Contents

1 Module `XmlRpc` : XmlRpc Light

XmlRpc Light is a minimal XmlRpc client based on Xml Light and Ocamlnet.

It provides a type for values, a client class with a simple calling interface, and low-level tools that can be used to implement a server.

High-level interface

Example:

```
let rpc = new XmlRpc.client "http://localhost:8000" in
let result = rpc#call "echo" [`String "hello!"] in
print_endline (XmlRpc.dump result)
```

```
exception Error of (int * string)
```

Raised for all errors including XmlRpc faults (code, string).

```
type value = [ `Array of value list
  | `Binary of string
  | `Boolean of bool
  | `DateTime of int * int * int * int * int * int
  | `Double of float
  | `Int of int
  | `String of string
  | `Struct of (string * value) list ]
```

Polymorphic variant type for XmlRpc values:

- `` `Array ``: An ordered list of values
- `` `Binary ``: A string containing binary data
- `` `Boolean ``: A boolean
- `` `DateTime ``: A date-time value (year, month, day, hour, minute, second, timezone offset in minutes)
- `` `Double ``: A floating-point value
- `` `Int ``: An integer
- `` `String ``: A string
- `` `Struct ``: An association list of (name, value) pairs

Note that base64-encoding of `` `Binary `` values is done automatically. You do not need to do the encoding yourself.

```
class client : string ->
  object
```

```
    val url : string
```

Url of the remote XmlRpc server.

```
    val mutable useragent : string
```

User-agent to send in request headers.

```
    val mutable debug : bool
```

If true, Xml messages will be printed to standard output.

```
    method url : string
```

Gets `url`.

```
    method useragent : string
```

Gets `useragent`.

```
    method set_useragent : string -> unit
```

Sets `useragent`.

```
    method debug : bool
```

Gets `debug`.

```
    method set_debug : bool -> unit
```

Sets `debug`.

```
    method set_base64_encode : (string -> string) -> unit
```

Sets an alternate Base-64 binary encoding function.

```
    method set_base64_decode : (string -> string) -> unit
```

Sets an alternate Base-64 binary decoding function.

```
    method set_datetime_encode :
      (int * int * int * int * int * int * int -> string) -> unit
```

Sets an alternate ISO-8601 date/time encoding function.

```
    method set_datetime_decode :
      (string -> int * int * int * int * int * int * int) -> unit
```

Sets an alternate ISO-8601 date/time decoding function.

```
    method call : string -> XmlRpc.value list -> XmlRpc.value
```

`call name params` invokes an XmlRpc method and returns the result, or raises `XmlRpc.Error`[1] on error.

```
  end
```

Class for XmlRpc clients. Takes a single argument, the Url.

Utility functions

```
val dump : value -> string
```

Converts an XmlRpc value to a human-readable string for debugging.

Low-level interface

```
type message =
  | MethodCall of (string * value list)
  | MethodResponse of value
  | Fault of (int * string)
```

Type for XmlRpc messages.

```
val message_of_xml_element :
  ?base64_decode:(string -> string) ->
  ?datetime_decode:(string -> int * int * int * int * int * int * int) ->
  Xml.xml -> message
```

Converts an Xml Light element to an XmlRpc message.

```
val xml_element_of_message :
  ?base64_encode:(string -> string) ->
  ?datetime_encode:(int * int * int * int * int * int * int -> string) ->
  message -> Xml.xml
```

Converts an XmlRpc message to an Xml Light element.

```
val value_of_xml_element :
  ?base64_decode:(string -> string) ->
  ?datetime_decode:(string -> int * int * int * int * int * int * int) ->
  Xml.xml -> value
```

Converts an Xml Light element to an XmlRpc value.

```
val xml_element_of_value :
  ?base64_encode:(string -> string) ->
  ?datetime_encode:(int * int * int * int * int * int * int -> string) ->
  value -> Xml.xml
```

Converts an XmlRpc value to an Xml Light element.

2 Module `XmlRpcBase64` : Base64 codec.

8-bit characters are encoded into 6-bit ones using ASCII lookup tables. Default tables maps 0..63 values on characters A-Z, a-z, 0-9, '+' and '/' (in that order).

`exception Invalid_char`

This exception is raised when reading an invalid character from a base64 input.

`exception Invalid_table`

This exception is raised if the encoding or decoding table size is not correct.

`type encoding_table = char array`

An encoding table maps integers 0..63 to the corresponding char.

`type decoding_table = int array`

A decoding table maps chars 0..255 to the corresponding 0..63 value or -1 if the char is not accepted.

`val str_encode : ?tbl:encoding_table -> string -> string`

Encode a string into Base64.

`val str_decode : ?tbl:decoding_table -> string -> string`

Decode a string encoded into Base64, raise `Invalid_char` if a character in the input string is not a valid one.

`val encode : ?tbl:encoding_table -> char Stream.t -> char Stream.t`

Generic base64 encoding over a character stream.

`val decode : ?tbl:decoding_table -> char Stream.t -> char Stream.t`

Generic base64 decoding over a character stream.

`val make_decoding_table : encoding_table -> decoding_table`

Create a valid decoding table from an encoding one.