Gamma

version 2.0, 20 March 2010

Sergey Poznyakoff.

Short Contents

Table of Contents

1 Overview

‘`Gamma`’ is a collection of assorted Guile modules. Version 2.0 provides a ‘`syslog`’ interface, a module for interfacing with SQL (more precisely: MySQL and PostgreSQL) databases and a module for writing XML parsers,

2 Syslog Interface

The '`(gamma syslog)`' module provides bindings for '`syslog`' functions:

```
(use-modules ((gamma syslog)))
```

`openlog` *tag option facility* [Scheme procedure]

Opens a connection to the system logger for Guile program. Arguments have the same meaning as in openlog(3):

tag Syslog *tag*: a string that will be prepended to every message.

option Flags that control the operation. A logical or (`logior`) of one or more of the following:

LOG_CONS
: Write directly to system console if there is an error while sending to system logger.

LOG_NDELAY
: Open the connection immediately (normally, the opening is delayed until when the first message is logged).

LOG_NOWAIT
: Don't wait for child processes that may have been created while logging the message.

LOG_ODELAY
: The converse of '`LOG_NDELAY`'; opening of the connection is delayed until `syslog` is called. This is the default.

LOG_PERROR
: Print to stderr as well. This constant may be absent if the underlying implementation does not support it.

LOG_PID Include PID with each message.

facility Specifies what type of program is logging the message. The facility must be one of:

Facility	**Meaning**
LOG_AUTH	Security/authorization messages.
LOG_AUTHPRIV	Same as `LOG_AUTH`.
LOG_CRON	Clock daemon.
LOG_DAEMON	System daemons without separate facility value.
LOG_FTP	FTP daemon.
LOG_LOCAL0 through LOG_LOCAL7	Reserved for local use.

LOG_LPR	Line printer subsystem.
LOG_MAIL	Mail subsystem.
LOG_NEWS	USENET news subsystem.
LOG_SYSLOG	Messages generated internally by `syslogd`.
LOG_USER	Generic user-level messages. This is the default.
LOG_UUCP	UUCP subsystem.

Example:

```
(openlog "reader" (logior LOG_PID LOG_CONS) LOG_DAEMON)
```

`syslog-tag` [Scheme procedure]
Returns the tag, used in the recent call to `openlog`.

`syslog` *prio text* [Scheme procedure]
Distribute a message via `syslogd`. The *text* supplies the message text. The *prio* specifies *priority* of the message. Its value must be one of the following:

Priority	**Meaning**
LOG_EMERG	system is unusable
LOG_ALERT	action must be taken immediately
LOG_CRIT	critical conditions
LOG_ERR	error conditions
LOG_WARNING	warning conditions
LOG_NOTICE	normal, but significant, condition
LOG_INFO	informational message
LOG_DEBUG	debug-level message

Example:

```
(syslog LOG_WARNING "This is a test message")
```

The priority argument may also be '`OR`'ed with a facility value, to override the one set by the `openlog` function, e.g.:

```
(syslog (logior LOG_DAEMON LOG_WARNING) "This is a test message")
```

It is common to use the `format` function to prepare the value of the *text* argument:

```
(syslog LOG_WARNING
   (format #f "operation reported: ~A" result))
```

`open-syslog-port` *prio* [Scheme procedure]
Create a *syslog port* for the given priority. Syslog port is a special output port such that any writes to it are transferred to the syslog with the given priority. The port is line buffered. For example, the following code:

```
(set-current-output-port (open-syslog-port LOG_ERR))
(display "A test ")
(display "message")
(newline)
```

results in sending the string '`A test message`' to the syslog priority `LOG_ERR`.

`openlog?` [Scheme procedure]
Return `#t` if `openlog` was previously called.

`closelog` [Scheme procedure]
Close the logging channel. The use of this function is optional.

3 SQL Interface

The '`(gamma sql)`' module provides interface with MySQL and PostgreSQL database management systems.

