20.5. `xml.parsers.expat` — Fast XML parsing using Expat¶

New in version 2.0.

The `xml.parsers.expat` module is a Python interface to the Expat non-validating XML parser. The module provides a single extension type, `xmlparser`, that represents the current state of an XML parser. After an `xmlparser` object has been created, various attributes of the object can be set to handler functions. When an XML document is then fed to the parser, the handler functions are called for the character data and markup in the XML document.

This module uses the `pyexpat` module to provide access to the Expat parser. Direct use of the `pyexpat` module is deprecated.

This module provides one exception and one type object:

exception `xml.parsers.expat.ExpatError`¶

The exception raised when Expat reports an error. See section *ExpatError Exceptions* for more information on interpreting Expat errors.

exception `xml.parsers.expat.error`¶

Alias for `ExpatError`.

`xml.parsers.expat.XMLParserType`¶

The type of the return values from the `ParserCreate()` function.

The `xml.parsers.expat` module contains two functions:

`xml.parsers.expat.ErrorString`(*errno*)¶

Returns an explanatory string for a given error number *errno*.

`xml.parsers.expat.ParserCreate`([*encoding*[, *namespace_separator*]])¶

Creates and returns a new `xmlparser` object. *encoding*, if specified, must be a string naming the encoding used by the XML data. Expat doesn't support as many encodings as Python does, and its repertoire of encodings can't be extended; it supports UTF-8, UTF-16, ISO-8859-1 (Latin1), and ASCII. If *encoding* [1] is given it will override the implicit or explicit encoding of the document.

Expat can optionally do XML namespace processing for you, enabled by providing a value for *namespace_separator*. The value must be a one-character string; a `ValueError` will be raised if the string has an illegal length (`None` is considered the same as omission). When namespace processing is enabled, element type names and attribute names that belong to a namespace will be expanded. The element name passed to the element handlers `StartElementHandler` and `EndElementHandler` will be the concatenation of the namespace URI, the namespace separator character, and the local part of the name. If the namespace separator is a zero byte (`chr(0)`) then the namespace URI and the local part will be concatenated without any separator.

For example, if *namespace_separator* is set to a space character (`' '`) and the following document is parsed:

```
<?xml version="1.0"?>
<root xmlns    = "http://default-namespace.org/"
     xmlns:py = "http://www.python.org/ns/">
 <py:elem1 />
 <elem2 xmlns="" />
</root>
```

`StartElementHandler` will receive the following strings for each element:

```
http://default-namespace.org/ root
http://www.python.org/ns/ elem1
elem2
```

See also

The Expat XML Parser

Home page of the Expat project.

20.5.1. XMLParser Objects¶

`xmlparser` objects have the following methods:

`xmlparser.Parse`(*data*[, *isfinal*])¶

Parses the contents of the string *data*, calling the appropriate handler functions to process the parsed data. *isfinal* must be true on the final call to this method. *data* can be the empty string at any time.

`xmlparser.ParseFile`(*file*)¶

Parse XML data reading from the object *file*. *file* only needs to provide the `read(nbytes)` method, returning the empty string when there's no more data.

`xmlparser.SetBase`(*base*)¶

Sets the base to be used for resolving relative URIs in system identifiers in declarations. Resolving relative identifiers is left to the application: this value will be passed through as the *base* argument to the `ExternalEntityRefHandler()`, `NotationDeclHandler()`, and `UnparsedEntityDeclHandler()` functions.

`xmlparser.GetBase`()¶

Returns a string containing the base set by a previous call to `SetBase()`, or `None` if `SetBase()` hasn't been called.

`xmlparser.GetInputContext`()¶

Returns the input data that generated the current event as a string. The data is in the encoding of the entity which contains the text. When called while an event handler is not active, the return value is `None`.

New in version 2.1.

