misc.import()`. This means that the mode argument is no longer supported.

- ssh URLs now required to standalone `execnet` package. The `ssh_config` parameter for ssh URLs is gone.

Changes to `ll.make`

- The two classes `ll.make.PoolAction` and `ll.make.XISTPoolAction` have been dropped. To update your code, replace:

```
make.XISTPoolAction(html)
```

with:

```
make.ObjectAction(xsc.Pool).call(html)
```

- The class `XISTParseAction` has been removed. This action can be replaced by a combination of `ObjectAction`, `CallAction` and `CallAttrAction` using the new parsing infrastructure.

Other changes

- `ll.xist.ns.specials.z` has been moved to the *ll.xist.ns.doc* module.

6.17.57 Migrating to version 3.7

Changes to the make module

- The division operator for actions is no longer implemented, so instead of:

```
t1 = make.FileAction(key=url.URL("file:foo.txt"))
t2 = t1 /
     make.DecodeAction("iso-8859-1") /
     make.EncodeAction("utf-8") /
     make.FileAction(key=url.URL("bar.txt"))
```

  you now have to write something like the following:

```
t1 = make.FileAction("file:foo.txt")
t2 = t1.callattr("decode", "iso-8859-1")
t2 = t2.callattr("encode", "utf-8")
t2 = make.FileAction("file:bar.txt", t2)
```

- Also the following classes have been removed from *ll.make*: `EncodeAction`, `DecodeAction`, `EvalAction`, `GZipAction`, `GUnzipAction`, `JavascriptMinifyAction`, `XISTBytesAction`, `XISTStringAction`, `JoinAction`, `UnpickleAction`, `PickleAction`, `TOXICAction`, `TOXICPrettifyAction`, `SplatAction`, `UL4CompileAction`, `UL4RenderAction`, `UL4DumpAction`, `UL4LoadAction`, `XISTTextAction` and `XISTConvertAction`. All of these actions can be executed by using `CallAction` or `CallAttrAction`.

6.17.58 Migrating to version 3.6

Changes to the color module

- The following `Color` class methods have been dropped: `fromrgba`, `fromrgba4`, `fromrgba8`, `fromint4`, `fromint8`.
- The following `Color` properties have been dropped: `r4`, `g4`, `b4`, `a4`, `r8`, `g8`, `b8`, `a8`, `r`, `g`, `b`, `a` `int4`, `int8`, `rgb4`, `rgba4`, `rgb8`, and `rgba8`. The new methods `r`, `g`, `b` and `a` return the 8 bit component values.
- The class methods `fromhsva` and `fromhlsa` have been renamed to `fromhsv` and `fromhls`.
- The property `css` has been dropped. The CSS string is returned by `__str__` now.
- Dividing colors now does a scalar division. Blending colors is now done with the modulo operator.

Removal of XPIT

- The XPIT templating language has been removed. You should replace all your XPIT templates with UL4 templates.

6.17.59 Migrating to version 3.5

Changes to UL4

- The UL4 function `csvescape` has been renamed to `csv`.

Changes to the color module

- *`ll.color.Color`* has been rewritten to create immutable objects with the components being 8 bit values (i.e. 0-255) instead of floating point values between 0 and 1.

6.17.60 Migrating to version 3.4

Changes to the make module

- `ll.make.CallMethAction` has been renamed to `CallAttrAction`.
- `ll.make.XISTPublishAction` has been renamed to `XISTBytesAction`.

Changes to UL4

- The templates available to the `<?render?>` tag are no longer passed as a separate argument to the render methods, but can be part of the normal variables.

Changes to XIST

- Building trees with `with` blocks has changed slightly. Unchanged code will lead to the following exception:

```
File "/usr/local/lib/python2.5/site-packages/ll/xist/xsc.py", line 1285, in __
↪enter__
   threadlocalnodehandler.handler.enter(self)
AttributeError: 'NoneType' object has no attribute 'enter'
```

To fix this, change your code from:

```
with html.html() as node:
   with html.head():
      +html.title("Foo")
   with html.body():
      +html.p("The foo page!")
```

to:

```
with xsc.build():
   with html.html() as node:
      with html.head():
         +html.title("Foo")
      with html.body():
         +html.p("The foo page!")
```

(i.e. wrap the outermost `with` block in another `with xsc.build()` block.)

6.17.61 Migrating to version 3.3

Changes to the make module

- `ll.make.ImportAction` has been dropped as now the module object can be used directly (e.g. as the input for an `XISTPoolAction` object).
- The constructor of most action classes now accept the input action as a parameter again. This means that you might have to change the calls. Usually it's safest to use keyword arguments. I.e. change:

```
make.FileAction(url.File("foo.txt"))
```

to:

```
make.FileAction(key=url.File("foo.txt"))
```

- The `targetroot` parameter for `ll.make.XISTConvertAction.__init__()` has been renamed to `root`.

Changes to TOXIC

- TOXIC has been split into a compiler and an XIST namespace module. Instead of calling the function `ll.xist.ns.toxic.xml2ora()` you now have to use `ll.toxicc.compile()`. (However using TOXIC with *`ll.make`* hasn't changed).

Changes to XIST

- The default parser for XIST is expat now. To switch back to sgmlop simply pass an `SGMLOPParser` object to the parsing functions:

```
>>> from ll.xist import parsers
>>> node = parsers.parsestring("<a>", parser=parsers.SGMLOPParser())
```

6.17.62 Migrating to version 3.2.6

Changes to escaping

The functions `ll.xist.helpers.escapetext` and `ll.xist.helpers.escapeattr` have been merged into `ll.misc.xmlescape` and all the characters <, >, &, " and ' are escaped now.

6.17.63 Migrating to version 3.1

Changes to URL handling

URLs containing processing instructions will no longer be transformed in any way. If you need the old behaviour you can wrap the initial part of the attribute value into a `specials.url` PI.

6.17.64 Migrating to version 3.0

Changes to tree traversal

You can no longer apply xfind expression directly to nodes, so instead of:

```
for node in root//html.p:
   print node
```

you have to write:

```
for node in root.walknode(html.p):
   print node
```

If you want the search anchored at the root node, you can do the following:

```
for node in root.walknode(root/html.p):
   print node
```

This will yield `html.p` elements only if they are immediate children of the `root` node.

Passing a callable to the `walk()` method now creates a *`ll.xist.xfind.CallableSelector`*. If you want the old tree traversal logic back, you have to put your code into the `filterpath()` method of a `WalkFilter` object.

Many of the XFind operators have been renamed (and all have been rewritten). See the `xfind` documentation for more info.

The death of namespace modules

It's no longer possible to turn modules into namespaces. Element classes belong to a namespace (in the XML sense) simply if their `xmlns` attribute have the same value. So a module definition like this:

```
from ll.xist import xsc

class foo(xsc.Element):
   def convert(self, converter):
      return xsc.Text("foo")

class xmlns(xsc.Namespace):
   xmlname = "foo"
   xmlurl = "http://xmlns.example.org/foo"
xmlns.makemod(vars())
```

has to be changed into this:

```
from ll.xist import xsc

class foo(xsc.Element):
   xmlns = "http://xmlns.example.org/foo"

   def convert(self, converter):
      return xsc.Text("foo")
```

Renamed doc classes

Many classes in the *ll.xist.ns.doc* module have been renamed. The following names have changed:

- `function` to `func`;
- `method` to `meth`;
- `module` to `mod`;
- `property` to `prop`;
- `title` to `h`;
- `par` to `p`;
- `olist` to `ol`;
- `ulist` to `ul`;
- `dlist` to `dl`;
- `item` to `li` or `dd` (depending on whether it's inside an `ol`, `ul` or `dl`);
- `term` to `dt`;
- `link` to `a`.

6.17.65 Migrating to version 2.15

Changes to plain text conversion

The node method `asText()` has been moved to the `html` namespace, so you have to replace:

```
print node.asText()
```

with:

```
from ll.xist.ns import html
print html.astext(node)
```

Changes to `htmlspecials.pixel`

If you've been using the `color` attribute for `htmlspecials.pixel`, you have to add a `#` in from of the value, as it is a CSS color value now. (And if've you've been using `color` and a CSS padding of a different color: This will no longer work).

6.17.66 Migrating to version 2.14

Changes to presenters

Presenters work differently now. Instead of:

```
print node.asrepr(presenters.CodePresenter)
```

simply do the following:

```
print presenters.CodePresenter(node)
```

6.17.67 Migrating to version 2.13

Changes to `ll.xist.xsc`

`xsc.Namespace.tokenize()` no longer has an `encoding` argument, but operates on a unicode string directly. You can either use the result of a `asString()` call or decode the result of an `asBytes()` call yourself.

6.17.68 Migrating to version 2.11

Changes to `ll.xist.xsc`

The function `ToNode()` has been renamed to `tonode()`.

`ll.xist.Context` no longer subclasses `list`. If you need a stack for your context, simply add the list as an attribute of the context object.

Code rearrangements

The iterator stuff from *`ll.xist.xfind`* has been moved to the `ll` package/module, i.e. you have to use `ll.first()` instead of `ll.xist.xfind.first()`.

Changes to the `walk()` method

The `walk()` method has changed again. There are no inmodes and outmodes any longer. Instead input and output are `Cursor` objects. If you're using your own `walk()` filters, you have to update them. For different output modes you can use the methods `walknode()`, `walkpath()` or `walkindex()` instead of using the cursor yielded by `walk()`.

The node methods `find()` and `findfirst()` have been removed. Use `xsc.Frag(node.walk(...))` or `node.walk(...)[0]` instead.

Changes to publishing

Publishing has changed: If you've used the method `repr()` before to get a string representation of an XML tree, you have to use `asrepr()` instead now (`repr()` is a generator which will produce the string in pieces).

Changes to the `xfind` module

The functions `item()`, `first()`, `last()`, `count()` and `iterone()` as well as the class `Iterator` have been moved to the `ll` module.

6.17.69 Migrating to version 2.10

Changes to publishing

Publishing has been changed from using a stream API to using a iterator API. If you've been using `Publisher.write()` or `Publisher.writetext()` (in your own `publish()` methods) you must update your code by replacing `publisher.write(foo)` with `yield publisher.encode(foo)` and `publisher.writetext(foo)` with `yield publisher.encodetext(foo)`.

Changes to the test suite

The test suite now uses py.test, so if you want to run it you'll need py.test.

Changes to `ll.xist.ns.code`

The code in a *`ll.xist.ns.code.pyexec`* object is no longer executed at construction time, but at conversion time. So if you relied on this fact (e.g. to make a namespace available for parsing of the rest of the XML file) you will have to change your code.

Removed namespaces

The namespace modules `ll.xist.ns.css` and `ll.xist.ns.cssspecials` have been removed.

6.17.70 Migrating to version 2.9

Changes to exceptions

All exception classes have been moved from `ll.xist.errors` to *`ll.xist.xsc`*.

Changes to XML name handling

The class attribute `xmlname` no longer gets replaced with a tuple containing both the Python and the XML name. If you want to get the Python name, use `foo.__class__.__name__`.

Changes to the methods `walk()`, `find()` and `findfirst()`

The argument `filtermode` has been renamed to `inmode` and (for `walk()`) `walkmode` has been renamed to `outmode`.

6.17.71 Migrating to version 2.8

Changes to display hooks

The way XIST uses `sys.displayhook()` has been enhanced. To make use of this, you might want to update your Python startup script. For more info see the installation instructions.