Usage:

```
(use-modules ((gamma sql)))
```

`sql-open-connection` *params* [Scheme procedure]

This function opens a connection to the SQL server and returns a connection object. This object is then used as argument to `sql-query` and `sql-close-connection` functions.

The `params` argument supplies the connection parameters. It is a list of conses, each of which is composed from a keyword and a value.

`#:iface` [Keyword]

Defines the type of the SQL interface. Valid values are: '`"mysql"`', to connect to a MySQL server, and '`"pgsql"`', to connect to a Postgres server.

`#:host` [Keyword]

Defines server host name. The value is a string, containing the host name or ASCII representation of the host IP address.

`#:port` [Keyword]

Defines the port number server is listening on. The value is a decimal port number.

`#:socket` [Keyword]

If the SQL server is listening on a socket, this keyword defines the UNIX pathname of the socket. This keyword cannot be used together with '`#:host`' or '`#:port`' keyword pairs.

`#:user` [Keyword]

Sets the SQL user name.

`#:pass` [Keyword]

Sets the SQL user password.

`#:db` [Keyword]

Sets the database name.

`#:ssl-cert` [Keyword]

Defines full pathname of the SSL certificate to use. If this keyword is present, the connection with the server will be encrypted using SSL.

Currently it is implemented only for MySQL connections.

`#:config-file` [Keyword]

Use the specified MySQL configuration file to obtain missing parameters.

`#:config-group` [Keyword]
Obtain missing parameters from the specified group in the MySQL configuration file (see '`#:config-file`', above).

`sql-close-connection` *conn* [Scheme procedure]
Close the SQL connection. The *conn* must be a connection descriptor returned from a previous call to `sql-open-connection`.

`sql-query` *conn query* [Scheme procedure]
Conn is a connection descriptor returned from a previous call to `sql-open-connection`, and *query* is a valid SQL query. This function executes the query and returns its results.

If *query* is a `SELECT` query (or a similar query, returning tuples), the return is a list, each element of which is a list representing a row. Elements of each row (*columns*) are string values.

If *query* results in some modifications to the database (e.g. an `UPDATE` statement), the `sql-query` function returns the number of affected database rows.

`sql-error` [Error Keyword]
An error of this type is raised when any of the above functions fails. Two arguments are supplied: a string describing the error, and error message from the underlying SQL implementation.

`sql-catch-failure` (*handler*) *expr* [Scheme syntax]
`sql-catch-failure` *expr* [Scheme syntax]
This syntax executes the Scheme expression *expr* and calls `handler` if a `gsql-error` exception occurs. In its second form, `sql-catch-failure` calls a function named `sql-error-handler` if a `sql-error` exception occurs. The `sql-error-handler` must be declared by the user.

The error handler must be declared as follows:

```
(define (handler key func fmt fmtargs data)
  ...)
```

where:

key The error key ('`sql-error`').

func Name of the Scheme function that encountered the error.

fmt Format string suitable for `format`.

fmtargs Arguments to *fmt*.

data Interface-specific error description. It is a list consisting of two elements. The first element is an integer code of the error, if supported by the underlying implementation, or `#f` if not. The second element is a textual description of the error obtained from the underlying implementation.

For example:

```
(define (sql-error-handler key func fmt fmtargs data)
  (apply format (current-error-port) fmt fmtargs))
```

`sql-ignore-failure` (*value*) *expr* [Scheme syntax]
`sql-ignore-failure` *expr* [Scheme syntax]

Evaluates Scheme expression *expr* and returns the result of evaluation, or *value* if a `gsql-error` exception occurs.

In its second form, returns `#f` in case of error.

4 Expat Interface

The '`(gamma expat)`' module provides interface to `libexpat`, a library for parsing XML documents. See `http://expat.sourceforge.net`, for a description of the library.