`xmlparser.ExternalEntityParserCreate`(*context*[, *encoding*])¶

Create a "child" parser which can be used to parse an external parsed entity referred to by content parsed by the parent parser. The *context* parameter should be the string passed to the `ExternalEntityRefHandler()` handler function, described below. The child parser is created with the `ordered_attributes`, `returns_unicode` and `specified_attributes` set to the values of this parser.

`xmlparser.UseForeignDTD`([*flag*])¶

Calling this with a true value for *flag* (the default) will cause Expat to call the `ExternalEntityRefHandler` with `None` for all arguments to allow an alternate DTD to be loaded. If the document does not contain a document type declaration, the `ExternalEntityRefHandler` will still be called, but the `StartDoctypeDeclHandler` and `EndDoctypeDeclHandler` will not be called.

Passing a false value for *flag* will cancel a previous call that passed a true value, but otherwise has no effect.

This method can only be called before the `Parse()` or `ParseFile()` methods are called; calling it after either of those have been called causes `ExpatError` to be raised with the `code` attribute set to `errors.XML_ERROR_CANT_CHANGE_FEATURE_ONCE_PARSING`.

New in version 2.3.

`xmlparser` objects have the following attributes:

`xmlparser.buffer_size`¶

The size of the buffer used when `buffer_text` is true. A new buffer size can be set by assigning a new integer value to this attribute. When the size is changed, the buffer will be flushed.

New in version 2.3.

Changed in version 2.6: The buffer size can now be changed.

`xmlparser.buffer_text`¶

Setting this to true causes the `xmlparser` object to buffer textual content returned by Expat to avoid multiple calls to the `CharacterDataHandler()` callback whenever possible. This can improve performance substantially since Expat normally breaks character data into chunks at every line ending. This attribute is false by default, and may be changed at any time.

New in version 2.3.

`xmlparser.buffer_used`¶

If `buffer_text` is enabled, the number of bytes stored in the buffer. These bytes represent UTF-8 encoded text. This attribute has no meaningful interpretation when `buffer_text` is false.

New in version 2.3.

`xmlparser.ordered_attributes`¶

Setting this attribute to a non-zero integer causes the attributes to be reported as a list rather than a dictionary. The attributes are presented in the order found in the document text. For each attribute, two list entries are presented: the attribute name and the attribute value. (Older versions of this module also used this format.) By default, this attribute is false; it may be changed at any time.

New in version 2.1.

`xmlparser.returns_unicode`¶

If this attribute is set to a non-zero integer, the handler functions will be passed Unicode strings. If `returns_unicode` is `False`, 8-bit strings containing UTF-8 encoded data will be passed to the handlers. This is `True` by default when Python is built with Unicode support.

Changed in version 1.6: Can be changed at any time to affect the result type.

`xmlparser.specified_attributes`¶

If set to a non-zero integer, the parser will report only those attributes which were specified in the document instance and not those which were derived from attribute declarations. Applications which set this need to be especially careful to use what additional information is available from the declarations as needed to comply with the standards for the behavior of XML processors. By default, this attribute is false; it may be changed at any time.

New in version 2.1.

The following attributes contain values relating to the most recent error encountered by an `xmlparser` object, and will only have correct values once a call to `Parse()` or `ParseFile()` has raised a `xml.parsers.expat.ExpatError` exception.

`xmlparser.ErrorByteIndex`¶

Byte index at which an error occurred.

`xmlparser.ErrorCode`¶

Numeric code specifying the problem. This value can be passed to the `ErrorString()` function, or compared to one of the constants defined in the `errors` object.

`xmlparser.ErrorColumnNumber`¶

Column number at which an error occurred.

`xmlparser.ErrorLineNumber`¶

Line number at which an error occurred.

The following attributes contain values relating to the current parse location in an `xmlparser` object. During a callback reporting a parse event they indicate the location of the first of the sequence of characters that generated the event. When called outside of a callback, the position indicated will be just past the last parse event (regardless of whether there was an associated callback).

New in version 2.4.

