Python/XML Reference Guide

The Python/XML Special Interest Group

xml-sig@python.org
(edited by akuchling@acm.org)

Abstract:

XML is the Extensible Markup Language, a subset of SGML, intended to allow the creation and processing of application-specific markup languages. Python makes an excellent language for processing XML data. This document is the reference manual for the Python/XML package, containing several XML modules.

This is a draft document; 'XXX' in the text indicates that something has to be filled in later, or rewritten, or verified, or something.

Contents

* 1 xml.dom Extensions
    + 1.1 UserDataHandler Objects
* 2 xml.dom.ext.c14n -- Canonical XML generation
* 3 xml.dom.minidom Extensions
* 4 xml.dom.xmlbuilder -- DOM Level 3 Load/Save Interface
    + 4.1 DOMImplementationLS Extensions
    + 4.2 DocumentLS Extensions
    + 4.3 DOMBuilder Objects
    + 4.4 Subclassing DOMBuilder
* 5 xml.ns -- XML Namespace constants
* 6 xml.parsers.expat Extensions
* 7 xml.parsers.sgmllib -- Accelerated SGML Parser
* 8 xml.parsers.sgmlop -- XML/SGML Parser Accelerator
* 9 xml.sax.saxexts
    + 9.1 ExtendedParser methods
    + 9.2 ParserFactory methods
* 10 xml.sax.saxlib
    + 10.1 AttributeList methods
    + 10.2 DocumentHandler methods
    + 10.3 DTDHandler methods
    + 10.4 EntityResolver methods
    + 10.5 ErrorHandler methods
    + 10.6 Locator methods
    + 10.7 Parser methods
    + 10.8 SAXException methods
    + 10.9 SAXParseException methods
* 11 xml.sax.saxutils
    + 11.1 Location methods
* 12 xml.utils.iso8601
* About this document ...

1 xml.dom Extensions

Note: The material in this section describes features of the xml.dom module that does not appear in the version of the module included with Python. This is written as a supplement to the corresponding section in the Python 2.2.1 documentation. We intend that all or most of these additions be added to the Python standard library in a future release.

Starting with PyXML 0.8, some features of the DOM Level 3 working drafts are being added. These features are based on working drafts and will be changed as the drafts are updated. Use at your own risk.

exception ValidationErr
: New exception. The validation features of the Level 3 DOM can raise this exception when validation constraints are not met. The rest of the validation features have not been implemented, but any Python DOM implementation would use the same exception, so it is offered at this point.

The additional exception code defined in the Level 3 DOM specification map to the exception described above:

Constant	Exception
VALIDATION_ERR	ValidationErr

The Node class has grown a number of additional constants useful for some of the new methods added in DOM Level 3. These constants are: TREE_POSITION_PRECEDING, TREE_POSITION_FOLLOWING, TREE_POSITION_ANCESTOR, TREE_POSITION_DESCENDENT, TREE_POSITION_EQUIVALENT, TREE_POSITION_SAME_NODE, TREE_POSITION_DISCONNECTED.

Additional classes have been added to provide access to constants and serve as base classes for implementations:

class UserDataHandler
: The DOM Level 3 Core adds a method setUserData() which accepts an instance conforming to the UserDataHandler interface. This class provides the constants used as arguments to the handle() of that interface. The constants provided are NODE_CLONED, NODE_IMPORTED, NODE_DELETED, and NODE_RENAMED. Implementation classes may choose to subclass from this class, but there is no reason for them to do so.

class DOMError
: The DOM Level 3 Core feature adds support for a user-settable error handler. This class provides the constants used to indicate the severity of an error, and may be used as a base class by DOM implementations. These constants should be used to test the value of the severity member of instances of this interface. The constants provided are SEVERITY_WARNING, SEVERITY_ERROR, and SEVERITY_FATA_ERROR.

1.1 UserDataHandler Objects

The UserDataHandler interface, introduced in the DOM Level 3 Core (draft) specification, is used to allow DOM clients to manage node-specific application data stored using the setUserData() method on nodes.

UserDataHandler implementations need only define one method:

handle(operation, key, data, src, dest)
: This method is called for a few particular events in the lifespan the node on which it was registered. For each type of

event, operation indicates the specific operation being performed on the node, key gives the key for which data and the handler object were registered, data is the actual data object, and src is the node on which the handler was registered. The dest argument will always be either None or a different node, depending on the specific operation.

If operation is NODE_CLONED, then dest is the clone of src; src and dest will always belong to the same document unless src is a DOCUMENT_TYPE_NODE.