Changes to the `xmlns` attribute

Each element (or entity, or processing instruction) class had an attribute `xmlns` that references the namespace module. This attribute has been renamed to `__ns__`.

Other minor changes

`ll.xist.ns.specials.x` has been renamed to *`ll.xist.ns.specials.ignore`*.

`ll.xist.xfind.item` no longer handles slices. If you've used that functionality, you may now use slices on XFind operators, and materialize the result, i.e. replace `xfind.slice(foo, 1, -1)` with `list(foo[1:-1])`, if `foo` is an XFind operator. Otherwise you can use `list(foo)[1:-1]`.

6.17.72 Migrating to version 2.7

Changes to `ll.xist.xfind`

The functions `xfind.first()` and `xfind.last()` now use `xfind.item()`, so they will raise an `IndexError` when no default value is passed. To get the old behaviour, simply pass `None` as the default.

6.17.73 Migrating to version 2.6

Changes to the publishing API

The top level publishing method in the publisher has been renamed from `dopublication()` to `publish()`. If you're using the publishing API directly (instead of the node methods `asBytes()` and `write()`), you'll have to update your code.

The method that writes a unicode object to the output stream has been renamed from `publish()` to `write()`. This is only relevant when you've overwritten the `publish()` method in your own node class (e.g. in JSP tag library directives or similar stuff, or for special nodes that publish some text literally).

Changes to the presentation API

The presentation API has been changed too: The top level presentation method in the presenter has been renamed from `dopresentation()` to `present()`. This is only relevant if you've written your own presenter, or are using the presentation API directly (instead of the node method `repr()`).

Parsing HTML

Parsing HTML is now done via libxml2's HTML parser, instead of using µTidyLib of mxTidy. You can no longer pass arguments to tidy. Only the boolean values of the `tidy` argument will be used. There are no other visible changes to the API but the result of parsing might have changed.

Removed APIs and scripts

The script `xscmake.py` has been removed.

The `visit()` method has been removed.

`ll.xist.xsc.FindOld()` has been removed.

`ll.xist.ns.xml.header` has been renamed to *`ll.xist.ns.xml.declaration`*.

6.17.74 Migrating to version 2.5

Changes to content model

The boolean class attribute `empty` for element classes has been replaced by an object `model`. `empty` is still supported, but issues a `PendingDeprecationWarning`. If you don't want to specify a proper content model for your own elements you can replace `empty = False` with `model = True` (which is a shortcut for `model = sims.Any()`) and `empty = True` with `model = False` (which is a shortcut for `model = sims.Empty()`).

6.17.75 Migrating to version 2.4

Changes to parsing

Parsing has changed internally, but the module level parsing functions in `ll.xist.parsers` are still available (and will create a parser on the fly), but a few arguments have changed:

handler
: This argument is no longer available, if you need a special handler, you have to subclass `ll.xist.parsers.Parser` and call its parsing methods.

parser
: This argument has been renamed to `saxparser` and is *not* a SAX2 parser instance any longer, but a callable that will create a SAX2 parser.

sysid
: `sysid` is now available for all parsing functions not just `parseString()`.

Changes to converter contexts

`ll.xist.converters.Converter.__getitem__()` now doesn't use the key passed in, but `key.Context` as the real dictionary key. This has the following consequences:

- If you want a unique context for your own element class, you *must* implement a new `Context` class (otherwise you'd get `ll.xist.xsc.Element.Context`):

```
class Foo(xsc.Element):
   empty = False

   class Context(xsc.Element.Context):
      def __init__(self):
         xsc.Element.Context.__init__(self)
         ...
```

- Subclasses that don't overwrite `Context` (as well as instances of those classes) can be passed to `ll.xist.converters.Converter.__getitem__()` and the unique base class context object will be returned.

Changed namespaces

The character reference classes from `ll.xist.ns.ihtml` that are duplicates of those in `ll.xist.ns.chars` have been removed, so you have to use `ll.xist.ns.chars` for those characters in addition to `ll.xist.ns.ihtml`

6.17.76 Migrating to version 2.3

Changes in namespace handling

Namespace handling has changed. There are no entity or processing instruction prefixes any longer and creating a proper `Prefixes` object has been simplified. For example:

```
prefixes = xsc.Prefixes()
prefixes.addElementPrefixMapping(None, html)
prefixes.addElementPrefixMapping("svg", svg)
```

can be simplified to:

```
prefixes = xsc.Prefixes(html, svg=svg)
```

The three arguments `elementmode`, `entitymode` and `procinstmode` for the publishing methods have been combined into `prefixmode`, which is used for elements only.

Changed namespaces

The character reference classes from *`ll.xist.ns.html`* have been moved to a separate namespace `ll.xist.ns.chars`.

The processing instructions `eval_` and `exec_` from the *`ll.xist.ns.code`* module have been renamed to `pyeval` and `pyexec`.

Changed method names

The method names `beginPublication()`, `endPublication()` and `doPublication()` have been lowercased.

6.17.77 Migrating to version 2.2

Attribute methods

The `Element` methods for accessing attributes have been deprecated. So instead of `node.hasattr("attr")`, you should use:

```
"attr" in node.attrs
```

The same holds for checking whether an attribute is allowed. You can use the following code:

```
"attr" in node.Attrs
```

or:

```
"attr" in NodeClass.Attrs
```

or:

```
NodeClass.isallowed("attr")
```

Many `Attrs` methods have gained an additional parameter `xml`, which specifies whether an attribute name should be treated as the XML or the Python name of the attribute. Make sure that you're not mixing up your arguments in the function call. The safest method for this is using keyword arguments, e.g.:

```
node.attr.get("attr", default=42)
```

JSP directive page element

A `contentType` attribute is no longer generated for the `ll.xist.ns.jsp.directive_page`. You have to explicitly use an attribute `contentType="text/html"` to get a `contentType` attribute in the resulting JSP. The `charset` option is generated automatically from the encoding specified in the publisher.

`autoimg` changes

`ll.xist.htmlspecials.autoimg` will no longer touch existing `width` or *height*` attributes, so e.g. setting the width to twice the image size via `width="2*%(width)s"` no longer works. You have to implement your own version of `autoimg` if you need this.

`find()` changes

`find()` has been completely rewritten to use the new tree traversal filters. For backwards compatibility a filter functor `ll.xist.xsc.FindOld` exists that takes the same arguments as the old `find()` method. I.e. you can replace:

```
node.find(
   type=html.a,
   attr={"href": None},
   searchchildren=True
)
```

with:

```
node.find(
   xsc.FindOld(
      type=html.a,
      attr={"href": None},
      searchchildren=True
   ),
   skiproot=True
)
```

But one minor difference remains: when `skiproot` is set to true in the new `find()` method, the attributes of the root element will *not* be traversed. With the old method they would be traversed.

`doc` changes

`programlisting` has been renamed to `prog`.

Namespace changes

Namespaces can no longer be instantiated. Instead you have to derive a class from `Namespace`. The `xmlprefix` argument from the constructor becomes a class attribute `xmlname` and the argument `xmlname` becomes `xmlurl`.

Adding element classes to the namespace is now done with the `Namespace` classmethod `update()`. If you want the turn a namespace into a module, you can use the classmethod `makemod()` instead of `update()`, i.e. replace:

```
xmlns = xsc.Namespace("foo", "http://www.foo.com/", vars())
```

with:

```
class xmlns(xsc.Namespace):
   xmlname = "foo"
   xmlurl = "http://www.foo.com/"
xmlns.makemod(vars())
```

6.17.78 Migrating to version 2.1

The method `withSep()` has been renamed to `withsep()`.

The argument `defaultEncoding` for the various parsing functions has been renamed to `encoding`.

6.17.79 Migrating to version 2.0

Attribute handling

The biggest change is in the way attributes are defined. In older versions you had to define a class attribute `attrHandlers` that mapped attribute names to attribute classes. This created problems with "illegal" attribute names (e.g. `class` and `http-equiv` in HTML), so for them an ugly workaround was implemented. With 2.0 this is no longer neccessary. Defining attributes is done via a class `Attrs` nested inside the element class like this:

```
class foo(xsc.Element):
   class Attrs(xsc.Element.Attrs):
      class bar(xsc.TextAttr)
         "The bar attribute"
         default = "spam"
         values = ("spam", "eggs")
         required = True
      class baz(xsc.URLAttr):
         "The baz attribute"
```

Default values, set of allowed attributes values and whether the attribute is required can be defined via class attributes as shown above. You should (directly or indirecty) inherit from `xsc.Element.Attrs`, because this class implements handling of global attributes. If you want to inherit some attributes (e.g. from your base class), you can derive from the appropriate `Attrs` class. Removing an attribute you inherited can be done like this:

```
class bar(foo):
   class Attrs(foo.Attrs):
      baz = None
```

This removes the attribute `baz` inherited from `foo`.

For attribute names that are no legal Python identifiers, the same method can be used as for element classes: Define the real XML name via a class attribute. This class attribute has been renamed from `name` to `xmlname`.

This also means that you always have to use the Python name when using attributes now. The XML name will only be used for parsing and publishing.

XIST 2.0 tries to be as backwards compatible as possible: An existing `attrHandlers` attribute will be converted to an `Attrs` class on the fly (and will generate a `DeprecationWarning` when the class is created). An `Attrs` class will automatically generate an `attrHandlers` attribute, so it's possible to derive from new element classes in the old way. The only situation where this won't work, is with attributes where the Python and XML name differ, you have to use "new style" attributes there.

Namespace support

XIST supports XML namespaces now and for parsing it's possible to configure which namespaces should be available for instantiating classes from. For more info about this refer to the documentation for the class `Prefixes`.

Before 2.0 the XML name for a namespace object was pretty useless, now it can be used as the namespace name in `xmlns` attributes and it will be used for that when publishing and specifying an `elementmode` of `2` in the call to the publishing method or the constructor of the publisher.

Namespace objects should now be named `xmlns` instead of `namespace` as before.

Global attributes

Global attributes are supported now, e.g. the attributes `xml:lang` and `xml:space` can be specified in an element constructor like this:

```
from ll.xist import xsc
from ll.xist.ns import html, xml

node = html.html(
   content,
   {(xml, "lang"): "en", (xml, "space"): "preserve"},
   lang="en"
)
```

Instead of the module object (which must contain a namespace object named `xmlns`), you can also pass the namespace object itself (i.e. `xml.xmlns`) or the namespace name (i.e. `"http://www.w3.org/XML/1998/namespace"`).

Namespace changes

The classes `XML` and `XML10` have been moved from *`ll.xist.xsc`* to *`ll.xist.ns.xml`*.

All the classes in *`ll.xist.ns.specials`* that are specific to HTML generation have been moved to the new module *`ll.xist.ns.htmlspecials`*.

The module *`ll.xist.ns.html`* has been updated to the XHTML specification, so there might be some changes. The new feature for specifying attribute restrictions has been used, so e.g. you'll get warnings for missing `alt` attributes in `img` elements. These warnings are issued via the warning framework. Refer to the documentation for the `warnings` module to find out how to configure the handling of these warnings.

Miscellaneous

XIST now requires at least Python 2.2.1 because the integer constants `True` and `False` are used throughout the code wherever appropriate. These constants will become instances of the new class `bool` in Python 2.3. You might want to change your code too, to use these new constant (e.g. when setting the element class attribute `empty`).

