Documentation V 1.2.5

eXtensible Concatenative Language Engine

Overview

XCLE provides a framework for software-based handling of executable code, combining easy generation and manipulation, with execution speed and memory efficiency at runtime. XCLE implements most of the basic data types: integers, floats, strings, recursive lists and executable primitives, encapsulated in a generic object type. The API provides the means to integrate program building capabilities into software, handling both the data and code aspects of program generation and execution. The library as a whole provides the necessary framework for manipulating concatenative code.

The object hierarchy tree is described below, showing object name, data, category (between brackets), and string representation (between quotes).

XCLE_Object		Generic object	**[none]**	---
	XCLE_Void	Undefined object	**[null]**	**'¬'**
	XCLE_Intg	Integer number on 32 bits	**[scal]**	**'123456'**
	XCLE_Fltp	Floating point number	**[scal]**	**'3.1415926e+00'**
	XCLE_Strg	Character string	**[scal]**	**'"any text"'**
	XCLE_List	All-purposes list	**[list]**	**'[obj1 obj2 ...]'**
	XCLE_Code	Dynamic executable code	**[exec]**	**'<CODE_NAME>' or '<CODE_NAME:data>'**

Note 1: objects in an **XCLE_List** can be separated by spaces, tabs, or linefeeds / carry returns

Note 2: here is a table of numeric values for special characters:

CHR	DEC	HEX	CHR	DEC	HEX
"	034	0x22	:	058	0x3A
<	060	0x3C	>	062	0x3E
[	091	0x5B	]	093	0x5D
¬	172	0xAC			

Programming Interface

Execution

XCLE_ListEval or **XCLE_ObjectEval** respectively take an **XCLE_List** or an **XCLE_Object** in addition to an **XCLE_Stack** and **XCLE_Hash**, on which they will execute the **XCLE_List** or **XCLE_Object**.

The **XCLE_List** type can contain any particular **XCLE_Object**. It also provides a mecanism to build a program, since when a List gets executed, all its elements are executed in turn.
All **XCLE_Code** executable instructions, as well as the execution of an **XCLE_List**, use an **XCLE_Stack** and an **XCLE_Hash**. They take arguments and return results on the **XCLE_Stack**, while having access to named variables in the **XCLE_Hash**. Execution of an **XCLE_List** is the act to take every member of this list and deposit it on the stack if it is an **XCLE_Void**, **XCLE_Intg**, **XCLE_Strg**, or **XCLE_List**, or execute it if it is an **XCLE_Code**.
Execution stops when the end of the list is reached, or the execution of an [exec] type failed, from lack of arguments, bad argument types, or some other error.

These execution routines return an exception (an **XCLE_Exception** structure), equal to **XCLE_EXCEPTION_OK** when the execution went well, or the raised exception in case of error. The full exception stack can be found in the execution context structure.

Compiler

XCLE_ListParse or **XCLE_ObjectParse** respectively take a character string and produce an **XCLE_List** or an **XCLE_Object**. The string is interpreted into objects so that printing these objects would produce back the original string, or something very similar.

Syntax

A valid string for **XCLE_ListParse** is one or more string representation of objects, among:

Instructions:	e.g. ' <HELP> ', the code base building blocks
Numbers:	e.g. ' 3.1425e+00 ', an integer or floating point number
Strings:	e.g. ' "HELLO" ', a litteral string
Lists:	e.g. ' [obj1 obj2] ', a list of other objects

During parsing, a bareword is looked-up as a number, then if not applicable in the set of available instructions, and finally parsed as a string if it could not be found.

Data

The memory representation of an **XCLE_Object** has two parts: a generic memory and tracking structure, used for reference counting and cross-referencing, and a content part, type-dependant.

The **XCLE_Void** data type is the default type: it is only used to mark internal **XCLE_Object** use, or in a standard context that something has gone wrong in memory management.
The three scalar data types are simple counted memory segments: fixed size for the **XCLE_Intg** and **XCLE_Fltp** types, variable size (which means keeping a 'size' register) for the **XCLE_Strg** type.

The **List** type is a variable-length vector, with buffer space before and after the section holding (in a consecutive manner) the **XCLE_Object**'s. The buffer spaces enables fast insertion and deletion with the pop/push and shift/unshift operations.