It operation is NODE_IMPORTED, then dest is the imported version of src, and belongs to a different document.

If operation is NODE_RENAMED, then src and dest are intended to represent the same node, but different node objects are used. This is not called when Document.renameNode() does not create a new node instance.

If operation is NODE_DELETED, then src is about to be deleted (not just removed from the document tree), and dest is None. It is not expected that Python implementations of the DOM will implement this, since doing so properly may interfere with the reclamation of unused nodes.

The constants passed as the values for the operation argument of this method are available from the xml.dom.UserDataHandler class.

2 xml.dom.ext.c14n -- Canonical XML generation

This module takes a DOM element node (and all its children) and generates canonical XML as defined by the W3C candidate recommendation http://www.w3.org/TR/xml-c14n. (Unlike the specification, however, general document subsets are not supported.)

The module name, c14n, comes from the standard way of abbreviating the word "canonicalization." This module is typically imported by doing from xml.dom.ext import Canonicalize.

Canonicalize(node[output[**keywords]])

This function generates the canonical format. If output is specified, the data is sent by invoking its write method, the the function will return None. If output is omitted or has the value None, then the Canonicalize will return the data as a string.

The keyword argument comments, if non-zero, directs the function to leave any comment nodes in the output. By default, they are removed.

The keyword argument stripspace, if non-zero, directs the function to strip all extra whitespace from text elements. By default, whitespace is preserved. This argument should be used with caution, as the canonicalization specification directs that whitespace be preserved.

The keyword argument nsdict may be used to provide a namespace dictionary that is assumed to be in the node’s containing

context. The keys are namespace prefixes, and the values are the namespace URI's. If nsdict is None or an empty dictionary, then an initial dictionary containing just the URI's for the xml and xmlns prefixes will be used.

3 xml.dom.minidom Extensions

The implementation of xml.dom.minidom in PyXML contains support for more of the DOM Level 2 Core specification than the version packaged in Python, and incorporates partial support for the draft DOM Level 3 Core and Load/Save specifications. This additional support includes the interfaces documented for the xml.dom.xmlbuilder module.

This section describes the major extensions beyond what's documented for this module in the Python Library Reference for Python 2.2.1.

The Entity and Notation node types are now supported.

These additional Node attributes have been implemented or changed:

toxml([encoding])

The optional encoding argument has been added to the toxml() method. When specified and not None, the output document uses the given encoding. Changed in version 0.8: the encoding argument was added.

isSupported(feature, version)

This is equivalent to calling hasFeature(feature, version) on the corresponding DOMImplementation object. Added in DOM Level 2 Core.

getInterface(feature)

Return the interface object for the current node that supports the feature feature if isSupported(feature, None) returns true, otherwise returns None. It is not expected that Python DOM implementations will normally need this, but a DOM implementation that adds substantial new functionality may want to require the use of this method to provide access to a helper object that implements extension-specific methods. Added in DOM Level 3.

getUserData(key)

Retrieve the data registered with the node for the key key using setUserData(). Returns None if no data was registered for key. Added in DOM Level 3.

setUserData(key, data, handler)

Set the object data to be associated with a key key on the current node. key can be any hashable object, and will be used as a dictionary key. Any previous value registered for the same value of key will be discarded.

The handler argument should be None or an implementation of the UserDataHandler interface. Added in DOM Level 3.

These additional Text attributes have been added based on the Level 3 drafts. These are also available on CDATASection nodes.

wholeText
: Returns the textual content for a contiguous range of nodes of type TEXT_NODE and CDATA_SECTION_NODE nodes containing the current node.

  replaceWholeText(content)

  Replace a contiguous range of nodes of type TEXT_NODE and CDATA_SECTION_NODE nodes containing the current node, with a single node containing the text content. Returns the node containing content. If content is an empty string, removes the entire range of affected nodes and returns None.

These methods and attributes have been added to the Document interface:

actualEncoding
: The encoding used by the parser, if it was overridden by source context (such as HTTP headers) rather than determined based on the bytes of the source document. If only the source document was used to determine the encoding, this is None Added in DOM Level 3.

encoding
: The encoding specified in the XML declaration, or None if the declaration or encoding pseudo-attribute were omitted. Added in DOM Level 3.

standalone
: If the standalone pseudo-attribute was given in the document's XML declaration, this will be True if the value was "yes" or False if the value was "no". If the XML declaration or standalone pseudo-attribute were omitted, this will be None. Added in DOM Level 3.

version
: The value of the version pseudo-attribute from the XML declaration, if present, or None. Added in DOM Level 3.

  importNode(node, deep)

  Imports a node node from another document. If deep is true, child nodes are recursively imported. Nodes of type DOCUMENT_NODE and DOCUMENT_TYPE_NODE cannot be imported; if node has one of these values for its nodeType attribute, xml.dom.NotSupportedErr will be raised. Added in DOM Level 2.

  renameNode(node, namespaceURI, name)

  Added in DOM Level 3.