Using mixed case method names was a bad idea, because this conflicts with Python's convention of using all lowercase names (without underscores). These method names will be fixed in the next few XIST versions. The first names that where changed were the element methods `getAttr()` and `hasAttr()`, which have been renamed to `getattr()` and `hasattr()` respectively. `getAttr()` and `hasAttr()` are still there and can be called without generating deprecation warnings, but they will start to generate warnings in the upcoming versions.

6.18 Old history

The following documents document changes or migration info for modules and packages that where merged into XIST (starting with XIST 3.2) or into the former core package (starting with XIST 2.12).

6.18.1 Changes in old packages

The following list of changes is for modules and packages that where merged into XIST (starting with XIST 3.2) or into the former core package (starting with XIST 2.12).

Changes in the new core package

Changes in 1.11.1 (released 01/11/2008)

- Added missing source file `_xml_codec_include.c` to the source distributions.

Changes in 1.11 (released 01/07/2008)

- `root:` URLs are now treated as local files for purposes of file i/o.
- *`ll.make.ModuleAction`* no longer supports modules that fiddle with `sys.modules`.
- The function *`ll.misc.tokenizepi()`* can be used to split a string according to the processing instruction tags in the string (this formerly was part of *`ll.xist.xsc`*).
- *`ll.make`* has been changed interally to store modification timestamp, which should fix the implementation of `PhonyAction` and makes it possible to use `PhonyAction` objects as inputs to other actions.
- *`ll.make`* has a new action `ImportAction`. This action doesn't take any input. It imports a module specified by name, e.g. `make.ImportAction("cStringIO")` (Note that you should not import one module via `ModuleAction` and `ImportAction` (or a normal import) as this will return two different module objects).
- Make actions don't have the input actions as a constructor parameter any longer. The input action can now only be set via `__div__()`.
- *`ll.make`* has been updated to support the actions required by XIST 3.0.
- The functions `misc.item()`, `misc.first()`, `misc.last()` and `misc.count()` have been reimplemented in C and should be faster now (especally `item()` for negative indexes).
- `misc.Namespace` is gone and has been replaced with `misc.Pool` which works similar to the pools use by XIST (in fact XISTs pool class is a subclass of `misc.Pool`).
- The module `xml_codec` has been added. It contains a complete codec for encoding and decoding XML.

Changes in 1.10.1 (released 07/20/2007)

- Fix option handling in *`ll.daemon`* (values from the optionparser where never used).

Changes in 1.10 (released 06/21/2007)

- *ll.daemon* now uses `optparse` to parse the command line options. Start options `restart` and `run` have been added.

Changes in 1.9.1 (released 04/03/2007)

- Fixed a bug in *ll.url.SshConnection*, which was missing a call to `urllib.url2pathname()`.

Changes in 1.9 (released 03/30/2007)

- *ll.url.Context* no longer relies on automatic cleanup for closing connections. Instead when a `Context` object is used in a `with` block, all connections will be closed at the end of the block. This should finally fix the problem with hanging threads at the end of the program.
- A script `ucp.py` has been added that can be used to copy stuff around:

```
$ ucp -v http://www.python.org ssh://root@www.example.net/~joe/public_html/index.
↪html -u joe -g users
```

Changes in 1.8 (released 03/12/2007)

- In calls to *ll.url.URL* methods that get forwarded to a connection it's now possible to pass keyword arguments for the connection constructor directly to the called method, i.e. you can do:

```
>>> u = url.URL("ssh://root@www.example.com/etc/passwd")
>>> u.size(identity="/root/.ssh/id_rsa")
1550
```

Changes in 1.7.5 (released 03/09/2007)

- *ll.url.Resource* now has a method `encoding()` that returns `None` (for "encoding unknown").

Changes in 1.7.4 (released 03/08/2007)

- *ll.url.SshConnection* objects now supports the `identity` parameter. This can be used to specify the filename to be used as the identity file (private key) for authentication.

Changes in 1.7.3 (released 02/22/2007)

- *ll.url.SshConnection* now has a new method `close()` which can be used to shut down the communication channel. As a `SshConnection` no longer stores a reference to the context, this means that ssh connections are shut down immediately after the end of the context in which they are stored. This avoids a problem with hanging threads.

Changes in 1.7.2 (released 02/02/2007)

- Fixed a bug in `ll.url._import()`.

Changes in 1.7.1 (released 01/24/2007)

- `ll.astyle` has been updated to the current trunk version of IPython.
- As the `new` module is deprecated, use `types` instead.

Changes in 1.7 (released 11/23/2006)

- Fixed a bug in the user switching in *`ll.daemon.Daemon`*.
- Added a new action class `GetAttrAction` to *`ll.make`*. This action gets an attribute of its input object.

Changes in 1.6.1 (released 11/22/2006)

- *`ll.make.ModuleAction`* now puts a real filename into the modules `__file__` attribute, so that source code can be displayed in stacktraces.
- `ll.astyle` has been fixed to work with the current trunk version of IPython.

Changes in 1.6 (released 11/08/2006)

- *`ll.url`* now supports ssh URLs which are files on remote hosts. This requires py.execnet. Most of the file data and metadata handling has been rewritten to support the requirements of ssh URLs.
- *`ll.make.ModeAction`* and *`ll.make.OwnerAction`* are subclasses of `ll.make.ExternalAction` now. This means they will execute even in "infoonly" mode.
- Fixed a bug in `ll.make.JoinAction.get()`.
- Remove the pid file for *`ll.sisyphus.Job()`* when a `KeyboardInterrupt` happens and we're running on Python 2.5.
- Fixed a longstanding bug in *`ll.sisyphus.Job()`* which resulted in the pid file not being written in certain situations.
- *`ll.daemon.Daemon`* now allows to switch the group too.

Changes in 1.5 (released 09/24/2006)

- `ll.make.XISTTextAction` is compatible to XIST 2.15 now.
- The functions `ll.url.Dirname()` and `ll.url.Filename()` have been removed (use *`ll.url.Dir()`* and *`ll.url.File()`* instead).
- The methods `ll.url.URL.isLocal()` and `ll.url.URL.asFilename()` have been removed (use *`ll.url.URL.islocal()`* and *`ll.url.URL.local()`* instead).

Changes in 1.4 (released 08/23/2006)

- A new module has been added: *ll.daemon* can be used on UNIX to fork a daemon running.

Changes in 1.3.2 (released 07/25/2006)

- *ll.make.ModuleAction* now normalizes line feeds, so that this action can now be used directly on Windows too.

Changes in 1.3.1 (released 07/06/2006)

- An option `showinfoonly` has been added to *ll.make.Project* (defaulting to `False`). This option determines whether actions that run in `infoonly` mode are reported or not.

Changes in 1.3 (released 06/28/2006)

- *ll.make* has been rewritten. Now there's no longer a distinction between `Targets` and `Actions`. Actions can be chained more easily and creating an action and registering it with the project are two separate steps. Actions can no longer be shared, as each action stores its own input actions (but output actions are not stored). "Ids" have been renamed to "keys" (and `DBID`/`OracleID` to `DBKey`/`OracleKey`). `ImportAction` has been renamed to `ModuleAction` and can now turn any string into a module.
- In *ll.url* modification dates for local files now include microseconds (if the OS supports it).
- A class `Queue` has been added to *ll.misc* which provides FIFO queues.
- A decorator `withdoc()` has been added to *ll.misc* that sets the docstring on the function it decorates.
- `setuptools` is now supported for installation.

Changes in 1.2 (released 12/13/2005)

- `None` is now allowed as a proper data object in *ll.make* actions.
- `ll.xpit` now supports conditionals (i.e. the new processing instruction targets `if`, `elif`, `else` and `endif`. Now there *must* be a space after the target name.

Changes in 1.1.1 (released 11/15/2005)

- Fixed a bug in *ll.make.Project.buildwithargs()*.

Changes in 1.1 (released 10/31/2005)

- `ll.make.TOXICAction` no longer takes an `encoding` argument in the constructor, but works on unicode strings directly.
- Two new actions (`DecodeAction` and `EncodeAction`) have been added to *ll.make*.

Changes in 1.0.2 (released 10/24/2005)

- Fixed a bug in `ll.make.Project.destroy()` that broke the `recreate()` method.

Changes in 1.0.1 (released 10/18/2005)

- Fixed a bug in `ll.make.Project.__contains__.()`

Changes in 1.0 (released 10/13/2005)

- This package now contains the following modules, that have been distributed as separate packages previously: `ansistyle`, `color`, `make`, `misc` (which contains the stuff from the old `ll` package), `sisyphus`, `url` and `xpit`.
- *`ll.misc.Iterator`* now has a method `get()` that will return a default value when the iterator doesn't have the appropriate item.
- In *`ll.make`* the output has been fixed: The `showactionfull` flag is checked before the `showaction` flag and target id's will always be output in this mode.

Changes in the old core package

Changes in ll-core 0.3 (released 05/24/2005)

- Functions will now no longer be turned into `staticmethod()` objects automatically when used in a `Namespace`.
- The iterator tools from *`ll.xist.xfind`* (`item()`, `first()`, `last()`, `count()` and `Iterator`) have been move here, as they are in no way specific to XIST.
- A test suite has been added.
- The wrapper function returned by `notimplemented()` will now have an attribute `__wrapped__` that points back to the original function.

Changes in ll-core 0.2.1 (released 01/21/2005)

- `__getitem__()` now raises a KeyError if the attribute doesn't exist.

Changes in ll-core 0.2 (released 01/11/2005)

- `Namespace` now has a `__getitem__()` method, so a `Namespace` class can be used in a call to the `eval()` function.

Changes in ll-core 0.1 (released 01/03/2005)

- Initial release

Changes in ll-ansistyle

Changes in ll-ansistyle 1.1 (released 06/28/2005)

- The methods `pushcolor()` and `popcolor()` have been resurrected. Without them switching to a new color and back would have to be done in a single call to `feed()`.

Changes in ll-ansistyle 1.0 (released 06/08/2005)

- `ll.ansistyle` has been completely reimplemented to use an iterator interface instead of a stream interface.
- Support for underlined and blinking text has been added.
- A py.test based test suite has been added.

Changes in ll-ansistyle 0.6.1 (released 03/22/2005)

- Added a note about the package init file to the installation documentation.

Changes in ll-ansistyle 0.6 (released 01/03/2005)

- `ansistyle` requires the core module and Python 2.4 now.

Changes in ll-ansistyle 0.5 (released 05/21/2004)

- `Text` has been derived from `list` directly, so it inherits all list methods.
- The method `getcolor()` has been dropped. The class attribute `color` is used now instead.

Changes in ll-ansistyle 0.4 (released 07/31/2003)

- The names of the methods `pushColor()`, `popColor()`, `getColor()` and `escapeChar()` have been changed to lowercase.
- `ansistyle` requires Python 2.3 now.

Changes in ll-ansistyle 0.3.1 (released 11/14/2002)

- Added source code encodings to all Python files.

Changes in ll-ansistyle 0.3 (released 08/27/2002)

- `ansistyle` has been moved to the `ll` package.

Changes in ll-ansistyle 0.2.2 (released 02/12/2002)

- Fixed a bug in `Text.insert()`.

Changes in ll-ansistyle 0.2.1 (released 04/11/2001)

- ansistyle now compiles under Windows with Visual C++. A binary distribution archive is available from the FTP directory.

Changes in ll-ansistyle 0.2 (released 04/02/2001)

- ansistyle now supports background colors. You can specify the background color via the bits 4-7 of the color, i.e. for the background color b = 0,...,7, and the foreground color f=0,...,15 the color value is `(b<<4)|f`.