Usage:

```
(use-modules ((gamma expat)))
```

4.1 Expat Basics

Parsing of XML documents using Expat is based on user-defined callback functions. You create a *parser* object, and associate *callback* (or *handler*) functions with the events he is interested in. Such events may be, for instance, encountering of a open or closing tag, encountering of a comment block, etc. Once the parser object is ready, you start feeding the document to it. As the parser recognizes XML constructs, it calls the callbacks that are registered for them.

Parsers are created using `xml-make-parser` function. In the simplest case, it takes no arguments, e.g.:

```
(let ((parser (xml-make-parser)))
  ...
```

The function `xml-parse` takes the parser as its argument, reads the document from the current input stream and feeds it to the parser. Thus, the simplest program for parsing XML documents is:

```
(use-modules ((gamma expat)))
(xml-parse (xml-make-parser))
```

This program is perhaps not so useful, but you may already use it to check whether its input is a correctly formed XML document. If `xml-parse` encounters an error, it signals the `gamma-xml-error` error. See Section 4.4 [errors], page 14, for a discussion on how to handle it.

The `xml-make-parser` function takes optional arguments, which allow to set callback functions for the new parser. For example, the following code sets function '`elt-start`' as a handler for start elements:

```
(xml-make-parser #:start-element-handler elt-start)
```

The `#:start-element-handler` keyword informs the function that the argument following it is a handler for start XML documents. Any number of handlers may be set this way, e.g.:

```
(xml-make-parser #:start-element-handler elt-start
                 #:end-element-handler elt-end
                 #:comment-handler comment)
```

Definitions of particular handler functions differ depending on their purpose, i.e. on the event they are defined to handle. For example, a start element handler must be defined as having two arguments. First of them is the name of the tag, and the second one is a list of attributes supplied for

that tag. Thus, for example, the following start handler prints the tag and the number of attributes:

```
(define (elt-start name attrs)
  (format #t "~A (~A)~%" name (length attrs)))
```

For a detailed description of all available handlers and handler keywords, see Section 4.5 [handlers], page 17.

To further improve our example, suppose you need a program that will take an XML document as its input and create a description of its structure on output, showing element nesting levels by indenting their description. Here is how to write it.

First, define handlers for start and end elements. Start element handler will print two indenting spaces for each level of ancestor elements, followed by the element name and its attributes and a newline. It will then increase the global level variable:

```
(define level 0)

(define (elt-start name attrs)
  (display (make-string (* 2 level) #\space))
  (display name)
  (for-each
   (lambda (x)
    (display " ")
    (display (car x))
    (display "=")
    (display (cdr x)))
   attrs)
  (newline)
  (set! level (1+ level)))
```

The handler for end tags is simpler: it must only decrease the level:

```
(define (elt-end name)
  (set! level (1- level)))
```

Finally, create a parser and parse the input:

```
(xml-parse (xml-make-parser #:start-element-handler elt-start
                            #:end-element-handler elt-end))
```

4.2 Creating XML Parsers

Gamma provides several functions for creating and modifying XML parsers. The `xml-primitive-make-parser` and `xml-primitive-set-handler` are lower level interfaces, provided for those who wish to further extend Gamma functionality. Higher level interfaces are `xml-make-parser` and `xml-set-handler` which we recommend for regular users.

`xml-primitive-make-parser` *enc sep* [Scheme procedure]
Return a new XML parser. If *enc* is given, it must be one of: '`US-ASCII`', '`UTF-8`', '`UTF-16`', '`ISO-8859-1`'. If *sep* is given, the returned parser has namespace processing in effect. In that case, *sep* is a character which is used as a separator between the namespace URI and the local part of the name in returned namespace element and attribute names.

`xml-set-encoding` *parser enc* [Scheme procedure]
Set the encoding to be used by the *parser*. The latter must be a value returned from a previous call to `xml-primitive-make-parser` or `xml-make-parser`.