As for the **XCLE_Code** (executable) type, this is a complex structure, containing, among other things, a pointer toward a segment of assembler code retaining the actual implementation, input and output arity and type information, and a formatted description.
The "system" types (**XCLE_Stack** and **XCLE_Hash**, or named variables table) play an essential role in memory management: objects can be freed when no reference for them in one of these tables exist any more. They also are the essential holders for instruction arguments and results.

DOWNLOAD

The latest version of XCLE can be obtained from
http://www.varkhan.net/software/xcl/XCLE/

AUTHOR

Name: **Yann LANDRIN-SCHWEITZER aka Varkhan**
Mail: **varkhan at varkhan dot net**
Home: **http://www.varkhan.net/**

BUGS

Innumerable. Don't forget to report them, even if each bug correction is the source for new ones...

TODO

Implement pseudo-code (list-compiled code) file dumping, in a cross-platform format.

SEE ALSO

XCLstd: a library defining a standard instruction set.

xcl:compiler/interpreter for the XCLE library.

gxcl: a Gtk interface for interactively executing XCLE code and manipulating stack and variable lists.

LICENSE

eXtensible Concatenative Language Engine

XCLE: Programmatic handling of executable code, combining easy generation and manipulation with execution speed and memory efficiency at runtime.

API documentation

Support objects and utilities

_ Splash
_ Containers

Operationnal types

_ Obj_generic
_ Obj_void
_ Obj_intg
_ Obj_fltp
_ Obj_strg
_ Obj_list
_ Obj_code

Modules

_ Mod_string
_ Mod_write
_ Mod_parse
_ Mod_exec
_ Mod_fileIO

Splash text utilities

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://www.varkhan.net/

XCLE_Splash utilities

Function XCLE_versionid

unsigned long XCLE_versionid(void) ;
Returns a binary identifier for the current version of XCLE.

Returns:
A binary version identifier.

Note:
The version identifier is composed of the major version number, on the first byte, the minor version number on the second byte, and the release number on the third byte.

Function XCLE_version

const char * XCLE_version(void) ;
Returns a version string for the current version of XCLE.

Returns:
A string version identifier.

Note:

The version identifier takes the form "X.Y.Z", where X is the major version number, Y the minor version number and Z the release number.

Function XCLE_copyright

```
const char * XCLE_copyright(void) ;
```

Returns a copyright string for the XCLE library.

Returns:

The copyright string for XCLE.

Function XCLE_license

```
const char * XCLE_license(void) ;
```

Returns a short license information for the XCLE library.

Returns:

The string "XCLE is distributed under the LGPL License"

Function XCLE_splash

```
const char * XCLE_splash(void) ;
```

Returns the full name of the XCLE library.

Returns:

The string "XCLE: eXtensible Concatenative Language Engine"

Function XCLE_author

```
const char * XCLE_author(void) ;
```

Returns the author(s) name and contact information.

Returns:

A string containing one author record per line.

Note:

Each author record should contain, separaed by double dashes:
_ the author's full name,
_ the author's mail address,
_ the author's web page.

Function XCLE_syntax

```
const char * XCLE_syntax(void) ;
```

Returns a long string displaying basic synatx information for the XCL language.

Returns:

A string containing one object recor dper line.

Note:

Each line contains each object’s type, common name, class and typical string representation.

Execution context structures

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

XCLE_Stack definition

Function XCLE_StackAlloc

XCLE_Stack XCLE_StackAlloc(unsigned long chksz) ;
Allocates a new, empty Stack.

Args:
_ **chksz** : buffer size increment

Returns:
_ **NULL** : if no memory was available
else the newly allocated Stack.

Errors:
_ **ENOMEM** : if no memory was available

Function XCLE_StackFree

void XCLE_StackFree(XCLE_Stack stk) ;
Frees a Stack, dereferences and frees its contents.

Args:

_ **stk** : Stack to free

Function XCLE_StackDepth

unsigned long XCLE_StackDepth(XCLE_Stack stk) ;
Returns the depth (number of elements) of a Stack.

Args:

_ **stk** : Stack we want to get information about

Returns:

_ **-1** : if a NULL Stack was passed as argument
else the Stack's depth.

