NAME

XML::Parser::Expat – Lowlevel access to James Clark's expat XML parser

SYNOPSIS

```
use XML::Parser::Expat;

$parser = XML::Parser::Expat->new;
$parser->setHandlers('Start' => \&sh,
                     'End'   => \&eh,
                     'Char'  => \&ch);
open(FOO, '<', 'info.xml') or die "Couldn't open";
$parser->parse(*FOO);
close(FOO);
# $parser->parse('<foo id="me"> here <em>we</em> go </foo>');

sub sh
{
  my ($p, $el, %atts) = @_;
  $p->setHandlers('Char' => \&spec)
    if ($el eq 'special');
  ...
}

sub eh
{
  my ($p, $el) = @_;
  $p->setHandlers('Char' => \&ch)  # Special elements won't contain
    if ($el eq 'special');         # other special elements
  ...
}
```

DESCRIPTION

This module provides an interface to James Clark's XML parser, expat. As in expat, a single instance of the parser can only parse one document. Calls to parsestring after the first for a given instance will die.

Expat (and XML::Parser::Expat) are event based. As the parser recognizes parts of the document (say the start or end of an XML element), then any handlers registered for that type of an event are called with suitable parameters.

METHODS

new

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

- ProtocolEncoding

  The protocol encoding name. The default is none. The expat 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. Setting the protocol encoding overrides any encoding in the XML declaration.

- Namespaces

  When this option is given with a true value, then the parser does namespace processing. By default, namespace processing is turned off. When it is turned on, the parser consumes *xmlns* attributes and strips off prefixes from element and attributes names where those prefixes have a defined namespace. A name's namespace can be found using the "namespace" method and two names can be checked for absolute equality with the "eq_name" method.

- NoExpand

  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 option 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.

- ErrorContext

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

- ParseParamEnt

  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.

- Base

  The base to use for relative pathnames or URLs. This can also be done by using the base method.

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

This method registers handlers for the various events. If no handlers are registered, then a call to parsestring or parsefile will only determine if the corresponding XML document is well formed (by returning without error.) This may be called from within a handler, after the parse has started.

Setting a handler to something that evaluates to false unsets that handler.

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

The recognized events and the parameters passed to the corresponding handlers are:

- Start (Parser, Element [, Attr, Val [,...]])

  This event is generated when an XML start tag is recognized. Parser is an XML::Parser::Expat instance. Element is the name of the XML element that is opened with the start tag. The Attr & Val pairs are generated for each attribute in the start tag.

- End (Parser, 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.

  There is always a lower level start and end handler installed that wrap the corresponding callbacks. This is to handle the context mechanism. A consequence of this is that the default handler (see below) will not see a start tag or end tag unless the default_current method is called.

- Char (Parser, 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 (Parser, Target, Data)

  This event is generated when a processing instruction is recognized.

- Comment (Parser, String)

This event is generated when a comment is recognized.

- CdataStart (Parser)

  This is called at the start of a CDATA section.

- CdataEnd (Parser)

  This is called at the end of a CDATA section.

- Default (Parser, 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 (Parser, 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 (Parser, 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 (Parser, 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).

- ExternEntFin (Parser)

  This is called after an external entity has been parsed. It allows applications to perform cleanup on actions performed in the above ExternEnt handler.

- Entity (Parser, 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 (Parser, 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::ContentModel object. See "XML::Parser::ContentModel Methods" for methods available for this class.

- Attlist (Parser, 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 (Parser, 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 will be true or false, indicating whether or not the doctype declaration contains an internal subset.

- DoctypeFin (Parser)

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

- XMLDecl (Parser, 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 is either undefined, or true or false. Undefined indicates that no standalone parameter was given in the XML declaration. True or false indicates "yes" or "no" respectively.

namespace(name)
Return the URI of the namespace that the name belongs to. If the name doesn't belong to any namespace, an undef is returned. This is only valid on names received through the Start or End handlers from a single document, or through a call to the generate_ns_name method. In other words, don't use names generated from one instance of XML::Parser::Expat with other instances.

eq_name(name1, name2)
Return true if name1 and name2 are identical (i.e. same name and from the same namespace.) This is only meaningful if both names were obtained through the Start or End handlers from a single document, or through a call to the generate_ns_name method.

generate_ns_name(name, namespace)
Return a name, associated with a given namespace, good for using with the above 2 methods. The namespace argument should be the namespace URI, not a prefix.

new_ns_prefixes
When called from a start tag handler, returns namespace prefixes declared with this start tag. If called elsewhere (or if there were no namespace prefixes declared), it returns an empty list. Setting of the default namespace is indicated with '#default' as a prefix.

expand_ns_prefix(prefix)
Return the uri to which the given prefix is currently bound. Returns undef if the prefix isn't currently bound. Use '#default' to find the current binding of the default namespace (if any).

current_ns_prefixes
Return a list of currently bound namespace prefixes. The order of the the prefixes in the list has no meaning. If the default namespace is currently bound, '#default' appears in the list.

recognized_string
Returns the string from the document that was recognized in order to call the current handler. For instance, when called from a start handler, it will give us the start-tag string. The string is encoded in UTF−8. This method doesn't return a meaningful string inside declaration handlers.

original_string
Returns the verbatim string from the document that was recognized in order to call the current handler. The string is in the original document encoding. This method doesn't return a meaningful string inside declaration handlers.

default_current

When called from a handler, causes the sequence of characters that generated the corresponding event to be sent to the default handler (if one is registered). Use of this method is deprecated in favor the recognized_string method, which you can use without installing a default handler. This method doesn't deliver a meaningful string to the default handler when called from inside declaration handlers.