Changes in ll-ansistyle 0.1.1 (released 03/21/2001)

- Fixed a minor bug in `ansistyle.Text.__repr__()`

Changes in ll-ansistyle 0.1 (released 02/18/2001)

- Initial release

Changes in ll-color

Changes in ll-color 0.3.1 (released 03/22/2005)

- Added a note about the package init file to the installation documentation.

Changes in ll-color 0.3 (released 01/21/2005)

- Two new methods (`abslum()` and `rellum()`) have been added that return a color with modified luminosity.

Changes in ll-color 0.2 (released 01/03/2005)

- `color` requires the core module and Python 2.4 now.
- Various bug fixes.

Changes in ll-color 0.1.1 (released 05/07/2004)

- Fixed a bug in the `css` property.

Changes in ll-color 0.1 (released 05/07/2004)

- Initial release.

Changes in ll-make

Changes in ll-make 1.1.2 (released 10/04/2005)

- Fixed a bug in the handling of color environment variables.

Changes in ll-make 1.1.1 (released 09/02/2005)

- Specifying colors via environment variables now works.
- It's possible to specify a default for the `show...` options via environment variables.
- `CacheAction` now drops the data in its `clear()` method.

Changes in ll-make 1.1 (released 09/01/2005)

- New action classes have been added: `PickleAction`, `UnpickleAction`, `NullAction` and `CacheAction`.
- During calls to `Target.clear()` and `Target.dirty()` the action methods with the same name are now called.

Changes in ll-make 1.0 (released 08/29/2005)

- `Target` objects may now cache the objects that they create, so it can be reused for different outputs.
- The `Action` chain has been split into four chains that will be used in different situations. Each target has an internal and external representation (e.g. the Python `str` object (the internal representation), that is the content of a file (the external representation). The read chain creates the internal representation from the external one, the write chain creates the external representation from the internal one. The convert chain converts between different internal representations. The use chain is called when external and internal representation exist and are up to date.
- The internal representation of a target is now available via the method `getdata()`.
- Importing Python modules is now done via an `ImportAction`.
- `ImportAction` and `UseModuleAction` can be used to automatically track module dependencies.
- During build operations the currently "running" project is available as `ll.make.currentproject`.
- Two new action classes are available: `SelectMainAction` and `JoinOrderedAction`, which can be used to select the input data at the start of a convert chain.

Changes in ll-make 0.26 (released 05/29/2005)

- Uses `ansistyle` 1.1 now.
- Introduced a new `Action` class named `ChainedAction` that consists of a list of other actions. Each `Target` now only has one action to update this target, but this action might be a `ChainedAction`. `Action` objects can be added (which results in a `ChainedAction`).

Changes in ll-make 0.25 (released 05/20/2005)

- `make` is compatible with XIST 2.10 now.

Changes in ll-make 0.24 (released 04/11/2005)

- `XPITAction` now works if there is no namespace available. In this case only the global namespace will be passed to the expressions.

Changes in ll-make 0.23.1 (released 03/22/2005)

- Added a note about the package init file to the installation documentation.

Changes in ll-make 0.23 (released 02/14/2005)

- Actions can now be displayed during the make process in two ways: a short name (this uses the method `desc()`) and a longer description (using the method `fulldesc()`). You can activate the full description via the command line option `-vf` and deactivate it with `-vF`. In interactive mode you can use the attribute `showactionsfull`.

Changes in ll-make 0.22 (released 01/21/2005)

- `XPITAction` will now pass the project, target and action to the embedded Python expression as global variables.

Changes in ll-make 0.21 (released 01/19/2005)

- An action class `XPITAction` has been added for use with `ll.xpit`.
- Setting a project dirty (so that out-of-dateness will be rechecked) has been factored into a separate method.

Changes in ll-make 0.20 (released 01/03/2005)

- `make` requires the core module and Python 2.4 now.

Changes in ll-make 0.19.1 (released 11/26/2004)

- Fixed print of tracebacks when `ignoreerrors` is true.

Changes in ll-make 0.19 (released 10/29/2004)

- *`ll.make`* is compatible with XIST 2.6 now (and incompatible with XIST 2.5).

Changes in ll-make 0.18.2 (released 10/12/2004)

- Retry with absolute and real URLs in `__candidates()` even if the argument is already an *ll.url.URL* object. This works around an URL normalization bug under Windows.

Changes in ll-make 0.18.1 (released 08/27/2004)

- `Target.actions` is now a list instead of a tuple.

Changes in ll-make 0.18 (released 07/06/2004)

- Added a new action class `TOXICPrettifyAction` that uses the new `prettify()` function introduced in `ll.toxic` version 0.3.

Changes in ll-make 0.17 (released 06/02/2004)

- Renamed `OracleTarget` to `DBTarget`.
- Reporting `PhonyTarget` objects has been moved to a separate method named `reportphonytargets()`.

Changes in ll-make 0.16 (released 05/31/2004)

- The method `buildWithArgs()` has been dropped. Use `buildwithargs()` now.
- Argument parsing has been made extensible. The method `optionparser()` must return an instance of `optparse.OptionParser`. The method `parseoptions()` parses the argument sequence passed in (defaults to `sys.argv[1:]` and returns a tuple with `(values, args)` (just like `optparse.OptionParser.parse_args()` does).
- The arguments `ignoreerrors`, `color`, `maxinputreport` have been removed from the `Project` constructor. If you really need different values for these, simply change the attributes after creating the `Project` object.
- `Project.__getitem__()` and `Project.__contains__()` now recognize database ids.

Changes in ll-make 0.15.1 (released 05/25/2004)

- Fixed formatting bugs in `OracleReadResource`.

Changes in ll-make 0.15 (released 05/25/2004)

- There's a new option `-vl` that reports the recursion level as an indentation during the build process. This makes it easier to see, what depends on what. The indentation per level can be specified with the environment variable `LL_MAKE_INDENT`.
- The environment variable `MAKE_REPRANSI` has been renamed to `LL_MAKE_REPRANSI`.

Changes in ll-make 0.14.2 (released 05/25/2004)

- If a target has prerequisites, the time to rebuild those will be reported in the progress report too (if time reporting is on (via the option `-vt`)).
- Fix a bug in `XISTPublishAction`.

Changes in ll-make 0.14.1 (released 05/21/2004)

- The default color for output has been removed.
- In the progress report URLs relative to the home directory are now tried too to find the shortest URL for display.
- Fix a bug in `JoinedReadAction` and various other bugs.

Changes in ll-make 0.14 (released 05/20/2004)

- Actions have been made much more atomic and flexible. For each target a chain of actions will be executed. The first action loads the file. The next actions transform the content, after that an action will save the result to a file. Finally other actions can modify this file (what has formerly been known as "secondary actions").

  For example: to transform an XIST file now you need a `ReadAction`, a `XISTParseAction`, a `XISTConvertAction`, a `XISTPublishAction` and a `WriteAction`. The base `URL`s for parsing and publishing have been moved from :class:`XISTTarget` to `XISTParseAction` and `XISTPublishAction`.
- `DBID` has been rewritten. For Oracle `DBID` objects it's possible to read and write functions and procedures via a file-like interface.
- Support for Apache FOP and ll-toxic has been added.
- The `Target` methods `sources()` and `targets()` have been renamed to `inputs()` and `outputs()` (related methods have been renamed too).
- The options for selecting the verbosity of the progress report have been combined into one option `-v`.
- The progress report tries to shorten URLs by displaying relative URLs (relative to the current directory) if those are shorter (which they usually are).

Changes in ll-make 0.13.1 (released 05/05/2004)

- Fixed a small bug in `Project.__contains__()`.

Changes in ll-make 0.13 (released 01/12/2004)

- Now after the build the import cache `ll.url.importcache` will be restored to the state before the call. This fixes a bug, where a module that was loaded from another module (not as a `PythonTarget`), didn't get cleared from the import cache.

Changes in ll-make 0.12 (released 01/02/2004)

- Adapted to XIST 2.4. `XISTTarget` now has two attributes `parser` and `publisher` which will be used by `XISTAction` for parsing and publishing targets.
- Changed the assertions that check that `XISTAction`, `CopyAction` and `SplatAction` have only one source into exceptions.
- `Project.__getitem__()` and related methods will now only try absolute file paths, if the URL really is local.
- Dropped the deprecated project method `has()`.
- For parsing the command line option `optparse` is used now instead of `getopt`.

Changes in ll-make 0.11.7 (released 12/15/2003)

- When building a target fails, the file will now only be removed if it exists.

Changes in ll-make 0.11.6 (released 12/08/2003)

- Remove the module from the import cache in `PythonTarget.clear()`, so that the module will be reloaded when `Project.recreate()` is used.
- Made compatible with XIST 2.3.

Changes in ll-make 0.11.5 (released 12/06/2003)

- Now when a project is rebuilt, all loaded Python modules will be removed from the import cache before rebuilding commences. This should fix intermodule dependencies.

Changes in ll-make 0.11.4 (released 12/06/2003)

- Added methods `itersources()`, `itertargets()`, `itersourcedeps()` and `itertargetdeps()` to the `Target` class.

Changes in ll-make 0.11.3 (released 11/22/2003)

- `__getitem__()` and `__contains__()` of the `Project` class now first try with an absolute filename and then with the real filename (i.e. all links resolved).

Changes in ll-make 0.11.2 (released 08/06/2003)

- A few of the `Project` attributes have been renamed to avoid name clashes when a class was derived from `Project` and *`ll.sisyphus.Job`*.

Changes in ll-make 0.11.1 (released 08/01/2003)

- Fixed a bug in `Project.build()`: Timestamps were not cleared on the second call to `build()` when the first one had failed.
- Timestamp handling was broken. Timestamps from the filesystem were in UTC, but the timestamp set after calls to `Target.update()` were in local time. This has been fixed now.

Changes in ll-make 0.11 (released 07/31/2003)

- Calling the XIST conversion in `XISTAction` has been moved from `execute()` to a new method `convert()` to be easier to customize.
- `make` requires Python 2.3 now.

Changes in ll-make 0.10 (released 07/02/2003)

- Targets will now be removed when building them fails.

Changes in ll-make 0.9.5 (released 05/02/2003)

- `Project.__getitem__()` now retries with a canonical filename (i.e. the result of the `real()`) before giving up.

Changes in ll-make 0.9.4 (released 04/24/2003)

- All primary actions now make sure that the output file is removed when an error happens. The next call to a make script will again try to generate the output instead of silently skipping the half finished (but seemingly up to date) file.

Changes in ll-make 0.9.3 (released 04/23/2003)

- Use the enhanced `import_()` method from *ll.url* 0.7.
- Add a `doc` attribute to `PhonyTarget` which can be used in help messages (e.g. when `buildWithArgs()` is called without arguments).

Changes in ll-make 0.9.2 (released 04/15/2003)

- Fixed a small bug in the deprecated `Project.has()`.

Changes in ll-make 0.9.1 (released 03/11/2003)

- Fixed a small bug in `Target.lastmodified()`.

Changes in ll-make 0.9 (released 03/10/2003)

- Generating a `Publisher` in an `XISTAction` has been moved to a separate method `publisher()`.
- Each target can now be assigned a sequence of actions. There are new action classes `ModeAction` and `OwnerAction` that change the access permissions or owner of a file that has been created by a previous action in an action sequence.
- Updated the timestamp functionality so that with Python 2.3 the `datetime` module will be used for timestamps.

