luasup

support routines for interfacing C with Lua
Edition 1.0.2, for version 1.0.2
20 April 2007

Diab Jerius

Table of Contents

1 Copying

2 Introduction

The `luasup` package contians support routines for use when interacting with and embedded Lua interpreter. In particular, it eases argument checking.

Many of the routines will return an error message via a user defined function (with a typedef of `SLua_ErrFunc`:

```
typedef void(*SLua_ErrFunc)( char *fmt, ... );
```

Generally the function is passed in to the `luasup` function. It is not required to return. `luasup` functions will return an error status if it does, however.

Typically, an error function will use the `lua_error` routine so that the line number at which the error ocurred will be printed. Here's a simple one:

```
/* --8<--8<--8<--8<--
 *
 * Copyright (C) 2006 Smithsonian Astrophysical Observatory
 *
 * This file is part of luasup
 *
 * luasup is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * luasup is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the
 *       Free Software Foundation, Inc.
 *       51 Franklin Street, Fifth Floor
 *       Boston, MA  02110-1301, USA
 *
 * -->8-->8-->8-->8-- */

#include <stdarg.h>

#include <lua.h>

void
errfunc( const char *fmt, ... )
{
  va_list ap;
  char buffer[1024];
```

```
  va_start( ap, fmt );

  vsprintf( buffer, fmt, ap );

  va_end( ap );

  /* this doesn't return */
  lua_error( buffer );
}
```

Appendix A Functions

A.1 Argument parsing

A.1.1 slua_args

read in the arguments passed to a C function from Lua

Synopsis

```
#include <luasup/luasup.h>

int slua_args(
  SLua_ErrFunc errfunc,
  const char *func,
  ...
);
```

Parameters

`SLua_ErrFunc errfunc`
: the error function

`const char *func`
: the name of the Lua called function. Used for error messages

`...`
: the list of arguments

Description

This routine will read the arguments passed by Lua to a C function when the function is called from Lua. It expects a list of triples, each of which contains the argument's name (for error reporting purposes), the argument's type (one of `SLua_STRING`, `SLua_NUMBER` or `SLua_TABLE`), and a pointer to storage for the argument. The list should be terminated by a `NULL` value.

Numeric values are stored as `double`. String values are returned as a pointer to a duplicate string. The calling routine is responsible for deallocating this memory.

As `slua_args` uses `slua_args_va`, see its write-up for more information.

Returns

It returns the number of arguments found, or '`-1`' if an error ocurred. If the number of arguments passed by Lua is different than in the passed list, an error message is generated.

Errors

Errors are returned via the passed `SLua_ErrFunc` function.

Author

Diab Jerius

Example

```
/* --8<--8<--8<--8<--
 *
 * Copyright (C) 2006 Smithsonian Astrophysical Observatory
 *
 * This file is part of luasup
 *
 * luasup is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * luasup is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the
 *       Free Software Foundation, Inc.
 *       51 Franklin Street, Fifth Floor
 *       Boston, MA  02110-1301, USA
 *
 * -->8-->8-->8-->8-- */

#include <stddef.h>
#include <stdarg.h>

#include <luasup.h>

extern SLua_ErrFunc errfunc;

void foo ( void )
{
  char *name;
  double energy, base_flux;

  slua_args( errfunc, "mono flux generator",
                "name",         SLua_STRING,     &name,
                "energy",       SLua_NUMBER,     &energy,
                "photon flux",  SLua_NUMBER,     &base_flux,
                NULL
                );

  free( name );
```

```
}
```

A.1.2 slua_args_n

read in the arguments passed to a C function from Lua

Synopsis

```
#include <luasup/luasup.h>

int slua_args_n(
  SLua_ErrFunc errfunc,
  const char *func,
  int n,
  ...
);
```

Parameters

`SLua_ErrFunc errfunc`
: the error function

`const char *func`
: the name of the Lua called function. Used for error messages

`int n`
: the Lua argument position at which to start. Unary based.

