

NAME

lua_udunits2 – Lua interface to the udunits2 library

SYNTAX

```
ut = require('udunits2')
uts = require('udunits2.system')
utu = require('udunits2.unit')
cv = require('udunits2.converter')
```

DESCRIPTION

lua_udunits2 provides a Lua interface for the UCAR UDUNITS–2 units conversion library. Its documentation should be used to understand its basic functionality and use. This document describes only the Lua interface.

lua_udunits2 distinguishes between *methods* (which operate on `ut_system`, `ut_unit`, or `cv_converter` objects) and *functions*, which do not. There is a small set of subroutines which may be interpreted as being in either category; both functions and methods are provided for those.

In most cases the names and arguments for the Lua routines are the same as for the C routines with the following changes:

- The `ut_` and `cv_` prefixes have been removed.
- Methods use the standard Lua method invocation, so

```
ut_unit* unit = ut_new_base_unit( system );
```

becomes

```
unit = system:new_base_unit()
```

Exceptions are noted in “API CHANGES”. See also “PITFALLS”.

Return Values and UDUNITS–2 Errors

UDUNITS–2 uses function return values and the *ut_get_status()* function to obtain the status of the last call to the function. Some functions return the status directly, others return a NULL pointer and the caller must use *ut_get_status()* to obtain further information.

The Lua interface strives to be a bit more Lua-like.

The Lua versions of UDUNITS–2 routines which return `ut_status` return a boolean indicating success or failure, and in the case of failure the status code and the value of the C system variable `errno`.

The Lua versions of UDUNITS–2 routines which return objects return the object upon success; upon failure they return `nil`, the status code, and the value of the C system variable `errno`.

The `ut_get_status()` and `ut_set_status` functions are still available.

Error Message Handlers

UDUNITS–2 supports user provided error message handlers. Lua functions may be used as handlers, but the API is different. See the documentation for `set_error_message_handler` in “API CHANGES” for more information.

Lua Modules

The UDUNITS–2 library is exposed via four Lua modules. When `require`’d these modules are *not* inserted into the global symbol table. So, do this:

```
ut = require('udunits')
```

not this:

```
require('udunits')
```

Only load a module to access its functions. Objects will automatically find their methods. For example, in the following code, it is not required to load `udunits2.unit` for the `unit1` object to have access to the `same_system()` method.

```

uts = require( 'udunits2.system' )
system = uts.new()
unit1 = system:new_base_unit()
unit2 = system:new_base_unit()
unit1:same_system( unit 2)

```

The Modules

udunits2

This module provides the following functions:

```

trim
encode_date
encode_clock
encode_time
decode_time
get_status
set_status
set_error_message_handler (see below for API change)
handle_error_message (see below for API change)

```

UDUNITS-2 provides a number of enumerated constants; these are provided as integers with the UT_ prefix removed.

```
ASCII ISO_8859_1 LATIN1 UTF8
```

```

SUCCESS BAD_ARG EXISTS NO_UNIT OS NOT_SAME_SYSTEM
MEANINGLESS NO_SECOND VISIT_ERROR CANT_FORMAT SYNTAX
UNKNOWN OPEN_ARG OPEN_ENV OPEN_DEFAULT PARSE

```

```
NAMES DEFINITION
```

This module provides an additional function not available in **UDUNITS-2**:

isa

```
boolean = isa( object, type )
```

returns true if *object* is a **UDUNITS-2** object of the specified type. *type* is a string and may take one of the following values:

```

converter
error_message_handler
system
unit

```

udunits2.system

This module provides the following functions:

```

new (same as ut_new_system() )
read_xml

```

and the following methods

add_name_prefix	new_base_unit
add_symbol_prefix	new_dimensionless_unit
get_dimensionless_unit_one	parse
get_unit_by_name	unmap_name_to_unit
get_unit_by_symbol	unmap_symbol_to_unit

udunits2.unit

This module provides the following functions:

```

are_convertible
compare
divide
get_converter
multiply
same_system

```

and the following methods:

are_convertible	map_symbol_to_unit (API change)
clone	map_unit_to_name
compare	map_unit_to_symbol
divide	multiply
format (API change)	offset
get_converter	offset_by_time
get_name	raise
get_symbol	root
get_system	same_system
invert	scale (API change)
is_dimensionless	set_second
log (API change)	unmap_unit_to_name
map_name_to_unit (API change)	unmap_unit_to_symbol

Note that there is overlap between the methods and the functions.

udunits2.converter

This module provides the following functions:

combine	get_galilean
get_trivial	get_converter
get_inverse	get_log
get_scale	get_pow
get_offset	

and the following methods:

```

combine
convert
get_expression

```

Note that there is overlap between the methods and the functions. The `cv_convert_float` and `cv_convert_double` functions are coalesced into the `convert` method.