Implementation specific behavior: The DOM Level 2 Core specification leaves some room for differing behaviors for how some node types are handled by the Node.cloneNode() method. Before PyXML 0.8.1, the specific behavior exhibited by PyXML varied as an accident of implementation. Starting with PyXML 0.8.1, the following behavior is considered intentional and will be maintained.

When called on a Document node, cloneNode() with the deep argument set to true will create a new document with the children recursively imported into the new document. When deep is false, cloneNode() will return None, as the operation is no reasonable meaning for the operation.

When called on a DocumentType node, cloneNode() will return None if it is owned by a document. If it is not owned, a new document type node is created with all of the attributes copied over, except for the entities and notations fields. If deep is true, these will be new NamedNodeMap objects which hold clones of the Entity and Notation nodes, as appropriate. If deep is false, these will be initialized to new, empty NamedNodeMap objects.

4 xml.dom.xmlbuilder -- DOM Level 3 Load/Save Interface

New in version 0.8.

Warning: The xml.dom.xmlbuilder represents the DOM Level 3 Load/Save interface, which is only defined in a W3C working draft at this time. The Python API presented here is modelled on the 25 July 2002 version of the working draft, and is expected to change as new drafts are released. Backward compatibility to support older versions of this interface will not be preserved.

This module provides API support for the DOM Level 3 Load/Save specification. It includes an implementation of the loading components of that specification only at this time.

Tow groups of classes are provided. The first group implements the objects specific to the Load/Save specification, and the second provides a group of mixin classes that can be used by a DOM implementation to make use of the first group of classes.

Implementation classes:

DOMInputSource DOMEntityResolver DOMBuilder DOMBuilderFilter

Mixin classes:

class DOMImplementationLS
: Class that can be mixed into an implementation of the DOMImplementation interface. This implementation provides the MODE_SYNCHRONOUS and MODE_ASYNCHRONOUS constants and the createDOMBuilder(), createDOMWriter(), and createDOMInputSource() methods. Most DOM implementations should be able to re-use the createDOMInputSource() method, and will need to override the createDOMBuilder() method if it will actually be using a different DOM builder. The createDOMWriter() method should be usable for all implementations once the DOMWriter has been implemented, but that has not yet been done in PyXML.

class DocumentLS
: Class that can be mixed in to a Document implementation to provide access to features of the Load/Save interface. There isn’t much that can be provided by this implementation, so most methods raise NotSupportedErr.

4.1 DOMImplementationLS Extensions

The DOMImplementationLS mixin class is designed to be used in an implementation of the DOMImplementation interface. This class provides the constants MODE_SYNCHRONOUS and MODE_ASYNCHRONOUS, and the following methods:

createDOMBuilder(mode, schemaType)

Returns a DOMBuilder instance. Specific DOM implementations will usually need to override this to return a specialized subclass of the DOMBuilder class; see the documentation on the DOMBuilder for information on how and when to write a subclass of that.

createDOMWriter( )

For now, raises NotImplementedError since the writer interface has not been implemented yet.

createDOMInputSource( )

Returns a DOMInputSource instance with all attributes set to None.

4.2 DocumentLS Extensions

The DocumentLS mixin class for a Document adds the following methods:

load(uri)

Load a new document from a URI into this DOM Document instance. This is not yet implemented in PyXML.

loadXML(source)

Load a new document from a DOMInputSource into this DOM Document instance. This is not yet implemented in PyXML.

saveXML(snode)

Return an XML serialization of the DOM node snode as a string. snode must belong to this document; if not, xml.dom.WrongDocumentErr will be raised.

The following attribute is also added:

async

If set to a true value, the load() and loadXML() methods should load documents asynchronously. If false, these methods will operate in a synchronous mode. PyXML does not support setting this to true.

4.3 DOMBuilder Objects

The DOMBuilder class provides support for configuring the DOM construction process, and allows alternate DOM construction methods to be provided by subclasses.