`...`
: the list of arguments

Description

This routine will read the arguments passed by Lua to a C function when the function is called from Lua. It expects a list of triples, each of which contains the argument's name (for error reporting purposes), the argument's type (one of `SLua_STRING`, `SLua_NUMBER`, or `SLua_TABLE`), and a pointer to storage for the argument. The list should be terminated by a `NULL` value.

Numeric values are stored as `double`. String values are returned as a pointer to a duplicate string. The calling routine is responsible for deallocating this memory.

It differs from `slua_args` in that it will start with the Lua argument at the position specified by `n`, and will not complain about extra passed arguments.

As `slua_args_n` uses `slua_args_va`, see its write-up for more information.

Returns

It returns the number of arguments found, or '`-1`' if an error ocurred.

Errors

Errors are returned via the passed `SLua_ErrFunc` function.

Author

Diab Jerius

Example

```
/* --8<--8<--8<--8<--
 *
 * Copyright (C) 2006 Smithsonian Astrophysical Observatory
 *
 * This file is part of luasup
 *
 * luasup is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * luasup is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the
 *       Free Software Foundation, Inc.
 *       51 Franklin Street, Fifth Floor
 *       Boston, MA  02110-1301, USA
 *
 * -->8-->8-->8-->8-- */

#include <stddef.h>
#include <stdarg.h>

#include <luasup.h>

extern SLua_ErrFunc errfunc;

void foo ( void )
{
  char *name;
  double energy, base_flux;

  slua_args_n( errfunc, "mono flux generator", 1,
               "name",         SLua_STRING,     &name,
               "energy",       SLua_NUMBER,     &energy,
               "photon flux",  SLua_NUMBER,     &base_flux,
               NULL
               );

  free( name );
```

```
  slua_args_n( errfunc, "mono flux generator", 4,
               "name",         SLua_STRING,     &name,
               "energy",       SLua_NUMBER,     &energy,
               "photon flux",  SLua_NUMBER,     &base_flux,
               NULL
               );

  free( name );
}
```

A.1.3 slua_args_va

read in the arguments passed to a C function from Lua

Synopsis

```
#include <luasup/luasup.h>

int slua_args_va(
  SLua_ErrFunc errfunc,
  const char *func,
  int n,
  va_list ap
);
```

Parameters

`SLua_ErrFunc errfunc`
: the error function

`const char *func`
: the name of the Lua called function. Used for error messages

`int n`
: the Lua argument position at which to start. Unary based.

`va_list ap`
: the list of arguments

Description

This routine will read the arguments passed by Lua to a C function when the function is called from Lua. It expects a `va_list` of triples, each of which contains the argument's name (for error reporting purposes), the argument's type (one of `SLua_STRING`, `SLua_NUMBER`, or `SLua_TABLE`) and a pointer to storage for the argument. The list should be terminated by a `NULL` value. Support is provided for optional arguements. These must be at the end of the arguement list, and should have their type or'd with `SLua_OPTIONAL` (e.g. `SLua_TABLE | SLua_OPTIONAL`). To determine if an optional argument was present, use `slua_args_va`'s return value.

Numeric values are stored as `double`. String values are returned as a pointer to a duplicate string. The calling routine is responsible for deallocating this memory.

Arguments are returned beginning with the Lua argument at the position specified by `n`.

Returns

It returns the number of arguments found, or '`-1`' if an error ocurred.

Errors

Errors are returned via the passed `SLua_ErrFunc` function.

Author

Diab Jerius

Example

```
/* --8<--8<--8<--8<--
 *
 * Copyright (C) 2006 Smithsonian Astrophysical Observatory
 *
 * This file is part of luasup
 *
 * luasup is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * luasup is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the
 *       Free Software Foundation, Inc.
 *       51 Franklin Street, Fifth Floor
 *       Boston, MA  02110-1301, USA
 *
 * -->8-->8-->8-->8-- */

#include <stdarg.h>

#include <luasup.h>

int foo ( SLua_ErrFunc errfunc, char *func, ... )
{
  va_list ap;
  int n;
```

```
  va_start( ap, func );

  n = slua_args_va( errfunc, func, 1, ap );

  va_end( ap );

  return n;
}
```

A.1.4 slua_check_n_arg

check the number of arguments passed to C from Lua.

