`!xml.sax.handler` --- Base classes for SAX handlers

- ContentHandler Objects
- DTDHandler Objects
- EntityResolver Objects
- ErrorHandler Objects
- LexicalHandler Objects

xml.sax.handler

Source code: `Lib/xml/sax/handler.py`

The SAX API defines five kinds of handlers: content handlers, DTD handlers, error handlers, entity resolvers and lexical handlers. Applications normally only need to implement those interfaces whose events they are interested in; they can implement the interfaces in a single object or in multiple objects. Handler implementations should inherit from the base classes provided in the module `!xml.sax.handler`, so that all methods get default implementations.

This is the main callback interface in SAX, and the one most important to applications. The order of events in this interface mirrors the order of the information in the document.

Handle DTD events.

This interface specifies only those DTD events required for basic parsing (unparsed entities and attributes).

Basic interface for resolving entities. If you create an object implementing this interface, then register the object with your Parser, the parser will call the method in your object to resolve all external entities.

Interface used by the parser to present error and warning messages to the application. The methods of this object control whether errors are immediately converted to exceptions or are handled in some other way.

Interface used by the parser to represent low frequency events which may not be of interest to many applications.

In addition to these classes, `!xml.sax.handler` provides symbolic constants for the feature and property names.

feature_namespaces

value: `"http://xml.org/sax/features/namespaces"`
true: Perform Namespace processing.
false: Optionally do not perform Namespace processing (implies namespace-prefixes; default).
access: (parsing) read-only; (not parsing) read/write

feature_namespace_prefixes

value: `"http://xml.org/sax/features/namespace-prefixes"`
true: Report the original prefixed names and attributes used for Namespace declarations.
false: Do not report attributes used for Namespace declarations, and optionally do not report original prefixed names (default).
access: (parsing) read-only; (not parsing) read/write

The parser based on `xml.parsers.expat` does not support this feature.

feature_string_interning

value: `"http://xml.org/sax/features/string-interning"`
true: All element names, prefixes, attribute names, Namespace URIs, and local names are interned in a dictionary (see `property_interning_dict`).
false: Names are not necessarily interned, although they may be (default).
access: (parsing) read-only; (not parsing) read/write

feature_validation

value: `"http://xml.org/sax/features/validation"`
true: Report all validation errors (implies external-general-entities and external-parameter-entities).
false: Do not report validation errors.
access: (parsing) read-only; (not parsing) read/write

The parser based on `xml.parsers.expat` does not support this feature, because Expat is a non-validating parser.

feature_external_ges

> Enabling opens a vulnerability to external entity attacks if the parser is used with user-provided XML content. Please reflect on your threat model before enabling this feature.

value: `"http://xml.org/sax/features/external-general-entities"`
true: Include all external general (text) entities.
false: Do not include external general entities.
access: (parsing) read-only; (not parsing) read/write

feature_external_pes

value: `"http://xml.org/sax/features/external-parameter-entities"`
true: Include all external parameter entities, including the external DTD subset.
false: Do not include any external parameter entities, even the external DTD subset.
access: (parsing) read-only; (not parsing) read/write

The parser based on `xml.parsers.expat` does not support this feature.

all_features

List of all features.

property_lexical_handler

value: `"http://xml.org/sax/properties/lexical-handler"`
data type: `~xml.sax.handler.LexicalHandler`
description: An optional extension handler for lexical events like comments.
access: read/write

property_declaration_handler

value: `"http://xml.org/sax/properties/declaration-handler"`
data type: an object implementing the SAX2 `DeclHandler` interface
description: An optional extension handler for DTD-related events other than notations and unparsed entities.
access: read/write

No parser in the standard library supports this property, and the standard library provides no such handler.

property_dom_node

value: `"http://xml.org/sax/properties/dom-node"`
data type: `xml.dom.Node`
description: When parsing, the current DOM node being visited if this is a DOM iterator; when not

parsing, the root DOM node for iteration.
access: (parsing) read-only; (not parsing) read/write

No parser in the standard library supports this property.

property_xml_string

value: `"http://xml.org/sax/properties/xml-string"`
data type: Bytes
description: The literal string of characters that was the source for the current event.
access: read-only, and only during a handler callback

property_encoding

value: `"http://www.python.org/sax/properties/encoding"`
data type: String
description: The name of the encoding to assume for input data.
access: read/write

No parser in the standard library supports this property.

property_interning_dict

value: `"http://www.python.org/sax/properties/interning-dict"`
data type: Dictionary
description: The dictionary used to intern names, or `None` if names are not interned. Setting it enables interning, as does the `feature_string_interning` feature.
access: read/write

all_properties

List of all known property names.

