NAME

XML::Parser - A perl module for parsing XML documents

SYNOPSIS

```
use XML::Parser;

$p1 = XML::Parser->new(Style => 'Debug');
$p1->parsefile('REC-xml-19980210.xml');
$p1->parse('<foo id="me">Hello World</foo>');

# Alternative
$p2 = XML::Parser->new(Handlers => {Start => \&handle_start,
End => \&handle_end,
Char => \&handle_char});
$p2->parse($socket);

# Another alternative
$p3 = XML::Parser->new(ErrorContext => 2);

$p3->setHandlers(Char => \&text,
Default => \&other);

open(FOO, 'xmlgenerator |');
$p3->parse(*FOO, ProtocolEncoding => 'ISO-8859-1');
close(FOO);

$p3->parsefile('junk.xml', ErrorContext => 3);
```

DESCRIPTION

This module provides ways to parse XML documents. It is built on top of XML::Parser::Expat, which is a lower level interface to James Clark's expat library. Each call to one of the parsing methods creates a new instance of XML::Parser::Expat which is then used to parse the document. Expat options may be provided when the XML::Parser object is created. These options are then passed on to the Expat object on each parse call. They can also be given as extra arguments to the parse methods, in which case they override options given at XML::Parser creation time.

The behavior of the parser is controlled either by `"STYLES"` and/or `"HANDLERS"` options, or by ''setHandlers'' method. These all provide mechanisms for XML::Parser to set the handlers needed by XML::Parser::Expat. If neither`Style` nor `Handlers` are specified, then parsing just checks the document for being well-formed.

When underlying handlers get called, they receive as their first parameter the *Expat* object, not the Parser object.

METHODS

new

This is a class method, the constructor for XML::Parser. Options are passed as keyword value pairs. Recognized options are:

- Style

  This option provides an easy way to create a given style of parser. The built in styles are: ''Debug'', ''Subs'', ''Tree'', ''Objects'', and ''Stream''. These are all defined in separate packages under `XML::Parser::Style::*`, and you can find further documentation for each style both below, and in those packages.

  Custom styles can be provided by giving a full package name containing at least one '::'. This package should then have subs defined for each handler it wishes to have installed. See ''STYLES'' below for a discussion of each built in style.

- Handlers

  When provided, this option should be an anonymous hash containing as keys the type of handler and as values a sub reference to handle that type of event. All the handlers get passed as their 1st parameter the instance of expat that is parsing the document. Further details on handlers can be found in ''HANDLERS''. Any handler set here overrides the corresponding handler set with the Style option.

- Pkg

  Some styles will refer to subs defined in this package. If not provided, it defaults to the package which called the constructor.

- ErrorContext

  This is an Expat option. When this option is defined, errors are reported in context. The value should be the number of lines to show on either side of the line in which the error occurred.

- ProtocolEncoding

  This is an Expat option. This sets the protocol encoding name. It defaults to none. The built-in encodings are: `UTF-8`, `ISO-8859-1`, `UTF-16`, and `US-ASCII`. Other encodings may be used if they have encoding maps in one of the directories in the `@Encoding_Path` list. Check ''ENCODINGS'' for more information on encoding maps. Setting the protocol encoding overrides any encoding in the XML declaration.

- Namespaces

  This is an Expat option. If this is set to a true value, then namespace processing is done during the parse. See ''Namespaces'' in XML::Parser::Expat for further discussion of namespace processing.

- NoExpand

  This is an Expat option. Normally, the parser will try to expand references to entities defined in the internal subset. If this option is set to a true value, and a default handler is also set, then the default handler will be called when an entity reference is seen in text. This has no effect if a default handler has not been registered, and it has no effect on the expansion of entity references inside attribute values.

- Stream_Delimiter

  This is an Expat option. It takes a string value. When this string is found alone on a line while parsing from a stream, then the parse is ended as if it saw an end of file. The intended use is with a stream of xml documents in a MIME multipart format. The string should not contain a trailing newline.