The sequence:

```
(let ((parser (xml-primitive-make-parser)))
  (xml-set-encoding parser encoding)
  ...
```

is equivalent to:

```
(let ((parser (xml-primitive-make-parser encoding)))
 ...
```

and to:

```
(let ((parser (xml-make-parser encoding)))
 ...
```

`xml-primitive-set-handler` *parser key handler* [Scheme procedure]
Set XML handler for an event. Arguments are:

parser A valid XML parser

key A key, identifying the event. For example, '`#:start-element-handler`' sets handler which is called for start tags.

See Section 4.5 [handlers], page 17, for its values and their meaning.

handler Handler procedure.

`xml-set-handler` *parser args...* [Scheme function]
Sets several handlers at once. Optional arguments (*args*) are constructed of keywords (as described in see [handler-keyword], page 13), followed by their arguments, for example:

```
(xml-set-handler parser
      #:start-element-handler elt-start
      #:end-element-handler elt-end)
```

`xml-make-parser` [*enc* [*sep*]] *args...* [Scheme function]
Create a parser and set its handlers. Optional *enc* and *sep* have the same meaning as in [xml-primitive-make-parser], page 12. The rest of arguments define handlers for the new parser. They must be supplied

in pairs: a keyword (as described in see [handler-keyword], page 13), followed by its argument. For example:

```
(xml-make-parser "US-ASCII"
      #:start-element-handler elt-start
      #:end-element-handler elt-end)
```

This call creates a new parser for documents in 'US-ASCII' encoding and sets two handlers: for element start and for element end. This call is equivalent to:

```
(let ((p (xml-primitive-make-parser "US-ASCII")))
   (xml-primitive-set-handler p #:start-element-handler elt-start)
   (xml-primitive-set-handler p #:end-element-handler elt-end)
   ...
```

4.3 Parser Functions

`xml-primitive-parse` *parser input isfinal* [Scheme procedure]
Parse next piece of input. Arguments are:

parser A parser returned from a previous call to `xml-primitive-make-parser` or `xml-make-parser`.

input A piece of input text.

isfinal Boolean value indicating whether *input* is the last part of input.

`xml-parse-more` *parser input* [Scheme function]
Equivalent to:

```
(xml-primitive-parse parser input #f)
```

unless *input* is an end-of-file object, in which case it is equivalent to:

```
(xml-primitive-parse parser "" #t)
```

`xml-parse` *parser* [*port*] [Scheme function]
Reads XML input from *port* (or the standard input port, if it is not given) and parses it using `xml-primitive-parse`.

4.4 Error Handling

When encountering an error. the '`gamma xml`' functions use Guile error reporting mechanism (see Section "Error Reporting" in *The Guile Reference Manual*). The *error key* indicates what type of error it was, and the rest of arguments supply additional information about the error. Recommended ways for handling errors in Guile are described in Section "Handling Errors" in *The Guile Reference Manual*). In this chapter we will describe how to handle errors in XML input and other errors reported by the underlying '`libexpat`' library.

`gamma-xml-error` [Error Key]
An error of this type is signalled when a of '`gamma xml`' functions encounters an XML-related error.

The arguments supplied with this error are:

key The error key (`gamma-xml-error`).

func Name of the function that generated the error.

fmt Format string

fmt-args Arguments for '`fmt`'.

descr Error description. If there are no additional information, it is `#f`. Otherwise it is a list of 5 elements which describes the error and its location in the input stream:

0. Error code (number).
1. Line number (starts at 1).
2. Column number (starts at 0).
3. Context in which the error occurred, i.e. a part of the input text which was found to contain the error.
4. Offset of point that caused the error within the context.

A special syntax is provided to extract parts of the '`descr`' list:

`xml-error-descr` *descr key* [Gamma Syntax]
Extract from *descr* the part identified by *key*. Use this macro in the error handlers. Valid values for *key* are:

`#:error-code` [xml-error-descr key]
Return the error code.

`#:line` [xml-error-descr key]
Return line number.