`xmlparser.CurrentByteIndex`¶

Current byte index in the parser input.

`xmlparser.CurrentColumnNumber`¶

Current column number in the parser input.

`xmlparser.CurrentLineNumber`¶

Current line number in the parser input.

Here is the list of handlers that can be set. To set a handler on an `xmlparser` object *o*, use `o.handlername = func`. *handlername* must be taken from the following list, and *func* must be a callable object accepting the correct number of arguments. The arguments are all strings, unless otherwise stated.

`xmlparser.XmlDeclHandler`(*version*, *encoding*, *standalone*)¶

Called when the XML declaration is parsed. The XML declaration is the (optional) declaration of the applicable version of the XML recommendation, the encoding of the document text, and an optional “standalone” declaration. *version* and *encoding* will be strings of the type dictated by the `returns_unicode` attribute, and *standalone* will be `1` if the document is declared standalone, `0` if it is declared not to be standalone, or `-1` if the standalone clause was omitted. This is only available with Expat version 1.95.0 or newer.

New in version 2.1.

`xmlparser.StartDoctypeDeclHandler`(*doctypeName*, *systemId*, *publicId*, *has_internal_subset*)¶

Called when Expat begins parsing the document type declaration (`<!DOCTYPE ...`). The *doctypeName* is provided exactly as presented. The *systemId* and *publicId* parameters give the system and public identifiers if specified, or `None` if omitted. *has_internal_subset* will be true if the document contains and internal document declaration subset. This requires Expat version 1.2 or newer.

`xmlparser.EndDoctypeDeclHandler`()¶

Called when Expat is done parsing the document type declaration. This requires Expat version 1.2 or newer.

`xmlparser.ElementDeclHandler`(*name*, *model*)¶

Called once for each element type declaration. *name* is the name of the element type, and *model* is a representation of the content model.

`xmlparser.AttlistDeclHandler`(*elname*, *attname*, *type*, *default*, *required*)¶

Called for each declared attribute for an element type. If an attribute list declaration declares three attributes, this handler is called three times, once for each attribute. *elname* is the name of the element to which the declaration applies and *attname* is the name of the attribute declared. The attribute type is a string passed as *type*; the possible values are `'CDATA'`, `'ID'`, `'IDREF'`, ... *default* gives the default value for the attribute used when the attribute is not specified by the document instance, or `None` if there is no default value (`#IMPLIED` values). If the attribute is required to be given in the document instance, *required* will be true. This requires Expat version 1.95.0 or newer.

`xmlparser.StartElementHandler`(*name*, *attributes*)¶

Called for the start of every element. *name* is a string containing the element name, and *attributes* is a dictionary mapping attribute names to their values.

`xmlparser.EndElementHandler`(*name*)¶
Called for the end of every element.

`xmlparser.ProcessingInstructionHandler`(*target*, *data*)¶
Called for every processing instruction.

`xmlparser.CharacterDataHandler`(*data*)¶
Called for character data. This will be called for normal character data, CDATA marked content, and ignorable whitespace. Applications which must distinguish these cases can use the `StartCdataSectionHandler`, `EndCdataSectionHandler`, and `ElementDeclHandler` callbacks to collect the required information.

`xmlparser.UnparsedEntityDeclHandler`(*entityName*, *base*, *systemId*, *publicId*, *notationName*)¶
Called for unparsed (NDATA) entity declarations. This is only present for version 1.2 of the Expat library; for more recent versions, use `EntityDeclHandler` instead. (The underlying function in the Expat library has been declared obsolete.)

`xmlparser.EntityDeclHandler`(*entityName*, *is_parameter_entity*, *value*, *base*, *systemId*, *publicId*, *notationName*)¶

Called for all entity declarations. For parameter and internal entities, *value* will be a string giving the declared contents of the entity; this will be `None` for external entities. The *notationName* parameter will be `None` for parsed entities, and the name of the notation for unparsed entities. *is_parameter_entity* will be true if the entity is a parameter entity or false for general entities (most applications only need to be concerned with general entities). This is only available starting with version 1.95.0 of the Expat library.