- ParseParamEnt

  This is an Expat option. Unless standalone is set to ''yes'' in the XML declaration, setting this to a true value allows the external DTD to be read, and parameter entities to be parsed and expanded.

- NoLWP

  This option has no effect if the ExternEnt or ExternEntFin handlers are directly set. Otherwise, if true, it forces the use of a file based external entity handler.

- Non-Expat-Options

  If provided, this should be an anonymous hash whose keys are options that shouldn't be passed to Expat. This should only be of concern to those subclassing XML::Parser.

setHandlers(TYPE, HANDLER [, TYPE, HANDLER [...]])

This method registers handlers for various parser events. It overrides any previous handlers registered through the Style or Handler options or through earlier calls to setHandlers. By providing a false or

undefined value as the handler, the existing handler can be unset.

This method returns a list of type, handler pairs corresponding to the input. The handlers returned are the ones that were in effect prior to the call.

See a description of the handler types in ''HANDLERS''.

parse(SOURCE [, OPT => OPT_VALUE [...]])
The SOURCE parameter should either be a string containing the whole XML document, or it should be an open IO::Handle. Constructor options to XML::Parser::Expat given as keyword-value pairs may follow the SOURCE parameter. These override, for this call, any options or attributes passed through from the XML::Parser instance.

A die call is thrown if a parse error occurs. Otherwise it will return 1 or whatever is returned from the **Final** handler, if one is installed. In other words, what parse may return depends on the style.

parsestring
This is just an alias for parse for backwards compatibility.

parsefile(FILE [, OPT => OPT_VALUE [...]])
Open FILE for reading, then call parse with the open handle. The file is closed no matter how parse returns. Returns what parse returns.

parse_start([OPT => OPT_VALUE [...]])
Create and return a new instance of XML::Parser::ExpatNB. Constructor options may be provided. If an init handler has been provided, it is called before returning the ExpatNB object. Documents are parsed by making incremental calls to the parse_more method of this object, which takes a string. A single call to the parse_done method of this object, which takes no arguments, indicates that the document is finished.

If there is a final handler installed, it is executed by the parse_done method before returning and the parse_done method returns whatever is returned by the final handler.

HANDLERS

Expat is an event based parser. As the parser recognizes parts of the document (say the start or end tag for an XML element), then any handlers registered for that type of an event are called with suitable parameters. All handlers receive an instance of XML::Parser::Expat as their first argument. See ''METHODS'' in XML::Parser::Expat for a discussion of the methods that can be called on this object.

Init (Expat)
This is called just before the parsing of the document starts.

Final (Expat)
This is called just after parsing has finished, but only if no errors occurred during the parse. Parse returns what this returns.

Start (Expat, Element [, Attr, Val [,...]])
This event is generated when an XML start tag is recognized. Element is the name of the XML element type that is opened with the start tag. The Attr & Val pairs are generated for each attribute in the start tag.

End (Expat, Element)
This event is generated when an XML end tag is recognized. Note that an XML empty tag (<foo/>) generates both a start and an end event.

Char (Expat, String)
This event is generated when non-markup is recognized. The non-markup sequence of characters is in String. A single non-markup sequence of characters may generate multiple calls to this handler. Whatever the encoding of the string in the original document, this is given to the handler in UTF-8.

Proc (Expat, Target, Data)
This event is generated when a processing instruction is recognized.

Comment (Expat, Data)
This event is generated when a comment is recognized.

CdataStart (Expat)
This is called at the start of a CDATA section.

CdataEnd (Expat)
This is called at the end of a CDATA section.

Default (Expat, String)
This is called for any characters that don't have a registered handler. This includes both characters that are part of markup for which no events are generated (markup declarations) and characters that could generate events, but for which no handler has been registered.

Whatever the encoding in the original document, the string is returned to the handler in UTF-8.

Unparsed (Expat, Entity, Base, Sysid, Pubid, Notation)
This is called for a declaration of an unparsed entity. Entity is the name of the entity. Base is the base to be used for resolving a relative URI. Sysid is the system id. Pubid is the public id. Notation is the notation name. Base and Pubid may be undefined.