`#:column` [xml-error-descr key]
Return column number.

`#:has-context?` [xml-error-descr key]
Return `#t` if the description has context part. Use the two keywords below only if

```
(xml-error-descr d #:has-context?
```

returned `#t`.

`#:context` [xml-error-descr key]
Return context string.

`#:error-offset` [xml-error-descr key]
Return the location within `#:context` where the error occurred.

If no special handler is set, the default `guile` error handler displays the error and its approximate location on the standard error port. For example, given the following input file:

```
$ cat input.xml
<input>
 <ref a=1/>
</input>
```

the '`xmlck.scm`' (see [xmlck.scm], page 11) produces:

```
$ guile -s examples/xmlck.scm < input.xml
ERROR: In procedure xml-primitive-parse:
ERROR: not well-formed (invalid token) near line 2
```

To provide a more detailed diagnostics, catch the `gamma-xml-error` code and use information from the '`descr`' list. For example:

```
(catch 'gamma-xml-error
       (lambda ()
 (xml-parse (xml-make-parser)))
       (lambda (key func fmt args descr)
 (with-output-to-port
     (current-error-port)
   (lambda ()
     (cond
      ((not descr)
       (apply format #t fmt args)
       (newline))
      (else
       (format #t
       "~A:~A: ~A~%"
       (xml-error-descr descr #:line)
       (xml-error-descr descr #:column)
       (xml-error-string (xml-error-descr descr #:error-code)))
       (if (xml-error-descr descr #:has-context?)
   (let ((ctx-text (xml-error-descr descr #:context))
 (ctx-pos  (xml-error-descr descr #:error-offset)))
     (format #t
     "Context (^ marks the point): ~A^~A~%"
     (substring ctx-text 0 ctx-pos)
     (substring ctx-text ctx-pos))))
       (exit 1)))))))
```

When applied to the same input document as in the previous example, this code produces:

```
$ guile -s examples/xml-check.scm < input.xml
2:8: not well-formed (invalid token)
Context (^ marks the point): <input>
 <ref a=^1/>
```

4.5 Expat Handlers

This section describes all available element handlers. For clarity, each handler is described in its own subsection. For each handler, we indicate a *keyword* that is used when registering this handler and the *handler prototype*.

To register handlers, use `xml-make-parser` or `xml-set-handler` functions. See Section 4.2 [creating parsers], page 12, for a detailed discussion of these functions.

4.5.1 start-element-handler

`#:start-element-handler` [Handler Keyword]
Sets handler for start (and empty) tags.

The handler must be defined as follows:

`start-element` *name attrs* [Handler prototype]
Arguments:

name Element name.

attrs A list of element attributes. Each attribute is represented by a cons ('`car`' holds attribute name, '`cdr`' holds its value).

4.5.2 end-element-handler

`#:end-element-handler` [Handler Keyword]
Sets handler for end (and empty) tags. An empty tag generates a call to both start and end handlers (in that order).

The handler must be defined as follows:

`end-element` *name* [Handler prototype]
Arguments:

name Element name

4.5.3 character-data-handler

`#:character-data-handler` [Handler Keyword]
Sets a text handler. A single block of contiguous text free of markup may result in a sequence of calls to this handler. So, if you are searching for a pattern in the text, it may be split across calls to this handler.

The handler itself is defined as:

`character-data` *text* [Handler prototype]
Arguments:

text The text.

4.5.4 processing-instruction-handler

`#:processing-instruction-handler` [Handler Keyword]
Set a handler for *processing instructions*.

`processing-instruction` *target data* [Handler prototype]
Arguments are:

target First word in the processing instruction.

data The rest of the characters in the processing instruction, after *target* and whitespace following it.

4.5.5 comment-handler

`#:comment-handler` [Handler Keyword]
Sets a handler for comments.

`comment` *text* [Handler prototype]

text The text inside the comment delimiters.