New in version 2.1.

`xmlparser.NotationDeclHandler`(*notationName*, *base*, *systemId*, *publicId*)¶
Called for notation declarations. *notationName*, *base*, and *systemId*, and *publicId* are strings if given. If the public identifier is omitted, *publicId* will be `None`.

`xmlparser.StartNamespaceDeclHandler`(*prefix*, *uri*)¶
Called when an element contains a namespace declaration. Namespace declarations are processed before the `StartElementHandler` is called for the element on which declarations are placed.

`xmlparser.EndNamespaceDeclHandler`(*prefix*)¶
Called when the closing tag is reached for an element that contained a namespace declaration. This is called once for each namespace declaration on the element in the reverse of the order for which the `StartNamespaceDeclHandler` was called to indicate the start of each namespace declaration's scope. Calls to this handler are made after the corresponding `EndElementHandler` for the end of the element.

`xmlparser.CommentHandler`(*data*)¶
Called for comments. *data* is the text of the comment, excluding the leading '`<!--`' and trailing '`-->`'.

`xmlparser.StartCdataSectionHandler`()¶
Called at the start of a CDATA section. This and `EndCdataSectionHandler` are needed to be able to identify the syntactical start and end for CDATA sections.

`xmlparser.EndCdataSectionHandler`()¶
Called at the end of a CDATA section.

`xmlparser.DefaultHandler`(*data*)¶
Called for any characters in the XML document for which no applicable handler has been specified. This means characters that are part of a construct which could be reported, but for which no handler has been supplied.

`xmlparser.DefaultHandlerExpand`(*data*)¶
This is the same as the `DefaultHandler()`, but doesn't inhibit expansion of internal entities. The entity reference will not be passed to the default handler.

`xmlparser.NotStandaloneHandler`()¶
Called if the XML document hasn't been declared as being a standalone document. This happens when there is an external subset or a reference to a parameter entity, but the XML declaration does not set standalone to `yes` in an XML declaration. If this handler returns `0`, then the parser will throw an `XML_ERROR_NOT_STANDALONE` error. If this handler is not set, no exception is raised by the parser for this condition.

`xmlparser.ExternalEntityRefHandler`(*context*, *base*, *systemId*, *publicId*)¶

Called for references to external entities. *base* is the current base, as set by a previous call to `SetBase()`. The public and system identifiers, *systemId* and *publicId*, are strings if given; if the public identifier is not given, *publicId* will be `None`. The *context* value is opaque and should only be used as described below.

For external entities to be parsed, this handler must be implemented. It is responsible for creating the sub-parser using `ExternalEntityParserCreate(context)`, initializing it with the appropriate callbacks, and parsing the entity. This handler should return an integer; if it returns `0`, the parser will throw an `XML_ERROR_EXTERNAL_ENTITY_HANDLING` error, otherwise parsing will continue.

If this handler is not provided, external entities are reported by the `DefaultHandler` callback, if provided.

20.5.2. ExpatError Exceptions¶

`ExpatError` exceptions have a number of interesting attributes:

`ExpatError.code`¶

Expat's internal error number for the specific error. This will match one of the constants defined in the `errors` object from this module.

New in version 2.1.

`ExpatError.lineno`¶

Line number on which the error was detected. The first line is numbered `1`.

New in version 2.1.

`ExpatError.offset`¶

Character offset into the line where the error occurred. The first column is numbered `0`.

New in version 2.1.