ContentHandler Objects

Users are expected to subclass `ContentHandler` to support their application. The following methods are called by the parser on the appropriate events in the input document:

ContentHandler.setDocumentLocator(locator)

Called by the parser to give the application a locator for locating the origin of document events.

SAX parsers are strongly encouraged (though not absolutely required) to supply a locator: if it does so, it must supply the locator to the application by invoking this method before invoking any of the other methods in the DocumentHandler interface.

The locator allows the application to determine the end position of any document-related event, even if the parser is not reporting an error. Typically, the application will use this information for reporting its own errors (such as character content that does not match an application's business rules). The information returned by the locator is probably not sufficient for use with a search engine.

Note that the locator will return correct information only during the invocation of the events in this interface. The application should not attempt to use it at any other time.

ContentHandler.startDocument()

Receive notification of the beginning of a document.

The SAX parser will invoke this method only once, before any other methods in this interface or in DTDHandler (except for `setDocumentLocator`).

ContentHandler.endDocument()

Receive notification of the end of a document.

The SAX parser will invoke this method only once, and it will be the last method invoked during the parse. The parser shall not invoke this method until it has either abandoned parsing (because of an unrecoverable error) or reached the end of input.

ContentHandler.startPrefixMapping(prefix, uri)

Begin the scope of a prefix-URI Namespace mapping.

The information from this event is not necessary for normal Namespace processing: the SAX XML reader will automatically replace prefixes for element and attribute names when the `feature_namespaces` feature is enabled (the default).

There are cases, however, when applications need to use prefixes in character data or in attribute values, where they cannot safely be expanded automatically; the `startPrefixMapping` and `endPrefixMapping` events supply the information to the application to expand prefixes in those contexts itself, if necessary.

Note that `startPrefixMapping` and `endPrefixMapping` events are not guaranteed to be properly nested relative to each-other: all `startPrefixMapping` events will occur before the corresponding `startElement` event, and all `endPrefixMapping` events will occur after the corresponding `endElement` event, but their order is not guaranteed.

ContentHandler.endPrefixMapping(prefix)

End the scope of a prefix-URI mapping.

See `startPrefixMapping` for details. This event will always occur after the corresponding `endElement` event, but the order of `endPrefixMapping` events is not otherwise guaranteed.

ContentHandler.startElement(name, attrs)

Signals the start of an element in non-namespace mode.

The *name* parameter contains the raw XML 1.0 name of the element type as a string and the *attrs* parameter holds an object of the `Attributes <attributes-objects>` interface containing the attributes of the element. The object passed as *attrs* may be re-used by the parser; holding on to a reference to it is not a reliable way to keep a copy of the attributes. To keep a copy of the attributes, use the `copy` method of the *attrs* object.

ContentHandler.endElement(name)

Signals the end of an element in non-namespace mode.

The *name* parameter contains the name of the element type, just as with the `startElement` event.

ContentHandler.startElementNS(name, qname, attrs)

Signals the start of an element in namespace mode.

The *name* parameter contains the name of the element type as a `(uri, localname)` tuple, the *qname* parameter contains the raw XML 1.0 name used in the source document, and the *attrs* parameter holds an instance of the `AttributesNS <attributes-ns-objects>` interface containing the attributes of the element. If no namespace is associated with the element, the *uri* component of *name* will be `None`. The object passed as *attrs* may be re-used by the parser; holding on to a reference to it is not a reliable way to keep a copy of the attributes. To keep a copy of the attributes, use the `copy` method of the *attrs* object.

Parsers may set the *qname* parameter to `None`, unless the `feature_namespace_prefixes` feature is activated.

ContentHandler.endElementNS(name, qname)

Signals the end of an element in namespace mode.

The *name* parameter contains the name of the element type, just as with the `startElementNS` method, likewise the *qname* parameter.