The general public API has two aspects, configuration and Document

creation. The configuration aspect provides the following attributes and methods:

canSetFeature(name, state)

Return true if the feature name can be set to state, otherwise returns false. If feature name is not supported, returns false.

getFeature(name)

Returns the state for the feature name. If name is unrecognized or not supported, xml.dom.NotFoundErr is raised.

setFeature(name, state)

Set the feature name to state. If feature name is not recognized, xml.dom.NotFoundErr is raised; if the specific state requested is not supported, xml.dom.NotSupportedErr is raised.

supportsFeature(name)

Returns true if the feature name is supported at all, otherwise returns false. A true return does not imply that any particular value for the feature is supported.

The Document-creation aspect of the public API is provided by these methods:

parse(input)

Returns a document based on the DOMInputSource given as input.

parseURI(uri)

Returns a document from the URI given by uri.

parseWithContext(input, cnode, action)

Note: Not implemented in the current version.

Legal values for action are given by the constants ACTION_REPLACE, ACTION_APPEND_AS_CHILDREN, ACTION_INSERT_AFTER, and ACTION_INSERT_BEFORE, all defined on the DOMBuilder class.

Subclasses are expected to define the following method to determine how the Document instances returned by the parse() method will be created.

Warning: The interface for subclasses is very preliminary, and should be considered likely to change in future releases.

_parse_bytestream(stream, options)

Returns a Document instance parsed from the file object given as stream using the configuration options in the options object. The default implementation uses xml.parsers.expat to construct documents using the xml.dom.minidom DOM implementation.

4.4 Subclassing DOMBuilder

There are two important aspects to subclassing the DOMBuilder: implementing a useful subclass and getting a DOMImplementation that uses it.

The first aspect is faily easy; to create a DOMBuilder that uses a different parser, use a subclass that overrides the _parse_bytestream() method, documented above. The implementation of a specialized DOM builder may not be trivial, but the integration with the provided DOMBuilder class will be reasonably direct.

The second aspect, creating a DOM implementation that uses the new DOMBuilder implementation, is a little more tedious, but not excessively so. The DOMImplementationLS mixin class can be used, and the createDOMBuilder() method overridden to use the new DOMBuilder implementation. This makes less sense if you want to re-use most of an existing DOM implementation, however.

To use a new DOMBuilder with existing DOM implementation code, such as xml.dom.minidom, the easiest approach is to subclass an existing DOMImplementation class. For xml.dom.minidom, this could be done this way:

```
import xml.dom

from xml.dom.minidom import DOMImplementation
from xml.dom.xmlbuilder import DOMBuilder

class MyDOMBuilder(DOMBuilder):
    def __init__(self, implementation):
        self._implementation = implementation

    def _parse_bytestream(self, stream, options):
        raise xml.dom.NotSupportedErr(
            "I'm just an example; don't expect much.")

class MyDOMImplementation(DOMImplementation):
    def createDOMBuilder(self, mode, schemaType):
        # check for the supported mode and schemaType
        if schemaType is not None:
            raise xml.dom.NotSupportedErr(
                "unsupported schema type: %s" % schemaType)
        if mode != DOMImplementation.MODE_SYNCHRONOUS:
            raise xml.dom.NotSupportedErr(
                "asynchronous loading not supported")
        return MyDOMBuilder(self)

# minidom just stores the implementation on the class instance,
# so we need to do the same to override that attribute on instances.
#
MyDOMImplementation.implementation = MyDOMImplementation()
```

5 xml.ns -- XML Namespace constants

This module contains the definitions of namespaces (and sometimes other URI's) used by a variety of XML standards. Each class has a short all-uppercase name, which should follow any (emerging) convention for how that standard is commonly used. For example, "ds" is almost always used as the namespace prefixes for items in XML

Signature, so "DS" is the class name. Attributes within that class define symbolic names (hopefully evocative) for ‘‘constants’’ used in that standard.

class XMLNS
: The Namespaces in XML recommendation defines the concept and syntactic constructs relating to XML namespaces.

BASE
: The namespace URI assigned to namespace declarations. This is assigned to attributes named xmlns and attributes which have a namespace prefix of xmlns.

XML
: The namespace bound to this URI is used for all elements and attributes which start with the letters "xml", regardless of case. No other elements or attributes are allowed to use this namespace.

HTML
: This namespace is recommended for use with HTML 4.0.

class XLINK
: The XML Linking Language defines document linking semantics and an attribute language that allows these semantics to be expressed in XML documents.

BASE
: The URI of the global attributes defined in the XLink specification. All attributes that define the presence and behavior of links are in this namespace.

class SOAP
: Simple Object Access Protocol defines a means of communicating with objects on servers. It can be used as a remote procedure call (RPC) mechanism, or as a basis for message passing systems.

