20.2. `sgmllib` — Simple SGML parser¶

Deprecated since version 2.6: The `sgmllib` module has been removed in Python 3.0.

This module defines a class `SGMLParser` which serves as the basis for parsing text files formatted in SGML (Standard Generalized Mark-up Language). In fact, it does not provide a full SGML parser — it only parses SGML insofar as it is used by HTML, and the module only exists as a base for the `htmllib` module. Another HTML parser which supports XHTML and offers a somewhat different interface is available in the `HTMLParser` module.

class `sgmllib.SGMLParser`¶

The `SGMLParser` class is instantiated without arguments. The parser is hardcoded to recognize the following constructs:

- Opening and closing tags of the form `<tag attr="value" ...>` and `</tag>`, respectively.
- Numeric character references of the form `&#name;`.
- Entity references of the form `&name;`.
- SGML comments of the form `<!--text-->`. Note that spaces, tabs, and newlines are allowed between the trailing `>` and the immediately preceding `--`.

A single exception is defined as well:

exception `sgmllib.SGMLParseError`¶

Exception raised by the `SGMLParser` class when it encounters an error while parsing.

New in version 2.1.

`SGMLParser` instances have the following methods:

`SGMLParser.reset()`¶
Reset the instance. Loses all unprocessed data. This is called implicitly at instantiation time.

`SGMLParser.setnomoretags()`¶
Stop processing tags. Treat all following input as literal input (CDATA). (This is only provided so the HTML tag `<PLAINTEXT>` can be implemented.)

`SGMLParser.setliteral()`¶
Enter literal mode (CDATA mode).

`SGMLParser.feed(`*data*`)`¶
Feed some text to the parser. It is processed insofar as it consists of complete elements; incomplete data is buffered until more data is fed or `close()` is called.

`SGMLParser.close()`¶
Force processing of all buffered data as if it were followed by an end-of-file mark. This method may be redefined by a derived class to define additional processing at the end of the input, but the redefined version should always call `close()`.

`SGMLParser.get_starttag_text()`¶
Return the text of the most recently opened start tag. This should not normally be needed for structured processing, but may be useful in dealing with HTML "as deployed" or for re-generating input with minimal changes (whitespace between attributes can be preserved, etc.).

`SGMLParser.handle_starttag(`*tag*, *method*, *attributes*`)`¶

This method is called to handle start tags for which either a `start_tag()` or `do_tag()` method has been defined. The *tag* argument is the name of the tag converted to lower case, and the *method* argument is the bound method which should be used to support semantic interpretation of the start tag. The *attributes* argument is a list of `(name, value)` pairs containing the attributes found inside the tag's `<>` brackets.

The *name* has been translated to lower case. Double quotes and backslashes in the *value* have been interpreted, as well as known character references and known entity references terminated by a semicolon (normally, entity references can be terminated by any non-alphanumerical character, but this would break the very common case of `<A HREF="url?spam=1&eggs=2">` when `eggs` is a valid entity name).

For instance, for the tag `<A HREF="http://www.cwi.nl/">`, this method would be called as `unknown_starttag('a', [('href', 'http://www.cwi.nl/')])`. The base implementation simply calls *method* with *attributes* as the only argument.

New in version 2.5: Handling of entity and character references within attribute values.

`SGMLParser.handle_endtag(`*tag*, *method*`)`¶

This method is called to handle endtags for which an `end_tag()` method has been defined. The *tag* argument is the name of the tag converted to lower case, and the *method* argument is the bound method which should be used to support semantic interpretation of the end tag. If no `end_tag()` method is defined for the closing element, this handler is not called. The base implementation simply calls *method*.

`SGMLParser.handle_data`(*data*)¶

This method is called to process arbitrary data. It is intended to be overridden by a derived class; the base class implementation does nothing.

`SGMLParser.handle_charref`(*ref*)¶

This method is called to process a character reference of the form `&#ref;`. The base implementation uses `convert_charref()` to convert the reference to a string. If that method returns a string, it is passed to `handle_data()`, otherwise `unknown_charref(ref)` is called to handle the error.

Changed in version 2.5: Use `convert_charref()` instead of hard-coding the conversion.