Errors:

_ **EINVAL** : if a NULL Stack was passed as argument

Function XCLE_StackPush

unsigned long XCLE_StackPush(XCLE_Stack stk, XCLE_Object obj) ;
Pushes and references an object onto a Stack.

Args:

_ **stk** : Stack to push on
_ **obj** : object to push

Returns:

_ **-1** : if some error occurred
else the new depth of the Stack.

Errors:

_ **EINVAL** : if a NULL Stack or object was passed as argument
_ **EACCES** : if the Stack was freezed and consequently cannot receive objects
_ **ENOMEM** : if no memory was available

Function XCLE_StackPop

XCLE_Object XCLE_StackPop(XCLE_Stack stk) ;
Pops and dereferences an object from a Stack.

Args:

_ **stk** : non-empty Stack

Returns:

_ **NULL** : if the Stack was NULL, empty or freezed
else the object popped from the stack.

Errors:

_ **EINVAL** : if a NULL Stack was passed as argument
_ **EACCES** : if the Stack was freezed and consequently cannot yield objects
_ **ENOMEM** : if a memory disallocation failed (non-fatal error)

Function XCLE_StackGet

XCLE_Object XCLE_StackGet(XCLE_Stack stk, unsigned long num) ;
Gets an object at a specified level of a Stack.

Args:
- _ **stk** : non-empty Stack
- _ **num** : level at which to look for an object

Returns:
- _ **NULL** : if the XCLE_Stack was NULL, or the level invalid

else the object at level 'num' of the Stack (first level is 1).

Errors:
- _ **EINVAL** : if a NULL or empty Stack was passed as argument, or the level was invalid

Function XCLE_StackPut

XCLE_Object XCLE_StackPut(XCLE_Stack stk, XCLE_Object obj, unsigned long num) ;
Puts an XCLE_Object at a specified level of a Stack.

Args:
- _ **stk** : Stack to put in
- _ **obj** : object to put
- _ **num** : level at which to put an object

Returns:
- _ **NULL** : if the Stack was NULL, or the level invalid

else the old object at the specified level of the Stack (first level is 1).

Errors:
- _ **EINVAL** : if a NULL Stack or object was passed as argument, or the level was invalid

Function XCLE_StackMap

unsigned long XCLE_StackMap(XCLE_Stack stk, , void * dat) ;
Executes a map handler function on all objects of a Stack.

Args:
- _ **stk** : Stack to map
- _ **obj** : object to put
- _ **map** : map handler
- _ **dat** : user data pointer

Returns:
- _ **-1** : if the Stack or the map handler were NULL
- _ **-1** : if the map handler retuned -1 at some time

else the sum of all the values returned by the map handler

Errors:
- _ **EINVAL** : if a NULL Stack or object was passed as argument, or the level was invalid

and any error generated by the map handler.

Note:

The map handler takes several arguments:

_ **lvl** : the Stack level being currently mapped
_ **obj** : the Object at this level
_ **dat** : the user data pointer passed to XCLE_StackMap

The mapping stops when all the non-empty levels have been mapped, or the map handler returns a value of ~0 (i.e. -1).

Function XCLE_StackChkType

unsigned long XCLE_StackChkType(XCLE_Stack stk, unsigned long argc, XCLE_Type * argt) ;
Checks the number and types of objects on an Stack.

Args:

_ **stk** : Stack to type-check
_ **argc** : number of objects to check
_ **argt** : types of objects to check (or-ed types of admissible objects, level by level)

Returns:

_ **-1** : if the Stack was NULL or there were not enough objects on Stack
else an error flag (0 if all arguments were OK, every bit standing as an individual error flag for each level).

Errors:

_ **EINVAL** : if a NULL Stack was passed as argument

Variables domain definition

Function XCLE_HashAlloc

XCLE_Hash XCLE_HashAlloc(unsigned long chksz) ;
Allocates an empty Hash.

Args:

_ **chksz** : buffer size increment

Returns:

_ **NULL** : if no memory was available
else a new empty Hash.

Errors:

_ **ENOMEM** : if no memory was available

Function XCLE_HashFree

void XCLE_HashFree(XCLE_Hash hsh) ;
Frees a Hash, dereferences and frees its contents.