ENV
: This URI is used for the namespace of the ‘‘envelope’’ which contains the message. Elements in this namespace provide for destination identification and other information needed to route and decode the message.

ENC
: The namespace URI used for the optional payload encoding defined in section 5 of the SOAP specification.

ACTOR_NEXT
: The URI specified in section 4.2.2 of the SOAP specification which is used to indicate the destination of a SOAP message.

class DSIG
: The namespace URIs given here are defined by the XML digital signature specification.

BASE
: The basic namespace defined by the specification.

C14N

The URI by which Canonical XML (Version 1.0) is identified when used as a transformation or canonicalization method.

C14N_COMM

This URI identifies ‘‘canonical XML with comments,’’ as described in Canonical XML (Version 1.0), section 2.1.

C14N_EXCL

The URI by which the canonicalization variant defined in Exclusive XML Canonicalization (Version 1.0) is identified when used as a transformation or canonicalization method.

The specification also assigns URIs to specific methods of computing message digests and signatures, and other encoding techniques used in the specification.

DIGEST_SHA1

The URI for the SHA-1 digest method.

DIGEST_MD2

The URI for the MD2 digest method.

DIGEST_MD5

The URI for the MD5 digest method.

SIG_DSA_SHA1

The URI used to specify the Digital Signature Algorithm (DSA) with the SHA-1 hash algorithm. DSA is specified in FIPS PUB 186-2, Digital Signature Standard (DSS).

SIG_RSA_SHA1

The URI indicating the RSA signature algorithm using SHA-1 for the secure hash.

HMAC_SHA1

URI for the SHA-1 HMAC algorithm.

ENC_BASE64

URI used to denote the base64 encoding and transform.

ENVELOPED

URI used to specify the enveloped signature transform method (section 6.6.4 of the specification).

XPATH

URI used to specify the XPath filtering transform method (section 6.6.3 of the specification).

XSLT

URI used to specify the XSLT transform method (section 6.6.5 of the specification).

class RNG

The URIs provided here are used with the Relax NG schema language.

BASE

The namespace URI of the elements defined by the Relax NG Specification.

class SCHEMA

BASE

XSD1

XSD2

XSD3

XSI1

XSI2

XSI3

Two additional convenience attributes are defined:

XSD_LIST
A sequence of all ... namespaces.

XSI_LIST
A sequence of all ... namespaces.

class XSLT
XSLT, defined in XML Stylesheet Language -- Transformations, defines a single namespace:

BASE
This URI is used as the namespace for all XSLT elements and for XSLT attributes attached to non-XSLT elements.

class WSDL
The Web Services Description Language (WSDL) defines a language to specify the logical interactions with applications that use Web technologies as their access mechanism; this can be thought of as an IDL for servers that speak HTTP instead of XDR or IIOP.

BASE
The basic namespace defined in this specification.

BIND_SOAP
The URI of the SOAP binding for WSDL.

BIND_HTTP
HTTP bindings for WSDL using the GET and POST methods.

BIND_MIME
The URI of the namespace for MIME-type bindings for WSDL.

6 xml.parsers.expat Extensions

The version of the xml.parsers.expat module currently shipped with PyXML extends that provided by the standard Python library by exposing features of recent versions of the C implementation of the underlying Expat parser. These extensions are described here; the base

documentation for this module is that published as part of the Python Library Reference.

The module provides a new data attribute:

features
: A list of name-value pairs giving some information about the compilation of Expat being used. This is generated directly from the feature information provided by Expat's XML_GetFeatureList() function. It is unlikely to be of interest to most applications.

  This was added in PyXML 0.8.1 to support Expat 1.95.5 and newer.

The parser provides a new method:

: UseForeignDTD([flag])

  Tells Expat whether is should attempt to load an application-provided external subset if one is not specified by the document type declaration. If flag is true or omitted, Expat will attempt to load an external subset by calling the ExternalEntityRefHandler callback with systemId and publicId both set to None; this is done only if the document does not specify a external subset. If flag is false (or this method is never called), Expat will not attempt such a load. This method should only be called before parsing has actually started; ExpatError will be raised if this is called after parsing has begun.