xpcroak(message)

Concatenate onto the given message the current line number within the XML document plus the message implied by ErrorContext. Then croak with the formed message.

xpcarp(message)

Concatenate onto the given message the current line number within the XML document plus the message implied by ErrorContext. Then carp with the formed message.

current_line

Returns the line number of the current position of the parse.

current_column

Returns the column number of the current position of the parse.

current_byte

Returns the current position of the parse.

base([NEWBASE]);

Returns the current value of the base for resolving relative URIs. If NEWBASE is supplied, changes the base to that value.

context

Returns a list of element names that represent open elements, with the last one being the innermost. Inside start and end tag handlers, this will be the tag of the parent element.

current_element

Returns the name of the innermost currently opened element. Inside start or end handlers, returns the parent of the element associated with those tags.

in_element(NAME)

Returns true if NAME is equal to the name of the innermost currently opened element. If namespace processing is being used and you want to check against a name that may be in a namespace, then use the generate_ns_name method to create the NAME argument.

within_element(NAME)

Returns the number of times the given name appears in the context list. If namespace processing is being used and you want to check against a name that may be in a namespace, then use the generate_ns_name method to create the NAME argument.

depth

Returns the size of the context list.

element_index

Returns an integer that is the depth-first visit order of the current element. This will be zero outside of the root element. For example, this will return 1 when called from the start handler for the root element start tag.

skip_until(INDEX)

INDEX is an integer that represents an element index. When this method is called, all handlers are suspended until the start tag for an element that has an index number equal to INDEX is seen. If a start handler has been set, then this is the first tag that the start handler will see after skip_until has been called.

position_in_context(LINES)

Returns a string that shows the current parse position. LINES should be an integer >= 0 that represents the number of lines on either side of the current parse line to place into the returned string.

xml_escape(TEXT [, CHAR [, CHAR ...]])

Returns TEXT with markup characters turned into character entities. Any additional characters provided as arguments are also turned into character references where found in TEXT.

parse (SOURCE)

The SOURCE parameter should either be a string containing the whole XML document, or it should be an open IO::Handle. Only a single document may be parsed for a given instance of XML::Parser::Expat, so this will croak if it's been called previously for this instance.

parsestring(XML_DOC_STRING)

Parses the given string as an XML document. Only a single document may be parsed for a given instance of XML::Parser::Expat, so this will die if either parsestring or parsefile has been called for this instance previously.

This method is deprecated in favor of the parse method.

parsefile(FILENAME)

Parses the XML document in the given file. Will die if parsestring or parsefile has been called previously for this instance.

is_defaulted(ATTNAME)

NO LONGER WORKS. To find out if an attribute is defaulted please use the specified_attr method.

specified_attr

When the start handler receives lists of attributes and values, the non-defaulted (i.e. explicitly specified) attributes occur in the list first. This method returns the number of specified items in the list. So if this number is equal to the length of the list, there were no defaulted values. Otherwise the number points to the index of the first defaulted attribute name.

finish

Unsets all handlers (including internal ones that set context), but expat continues parsing to the end of the document or until it finds an error. It should finish up a lot faster than with the handlers set.

release

There are data structures used by XML::Parser::Expat that have circular references. This means that these structures will never be garbage collected unless these references are explicitly broken. Calling this method breaks those references (and makes the instance unusable.)

Normally, higher level calls handle this for you, but if you are using XML::Parser::Expat directly, then it's your responsibility to call it.

XML::Parser::ContentModel Methods

The element declaration handlers are passed objects of this class as the content model of the element declaration. They also represent content particles, components of a content model.

When referred to as a string, these objects are automagicly converted to a string representation of the model (or content particle).

isempty

This method returns true if the object is "EMPTY", false otherwise.

isany

This method returns true if the object is "ANY", false otherwise.

ismixed

This method returns true if the object is "(#PCDATA)" or "(#PCDATA|...)*", false otherwise.

isname

This method returns if the object is an element name.

ischoice

This method returns true if the object is a choice of content particles.

isseq

This method returns true if the object is a sequence of content particles.

quant

This method returns undef or a string representing the quantifier ('?', '*', '+') associated with the model or particle.

children

This method returns undef or (for mixed, choice, and sequence types) an array of component content particles. There will always be at least one component for choices and sequences, but for a mixed content model of pure PCDATA, "(#PCDATA)", then an undef is returned.

XML::Parser::ExpatNB Methods

The class XML::Parser::ExpatNB is a subclass of XML::Parser::Expat used for non-blocking access to the expat library. It does not support the parse, parsestring, or parsefile methods, but it does have these additional methods:

parse_more(DATA)

Feed expat more text to munch on.

parse_done

Tell expat that it's gotten the whole document.

FUNCTIONS

XML::Parser::Expat::load_encoding(ENCODING)

Load an external encoding. ENCODING is either the name of an encoding or the name of a file. The basename is converted to lowercase and a '.enc' extension is appended unless there's one already there. Then, unless it's an absolute pathname (i.e. begins with '/'), the first file by that name discovered in the `@Encoding_Path` path list is used.

The encoding in the file is loaded and kept in the `%Encoding_Table` table. Earlier encodings of the same name are replaced.

This function is automatically called by expat when it encounters an encoding it doesn't know about. Expat shouldn't call this twice for the same encoding name. The only reason users should use this function is to explicitly load an encoding not contained in the `@Encoding_Path` list.

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.