Changes in ll-make 0.8 (released 03/03/2003)

- The project method `has()` has been deprecated. Use `has_key()` or the new `__contains__()` for that. This means that all dictionary access method try strings, URLs and absolute URLs now.
- Populating a project can now be done in the overwritable method `create()`. There is a new method `clear()` which removes all targets from the project. Use the method `recreate()` to recreate a project, i.e. call `clear()` and `create()`.

Changes in ll-make 0.7 (released 02/26/2003)

- Made compatible with XIST 1.5 again: `prefixes` is only passed to the parser, when it is not `None`.
- `has()` and `has_key()` have been changed to do the same as `__getitem__()`, i.e. retry with an URL or absolute URL in case of an error.
- `build()` can now be called multiple times and will reset timestamp information on all subsequent calls. This makes it possible to rerun a build process without having to recreate the project with its targets and dependencies (provided that no targets have to be added or removed).

Changes in ll-make 0.6.1 (released 02/14/2003)

- `XISTTarget` has new attributes `parser`, `handler` and `prefixes` that can be specified in the constructor and will be used for parsing.

Changes in ll-make 0.6 (released 11/20/2002)

- `Project.__getitem__()` now raises an `UndefinedTargetError` exception with the original key if retrying with an `URL` object fails.
- The methods `Target.sources()` and `Target.targets()` have been changed to return the `Target` objects instead of the `Dep` objects. The old functionality is still available as `Target.sourcedeps()` and `Target.targetdeps()`. The same has been done for the method `Target.newerSources()` (and the method name has been made lowercase).

Changes in ll-make 0.5 (released 11/13/2002)

- `Project` is derived from `dict` now.
- Calling `Project.buildWithArgs()` with an empty argument list now lists all `PhonyTarget` objects.

Changes in ll-make 0.4.2 (released 11/11/2002)

- Added a new target class `JavaPropAction`, for Java property files.
- Added a `__len__()` to the `Project` class.

Changes in ll-make 0.4.1 (released 10/25/2002)

- Added a new action class `SplatAction`, that can be used for replacing strings in files.
- Speed up dependency creation by adding slot declarations.

Changes in ll-make 0.4 (released 08/27/2002)

- Adapted to XIST 2.0.

Changes in ll-make 0.3.2 (released 06/16/2002)

- Work around a problem with unicode objects in `sys.path`. This workaround will disappear as soon as Python 2.3 is released.
- Use the method `doPublication()` for publishing nodes. (This requires XIST 1.4.4.)

Changes in ll-make 0.3.1 (released 03/28/2002)

- Added a warning when the id of a new target already exists in the project, i.e. when the target is redefined.
- Added a warning for file modification timestamps from the future.

Changes in ll-make 0.3 (released 03/18/2002)

- Now `url.URL` is used everywhere instead of `fileutils.Filename`

Changes in ll-make 0.2.3 (released 02/22/2002)

- Added a new class `DBID` that can be used as an id for database content.
- Ported to Python 2.2

Changes in ll-make 0.2.2 (released 01/25/2001)

- Verbosity can now be specified via several `Project` constructor arguments.
- `Action.converter()` now sets the attribute `makeaction` on the returned `Converter` object.

Changes in ll-make 0.2.1 (released 10/03/2001)

- Support for the `root` paramenter for the `convert()` method in `XISTAction`.

Changes in ll-make 0.2 (released 10/02/2001)

- Dependencies now have a type (a subclass of `Dep`). This allows to mark certain dependencies as "special".
- `Project.build()` can now be called with a `Target` or a filename as a string.

Changes in ll-make 0.1 (released 07/27/2001)

- Initial release.

Changes in ll-sisyphus

Changes in ll-sisyphus 0.10.1 (released 03/22/2005)

- Added a note about the package init file to the installation documentation.

Changes in ll-sisyphus 0.10 (released 01/03/2005)

- `sisyphus` requires the core module and Python 2.4 now.

Changes in ll-sisyphus 0.9.1 (released 04/28/2004)

- Fixed a bug related to logging empty strings.

Changes in ll-sisyphus 0.9 (released 11/13/2003)

- Lowercased the constructor arguments `maxRuntime`, `raiseErrors` and `printKills`.
- When the job is started it checks whether it's predecessor is still running (i.e. it checks whether the pid from the run file really exists).
- Added a method `logErrorOnly()` that writes to the error log only (this is used when the message about a job still running is written to the error log, so the progress log from the previous job execution won't be disturbed).
- The loop log now contains the exception value in case of an error.

Changes in ll-sisyphus 0.8 (released 07/31/2003)

- `sisyphus` now uses and requires Python 2.3.
- The logging methods can now log everything. If the logged object is not a string, `pprint` is used for formatting.
- The number of seconds is now properly formatted with hours, minutes and seconds in the logfiles.
- A few methods have been lowercased.
- When a job fails the method `failed()` is called now. This gives the job the change to clean up.

Changes in ll-sisyphus 0.7 (released 03/11/2003)

- `sisyphus` now uses the *`ll.url`* module, `ll.fileutils` is no longer required.

Changes in ll-sisyphus 0.6.2 (released 12/03/2002)

- error reports are now logged to the process log too.

Changes in ll-sisyphus 0.6.1 (released 09/10/2002)

- The `Job` constructor has a new argument `printKills` which specifies whether killing a previous job should be printed (i.e. mailed from cron).

Changes in ll-sisyphus 0.6 (released 08/27/2002)

- `sisyphus` has been moved to the `ll` package.

Changes in ll-sisyphus 0.5.3 (released 05/07/2002)

- Derive `Job` from `object` to be able to use new style classes in mixins in subclasses.

Changes in ll-sisyphus 0.5.2 (released 07/19/2001)

- Made compatible with fileutils 0.2.

Changes in ll-sisyphus 0.5.1 (released 04/12/2001)

- Fixed a severe bug (missing call to `os.path.expanduser()`), that prevented `Job` from working.

Changes in ll-sisyphus 0.5 (released 03/29/2001)

- The `Job` constructor has a new parameter `raiseErrors`. When set to true exceptions will not only be written to the logfile but raised, which results in a output to the terminal and an email from the cron daemon.

Changes in ll-sisyphus 0.4 (released 03/26/2001)

- The class `LogFile` has been moved to a seperate module named `fileutils`.

Changes in ll-sisyphus 0.3 (released 02/16/2001)

- Initial public release

Changes in ll-url

Changes in ll-url 0.15.1 (released 03/22/2005)

- Added a note about the package init file to the installation documentation.

Changes in ll-url 0.15 (released 02/24/2005)

- The `mimetype` property of `ReadResource` is no longer a tuple, but a plain string.
- `ReadResource` has a new property `encoding`, which is the character encoding of the resource.
- A bug in the `lastmodified` property of `WriteResource` has been fixed.

Changes in ll-url 0.14.2 (released 02/22/2005)

- `url.URL("file:foo/").local()` will now always end in a directory separator. This didn't work on Windows before.

Changes in ll-url 0.14.1 (released 01/13/2005)

- On Windows `url.File("c:\\foo").abs()` generated `URL('file:///C|/foo')`. Now the result will always be `URL('file:/C|/foo')`. The same fix has been made for `real()` and the constructor.

Changes in ll-url 0.14 (released 01/03/2005)

- `url` requires the core module and Python 2.4 now.

Changes in ll-url 0.13 (released 11/25/2004)

- The helper function `_unescape()` will now interpret `%u` escapes (produced by Microsoft software). The patch has been contributed by Artiom Morozov.

Changes in ll-url 0.12.1 (released 11/03/2004)

- Fixed a bug in the C helper function `_unescape()` (forget to clear the exception).
- Dropped the system default encoding from the list of encodings that will be tried when UTF-8 fails in `_unescape()`.

Changes in ll-url 0.12 (released 01/12/2004)

- `removefromimportcache()` has been dropped, now you can assign the import cache directly (as the module level attribute `importcache`. Removing modules from the import cache can now be done via `url.importcache.remove(mod)`.

Changes in ll-url 0.11.7 (released 12/23/2003)

- Fixed a bug in `Path.real()` that only surfaced on Windows.

Changes in ll-url 0.11.6 (released 12/06/2003)

- Added a function `removefromimportcache()`.

Changes in ll-url 0.11.5 (released 11/22/2003)

- Fixed a bug with the `scheme` argument of the methods `real()` and `abs()`.

Changes in ll-url 0.11.4 (released 11/19/2003)

- `realurl` has been renamed to `finalurl` and now works for local URLs too (it will be the same as the original URL).

Changes in ll-url 0.11.3 (released 11/17/2003)

- Added an attribute `realurl` to `ReadResource` which contains the real URL (which might be different from the URL passed to the constructor, because of a redirect).

Changes in ll-url 0.11.2 (released 11/17/2003)

- URLs that have an authority part but a relative path will be properly formatted, i.e. the leading `/` will be included.

Changes in ll-url 0.11.1 (released 08/13/2003)

- The `URL` method `rename()` has been fixed.
- A bug has been fixed that created relative paths for HTTP URLs that didn't have a trailing `/`.

Changes in ll-url 0.11 (released 08/04/2003)

- A method `withoutfrag()` has been added. `withFragment()` has been renamed to `withfrag()` and the property `fragment` has been renamed to `frag`.

Changes in ll-url 0.10 (released 07/31/2003)

- `url` requires Python 2.3 now.
- The method `insert` has been fixed.

Changes in ll-url 0.9.1 (released 07/17/2003)

- Fixed a bug that drops the filename in `relative()` when both URLs have the same filenames but a different query.
- The fragment is now properly escaped when the URL is regenerated.

Changes in ll-url 0.9 (released 07/09/2003)

- `withExt()` and friends have been lowercased.
- The `path` has been changed from a string to an object of the new class `Path`. This new class provides many of the path related functionality of URLs.
- The method `URL.import_()` no longer uses the import machinery (from the `imp` module), but `execfile()`. This has the following consequences:
  - You can only import files with the extension `.py`.
  - The imported module no longer retains deleted attributes of the previous version.
  - The file will be compiled even if a bytecode file exists.

Changes in ll-url 0.8 (released 06/04/2003)

- Added methods `abs()` and `__rdiv__()` to `URL`.
- The method `real()` now has an argument `scheme` that specifies which scheme should the use for the resulting URL.
- Now the query part of an `URL` will be parsed into the attribute `query_parts` (which is a dictionary). If the query can't be parsed, `query_parts` will be `False`, but `query` will still contain the complete query part.

Changes in ll-url 0.7.1 (released 05/01/2003)

- Made `clearimportcache()` a class method.

Changes in ll-url 0.7 (released 04/23/2003)

- Introduced `local()` as a synonym for `asFilename()`, `Dir()` as a synonym for `Dirname()` and `File()` as a synonym for `Filename()`.
- Added functions `first()`, `firstdir()` and `firstfile()`, that returns the first URL from a list that exists, is a directory or a file.
- The method `import_()` uses a cache now. Different caching strategies can be chosen through the `mode` parameter.

Changes in ll-url 0.6.2 (released 03/07/2002)

- The method `real()` checked whether the referenced file really is a directory. This has the problem that the directory/file must exist. Now the directoryness of the URL itself is used.

Changes in ll-url 0.6.1 (released 03/06/2002)

- Fixed a bug in `chown()`: Attributes are not available for `pwd.getpwnam()` and `grp.getgrnam()` results under Python 2.2. Use the tuple entry instead.
- Added methods `mtime()`, `atime()` and `size()` to `URL`.