  This was added in PyXML 0.8.1 to support Expat 1.95.5 and newer.

and one new attribute:

namespace_prefixes
: Set to true on a parser with namespaces enabled to request that the actual prefix is reported as well as the namespace URI for each namespace-qualified name. This modifies the element or attribute names passed to the StartElementHandler and EndElementHandler callbacks, if true. If set to a true value, the names passed to these callbacks will have the prefix added to the end, separated from the local name by the value of the namespace_separator passed to the parser constructor. No additional information will be added if the name uses the default namespace. This should only be called before parsing has begun.

  This was added in PyXML 0.8 to support Expat 1.95.4 and newer.

One new callback method has been added as well:

: SkippedEntityHandler(name, is_param_entity)

  This is called when an external entity is not read, and gives information about the entity. The entity name will have been previously reported by the EntityDeclHandler, if set.

  This was added in PyXML 0.8 to support Expat 1.95.4 and newer.

See Also:

Expat Home Page
: The home page for the Expat project provides information about new versions of the library and user resources such as 3rd-party language bindings and mailing lists.

7 xml.parsers.sgmllib -- Accelerated SGML Parser

This module is an alternate implementation of the sgmllib module from the Python standard library. This implementation uses the xml.parsers.sgmlop accelerator to improve performance.

This module does create a cyclic reference. In order to break the cycle, be sure to call the close() method of the parser instances when done with them.

8 xml.parsers.sgmlop -- XML/SGML Parser Accelerator

The xml.parsers.sgmlop module is a C implementation of a parser similar in interface to the xmllib.XMLParser and sgmllib.SGMLParser classes from the Python standard library. Additional support is provided for a basic tree-constructor which is intended to be used in conjunction with the parsers implemented by this module.

9 xml.sax.saxexts

make_parser([parser])

: A utility function that returns a Parser object for a non-validating XML parser. If parser is specified, it must be a parser name; otherwise, a list of available parsers is checked and the fastest one chosen.

HTMLParserFactory
: An instance of the ParserFactory class that’s already been prepared with a list of HTML parsers. Simply call its make_parser() method to get a Parser object.

class ParserFactory( )

: A general class to be used by applications for creating parsers on foreign systems where the list of installed parsers is unknown.

SGMLParserFactory
: An instance of the ParserFactory class that’s already been prepared with a list of SGML parsers. Simply call its make_parser() method to get a parser object.

XMLParserFactory
: An instance of the ParserFactory class that’s already been prepared with a list of nonvalidating XML parsers. Simply call its make_parser() method to get a parser object.

XMLValParserFactory
: An instance of the ParserFactory class that’s already been prepared with a list of validating XML parsers. Simply call its make_parser() method to get a parser object.

class ExtendedParser( )

This class is an experimental extended parser interface, that offers additional functionality that may be useful. However, it’s not specified by the SAX specification.

9.1 ExtendedParser methods

close( )

Called after the last call to feed, when there are no more data.

feed(data)

Feeds data to the parser.

get_parser_name( )

Returns a single-word parser name.

get_parser_version( )

Returns the version of the imported parser, which may not be the one the driver was implemented for.

is_dtd_reading( )

True if the parser is non-validating, but conforms to the XML specification by reading the DTD.

is_validating( )

Returns true if the parser is validating, false otherwise.

reset( )

Makes the parser start parsing afresh.

9.2 ParserFactory methods

get_parser_list( )

Returns the list of possible drivers. Currently this starts out as ["xml.sax.drivers.drv_xmltok",
"xml.sax.drivers.drv_xmlproc",
"xml.sax.drivers.drv_xmltoolkit",
"xml.sax.drivers.drv_xmllib"].

make_parser([driver_name])

Returns a SAX driver for the first available parser of the parsers in the list. Note that the list contains drivers, so it first tries the driver and if that exists imports it to see if the parser also exists. If no parsers are available a SAXException is thrown.

Optionally, driver_name can be a string containing the name of the driver to be used; the stored parser list will then not be used at all.

set_parser_list(list)

Sets the driver list to list.

class AttributeList( )

Interface for an attribute list. This interface provides information about a list of attributes for an element (only specified or defaulted attributes will be reported). Note that the information returned by this object will be valid only during the scope of the DocumentHandler.startElement callback, and the attributes will not necessarily be provided in the order declared or specified.

class DocumentHandler( )

Handle general document events. This is the main client interface for SAX: it contains callbacks for the most important document events, such as the start and end of elements. You need to create an object that implements this interface, and then register it with the Parser. If you do not want to implement the entire interface, you can derive a class from HandlerBase, which implements the default functionality. You can find the location of any document event using the Locator interface supplied by setDocumentLocator().

class DTDHandler( )