Synopsis

```
#include <luasup/luasup.h>

int slua_check_n_arg(
  SLua_ErrFunc errfunc,
  const char *func,
  int n
);
```

Parameters

`SLua_ErrFunc errfunc`
: Not Documented.

`const char *func`
: the name of the lua function

`int n`
: the number of arguments expected

Description

This routine compares the number of arguments expected (passed via the `n` parameter) to the actual number of arguments passed to the C function by Lua. It's a simple wrapper around `lua_getparam`.

Returns

If the number of arguments matches is not less than that passed via `n`, it returns zero. If there are fewer arguments than expected, it calls the supplied error function and returns -1.

Author

Diab Jerius

Example

```
/* --8<--8<--8<--8<--
 *
```

```
 * Copyright (C) 2006 Smithsonian Astrophysical Observatory
 *
 * This file is part of luasup
 *
 * luasup is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 *
 * luasup is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the
 *       Free Software Foundation, Inc.
 *       51 Franklin Street, Fifth Floor
 *       Boston, MA  02110-1301, USA
 *
 * -->8-->8-->8-->8-- */

#include <stddef.h>
#include <stdarg.h>

#include <luasup.h>

extern SLua_ErrFunc errfunc;

void foo ( void )
{
  char *name;
  double energy, base_flux;

  slua_args_n( errfunc, "mono flux generator", 1,
               "name",         SLua_STRING,     &name,
               "energy",       SLua_NUMBER,     &energy,
               "photon flux",  SLua_NUMBER,     &base_flux,
               NULL
               );

  free( name );

  slua_args_n( errfunc, "mono flux generator", 4,
               "name",         SLua_STRING,     &name,
               "energy",       SLua_NUMBER,     &energy,
               "photon flux",  SLua_NUMBER,     &base_flux,
```

```
                NULL
                );

  free( name );
}
```

A.2 Manipulating Tables

A.2.1 slua_tbl_nelem

return the number of elements in a Lua table

Synopsis

```
#include <luasup/luasup.h>

int slua_tbl_nelem(lua_Object table);
```

Parameters

`lua_Object` `table`
: the table to iterate over

Description

This routine counts the number of elements in a Lua table.

Returns

It returns the number of elements in the table

Author

Diab Jerius

A.2.2 slua_tbl_next

an implementation of the Lua `next` function

Synopsis

```
#include <luasup/luasup.h>

lua_Object slua_tbl_next(
  lua_Object table,
  lua_Object *index
);
```

Parameters

`lua_Object` `table`
: the table to iterate over

`lua_Object *index`
: A pointer to the last index used; set the index's value to LUA_NOOBJECT on the first pass

Description

This routine is a wrapper around the Lua `next` function, for which there are no default bindings for C. It is passed a table and an index. The index should be set to `LUA_NOOBJECT` to start the scan in the table. `slua_tbl_next` will update the index to point to the next element in the table, and will return the value in the table at that index. Note that the *address* of the index is passed to `slua_tbl_next`.

If there are no more elements in the table, the returned value and the index are set to `LUA_NOOBJECT`.

Returns

It returns the value of the table at the next index, or `LUA_NOOBJECT` if there are no more elements.