4.5.6 start-cdata-section-handler

`#:start-cdata-section-handler` [Handler Keyword]
Sets a handler that gets called at the beginning of a CDATA section.

The handler is defined as follows:

`start-cdata-section` [Handler prototype]

4.5.7 end-cdata-section-handler

`#:end-cdata-section-handler` [Handler Keyword]
Sets a handler that gets called at the end of a CDATA section.

The handler is defined as:

`end-cdata-section` [Handler prototype]

4.5.8 default-handler

`#:default-handler` [Handler Keyword]
Sets a handler for any characters in the document which wouldn't otherwise be handled. This includes both data for which no handlers can be set (like some kinds of DTD declarations) and data which could be reported but which currently has no handler set.

`default` *text* [Handler prototype]

text A string containing all non-handled characters, which are passed exactly as they were present in the input XML document except that they will be encoded in UTF-8 or UTF-16. Line boundaries are not normalized. Note that a byte order mark character is not passed to the default handler. There are no guarantees about how characters are divided between calls to the default handler: for example, a comment might be split between multiple calls. Setting the '`default`' handler has the side effect of turning off expansion of references to internally defined general entities. Such references are passed to the default handler verbatim.

4.5.9 default-handler-expand

`#:default-handler-expand` [Handler Keyword]
This sets a default handler as above, but does not inhibit the expansion of internal entity references. Any entity references are not passed to the handler.

The handler prototype is the same as in Section 4.5.8 [default-handler], page 18.

4.5.10 skipped-entity-handler

`#:skipped-entity-handler` [Handler Keyword]
Set a skipped entity handler, i.e. a handler which is called if:

- An entity reference is encountered for which no declaration has been read and this is not an error.
- An internal entity reference is read, but not expanded, because a '`#:default-handler`' has been set.

`skipped-entity` *entity-name parameter?* [Handler prototype]
Arguments are:

entity-name
Name of the entity.

parameter?
This argument is `#t` if the entity is a parameter, and `#f` otherwise.

4.5.11 start-namespace-decl-handler

`#:start-namespace-decl-handler` [Handler Keyword]
Set a handler to be called when a namespace is declared.

`start-namespace-decl` *prefix uri* [Handler prototype]
Arguments:

prefix Namespace prefix.

uri Namespace URI.

4.5.12 end-namespace-decl-handler

`#:end-namespace-decl-handler` [Handler Keyword]
Set a handler to be called when leaving the scope of a namespace declaration. This will be called, for each namespace declaration, after the handler for the end tag of the element in which the namespace was declared.

The handler prototype is:

`end-namespace-decl` *prefix* [Handler prototype]

4.5.13 xml-decl-handler

`#:xml-decl-handler` [Handler Keyword]
Sets a handler that is called for XML declarations and also for text declarations discovered in external entities.

`xml-decl` *version encoding . detail* [Handler prototype]
Arguments:

version Version specification (string), or `#f`, for text declarations.

encoding Encoding. May be `#f`.

detail '`Unspecified`', if there was no standalone parameter in the declaration. Otherwise, `#t` or `#f` depending on whether it was given as '`yes`' or '`no`'.

4.5.14 start-doctype-decl-handler

`#:start-doctype-decl-handler` [Handler Keyword]
Set a handler that is called at the start of a '`DOCTYPE`' declaration, before any external or internal subset is parsed.

`start-doctype-decl` *name sysid pubid has-internal-subset?* [Handler prototype]
Arguments:

name Declaration name.

sysid System ID. May be `#f`.

pubid Public ID. May be `#f`.

has-internal-subset?
`#t` if the '`DOCTYPE`' declaration has an internal subset, `#f` otherwise.

4.5.15 end-doctype-decl-handler

`#:end-doctype-decl-handler` [Handler Keyword]
Set a handler that is called at the end of a '`DOCTYPE`' declaration, after parsing any external subset.