API CHANGES

ut_map_name_to_unit

```

C:   ut_map_name_to_unit( name, encoding, const unit)
Lua: unit:map_name_to_unit( name, encoding )

```

ut_map_symbol_to_unit

```

C:   ut_map_symbol_to_unit( symbol, encoding, const unit)
Lua: unit:map_symbol_to_unit( symbol, encoding )

```

ut_scale

```

C:   ut_scale( factor, unit )
Lua: unit:scale( factor )

```

ut_log

```

C:   ut_log( base, unit )
Lua: unit:log( base )

```

ut_format

C: `ut_format(unit, buf, size, opts)`
 Lua: `buf = unit:format(opts)`

The output string is limited to 1023 bytes.

ut_handle_error_message

In Lua, pass a simple string

C: `ut_handle_error_message(fmt, ...)`
 Lua: `handle_error_message(string)`

ut_set_error_message_handler

In C, a pointer to a `ut_error_message_handler` is passed and returned.

In Lua, pass in one of the following:

the string `stderr`

Error messages are written to the standard error string (equivalent to `ut_write_to_stderr`).

the string `ignore`

Error messages are discarded (equivalent to `ut_ignore`).

a Lua function

The function will be called with a single argument, the formatted error message string. Its length is limited to 1023 characters.

an `error_message_handler` object

This must come from a previous call to `set_error_message_handler`. Any operations on such an object are meaningless.

`set_error_message_handler` returns the last message handler set.

It may be either of the strings `stderr`, `ignore`, a Lua function, or a special `error_message_handler` object, which is used to encapsulate arbitrary handlers set from C. No operations on this object other than passing back to `set_error_message_handler` are defined.

cv_get_expression

C: `cv_get_expression(converter, buf, max, variable)`
 Lua: `buf = converter:get_expression(variable)`

C Library support routines

This package provides C side support for manipulating UDUNITS-2 objects on the Lua stack. This requires linking against the `lib_lua_udunits2` library.

The following functions are declared in `lua_udunits2.h`:

Pushing a UDUNITS-2 object onto the Lua stack

```
lua_tpush(lua_State *L, ut_system* obj, int is_lua_managed);
lua_tpush(lua_State *L, ut_unit* obj, int is_lua_managed);
lua_tpush(lua_State *L, cv_converter* obj,
           int is_lua_managed);
lua_tpush(lua_State *L, ut_error_message_handler* obj,
           int is_lua_managed);
```

Pushes a UDUNITS-2 object pointer onto the Lua stack. If `is_lua_managed` is non-zero then deallocation of the object will be handled by Lua.

Extracting a UDUNITS-2 object from the Lua stack

```
ut_system* lua_tosystem(lua_State *L, int idx);
ut_unit*    lua_tounit(lua_State *L, int idx);
cv_converter* lua_toconverter(lua_State *L, int idx);
ut_error_message_handler lua_tomessage_handler(lua_State *L, int idx);
```

Return the UDUNITS-2 object pointer at the given stack location.

Type checking

```
int luaut_is_system(L, idx);
int luaut_is_unit(L, idx);
int luaut_is_converter(L, idx);
int luaut_is_message_handler(L, idx);
```

Return true if the object on the Lua stack at the given index is the specific type.

Function argument checking

```
ut_system* luaut_check_system(L, nargs);
ut_unit*    luaut_check_unit(L, nargs);
cv_converter* luaut_check_converter(L, nargs);
ut_error_message_handler* luaut_check_message_handler(L, nargs);
```

Raise an error if the given function is not of the specified type, else return the pointer to the object. Similar in spirit to the `luaL_check*` functions.

PITFALLS**Lua Thread safety**

There is none. **UDUNITS-2** stores some state in global variables. No attempt has been made to maintain consistency across multiple Lua threads or interpreters.

Error Messages

If multiple Lua interpreters call the library and one or more install Lua functions as error message handlers, interpreters which have *not* done so will throw an error when calling `set_error_message_handler`. This latter is a feature to prevent confused code.

Unimplemented Functionality**Visitors**

None of the visitor functionality has been exposed.

`cv_convert_floats`

`cv_convert_doubles`

Conversions of arrays is not supported.

AUTHOR

Diab Jerius, <djerius@cfa.harvard.edu>

COPYRIGHT AND LICENSE

Copyright (C) 2011 by the Smithsonian Astrophysical Observatory

This software is released under the GNU General Public License. You may find a copy at <<http://www.fsf.org/copyleft/gpl.html>>.