Args:

_ **hsh** : XCLE_Hash to free

Function XCLE_HashGet

XCLE_Object XCLE_HashGet(XCLE_Hash hsh, char * name) ;
Gets an elements by its key in a Hash.

Args:
- _ **hsh** : Hash to query
- _ **name** : key, or "Object name"

Returns:
- _ **NULL** : if a NULL Hash or name was passed as argument, or no object was indexed by this key

else the Object under this 'name'.

Errors:
- _ **EINVAL** : if a NULL Hash or name was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_HashSet

XCLE_Object XCLE_HashSet(XCLE_Hash hsh, char * name, XCLE_Object obj) ;
Indexes an object under a key in a Hash, and reference it.

Args:
- _ **hsh** : Hash to query
- _ **name** : key, or "Object name"
- _ **obj** : object to set under its "name"

Returns:
- _ **NULL** : if some error occurred

else the old object under this key, if it existed (and it is then dereferenced)
or the object passed as argument.

Errors:
- _ **EINVAL** : if a NULL Hash, name or object was passed as argument
- _ **EACCES** : if the Hash was freezed and consequently cannot receive objects
- _ **ENOMEM** : if no memory was available

Function XCLE_HashDel

XCLE_Object XCLE_HashDel(XCLE_Hash hsh, char * name) ;
Deletes a key and dereferences its contents in a Hash.

Args:
- _ **hsh** : Hash to query
- _ **name** : key, or "Object name"

Returns:
- _ **NULL** : if a NULL Hash or name was passed as argument, or the key was not found

else the object under name 'name'.

Errors:
- _ **EINVAL** : if a NULL Hash or name was passed as argument
- _ **EACCES** : if the Hash was freezed and consequently cannot yield objects

_ **ENOMEM** : if a disallocation failed (non-fatal error)

Function XCLE_HashMap

unsigned long XCLE_HashMap(XCLE_Hash hsh, , void * dat) ;
Executes a map handler function on all objects of a Hash.

Args:

_ **hsh** : Hash to map
_ **obj** : object to put
_ **map** : map handler
_ **dat** : user data pointer

Returns:

_ **-1** : if the Hash or the map handler were NULL
_ **-1** : if the map handler retuned -1 at some time
else the sum of all the values returned by the map handler

Errors:

_ **EINVAL** : if a NULL Hash or object was passed as argument, or the level was invalid
and any error generated by the map handler.

Note:

The map handler takes several arguments:
_ **name** : the Hash key being currently mapped
_ **obj** : the Object at this level
_ **dat** : the user data pointer passed to XCLE_HashMap
The mapping stops when all the keys have been mapped, or the map handler returns a value of ~0 (i.e. -1).

Generic Object Container

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

Object type definitions

Macro XCLE_OT_VOID

XCLE_OT_VOID
The Void type identifier.

Macro XCLE_OT_INTG

XCLE_OT_INTG
The Intg type identifier.

Macro XCLE_OT_FLTP

XCLE_OT_FLTP
The Fltp type identifier.

Macro XCLE_OT_STRG

XCLE_OT_STRG

The Strg type identifier.

Macro XCLE_OT_LIST

XCLE_OT_LIST
The List type identifier.

Macro XCLE_OT_CODE

XCLE_OT_CODE
The Code type identifier.

Generic Object methods

Function XCLE_ObjectAlloc

XCLE_Object XCLE_ObjectAlloc(XCLE_Type typ) ;
Allocates an empty, undefined object.

Args:
_ **typ** : the type of object to allocate

Returns:
_ **NULL** : if no memory was available or the type was not recognized
else a new XCLE_Object of type 'typ'.

Errors:
_ **ENOMEM** : if no memory was available

OOPS:
Something was wrong here

Function XCLE_ObjectClone

XCLE_Object XCLE_ObjectClone(XCLE_Object obj) ;
Clones an object.

Args:
_ **obj** : object to clone

Returns:
_ **NULL** : if a NULL object was passed as argument, or no memory was available
else a cloned version of the object (flat copy).

Errors:
_ **EINVAL** : if a NULL object was passed as argument
_ **ENOMEM** : if no memory was available

OOPS:
Something was wrong here

Function XCLE_ObjectDnRef

XCLE_Object XCLE_ObjectDnRef(XCLE_Object obj) ;
Decreases the references count of an object.