20.5.3. Example¶

The following program defines three handlers that just print out their arguments.

```
import xml.parsers.expat

# 3 handler functions
def start_element(name, attrs):
   print 'Start element:', name, attrs
def end_element(name):
   print 'End element:', name
def char_data(data):
   print 'Character data:', repr(data)

p = xml.parsers.expat.ParserCreate()

p.StartElementHandler = start_element
p.EndElementHandler = end_element
p.CharacterDataHandler = char_data

p.Parse("""<?xml version="1.0"?>
<parent id="top"><child1 name="paul">Text goes here</child1>
<child2 name="fred">More text</child2>
</parent>""", 1)
```

The output from this program is:

```
Start element: parent {'id': 'top'}
Start element: child1 {'name': 'paul'}
Character data: 'Text goes here'
End element: child1
Character data: '\n'
Start element: child2 {'name': 'fred'}
Character data: 'More text'
End element: child2
Character data: '\n'
End element: parent
```

20.5.4. Content Model Descriptions¶

Content modules are described using nested tuples. Each tuple contains four values: the type, the quantifier, the name, and a tuple of children. Children are simply additional content module descriptions.

The values of the first two fields are constants defined in the `model` object of the `xml.parsers.expat` module. These constants can be collected in two groups: the model type group and the quantifier group.

The constants in the model type group are:

`xml.parsers.expat.XML_CTYPE_ANY`

The element named by the model name was declared to have a content model of `ANY`.

`xml.parsers.expat.XML_CTYPE_CHOICE`

The named element allows a choice from a number of options; this is used for content models such as `(A | B | C)`.

`xml.parsers.expat.XML_CTYPE_EMPTY`

Elements which are declared to be `EMPTY` have this model type.

`xml.parsers.expat.XML_CTYPE_MIXED`

`xml.parsers.expat.XML_CTYPE_NAME`

`xml.parsers.expat.XML_CTYPE_SEQ`

Models which represent a series of models which follow one after the other are indicated with this model type. This is used for models such as `(A, B, C)`.

The constants in the quantifier group are:

`xml.parsers.expat.XML_CQUANT_NONE`
No modifier is given, so it can appear exactly once, as for `A`.
`xml.parsers.expat.XML_CQUANT_OPT`
The model is optional: it can appear once or not at all, as for `A?`.
`xml.parsers.expat.XML_CQUANT_PLUS`
The model must occur one or more times (like `A+`).
`xml.parsers.expat.XML_CQUANT_REP`
The model must occur zero or more times, as for `A*`.

20.5.5. Expat error constants¶

The following constants are provided in the `errors` object of the `xml.parsers.expat` module. These constants are useful in interpreting some of the attributes of the `ExpatError` exception objects raised when an error has occurred.

The `errors` object has the following attributes:

`xml.parsers.expat.XML_ERROR_ASYNC_ENTITY`
`xml.parsers.expat.XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF`
An entity reference in an attribute value referred to an external entity instead of an internal entity.
`xml.parsers.expat.XML_ERROR_BAD_CHAR_REF`
A character reference referred to a character which is illegal in XML (for example, character `0`, or '`&#0;`').
`xml.parsers.expat.XML_ERROR_BINARY_ENTITY_REF`
An entity reference referred to an entity which was declared with a notation, so cannot be parsed.
`xml.parsers.expat.XML_ERROR_DUPLICATE_ATTRIBUTE`
An attribute was used more than once in a start tag.
`xml.parsers.expat.XML_ERROR_INCORRECT_ENCODING`
`xml.parsers.expat.XML_ERROR_INVALID_TOKEN`
Raised when an input byte could not properly be assigned to a character; for example, a NUL byte (value `0`) in a UTF-8 input stream.
`xml.parsers.expat.XML_ERROR_JUNK_AFTER_DOC_ELEMENT`
Something other than whitespace occurred after the document element.
`xml.parsers.expat.XML_ERROR_MISPLACED_XML_PI`
An XML declaration was found somewhere other than the start of the input data.
`xml.parsers.expat.XML_ERROR_NO_ELEMENTS`
The document contains no elements (XML requires all documents to contain exactly one top-level element)..
`xml.parsers.expat.XML_ERROR_NO_MEMORY`
Expat was not able to allocate memory internally.
`xml.parsers.expat.XML_ERROR_PARAM_ENTITY_REF`
A parameter entity reference was found where it was not allowed.
`xml.parsers.expat.XML_ERROR_PARTIAL_CHAR`
An incomplete character was found in the input.
`xml.parsers.expat.XML_ERROR_RECURSIVE_ENTITY_REF`
An entity reference contained another reference to the same entity; possibly via a different name, and possibly indirectly.
`xml.parsers.expat.XML_ERROR_SYNTAX`
Some unspecified syntax error was encountered.
`xml.parsers.expat.XML_ERROR_TAG_MISMATCH`
An end tag did not match the innermost open start tag.
`xml.parsers.expat.XML_ERROR_UNCLOSED_TOKEN`
Some token (such as a start tag) was not closed before the end of the stream or the next token was encountered.
`xml.parsers.expat.XML_ERROR_UNDEFINED_ENTITY`
A reference was made to a entity which was not defined.
`xml.parsers.expat.XML_ERROR_UNKNOWN_ENCODING`
The document encoding is not supported by Expat.
`xml.parsers.expat.XML_ERROR_UNCLOSED_CDATA_SECTION`
A CDATA marked section was not closed.
`xml.parsers.expat.XML_ERROR_EXTERNAL_ENTITY_HANDLING`
`xml.parsers.expat.XML_ERROR_NOT_STANDALONE`

