:mod:`xml.sax.handler` --- Base classes for SAX handlers

Source code: :source:`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 :mod:`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, :mod:`xml.sax.handler` provides symbolic constants for the feature and property names.

Unknown interpreted text role "mod".

Unknown directive type "data".

```
.. data:: 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
```

Unknown directive type "data".

```
.. data:: all_features

   List of all features.
```

Unknown directive type "data".

```
.. data:: property_lexical_handler

   | value: ``"http://xml.org/sax/properties/lexical-handler"``
   | data type: xml.sax.handler.LexicalHandler (not supported in Python 2)
   | description: An optional extension handler for lexical events like
     comments.
   | access: read/write
```

Unknown directive type "data".

```
.. data:: property_declaration_handler

   | value: ``"http://xml.org/sax/properties/declaration-handler"``
   | data type: xml.sax.sax2lib.DeclHandler (not supported in Python 2)
   | description: An optional extension handler for DTD-related events other
     than notations and unparsed entities.
   | access: read/write
```

Unknown directive type "data".

```
.. data:: property_dom_node

   | value: ``"http://xml.org/sax/properties/dom-node"``
   | data type: org.w3c.dom.Node (not supported in Python 2)
   | 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
```

Unknown directive type "data".

```
.. data:: property_xml_string

   | value: ``"http://xml.org/sax/properties/xml-string"``
   | data type: String
   | description: The literal string of characters that was the source for the
     current event.
   | access: read-only
```

Unknown directive type "data".

```
.. data:: all_properties

   List of all known property names.
```

ContentHandler Objects

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

Unknown interpreted text role "class".

Unknown directive type "method".

```
.. method:: 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.
```

Unknown directive type "method".

```
.. method:: 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 :meth:`setDocumentLocator`).
```

Unknown directive type "method".

```
.. method:: 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.
```

Unknown directive type "method".

```
.. method:: 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 :meth:`startPrefixMapping` and :meth:`endPrefixMapping` events supply the
   information to the application to expand prefixes in those contexts itself, if
   necessary.

   .. XXX This is not really the default, is it? MvL

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

Unknown directive type "method".

```
.. method:: ContentHandler.endPrefixMapping(prefix)

   End the scope of a prefix-URI mapping.

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

Unknown directive type "method".

```
.. method:: 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
   :class:`~xml.sax.xmlreader.Attributes`
   interface (see :ref:`attributes-objects`) 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 :meth:`copy` method of the *attrs*
```

object.

Unknown directive type "method".

```
.. method:: 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
   :meth:`startElement` event.
```

Unknown directive type "method".

```
.. method:: 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
   :class:`~xml.sax.xmlreader.AttributesNS` interface (see
   :ref:`attributes-ns-objects`)
   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 :meth:`copy` method of the *attrs* object.

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

Unknown directive type "method".

```
.. method:: 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
   :meth:`startElementNS` method, likewise the *qname* parameter.
```

Unknown directive type "method".

```
.. method:: 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.

   .. note::
```

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.

```
.. method:: 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.
```

```
.. method:: 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.
```

```
.. 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

:class:`DTDHandler` instances provide the following methods:

Unknown directive type "method".

```
.. method:: DTDHandler.notationDecl(name, publicId, systemId)

   Handle a notation declaration event.
```

Unknown directive type "method".

```
.. method:: DTDHandler.unparsedEntityDecl(name, publicId, systemId, ndata)

   Handle an unparsed entity declaration event.
```

EntityResolver Objects

Unknown directive type "method".

```
.. method:: 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 :class:`~xml.sax.xmlreader.XMLReader`. If you create an object that implements this interface, then register the object with your :class:`~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 :exc:`SAXParseException` as the only parameter. Errors and warnings may be converted to an exception by raising the passed-in exception object.

Unknown interpreted text role "class".

Unknown interpreted text role "class".

Unknown interpreted text role "exc".

Unknown directive type "method".

```
.. method:: 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.
```

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'`.

Unknown directive type "method".

```
.. method:: LexicalHandler.startCDATA()

   Reports the start of a CDATA marked section.

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

Unknown directive type "method".

```
.. method:: LexicalHandler.endCDATA()

   Reports the end of a CDATA marked section.
```