Notation (Expat, Notation, Base, Sysid, Pubid)
This is called for a declaration of notation. Notation is the notation name. Base is the base to be used for resolving a relative URI. Sysid is the system id. Pubid is the public id. Base, Sysid, and Pubid may all be undefined.

ExternEnt (Expat, Base, Sysid, Pubid)
This is called when an external entity is referenced. Base is the base to be used for resolving a relative URI. Sysid is the system id. Pubid is the public id. Base, and Pubid may be undefined.

This handler should either return a string, which represents the contents of the external entity, or return an open filehandle that can be read to obtain the contents of the external entity, or return undef, which indicates the external entity couldn't be found and will generate a parse error.

If an open filehandle is returned, it must be returned as either a glob (*FOO) or as a reference to a glob (e.g. an instance of IO::Handle).

A default handler is installed for this event. The default handler is XML::Parser::lwp_ext_ent_handler unless the NoLWP option was provided with a true value, otherwise XML::Parser::file_ext_ent_handler is the default handler for external entities. Even without the NoLWP option, if the URI or LWP modules are missing, the file based handler ends up being used after giving a warning on the first external entity reference.

The LWP external entity handler will use proxies defined in the environment (http_proxy, ftp_proxy, etc.).

Please note that the LWP external entity handler reads the entire entity into a string and returns it, where as the file handler opens a filehandle.

Also note that the file external entity handler will likely choke on absolute URIs or file names that don't fit the conventions of the local operating system.

The expat base method can be used to set a basename for relative pathnames. If no basename is given, or if the basename is itself a relative name, then it is relative to the current working directory.

ExternEntFin (Expat)
This is called after parsing an external entity. It's not called unless an ExternEnt handler is also set. There is a default handler installed that pairs with the default ExternEnt handler.

If you're going to install your own ExternEnt handler, then you should set (or unset) this handler too.

Entity (Expat, Name, Val, Sysid, Pubid, Ndata, IsParam)
This is called when an entity is declared. For internal entities, the Val parameter will contain the value and the remaining three parameters will be undefined. For external entities, the Val parameter will be undefined, the Sysid parameter will have the system id, the Pubid parameter will have the public id if it was provided (it will be undefined otherwise), the Ndata parameter will contain the notation for unparsed entities. If this

is a parameter entity declaration, then the IsParam parameter is true.

Note that this handler and the Unparsed handler above overlap. If both are set, then this handler will not be called for unparsed entities.

Element (Expat, Name, Model)

The element handler is called when an element declaration is found. Name is the element name, and Model is the content model as an XML::Parser::Content object. See ''XML::Parser::ContentModel Methods'' in XML::Parser::Expat for methods available for this class.

Attlist (Expat, Elname, Attname, Type, Default, Fixed)

This handler is called for each attribute in an ATTLIST declaration. So an ATTLIST declaration that has multiple attributes will generate multiple calls to this handler. The Elname parameter is the name of the element with which the attribute is being associated. The Attname parameter is the name of the attribute. Type is the attribute type, given as a string. Default is the default value, which will either be ''#REQUIRED'', ''#IMPLIED'' or a quoted string (i.e. the returned string will begin and end with a quote character). If Fixed is true, then this is a fixed attribute.

Doctype (Expat, Name, Sysid, Pubid, Internal)

This handler is called for DOCTYPE declarations. Name is the document type name. Sysid is the system id of the document type, if it was provided, otherwise it's undefined. Pubid is the public id of the document type, which will be undefined if no public id was given. Internal is the internal subset, given as a string. If there was no internal subset, it will be undefined. Internal will contain all whitespace, comments, processing instructions, and declarations seen in the internal subset. The declarations will be there whether or not they have been processed by another handler (except for unparsed entities processed by the Unparsed handler). However, comments and processing instructions will not appear if they've been processed by their respective handlers.