Author

Diab Jerius

A.2.3 slua_tbl_el

get an element from a table

Synopsis

```
#include <luasup/luasup.h>

SLuaError slua_tbl_el(
  SLua_ErrFunc errfunc,
  lua_Object table,
  const char *err_pfx,
  const char *index,
  SLuaArgType type,
  void *dest
);
```

Parameters

`SLua_ErrFunc errfunc`
: the error function

`lua_Object table`
: the table

`const char *err_pfx`
: a prefix for error messages

`const char *index`
: the index of the element to delete

`SLuaArgType type`
: the type of argument to expect (with or'd flags)

  Possible values for a `SLuaArgType` are as follows: `SLua_STRING`, `SLua_NUMBER`, `SLua_TABLE`, `SLua_NIL`, `SLua_FUNCTION`, `SLua_CFUNCTION`, `SLua_USERDATA`, `SLua_TYPES`, `SLua_OPTIONAL`, `SLua_DELENTRY`, `SLua_COPY`, `SLua_BOGUS`

`void *dest`
: where to store the retrieved data

Description

This function retrieves the value of an element with index `index` in a table. The caller must specify the type of argument to expect. If the type is or'd with `SLua_OPTIONAL` the element need not be present. If the type is `SLua_STRING` and is or'd with `SLua_COPY`, a copy of the string value is returned (the caller must handle deallocation).

The value of the element is stored in the passed `dest` variable.

Upon error, the passed function `errfunc` is called with an error message. The message will be prefixed with the string in `err_pfx` (using `"%s: error message"`). `errfunc` may be `NULL`, in which case it won't be called.

Returns

It returns one of the following values:

`0`
: everything went ok or the passed table was equal to `LUA_NOOBJECT`.

`SLua_ENOEL`
: The element was optional, but not present.

`SLua_ENOENT`
: The element was not in the table.

`SLua_ENOMEM`
: The type was `SLua_STRING|SLua_COPY` and the string could not be duplicated.

`SLua_ENOTYPE`
: The type is unknown.

`SLua_ENOMATCH`
: The requested type and the actual type do not agree.

Possible values for a `SLuaError` are as follows: `SLua_ENOERR`, `SLua_ENOENT`, `SLua_ENOMEM`, `SLua_ENOTYPE`, `SLua_ENOEL`, `SLua_ENOMATCH`

Author

Diab Jerius

A.2.4 slua_tbl_el_s

get a string element from a table

Synopsis

```
#include <luasup/luasup.h>

int slua_tbl_el_s(
  lua_Object table,
  const char *index,
  char **value
);
```

Parameters

`lua_Object table`
: the table containing the data

`const char *index`
: the index of the element

`char **value`
: the output value of the element

Description

retrieve the value of an element in a table. The value should be representable as a string, as a pointer to a dynamically allocated string is returned. The calling routine is responsible for deallocating the string.

Returns

It returns zero if all went well, '`1`' if the element didn't exist, and '`-1`' if it was unable to allocate enough memory for the string. If the element doesn't exist, `*value` is unchanged.

Author

Diab Jerius

A.2.5 slua_tbl_el_si

get a string element from a table

Synopsis

```
#include <luasup/luasup.h>

char *slua_tbl_el_si(
  lua_Object table,
  const char *index
);
```

Parameters

`lua_Object table`
: the table containing the data

`const char *index`
: the index of the element

Description

retrieve the value of an element in a table. The value should be representable as a string, as a pointer to the string is returned. This points to the Lua object; it differs from `slua_tbl_el_s` in that it doesn't allocate any memory for the string.

Returns

It returns a pointer to the string upon succes, `NULL` if the element didn't exist.

Author

Diab Jerius

A.2.6 slua_tbl_el_n

get a numeric element from a table

Synopsis

```
#include <luasup/luasup.h>

int slua_tbl_el_n(
  lua_Object table,
  const char *index,
  double *value
);
```

Parameters

`lua_Object table`
: the table containing the data

`const char *index`
: the index of the element

`double *value`
: the output value of the element

Description

retrieve the value of an element in a table. The value should be representable as a number.

Returns

It returns zero if all went well and '`1`' if the element didn't exist. If the element doesn't exist, `*value` is unchanged.

Author

Diab Jerius

A.2.7 slua_tbl_el_tbl

get a table element from a table

Synopsis

```
#include <luasup/luasup.h>

lua_Object slua_tbl_el_tbl(
  lua_Object table,
  const char *index
);
```

Parameters

`lua_Object table`
: the table containing the data

`const char *index`
: the index of the element

Description

retrieve the value of an element in a table. The value should be a table. The table handle is returned (no copy is made).