The handler takes no arguments:

`end-doctype-decl` [Handler prototype]

4.5.16 attlist-decl-handler

`#:attlist-decl-handler` [Handler Keyword]
Sets a handler for '`attlist`' declarations in the DTD. This handler is called for each attribute, which means, in particular, that a single attlist declaration with multiple attributes causes multiple calls to this handler.

The handler prototype is:

`attlist-decl` *el-name att-name att-type detail* [Handler prototype]
Argument:

el-name Name of the element for which the attribute is being declared.

att-name Attribute name.

detail Default value, if *el-name* is a '`#FIXED`' attribute, `#t`, if it is a '`#REQUIRED`' attribute, and `#f`, if it is a '`#IMPLIED`' attribute.

4.5.17 entity-decl-handler

`#:entity-decl-handler` [Handler Keyword]
Sets a handler that will be called for all entity declarations.

`entity-decl` *name param? value base sys-id pub-id notation* [Handler prototype]
Arguments:

name Entity name.

param? For parameter entities, `#t`. Otherwise, `#f`.

value For internal entities, entity value. Otherwise, `#f`.

base Base.

sys-id System ID. For internal entities – `#f`.

pub-id Public ID. For internal entities – `#f`.

notation Notation name, for unparsed entity declarations. Otherwise, `#f`. Unparsed are entity declarations that have a notation ('`NDATA`') field, such as:

```
<!ENTITY logo SYSTEM "images/logo.gif" NDATA gif>
```

4.5.18 notation-decl-handler

`#:notation-decl-handler` [Handler Keyword]
Sets a handler that receives notation declarations.

Handler prototype is:

`notation-decl` *notation-name base system-id public-id* [Handler prototype]

4.5.19 not-standalone-handler

`#:not-standalone-handler` [Handler Keyword]
Sets a handler that is called if the document is not *standalone*, i.e. when there is an external subset or a reference to a parameter entity, but does not have '`standalone`' set to `"yes"` in an XML declaration.

The handler takes no arguments:

`not-standalone` [Handler prototype]

4.6 miscellaneous functions

`xml-expat-version-string` [Scheme function]
Return the version of the expat library as a string.

For example:

```
(xml-expat-version-string) ⇒ "expat_2.0.1"
```

`xml-expat-version` [Scheme function]
Return the version of the expat library as a triplet: '`(major minor micro)`'.

For example:

```
(xml-expat-version) ⇒ (2 0 1)
```

`xml-default-current` [Scheme function]
Pass current markup to the default handler (see Section 4.5.8 [default-handler], page 18). This function may be called only from a callback handler.

`xml-error-string` *code*) [Scheme function]
Return a textual description corresponding to the *code* argument. See [catching gamma-xml-error], page 16, for an example of using this function.

`xml-current-line-number` *parser* [Scheme function]
Return number of the current input line in *parser*. Input lines are numbered from '`1`'.

`xml-current-column-number` *parser* [Scheme function]
Return number of column in the current input line.

`xml-current-byte-count` *parser* [Scheme function]
Return the number of bytes in the current event. Returns '`0`' if the event is inside a reference to an internal entity and for the end-tag event for empty element tags (the later can be used to distinguish empty-element tags from empty elements using separate start and end tags).

5 How to Report a Bug

If you think you've found a bug, please report it to gray+gamma@gnu.org.ua. Be sure to include maximum information needed to reliably reproduce it, or at least to analyze it. The information needed is:

- Version of the package you are using.
- Compilation options used when configuring the package.
- Run-time configuration.
- Conditions under which the bug appears.

Appendix A GNU Free Documentation License

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within

that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque

copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If

there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire

aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

A.1 ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (C) *year* *your name*.
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled ‘‘GNU Free Documentation License’’.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being *list their titles*, with the Front-Cover Texts being *list*, and with the Back-Cover Texts being *list*.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Concept Index

This is a general index of all issues discussed in this manual

M

N

O

P

Q

S

X