The parser determined that the document was not “standalone” though it declared itself to be in the XML declaration, and the `NotStandaloneHandler` was set and returned `0`.

`xml.parsers.expat.XML_ERROR_UNEXPECTED_STATE`

`xml.parsers.expat.XML_ERROR_ENTITY_DECLARED_IN_PE`

`xml.parsers.expat.XML_ERROR_FEATURE_REQUIRES_XML_DTD`

An operation was requested that requires DTD support to be compiled in, but Expat was configured without DTD support. This should never be reported by a standard build of the `xml.parsers.expat` module.

`xml.parsers.expat.XML_ERROR_CANT_CHANGE_FEATURE_ONCE_PARSING`

A behavioral change was requested after parsing started that can only be changed before parsing has started. This is (currently) only raised by `UseForeignDTD()`.

`xml.parsers.expat.XML_ERROR_UNBOUND_PREFIX`

An undeclared prefix was found when namespace processing was enabled.

`xml.parsers.expat.XML_ERROR_UNDECLARING_PREFIX`

The document attempted to remove the namespace declaration associated with a prefix.

`xml.parsers.expat.XML_ERROR_INCOMPLETE_PE`

A parameter entity contained incomplete markup.

`xml.parsers.expat.XML_ERROR_XML_DECL`

The document contained no document element at all.

`xml.parsers.expat.XML_ERROR_TEXT_DECL`

There was an error parsing a text declaration in an external entity.

`xml.parsers.expat.XML_ERROR_PUBLICID`

Characters were found in the public id that are not allowed.

`xml.parsers.expat.XML_ERROR_SUSPENDED`

The requested operation was made on a suspended parser, but isn’t allowed. This includes attempts to provide additional input or to stop the parser.

`xml.parsers.expat.XML_ERROR_NOT_SUSPENDED`

An attempt to resume the parser was made when the parser had not been suspended.

`xml.parsers.expat.XML_ERROR_ABORTED`

This should not be reported to Python applications.

`xml.parsers.expat.XML_ERROR_FINISHED`

The requested operation was made on a parser which was finished parsing input, but isn’t allowed. This includes attempts to provide additional input or to stop the parser.

`xml.parsers.expat.XML_ERROR_SUSPEND_PE`

Footnotes

[1] The encoding string included in XML output should conform to the appropriate standards. For example, “UTF-8” is valid, but “UTF8” is not. See http://www.w3.org/TR/2006/REC-xml11-20060816/#NT-EncodingDecl and http://www.iana.org/assignments/character-sets .

Table Of Contents