ContentHandler.characters(content)

Receive notification of character data.

The Parser will call this method to report each chunk of character data. SAX parsers may return all contiguous character data in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity so that the Locator provides useful information.

content may be a string or bytes instance; the `expat` reader module always produces strings.

> The earlier SAX 1 interface provided by the Python XML Special Interest Group used a more Java-like interface for this method. Since most parsers used from Python did not take advantage of the older interface, the simpler signature was chosen to replace it. To convert old code to the new interface, use *content* instead of slicing content with the old *offset* and *length* parameters.

ContentHandler.ignorableWhitespace(whitespace)

Receive notification of ignorable whitespace in element content.

Validating Parsers must use this method to report each chunk of ignorable whitespace (see the W3C XML 1.0 recommendation, section 2.10): non-validating parsers may also use this method if they are capable of parsing and using content models.

SAX parsers may return all contiguous whitespace in a single chunk, or they may split it into several chunks; however, all of the characters in any single event must come from the same external entity, so that the Locator provides useful information.

ContentHandler.processingInstruction(target, data)

Receive notification of a processing instruction.

The Parser will invoke this method once for each processing instruction found: note that processing instructions may occur before or after the main document element.

A SAX parser should never report an XML declaration (XML 1.0, section 2.8) or a text declaration (XML 1.0, section 4.3.1) using this method.

ContentHandler.skippedEntity(name)

Receive notification of a skipped entity.

The Parser will invoke this method once for each entity skipped. Non-validating processors may skip entities if they have not seen the declarations (because, for example, the entity was declared in an external DTD subset). All processors may skip external entities, depending on the values of the `feature_external_ges` and the `feature_external_pes` properties.

DTDHandler Objects

`DTDHandler` instances provide the following methods:

DTDHandler.notationDecl(name, publicId, systemId)

Handle a notation declaration event.

DTDHandler.unparsedEntityDecl(name, publicId, systemId, ndata)

Handle an unparsed entity declaration event.

EntityResolver Objects

EntityResolver.resolveEntity(publicId, systemId)

Resolve the system identifier of an entity and return either the system identifier to read from as a string, or an InputSource to read from. The default implementation returns *systemId*.

ErrorHandler Objects

Objects with this interface are used to receive error and warning information from the `~xml.sax.xmlreader.XMLReader`. If you create an object that implements this interface, then register the object with your `~xml.sax.xmlreader.XMLReader`, 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 `~xml.sax.SAXParseException` as the only parameter. Errors and warnings may be converted to an exception by raising the passed-in exception object.

ErrorHandler.error(exception)

Called when the parser encounters a recoverable error. If this method does not raise an exception, parsing may continue, but further document information should not be expected by the application. Allowing the parser to continue may allow additional errors to be discovered in the input document.

ErrorHandler.fatalError(exception)

Called when the parser encounters an error it cannot recover from; parsing is expected to terminate when this method returns.

ErrorHandler.warning(exception)

Called when the parser presents minor warning information to the application. Parsing is expected to continue when this method returns, and document information will continue to be passed to the application. Raising an exception in this method will cause parsing to end.

LexicalHandler Objects

Optional SAX2 handler for lexical events.

This handler is used to obtain lexical information about an XML document. Lexical information includes information describing the document encoding used and XML comments embedded in the document, as well as section boundaries for the DTD and for any CDATA sections. The lexical handlers are used in the same manner as content handlers.

Set the LexicalHandler of an XMLReader by using the setProperty method with the property identifier `'http://xml.org/sax/properties/lexical-handler'`.

LexicalHandler.comment(content)

Reports a comment anywhere in the document (including the DTD and outside the document element).

LexicalHandler.startDTD(name, public_id, system_id)

Reports the start of the DTD declarations if the document has an associated DTD.

LexicalHandler.endDTD()

Reports the end of DTD declaration.

LexicalHandler.startCDATA()

Reports the start of a CDATA marked section.

The contents of the CDATA marked section will be reported through the characters handler.

LexicalHandler.endCDATA()

Reports the end of a CDATA marked section.

□□□ #2
□ □□□ □□□ 2026-10-31 17:24:44 UTC
□ □□□ □□□ 2026-10-31 17:24:46 UTC