Returns

It returns the table handle, or `LUA_NOOBJECT` if the element didn't exist or wasn't a table. exist, and '`-1`' if it was unable to allocate enough memory for the string. If the element doesn't exist, `*value` is unchanged.

Author

Diab Jerius

A.2.8 slua_tbl_el_exists

determine existence of an element in a table

Synopsis

```
#include <luasup/luasup.h>

int slua_tbl_el_exists(
  lua_Object table,
  const char *index
);
```

Parameters

`lua_Object table`
: the table containing the data

`const char *index`
: the index of the element

Description

This routine checks for the existence of an element in a table.

Returns

It returns zero if the element doesn't exist, non-zero otherwise.

Author

Diab Jerius

A.2.9 slua_tbl_el_delete

delete an element from a table

Synopsis

```
#include <luasup/luasup.h>

void slua_tbl_el_delete(
  lua_Object table,
  const char *index
);
```

Parameters

`lua_Object table`
: the table containing the data

`const char *index`
: the index of the element to delete

Description

This routine sets the value of a specified element in a table to '`nil`', effectively deleting it.

Author

Diab Jerius

A.2.10 slua_tbl_cmp

compare two tables for equality

Synopsis

```
#include <luasup/luasup.h>
```

```
int slua_tbl_cmp(
  lua_Object table1,
  lua_Object table2
);
```

Parameters

`lua_Object` `table1`
: the first table

`lua_Object` `table2`
: the second table

Description

This routine compares two tables for equality. it is a deep comparison.

Returns

`0` if the two tables are equivalent, `1` if they are not.

Author

Diab Jerius

A.2.11 slua_tbl_copy

make a shallow copy of a table

Synopsis

```
#include <luasup/luasup.h>

lua_Object slua_tbl_copy(lua_Object table);
```

Parameters

`lua_Object` `table`
: the table to copy

Description

This routine makes a shallow copy of a table. It doesn't not attempt to recursively create tables contained within the table.

Returns

It returns a table object upon success, or `LUA_NOOBJECT` upon error.

Author

Diab Jerius

A.3 Miscellaneous Utilities

A.3.1 slua_type

return the type of a Lua object

Synopsis

```
#include <luasup/luasup.h>

SLuaArgType slua_type(lua_Object object);
```

Parameters

`lua_Object object`
: the object to test

Description

This routine returns the type (`SLuaArgType`) of a Lua object. of the Lua types. If the type is not recognized, it returns `SLua_BOGUS`.

Returns

Possible values for a `SLuaArgType` are as follows: `SLua_STRING`, `SLua_NUMBER`, `SLua_TABLE`, `SLua_NIL`, `SLua_FUNCTION`, `SLua_CFUNCTION`, `SLua_USERDATA`, `SLua_TYPES`, `SLua_OPTIONAL`, `SLua_DELENTRY`, `SLua_COPY`, `SLua_BOGUS`

Author

Diab Jerius

A.3.2 slua_type_name

return a string containing the name of a type

Synopsis

```
#include <luasup/luasup.h>

const char *slua_type_name(SLuaArgType type);
```

Parameters

`SLuaArgType type`
: Possible values for a `SLuaArgType` are as follows: `SLua_STRING`, `SLua_NUMBER`, `SLua_TABLE`, `SLua_NIL`, `SLua_FUNCTION`, `SLua_CFUNCTION`, `SLua_USERDATA`, `SLua_TYPES`, `SLua_OPTIONAL`, `SLua_DELENTRY`, `SLua_COPY`, `SLua_BOGUS`

Description

This routine returns a pointer to a character string identifying one of the Lua types. If the type is not recognized, it returns `NULL`.

The values of type that are supported are `SLua_STRING`, `SLua_NUMBER`, `SLua_TABLE`, `SLua_NIL`, `SLua_FUNCTION`, `SLua_CFUNCTION`, `SLua_USERDATA` .

Author

Diab Jerius