Handle DTD events. This interface specifies only those DTD events required for basic parsing (unparsed entities and attributes). If you do not want to implement the entire interface, you can extend HandlerBase, which implements the default behaviour.

class EntityResolver( )

This is the basic interface for resolving entities. If you create an object implementing this interface, then register the object with your Parser instance, the parser will call the method in your object to resolve all external entities. Note that HandlerBase implements this interface with the default behaviour.

class ErrorHandler( )

This is the basic interface for SAX error handlers. If you create an object that implements this interface, then register the object with your Parser, the parser will call the methods in your object to report all warnings and errors. There are three levels of errors available: warnings, (possibly) recoverable errors, and unrecoverable errors. All methods take a SAXParseException as the only parameter.

class HandlerBase( )

Default base class for handlers. This class implements the default behaviour for four SAX interfaces, inheriting from them

all: EntityResolver, DTDHandler, DocumentHandler, and ErrorHandler. Rather than implementing those full interfaces, you may simply extend this class and override the methods that you need. Note that the use of this class is optional, since you are free to implement the interfaces directly if you wish.

class Locator( )

Interface for associating a SAX event with a document location. A locator object will return valid results only during calls to methods of the SAXDocumentHandler class; at any other time, the results are unpredictable.

class Parser( )

Basic interface for SAX parsers. All SAX parsers must implement this basic interface: it allows users to register handlers for different types of events and to initiate a parse from a URI, a character stream, or a byte stream. SAX parsers should also implement a zero-argument constructor.

class SAXException(msg, exception, locator)

Encapsulate an XML error or warning. This class can contain basic error or warning information from either the XML parser or the application: you can subclass it to provide additional functionality, or to add localization. Note that although you will receive a SAXException as the argument to the handlers in the ErrorHandler interface, you are not actually required to throw the exception; instead, you can simply read the information in it.

class SAXParseException(msg, exception, locator)

Encapsulate an XML parse error or warning.

This exception will include information for locating the error in the original XML document. Note that although the application will receive a SAXParseException as the argument to the handlers in the ErrorHandler interface, the application is not actually required to throw the exception; instead, it can simply read the information in it and take a different action.

Since this exception is a subclass of SAXException, it inherits the ability to wrap another exception.

10.1 AttributeList methods

The AttributeList class supports some of the behaviour of Python dictionaries; the len() function and has_key(), keys() methods are available, and attr['href'] will retrieve the value of the href attribute. There are also additional methods specific to AttributeList:

getLength( )

Return the number of attributes in the list.

getName(i)

Return the name of attribute i in the list.

getType(i)

Return the type of an attribute in the list. i can be either the integer index or the attribute name.

getValue(i)

Return the value of an attribute in the list. i can be either the integer index or the attribute name.

10.2 DocumentHandler methods

characters(ch, start, length)

Handle a character data event.

endDocument( )

Handle an event for the end of a document.

endElement(name)

Handle an event for the end of an element.

ignorableWhitespace(ch, start, length)

Handle an event for ignorable whitespace in element content.

processingInstruction(target, data)

Handle a processing instruction event.

setDocumentLocator(locator)

Receive an object for locating the origin of SAX document events. You'll probably want to store the value of locator as an attribute of the handler instance.

startDocument( )

Handle an event for the beginning of a document.

startElement(name, attrs)

Handle an event for the beginning of an element.

10.3 DTDHandler methods

notationDecl(name, publicId, systemId)

Handle a notation declaration event.

unparsedEntityDecl(publicId, systemId, notationName)

Handle an unparsed entity declaration event.

10.4 EntityResolver methods

resolveEntity(name, publicId, systemId)

Resolve the system identifier of an entity.

10.5 ErrorHandler methods

error(exception)

Handle a recoverable error.

fatalError(exception)

Handle a non-recoverable error.

warning(exception)

Handle a warning.

10.6 Locator methods

getColumnNumber( )

Return the column number where the current event ends.

getLineNumber( )

Return the line number where the current event ends.

getPublicId( )

Return the public identifier for the current event.

getSystemId( )

Return the system identifier for the current event.

10.7 Parser methods

parse(systemId)

Parse an XML document from a system identifier.

parseFile(fileobj)

Parse an XML document from a file-like object.

setDocumentHandler(handler)

Register an object to receive basic document-related events.

setDTDHandler(handler)

Register an object to receive basic DTD-related events.

setEntityResolver(resolver)

Register an object to resolve external entities.

setErrorHandler(handler)

Register an object to receive error-message events.

setLocale(locale)

Allow an application to set the locale for errors and warnings.