*** DoctypeFin (Parser)**

This handler is called after parsing of the DOCTYPE declaration has finished, including any internal or external DTD declarations.

XMLDecl (Expat, Version, Encoding, Standalone)

This handler is called for xml declarations. Version is a string containing the version. Encoding is either undefined or contains an encoding string. Standalone will be either true, false, or undefined if the standalone attribute is yes, no, or not made respectively.

STYLES

Debug

This just prints out the document in outline form. Nothing special is returned by parse.

Subs

Each time an element starts, a sub by that name in the package specified by the Pkg option is called with the same parameters that the Start handler gets called with.

Each time an element ends, a sub with that name appended with an underscore (''_''), is called with the same parameters that the End handler gets called with.

Nothing special is returned by parse.

Tree

Parse will return a parse tree for the document. Each node in the tree takes the form of a tag, content pair. Text nodes are represented with a pseudo-tag of ''0'' and the string that is their content. For elements, the content is an array reference. The first item in the array is a (possibly empty) hash reference containing attributes. The remainder of the array is a sequence of tag-content pairs representing the content of the element.

So for example the result of parsing:

```
<foo><head id="a">Hello <em>there</em></head><bar>Howdy<ref/></bar>do</foo>
```

would be:

```
Tag Content
=============================================================
[foo, [{}, head, [{id => "a"}, 0, "Hello ", em, [{}, 0, "there"]],
bar, [ {}, 0, "Howdy", ref, [{}]],
0, "do"
]
]
```

The root document ‘‘foo’’, has 3 children: a ‘‘head’’ element, a ‘‘bar’’ element and the text ‘‘do’’. After the empty attribute hash, these are represented in it’s contents by 3 tag-content pairs.

Objects

This is similar to the Tree style, except that a hash object is created for each element. The corresponding object will be in the class whose name is created by appending ‘‘::’’ and the element name to the package set with the Pkg option. Non-markup text will be in the ::Characters class. The contents of the corresponding object will be in an anonymous array that is the value of the Kids property for that object.

Stream

This style also uses the Pkg package. If none of the subs that this style looks for is there, then the effect of parsing with this style is to print a canonical copy of the document without comments or declarations. All the subs receive as their 1st parameter the Expat instance for the document they’re parsing.

It looks for the following routines:

- StartDocument

  Called at the start of the parse .

- StartTag

  Called for every start tag with a second parameter of the element type. The $_ variable will contain a copy of the tag and the %_ variable will contain attribute values supplied for that element.

- EndTag

  Called for every end tag with a second parameter of the element type. The $_ variable will contain a copy of the end tag.

- Text

  Called just before start or end tags with accumulated non-markup text in the $_ variable.

- PI

  Called for processing instructions. The $_ variable will contain a copy of the PI and the target and data are sent as 2nd and 3rd parameters respectively.

- EndDocument

  Called at conclusion of the parse.

ENCODINGS

XML documents may be encoded in character sets other than Unicode as long as they may be mapped into the Unicode character set. Expat has further restrictions on encodings. Read the xmlparse.h header file in the expat distribution to see details on these restrictions.

Expat has built-in encodings for: `UTF-8`, `ISO-8859-1`, `UTF-16`, and `US-ASCII`. Encodings are set either through the XML declaration encoding attribute or through the ProtocolEncoding option to XML::Parser or XML::Parser::Expat.

For encodings other than the built-ins, expat calls the function load_encoding in the Expat package with the encoding name. This function looks for a file in the path list `@XML::Parser::Expat::Encoding_Path`, that matches the lower-cased name with a ’.enc’ extension. The first one it finds, it loads.

If you wish to build your own encoding maps, check out the XML::Encoding module from CPAN.

AUTHORS

Larry Wall <*larry@wall.org*> wrote version 1.0.

Clark Cooper <*coopercc@netheaven.com*> picked up support, changed the API for this version (2.x), provided documentation, and added some standard package features.

Matt Sergeant <*matt@sergeant.org*> is now maintaining XML::Parser