Changes in ll-url 0.6 (released 03/05/2002)

- Now all arguments for `walk()` default to `False`.
- Added new convenience methods `walkfiles()` and `walkdirs()`.
- An `URL` can now be iterated. This is equivalent to `walk(dirsbefore=True, files=True)`.
- Many functions from `os` and `os.path` have been added as methods to *`ll.url`*. This was inspired by Jason Orendorff's `path` module.
- The method `import_()` is now available in the `URL` class too.
- When Python 2.3 is used timestamp will now be `datetime.datetime` objects and `mx.DateTime` is no longer required. With Python 2.2 `mx.DateTime` will still be used.

Changes in ll-url 0.5.1 (released 01/07/2002)

- Added a `LICENSE` file.

Changes in ll-url 0.5 (released 11/14/2002)

- `WriteResource` has been largely rewritten to elminate the overhead of calls the `write()`. Access to properties might be a little slower now, because `WriteResource` has been optimized for maximum writing speed.
- Added source code encoding statements to the Python files.

Changes in ll-url 0.4.3 (released 11/11/2002)

- Fixed a refcounting leak in the new version of `_normalizepath()`.

Changes in ll-url 0.4.2 (released 11/08/2002)

- `_normalizepath()` has been reimplemented in C for performance reasons.

Changes in ll-url 0.4.1 (released 10/29/2002)

- `ReadResource` and `WriteResource` now have a method `import_()`, that imports the file as a Python module (ignoring the file extension).

Changes in ll-url 0.4 (released 10/18/2002)

- Added a `HOWTO` file.
- Made the docstrings compatible with XIST 2.0.
- The `imagesize` property now raises an `IOError` if the PIL is not available.

Changes in ll-url 0.3.1 (released 09/09/2002)

- `WriteResource` will now generate an empty file, even if `write()` is never called. This is checked in `close()`.
- `WriteResource` gained a destructor that will call `close()`.

Changes in ll-url 0.3 (released 08/27/2002)

- `url` has been moved to the `ll` package.

Changes in ll-url 0.2 (released 06/18/2002)

- `_escape()` now always uses unicode strings. 8bit strings will be converted to unicode before the UTF-8 version will be encoded.
- `_unescape()` now always emits unicode strings. If the UTF-8 decoding does not work, the system default encoding will be tried, and finally Latin-1 will be used.
- `_escape()` and `_unescape()` have been rewritten in C for performance reasons.

Changes in ll-url 0.1.8 (released 05/07/2002)

- Illegal `%` escapes now only issue a warning and will be used literally when the warning framework doesn't raise an exception.

Changes in ll-url 0.1.7 (released 04/30/2002)

- Removed the illegal scheme handling change from 0.1.6 again. Now this has to be done before constructing an `URL`.

Changes in ll-url 0.1.6 (released 04/26/2002)

- Now when the parser discovers an illegal scheme, you get another chance: Beginning whitespace will be stripped and it will be retried.

Changes in ll-url 0.1.5 (released 04/25/2002)

- Fixed a bug in `__div__()`: Now `URL("http://foo/bar")/"/baz"` works.

Changes in ll-url 0.1.4 (released 04/15/2002)

- When assigning to the `url` property, the scheme will now only be set when it consists of legal characters. This means that parsing `/foo.php?x=http://www.bar.com` won't try to set a scheme `/foo.php?x=http`, but will use an empty scheme.

Changes in ll-url 0.1.3 (released 04/09/2002)

- Make `ext` and `file` work with opaque `URL`s.
- Forgot the make `resdata` assignable. Fixed.
- Now the scheme to be used can be specified for the various filename functions.
- Added a method `withFragment()` that returns a copy of the `URL` with a new fragment.
- Use the `email` package instead of `rfc822` for `formatdate()`.
- No longer quote `[` and `]` to be compatible with the ezt templates from ViewCVS.
- When joining URLs the right hand URL no longer inherits the scheme, if it has not scheme, but the path is absolute:

```
>>> url.URL("root:foo.html")/url.URL("/cgi-bin/")
URL('/cgi-bin/')
```

Changes in ll-url 0.1.2 (released 03/26/2002)

- Fixed a bug in `URL.__eq__()` and `URL.__hash__()`: `query` and `fragment` were not used. This has been fixed.

Changes in ll-url 0.1.1 (released 03/20/2002)

- Fixed a bug in `ReadResource.contentlength`, which tried to convert the `stat()` result to a `DateTime` object.

Changes in ll-url 0.1 (released 03/18/2002)

- Initial release

Changes in ll-xpit

Changes in ll-xpit 0.2.1 (released 03/22/2005)

- Added a note about the package init file to the installation documentation.

Changes in ll-xpit 0.2 (released 01/21/2005)

- `convert()` now takes both a global and a local namespace and will raise an exception when an unknown processing instruction target is encountered.

Changes in ll-xpit 0.1 (released 01/19/2005)

- Initial release.

Changes to ll-orasql

Changes in ll-orasql 1.27.1 (released 03/31/2009)

- Fixed a bug in the dependency checking for `Connnection.itertables()`.
- `oradelete` now has a new option to use `truncate table` instead of `delete from`.

Changes in ll-orasql 1.27 (released 03/31/2009)

- Added a new script `oradelete` that can be used to delete all records from all tables and to reset all sequences.
- `Connection` has a new method `itersequences()`.
- Fixed a bug in the generated SQl code for triggers (the name always included the name of the original schema).

Changes in ll-orasql 1.26 (released 03/27/2009)

- *ll.orasql* now requires cx_Oracle 5.0 compiled in Unicode mode (i.e. with `WITH_UNICODE=1`). Lots of unicode handling stuff has been rewritten to take advantage of Unicode mode.
- `orafind` has a new option `--encoding` to decode the searchstring on the commandline.
- The `Pool` constructor now supports the additional arguments `getmode` and `homogeneous` from cx_Oracle 5.0.
- Fix a typo in `Privilege.grantddl()`.

Changes in ll-orasql 1.25.4 (released 01/21/2009)

- Procedures and functions with timestamp arguments can now be called.

Changes in ll-orasql 1.25.3 (released 11/07/2008)

- Procedures and functions now should handle arguments of type `BLOB` correctly.

Changes in ll-orasql 1.25.2 (released 08/29/2008)

- `Record` has a new method `get()` which works like the dictionary method `get()`.

Changes in ll-orasql 1.25.1 (released 07/21/2008)

- `orafind.py` now has an additional options `readlobs` (defaulting to false). If this option is set, the value of LOBs in the records found, will be printed.

Changes in ll-orasql 1.25 (released 06/17/2008)

- A new script has been added: `orafind.py` will search for a specified string in all columns of all tables in a schema.

Changes in ll-orasql 1.24.1 (released 05/30/2008)

- Fixed two bugs in `Callable._calcargs()` and `Connection.getobject()`.

Changes in ll-orasql 1.24 (released 05/20/2008)

- `Connection.getobject()`, `Procedure` and `Function` now support functions and procedures in packages.
- Added `__repr__()` to the exception classes.

Changes in ll-orasql 1.23.4 (released 04/04/2008)

- All database scripts now have an additional option `encoding` that specifies the encoding for the output script.

Changes in ll-orasql 1.23.3 (released 04/03/2008)

- Fixed a regression in the scripts `oracreate.py`, `oradrop.py` and `oragrant.py`.

Changes in ll-orasql 1.23.2 (released 04/01/2008)

- When calling functions/procedures, arguments are now wrapped in variable objects for their real type instead of ones for the type the function or procedure expects.

Changes in ll-orasql 1.23.1 (released 03/25/2008)

- Added a `__contains__()` to `Record` for checking the existence of a field.

Changes in ll-orasql 1.23 (released 03/25/2008)

- Calling procedures and functions has been rewritten: *`ll.orasql`* will only pass those parameters to the procedure/function that are passed to the call (or variables for out parameters). Internally this is handled by executing the call as a parameterized query calling the procedure/function with named arguments.
- `FetchRecord` has been renamed to `Record` (and is used for the result of procedure and function calls now, which required some internal changes to `FetchRecord`). The former `Record` has been renamed to `Args` as its only use now is collecting arguments for procedure/function calls. (The method `fromdata()` has been dropped.)
- The `__repr__()` output of `Argument` objects now shows the datatype.

Changes in ll-orasql 1.22 (released 03/19/2008)

- Added a new method `_getobject()` to `Connection` that does what `getobject()` does, but is case sensitive (This is used internally by `Synonym.getobject()`).
- The methods `xfetchone()`, `xfetchmany()`, `xfetchall()`, `xfetch()`, `xexecute()` and `xexecutemany()` have been dropped again. Fetch result objects are now of type `FetchRecord`. Field access is available via index (i.e. `row[0]`), key (`row["name"]`) and attribute (`row.name`). These result objects are generated via the `rowfactory` attribute (which was added in cx_Oracle 4.3.2). All fetch and execute methods support unicode values.

Changes in ll-orasql 1.21.1 (released 03/17/2008)

- Updated the scripts to work with the new execute methods.

Changes in ll-orasql 1.21 (released 03/13/2008)

- `Connection` has a new method `getobject()`, which returns the schema object with a specified name.
- `Synonym` has a new method `getobject()`, that returns the object for which the `Synonym` object is a synonym.
- The name of `Procedure` and `Function` objects now is case sensitive when calling the procedure or function.

Changes in ll-orasql 1.20 (released 02/07/2008)

- The fancy fetch methods have been renamed to `xfetchone()`, `xfetchmany()`, `xfetchall()` and `xfetch()`. `__iter__()` no longer gets overwritten. New methods `xexecute()` and `xexecutemany()` have been added, that support passing unicode parameters.

Changes in ll-orasql 1.19 (released 02/01/2008)

- All docstrings use ReST now.

Changes in ll-orasql 1.18 (released 01/07/2008)

- Updated the docstrings to XIST 3.0.
- Added ReST versions of the documentation.

Changes in ll-orasql 1.17.5 (released 08/09/2007)

- Fixed a bug in the error handling of wrong arguments when calling functions or procedures.

Changes in ll-orasql 1.17.4 (released 04/30/2007)

- The threshold for string length for procedure and function arguments has been reduced to 4000.

Changes in ll-orasql 1.17.3 (released 03/08/2007)

- `BLOB` arguments for procedures and functions are always passed as variables now.

Changes in ll-orasql 1.17.2 (released 03/07/2007)

- Arguments for procedures and functions that are longer that 32000 characters are passed as variables now (the threshold was 32768 before and didn't work).

Changes in ll-orasql 1.17.1 (released 03/02/2007)

- Fix an inverted logic bug in `Record.fromdata()` that surfaced in unicode mode: `BLOB`s were treated as string and `CLOB`s as binary data.

Changes in ll-orasql 1.17 (released 02/23/2007)

- The `readlobs` and `unicode` parameters are now honored when calling procedures and functions via `Procedure` and `Function` objects.

Changes in ll-orasql 1.16 (released 02/21/2007)

- A parameter `unicode` has been added to various constructors and methods. This parameter can be used to get strings (i.e. `VARCHAR2` and `CLOB`s) as `unicode` object instead of `str` objects.

Changes in ll-orasql 1.15 (released 02/17/2007)

- Fixed an output bug in `oradiff.py` when running in full output mode.
- A parameter `readlobs` has been added to various constructors and methods that can be used to get small (or all) `LOB` values as strings in cursor fetch calls.