`SGMLParser.convert_charref`(*ref*)¶

Convert a character reference to a string, or `None`. *ref* is the reference passed in as a string. In the base implementation, *ref* must be a decimal number in the range 0-255. It converts the code point found using the `convert_codepoint()` method. If *ref* is invalid or out of range, this method returns `None`. This method is called by the default `handle_charref()` implementation and by the attribute value parser.

New in version 2.5.

`SGMLParser.convert_codepoint`(*codepoint*)¶

Convert a codepoint to a `str` value. Encodings can be handled here if appropriate, though the rest of `sgmllib` is oblivious on this matter.

New in version 2.5.

`SGMLParser.handle_entityref`(*ref*)¶

This method is called to process a general entity reference of the form `&ref;` where *ref* is an general entity reference. It converts *ref* by passing it to `convert_entityref()`. If a translation is returned, it calls the method `handle_data()` with the translation; otherwise, it calls the method `unknown_entityref(ref)`. The default `entitydefs` defines translations for `&amp;`, `&apos`, `&gt;`, `&lt;`, and `&quot;`.

Changed in version 2.5: Use `convert_entityref()` instead of hard-coding the conversion.

`SGMLParser.convert_entityref`(*ref*)¶

Convert a named entity reference to a `str` value, or `None`. The resulting value will not be parsed. *ref* will be only the name of the entity. The default implementation looks for *ref* in the instance (or class) variable `entitydefs` which should be a mapping from entity names to corresponding translations. If no translation is available for *ref*, this method returns `None`. This method is called by the default `handle_entityref()` implementation and by the attribute value parser.

New in version 2.5.

`SGMLParser.handle_comment`(*comment*)¶

This method is called when a comment is encountered. The *comment* argument is a string containing the text between the `<!--` and `-->` delimiters, but not the delimiters themselves. For example, the comment `<!--text-->` will cause this method to be called with the argument `'text'`. The default method does nothing.

`SGMLParser.handle_decl`(*data*)¶

Method called when an SGML declaration is read by the parser. In practice, the `DOCTYPE` declaration is the only thing observed in HTML, but the parser does not discriminate among different (or broken) declarations. Internal subsets in a `DOCTYPE` declaration are not supported. The *data* parameter will be the entire contents of the declaration inside the `<!`...`>` markup. The default implementation does nothing.

`SGMLParser.report_unbalanced`(*tag*)¶

This method is called when an end tag is found which does not correspond to any open element.

`SGMLParser.unknown_starttag`(*tag*, *attributes*)¶

This method is called to process an unknown start tag. It is intended to be overridden by a derived class; the base class implementation does nothing.

`SGMLParser.unknown_endtag`(*tag*)¶

This method is called to process an unknown end tag. It is intended to be overridden by a derived class; the base class implementation does nothing.

`SGMLParser.unknown_charref`(*ref*)¶

This method is called to process unresolvable numeric character references. Refer to `handle_charref()` to determine what is handled by default. It is intended to be overridden by a derived class; the base class implementation does nothing.

`SGMLParser.unknown_entityref`(*ref*)¶

This method is called to process an unknown entity reference. It is intended to be overridden by a derived class; the base class implementation does nothing.

Apart from overriding or extending the methods listed above, derived classes may also define methods of the following form to define processing of specific tags. Tag names in the input stream are case independent; the *tag* occurring in method names must be in lower case:

`SGMLParser.start_tag`(*attributes*)

This method is called to process an opening tag *tag*. It has preference over `do_tag()`. The *attributes* argument has the same meaning as described for `handle_starttag()` above.

`SGMLParser.do_tag`(*attributes*)
This method is called to process an opening tag *tag* for which no `start_tag()` method is defined. The *attributes* argument has the same meaning as described for `handle_starttag()` above.

`SGMLParser.end_tag`()
This method is called to process a closing tag *tag*.

Note that the parser maintains a stack of open elements for which no end tag has been found yet. Only tags processed by `start_tag()` are pushed on this stack. Definition of an `end_tag()` method is optional for these tags. For tags processed by `do_tag()` or by `unknown_tag()`, no `end_tag()` method must be defined; if defined, it will not be used. If both `start_tag()` and `do_tag()` methods exist for a tag, the `start_tag()` method takes precedence.