SAX parsers are not required to provide localisation for errors and warnings; if they cannot support the requested locale, however, they must throw a SAX exception. Applications may request a locale change in the middle of a parse.

10.8 SAXException methods

getException( )

Return the embedded exception, if any.

getMessage( )

Return a message for this exception.

10.9 SAXParseException methods

The SAXParseException class has a locator attribute, containing an instance of the Locator class, which represents the location in the document where the parse error occurred. The following methods are delegated to this instance.

getColumnNumber( )

Return the column number of the end of the text where the exception occurred.

getLineNumber( )

Return the line number of the end of the text where the exception occurred.

getPublicId( )

Return the public identifier of the entity where the exception occurred.

getSystemId( )

Return the system identifier of the entity where the exception occurred.

11 xml.sax.saxutils

escape(data[, entities])

Escape "&", "<", and ">" in a string of data.

You can escape other strings of data by passing a dictionary as the optional entities parameter. The keys and values must all be strings; each key will be replaced with its corresponding value.

quoteattr(data[, entities])

Similar to escape(), but also prepares data to be used as an attribute value. The return value is a quoted version of data with any additional required replacements. quoteattr() will select a quote character based on the content of data, attempting to avoid encoding any quote characters in the string. If both single- and double-quote characters are already in data, the double-quote characters will be encoded and data will be wrapped in doule-quotes. The resulting string can be used directly as an attribute value:

```
>>> print "<element attr=%s>" % quoteattr("ab ’ cd \" ef")
<element attr="ab ’ cd &quot; ef">
```

This function is useful when generating attribute values for HTML or any SGML using the reference concrete syntax.

class Canonizer(writer)

A SAX document handler that produces canonicalized XML output. writer must support a write() method which accepts a single string.

class ErrorPrinter( )

A simple class that just prints error messages to standard error (sys.stderr).

class ESISDocHandler(writer)

A SAX document handler that produces naive ESIS output. writer must support a write() method which accepts a single string.

class EventBroadcaster(list)

Takes a list of objects and forwards any method calls received to all objects in the list. The attribute list holds the list and can freely be modified by clients.

class Location(locator)

Represents a location in an XML entity. Initialized by being passed a locator, from which it reads off the current location, which is then stored internally.

11.1 Location methods

getColumnNumber( )

Return the column number of the location.

getLineNumber( )

Return the line number of the location.

getPublicId( )

Return the public identifier for the location.

getSystemId( )

Return the system identifier for the location.

12 xml.utils.iso8601

The xml.utils.iso8601 module provides conversion routines between the ISO 8601 representations of date/time values and the floating point values used elsewhere in Python. The floating point representation is particularly useful in conjunction with the standard time module.

Currently, this module supports a small superset of the ISO 8601 profile described by the World Wide Web Consortium (W3C). This is a subset of ISO 8601, but covers the cases expected to be used most often in the context of XML processing and Web applications. Future versions of this module may support a larger subset of ISO 8601-defined formats.

parse(s)

Parse an ISO 8601 date representation (with an optional time-of-day component) and return the date in seconds since the epoch.

parse_timezone(timezone)

Parse an ISO 8601 time zone designator and return the offset relative to Universal Coordinated Time (UTC) in seconds. If timezone is not valid, ValueError is raised.

tostring(t[, timezone])

Return formatted date/time value according to the profile described by the W3C. If timezone is provided, it must be the offset from UTC in seconds specified as a number, or time zone designator which can be parsed by parse_timezone(). If timezone is specified as a string and cannot be parsed by parse_timezone(), ValueError will be raised.

ctime(t)

Return formatter date/time value using the local timezone. This is equivalent to "tostring(t, time.timezone)".

See Also:

Data elements and interchange formats -- Information interchange -- Representation of dates and times.
The actual ISO 8601 standard published by the International Organization for Standardization, 1988.

ISO 8601 date/time representations
Gary Houston’s description of the ISO 8601 formats for humans, written in January 1993.

A Summary of the International Standard Date and Time Notation
Markus Kuhn’s excellent discussion of international date/time representations.

Date and Time Formats
World Wide Web Consortium Technical Note from September 1998, written by Misha Wolf and Charles Wicksteed.

About this document ...

Python/XML Reference Guide

This document was generated using the LaTeX2HTML translator.

The application of LaTeX2HTML to the Python documentation has been heavily tailored by Fred L. Drake, Jr. Original navigation icons were contributed by Christopher Petrilli.

Python/XML Reference Guide

Release 0.8.