Changes in ll-orasql 1.14 (released 02/01/2007)

- A new method `iterprivileges()` has been added to `Connection`.
- A script `oragrant.py` has been added for copying privileges.

Changes in ll-orasql 1.13 (released 11/06/2006)

- Two new methods (`itertables()` and `iterfks()`) have been added to `Connection`. They yield all table definitions or all foreign keys respectively.
- A new method `isenabled()` has been added to `ForeignKey`.
- A `__str__()` method has been added to `Object`.
- A bug in `oramerge.py` has been fixed: In certain situations `oramerge.py` used merging actions that were meant to be used for the preceeding object.

Changes in ll-orasql 1.12.2 (released 10/18/2006)

- Fixed a bug that showed up when an index and a foreign key of the same name existed.

Changes in ll-orasql 1.12.1 (released 09/19/2006)

- Fixed a bug in `Index.__xattrs__()`.

Changes in ll-orasql 1.12 (released 09/06/2006)

- `Function` objects are now callable too. They return the return value and a `Record` containing the modified input parameters.

Changes in ll-orasql 1.11.1 (released 08/29/2006)

- Fixed a bug in `Column.modifyddl()`.

Changes in ll-orasql 1.11 (released 08/22/2006)

- The class `Column` has gained a few new methods: `datatype()`, `default()`, `nullable()` and `comment()`.
- Calling a procedure will now raise a `SQLObjectNotFoundError` error, if the procedure doesn't exist.

Changes in ll-orasql 1.10 (released 08/11/2006)

- The classes `Proc` and `LLProc` have been removed. The functionality of `Proc` has been merged into `ProcedureDefinition` (with has been renamed to `Procedure`). Information about the procedure arguments is provided by the `iteraguments()` method.
- All other subclasses of `Definition` have been renamed to remove the "Definition" for the name to reduce typing. (Methods have been renamed accordingly too.)
- `oramerge.main()` and `oradiff.main()` now accept option arrays as arguments.
- `oradiff.py` has finally been fixed.

Changes in ll-orasql 1.9.4 (released 08/09/2006)

- Fixed a bug in `oradiff.py`.

Changes in ll-orasql 1.9.3 (released 08/08/2006)

- Fixed a bug in `oramerge.py`.

Changes in ll-orasql 1.9.2 (released 08/04/2006)

- Fixed a bug in `TableDefinition.iterdefinitions()`.

Changes in ll-orasql 1.9.1 (released 08/02/2006)

- Fixed a bug in `oracreate.py`.

Changes in ll-orasql 1.9 (released 07/24/2006)

- Dependencies involving `MaterializedViewDefinition` and `IndexDefinition` objects generated by constraints work properly now, so that iterating all definitions in create order really results in a working SQL script.
- A method `table()` has been added to `PKDefinition`, `FKDefinition`, `UniqueDefinition` and `IndexDefinition`. This method returns the `TableDefinition` to object belongs to.
- A method `pk()` has been added to `FKDefinition`. It returns the primary key that this foreign key references.
- Indexes and constraints belonging to skipped tables are now skipped too in `oracreate.py`.
- Arguments other than `sys.argv[1:]` can now be passed to the `oracreate.py` and `oradrop.py` `main()` functions.

Changes in ll-orasql 1.8.1 (released 07/17/2006)

- *`ll.orasql`* can now handle objects name that are not in uppercase.

Changes in ll-orasql 1.8 (released 07/14/2006)

- `Connection.iterobjects()` has been renamed to `iterdefinitions()`.
- Each `Definition` subclass has a new classmethod `iterdefinitions()` that iterates through all definitions of this type in a schema (or all schemas).
- Each `Definition` subclass has new methods `iterreferences()` and `iterreferencedby()` that iterate through related definitions. The methods `iterreferencesall()` and `iterreferencedbyall()` do this recursively. The method `iterdependent()` is gone now.
- The method `iterschema()` of `Connection` now has an additional parameter `schema`. Passing `"all"` for `schema` will give you statistics for the complete database not just one schema.
- A new definition class `MaterializedViewDefinition` has been added that handles materialized views. Handling of create options is rudimentary though. Patches are welcome.
- `TableDefinition` has a three new methods: `ismview()` returns whether the table is a materialized view; `itercomments()` iterates through comments and `iterconstraints()` iterates through primary keys, foreign keys and unique constraints.
- The method `getcursor()` will now raise a `TypeError` if it can't get a cursor.

Changes in ll-orasql 1.7.2 (released 07/05/2006)

- `RAW` fields in tables are now output properly in `TableDefinition.createddl()`.
- A class `PackageBodyDefinition` has been added. `oracreate.py` will output package body definitions and `oradrop.py` will drop them.

Changes in ll-orasql 1.7.1 (released 07/04/2006)

- Duplicate code in the scripts has been removed.
- Fixed a bug in `oramerge.py`: If the source to be diffed was long enough the call to `diff3` deadlocked.

Changes in ll-orasql 1.7 (released 06/29/2006)

- The method `iterobjects()` has been moved from `Cursor` to `Connection`.
- The method `itercolumns()` has been moved from `Cursor` to `TableDefinition`.
- `LLProc` now recognizes the `c_out` parameter used by `ll.toxic` 0.8.
- Support for positional arguments has been added for `Proc` and `LLProc`. Error messages for calling procedures have been enhanced.
- `SequenceDefinition` now has a new method `createddlcopy()` that returns code that copies the sequence value. `oracreate.py` has a new option `-s/--seqcopy` that uses this feature.
- `setuptools` is now supported for installation.

Changes in ll-orasql 1.6 (released 04/26/2006)

- Added a `SessionPool` (a subclass of `SessionPool` in `cx_Oracle`) whose `acquire()` method returns *`ll.orasql.Connection`* objects.

Changes in ll-orasql 1.5 (released 04/05/2006)

- Added a class `IndexDefinition` for indexes. `oracreate.py` will now issue create statements for indexes.

Changes in ll-orasql 1.4.3 (released 12/07/2005)

- Fixed a bug with empty lines in procedure sources.
- Remove spurious spaces at the start of procedure and function definitions.

Changes in ll-orasql 1.4.2 (released 12/07/2005)

- Fixed a bug that the DDL output of Java source.
- Trailing whitespace in each line of procedures, functions etc. is now stripped.

Changes in ll-orasql 1.4.1 (released 12/06/2005)

- Fixed a bug that resulted in omitted field lengths.

Changes in ll-orasql 1.4 (released 12/05/2005)

- The option `-m/--mode` has been dropped from the script `oramerge.py`.
- A new class `ColumnDefinition` has been added to *`ll.orasql`*. The `Cursor` class has a new method `itercolumns()` that iterates the `ColumnDefinition` objects of a table.
- `oramerge.py` now doesn't output a merged `create table` statement, but the appropriate `alter table` statements.

Changes in ll-orasql 1.3 (released 11/24/2005)

- Added an option `-i` to `oracreate.py` and `oradrop.py` to ignore errors.
- The argument `all` of the cursor method `iterobjects()` is now named `schema` and may have three values: `"own"`, `"dep"` and `"all"`.
- Added an script `oramerge.py` that does a three way merge of three database schemas and outputs the resulting script.
- DB links are now copied over in `SynonymDefinition` objects.

Changes in ll-orasql 1.2 (released 10/24/2005)

- Added a argument to `createddl()` and `dropddl()` to specify if terminated or unterminated DDL is wanted (i.e. add `;` or `/` or not).
- `CommentsDefinition` has been renamed to `CommentDefinition` and holds the comment for one field only.
- `JavaSourceDefinition` has been added.
- The scripts `oracreate.py`, `oradrop.py` and `oradiff.py` now skip objects with `"$"` in their name by default. This can be changed with the `-k` option (but this will lead to unexecutable scripts).
- `oradiff.py` has a new options `-b`: This allows you to specify how whitespace should be treated.
- Added an option `-x` to `oracreate.py` to make it possible to directly execute the DDL in another database.
- Fixed a bug in `SequenceDefinition` when the `CACHE` field was `0`.

Changes in ll-orasql 1.1 (released 10/20/2005)

- A script `oradiff.py` has been added which can be used for diffing Oracle schemas.
- Definition classes now have two new methods `cdate()` and `udate()` that give the creation and modification time of the schema object (if available).
- A `"flat"` iteration mode has been added to `Cursor.iterobjects()` that returns objects unordered.
- `Connection` has a new method `connectstring()`.
- A class `LibraryDefinition` has been added.
- `CommentsDefinition.createddl()` returns `""` instead of `"\n"` now if there are no comments.
- `SQLObjectNotfoundError` has been renamed to `SQLObjectNotFoundError`.

Changes in ll-orasql 1.0 (released 10/13/2005)

- *ll.orasql* requires version 1.0 of the core package now.
- A new generator method `iterobjects()` has been added to the `Cursor` class. This generator returns "definition objects" for all the objects in a schema in topological order (i.e. if the name of an object (e.g. a table) is generated it will only depend on objects whose name has been yielded before). SQL for recreating and deleting these SQL objects can be generated from the definition objects.
- Two scripts (`oracreate.py` and `oradrop.py`) have been added, that create SQL scripts for recreating or deleting the content of an Oracle schema.

Changes in ll-orasql 0.7 (released 08/09/2005)

- The commands generated by `iterdrop()` no longer have a terminating `;`, as this seems to confuse Oracle/cx_Oracle.

Changes in ll-orasql 0.6 (released 06/20/2005)

- Two new functions have been added: `iterdrop()` is a generator that yields information about how to clear the schema (i.e. drop all table, sequences, etc.). `itercreate()` yields information about how to recreate a schema.

Changes in ll-orasql 0.5 (released 06/07/2005)

- Date values are now supported as `OUT` parameters.

Changes in ll-orasql 0.4.1 (released 03/22/2005)

- Added a note about the package init file to the installation documentation.

Changes in ll-orasql 0.4 (released 01/03/2005)

- *ll.orasql* now requires ll-core.
- Procedures can now be called with string arguments longer that 32768 characters. In this case the argument will be converted to a variable before the call. The procedure argument must be a `CLOB` in this case.
- Creating `Record` instances from database data is now done by the class method `Record.fromdata()`. This means it's now possible to use any other class as long as it provides this method.

Changes in ll-orasql 0.3 (released 12/09/2004)

- *ll.orasql* requires cx_Oracle 4.1 now.

Changes in ll-orasql 0.2.1 (released 09/09/2004)

- Fixed a regression bug in `Proc._calcrealargs()` as cursors will now always return `Record` objects.

Changes in ll-orasql 0.2 (released 09/08/2004)

- Now generating `Record` object is done automatically in a subclass of `cx_Oracle.Cursor`. So now it's possible to use *`ll.orasql`* as an extended `cx_Oracle`.

Changes in ll-orasql 0.1 (released 07/15/2004)

- Initial release.

Changes to ll-nightshade

Changes in ll-nightshade 0.14.1 (released 03/09/2009)

- *`ll.nightshade.Call`* now commits any changes that might have been done by the function or procedure.

Changes in ll-nightshade 0.14 (released 01/14/2009)

- `ll.nightshade.Connection` has new methods `commit()`, `rollback()`, `close()` and `cancel()`.

Changes in ll-nightshade 0.13.1 (released 08/29/2008)

- `Connect.cursor()` now passes keyword arguments through to *`ll.orasql.Connection.cursor()`*.

Changes in ll-nightshade 0.13 (released 02/15/2008)

- CherryPy 3.0 is required now.
- The `conditional()` decorator has been removed. You can use CherryPy's `tools.etags` tool.
- The `cache()` decorator has been removed. You can use CherryPy's `tools.caching` tool.

Changes in ll-nightshade 0.12 (released 02/01/2008)

- All docstrings use ReST now.

Changes in ll-nightshade 0.11 (released 01/07/2008)

- Updated the docstrings to XIST 3.0.
- Added ReST versions of the documentation.

Changes in ll-nightshade 0.10 (released 09/04/2007)

- When a `Connect` object is used as a decorator the database connection is no longer passed to the decorated function. This means that there will no longer be any signature mismatch between the original function and the decorated function. However the `Connect` object must be stored somewhere else and the user must call the new `cursor()` method to get a cursor.
- Keyword argument in the `Connect` constructor are passed on to the `connect()` call.

Changes in ll-nightshade 0.9 (released 07/18/2007)

- Added support for the `Cache-Control` header.

Changes in ll-nightshade 0.8.1 (released 06/26/2007)

- Fixed a bug in `Call.__call__()` (calling the procedure wasn't retried after the connection got lost).

Changes in ll-nightshade 0.8 (released 06/21/2007)

- `withconnection` has been renamed to `Connect` and the implementation of `__call__()` has been fixed.
- `Call` now needs a `Connect` object as the second argument in the constructor (instead of taking `connectstring`, `pool` and `retry` arguments).

Changes in ll-nightshade 0.7.1 (released 05/12/2007)

- Fixed a bug that surfaced after the connection to the database was lost.

Changes in ll-nightshade 0.7 (released 03/16/2007)

- A new decorator `withconnection` has been added. This can be used to retry database operations in case of stale connections.

Changes in ll-nightshade 0.6 (released 03/12/2007)

- Initial public release.

6.18.2 Old migration info

The following is migration info for modules and packages that where merged into XIST (starting with XIST 3.2) or into the former core package (starting with XIST 2.12).

Migration info for ll-core

Migrating to ll-core 1.6

Handling of file data and file metadata in *ll.url* has been largely rewritten. The most significant visible change is that the `ReadResource` properties `resdata`, `resheaders`, `imagesize`, `mimetype`, `encoding` and `finalurl` are methods now. A few properties have been turned into methods and have been renamed: `lastmodified` has been renamed to `mdate()`, `contentlength` has been renamed to `size()` and `stats` has been renamed to `stat()`.

Migrating to ll-core 1.5

The functions `ll.url.Dirname()` and `ll.url.Filename()` have been removed (use *ll.url.Dir()* and *ll.url.File()* instead).

The methods `ll.url.URL.isLocal()` and `ll.url.URL.asFilename()` have been removed (use *ll.url.URL.islocal()* and *ll.url.URL.local()* instead).

Migrating to ll-core 1.3

ll.make has been largely rewritten, so you have to adapt your make scripts. For examples demonstrating how to do this, take a look at either the small example in the module itself or the make script for the website.

Migrating to ll-core 1.2

Processing instruction targets in `ll.xpit` now require whitespace after the target name. This means that you have to replace <?=foo?> with <?= foo?> in your xpit strings.

Migrating to ll-core 1.1

If you've been using `TOXICAction` from *ll.make*, you have to use a `DecodeAction` before the `TOXICAction` to decode the `str` object into a `unicode` object and use a `EncodeAction` afterwards to encode it again as the constructor of `TOXICAction` no longer takes an `encoding` argument, but operates on unicode strings directly.

Migrating to ll-core 1.0

The content of the `ll` module has been move to *ll.misc*, so you have to replace e.g. `ll.notimplemented()` with `misc.notimplemented()` etc.

Migrating to ll-core 0.3

Changes to namespaces

Functions will no longer will turned into `staticmethod` objects automatically, so you have to decorate them yourself.

Migration info for ll-make

Migrating to ll-make 1.0

`Targets` now have four action chains instead of one, so you have to rewrite your `Target` constructors. How the new call looks depends on the target itself. For example a simple copy operation might look like this:

```
source = make.FileTarget(project, "foo", readaction=make.ReadAction())
target = make.FileTarget(project, "bar", convertaction=make.SelectMainAction(),␣
↪writeaction=make.WriteAction())
target.dependOn(make.MainDep, source)
```

Importing modules from other modules can now be done like this:

```
from ll import make

foo = make.currentproject["build/foo.py"].getdata()
```

Furthermore if `build/foo.py` itself is generated by other actions, these actions will be executed before `build/foo.py` is imported. For this to work you need to use the correct action chains for your target:

```
srcfoo = make.PythonTarget(
   project,
   "src/foo.py",
   readaction=make.ReadAction()
)
buildfoo = make.PythonTarget(
   project,
   "build/foo.py",
   cache=True,
   convertaction=make.SelectMainAction()+make.WriteAction()+make.ImportAction()+make.
↪UseModuleAction(),
   readaction=make.ImportAction()+make.UseModuleAction(),
   useaction=make.UseModuleAction()
)
buildfoo.dependOn(make.MainDep, srcfoo)
```

Migrating to ll-make 0.26

All `Target` constructors expect to be passed *one* `Action` instance only now, so instead of:

```
t = make.FileTarget(project, id, action1, action2, action3)
```

you should use:

```
t = make.FileTarget(project, id, action=action1+action2+action3)
```

Adding targets will create an appropriate `ChainedAction` object from the added actions.

Migrating to ll-make 0.23

A class variable `name` in an action class will be ignored now. You have to implement a method `desc()` (and might implement `fulldesc()` to give a longer description).

Migrating to ll-make 0.17

`OracleTarget` has been renamed to `DBTarget`.

Migrating to ll-make 0.15

The environment variable `MAKE_REPRANSI` has been renamed to `LL_MAKE_REPRANSI`.

Migrating to ll-make 0.14

The way actions are handled has changed completely. Instead of a single action that loads the input, does something and saves to output, each of these steps is done by a separate action.

XIST transformations will now look something like this:

```
from ll import make
p = make.Project()
t0 = make.XISTTarget(p, url.File("foo.htmlxsc"))
t1 = make.XISTTarget(p,
   url.File("../install/foo.html",
   make.ReadAction(),
   make.XISTParseAction(base=url.File("root:foo.html")),
   make.XISTConvertAction(),
   make.XISTPublishAction(
      publisher=publishers.Publisher(encoding="us-ascii"),
      base=url.File("root:foo.html")
   ),
   make.WriteAction(),
   make.ModeAction(0644)
)
t1.dependOn(make.MainDep, t0)
```

Several `Target` methods have been renamed: `sources()` has been renamed to `inputs()`. `targets()` has been renamed to `outputs()`. Several related methods and options have been renamed too.

The output during the build has changed. Instead of newer sources, the main sources will always be displayed now.

The options controlling the output during the build have beed changed and joined into one option, where letters in the option value switch certain output on and off. For more info simply invoke the build script with the option `--help`.

Migrating to ll-make 0.12

`make` has been updated for XIST 2.4: Parsing and publishing XIST files is now no longer the job of the `XISTAction` class itself, but is done through the attributes `parser` and `publisher` of the `XISTTarget` object, which must be an XIST parser and XIST publisher respectively.

Migrating to ll-make 0.8

All dictionary access method now try the literal id first, and if it's a string, they will retry with an &url; and an absolute &url;. So now you can no longer have a phony target and a file target with the same name (which shouldn't be a problem anyway, because a file target should include the full path).

Migrating to ll-make 0.6

The `Target` methods `sources()` and `targets()` have been changed, so that they return the source and target `Target` objects instead of the dependency objects.

This should be more convenient, because in most cases the targets are needed anyway. The old functionality is available through the new methods `sourcedeps()` and `targetdeps()`. If you've defined your own action classes you'll probably have to update them.

The same change has been made for the method `newerSources()` (and the method name has been made lowercase). So `newersources()` will return a list of `Target`s and :meth:`newersourcedeps` will return the list of dependencies accordingly.

Migration info for ll-nightshade

Migrating to ll-nightshade version 0.13

The decorators `cache()` and `conditional()` no longer exist. Use CherryPy's tools `tools.etag` and `tools.caching` instead.

Migrating to ll-nightshade version 0.10

When a `Connect` object is used as a decorator the database connection is no longer passed to the decorated function. You have to store the `Connect` object somewhere and call it's new `cursor()` method explicitly.

Migrating to ll-nightshade version 0.8

The class `withconnection` has been renamed to `Connect`.

Calling functions and procedures has changed a bit. Replace the following old code:

```
proc = nightshade.Call(orasql.Procedure("proc"), connectstring=connectstring)

@cherrypy.expose
def foo(arg):
   return proc(arg)
```

with:

```
connection = nightshade.Connect(connectstring=connectstring)
proc = nightshade.Call(orasql.Procedure("proc"), connection)

@cherrypy.expose
def foo(arg):
   return proc(arg)
```

6.19 Source

All our XIST/UL4 related repositories are available on GitHub:

XIST
https://github.com/LivingLogic/LivingLogic.Python.xist

Javascript
https://github.com/LivingLogic/LivingLogic.Javascript.ul4

Java
https://github.com/LivingLogic/LivingLogic.Java.ul4

PHP
https://github.com/LivingLogic/LivingLogic.PHP.ul4

Oracle
https://github.com/LivingLogic/LivingLogic.Oracle.ul4

6.20 About us

6.20.1 Responsibility for contents

LivingLogic AG
Handelsregister Bayreuth HRB 3274
VAT number: DE208246480
Executive board: Dr. Alois Kastner-Maresch

Postal address

Markgrafenallee 44
95448 Bayreuth

6.20.2 Copyright and trademark protection

6.20.3 Liability for contents

The websites of the LivingLogic AG are maintained constantly and have been prepared with utmost care. LivingLogic AG will however not assume liability for accuracy and completeness of the information.

6.20.4 Disassociation

The hyperlinks found on the websites of the LivingLogic AG are not to be regarded as recommendations. The accuracy and correctness of the information on the linked pages is not monitored by the LivingLogic AG. The LivingLogic AG is not responsible for the contents of the linked websites and accepts no liability for damages that might arise from the use of such information.

6.21 Datenschutzerklärung

Wir respektieren Ihre Privatsphäre.

Unsere Datenschutzpraxis steht im Einklang mit dem Bundesdatenschutzgesetz (BDSG) und dem Telemediengesetz (TMG).

Verantwortliche Stelle im Sinne der Datenschutzgesetze ist die LivingLogic AG, Markgrafenallee 44, 95448 Bayreuth, vertreten durch den Vorstand Dr. Alois Kastner-Maresch.

Wir haben einen betrieblichen Datenschutzbeauftragten bestellt.

Wenn Sie eine Webseite besuchen, speichert der zugrundeliegende Webserver immer die IP-Adresse Ihres Providers, die Webseite, die Sie vorher besucht haben, die Webseiten, die Sie auf dem Webserver abrufen sowie das Datum und die Dauer Ihres Besuchs. Diese Informationen sind für die technische Übermittlung der Webseiten und den Serverbetrieb erforderlich. LivingLogic garantiert Ihnen, dass keine personalisierte Auswertung dieser Daten erfolgt und dass diese Daten weder im Rohzustand noch in einem ausgewerteten Zustand an Dritte weitergegeben werden. Die gespeicherten Daten werden nur zur Weiterentwicklung der Website von LivingLogic im Interesse der Nutzer ausgewertet.

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z