Args:
_ **obj** : XCLE_Object to dereference

Returns:
_ **NULL** : if a NULL object was passed as argument
else the dereferenced object.

Errors:
_ **EINVAL** : if a NULL object was passed as argument

Function XCLE_ObjectFree

void XCLE_ObjectFree(XCLE_Object obj) ;
Frees an object if it is unreferenced.

Args:
_ **obj** : object to free

Note:
The object is freed if and only if its reference count has fallen to zero, meaning no other referenced object and no object storage structure contains it anymore.

Function XCLE_ObjectType

XCLE_Type XCLE_ObjectType(XCLE_Object obj) ;
Returns the type of an object.

Args:
_ **obj** : object to query

Returns:
_ **-1** : if a NULL object was passed as argument
else the numeric identifier for the object's type.

Errors:
_ **EINVAL** : if a NULL object was passed as argument

Function XCLE_ObjectTypeName

char * XCLE_ObjectTypeName(XCLE_Type typ) ;
Returns a name for a numeric type identifier.

Args:
_ **typ** : numeric identifier for a type

Returns:

_ **""** : if an unknown type was given
else a string name for that type.

Function XCLE_ObjectEqual

unsigned char XCLE_ObjectEqual(XCLE_Object obj, XCLE_Object objw) ;
Tests the equality between objects.

Args:
_ **obj** : object to test
_ **objw** : reference object

Returns:
_ **-1** : if a NULL object was passed as argument
_ **0** : if the two objects were distinct
_ **+1** : if the two objects were equal

Errors:
_ **EINVAL** : if a NULL object was passed as argument

Macro XCLE_AnyToObject

XCLE_Object XCLE_AnyToObject(XCLE_<Type> obj) ;
Converts any object into its generic version.

Args:
_ **obj** : object to convert

Returns:
a generic object containing 'obj'

Function XCLE_ObjectTo<Type>

XCLE_<Type> XCLE_ObjectTo<Type>(XCLE_Object obj) ;
Transforms a generic object into a <Type> object, checking that the conversion is possible.

Args:
_ **obj** : the object to convert

Returns:
_ **NULL** : if the object was NULL, or of a different type
else the converted object.

Errors:
_ **EINVAL** : if the object was NULL, or of an incompatible type

Void Object structure and methods

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

XCLE_Void object type

Function XCLE_VoidAlloc

XCLE_Void XCLE_VoidNew(void) ;
Allocates a new Void object.

Returns:
_ **NULL** : if no memory was available
else a new Void object.

Errors:
_ **EINVAL** : if a NULL Void was passed as argument
_ **ENOMEM** : if no memory was available

Note:
A Void object retains no individual information. It is simply an undefined object.
As such, no Copy and Clone methods are needed. Note that the numeric identifier
for a Void object is 0, to provide a simple default object type.

Function XCLE_VoidFree

void XCLE_VoidFree(XCLE_Void vd) ;
Frees a Void object.

Args:
_ **vd** : Void object to free

Intg Object structure and methods

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

XCLE_Intg object type

Function XCLE_IntgAlloc

XCLE_Intg XCLE_IntgAlloc(void) ;
Allocates a new Intg object, with value zero.

Returns:
_ **NULL** : if no memory was available
else a new Intg object with value zero.

Errors:
_ **ENOMEM** : if no memory was available

Note:
The Intg object type is a double capacity (64 bits) integer.

Function XCLE_IntgCopy

XCLE_Intg XCLE_IntgCopy(XCLE_Intg in) ;
Copies the given Intg.

Args:
_ **in** : the object to copy

Returns:
_ **NULL** : if no memory was available
else a new Intg whose value is the same as 'in'.

Errors:
_ **EINVAL** : if a NULL Void was passed as argument
_ **ENOMEM** : if no memory was available

Note:
XCLE_IntgCopy and XCLE_IntgClone do exactly the same thing.

Function XCLE_IntgClone

XCLE_Intg XCLE_IntgClone(XCLE_Intg in) ;
Clones the given Intg.

Args:
_ **in** : the object to clone

Returns:
_ **NULL** : if no memory was available
else a new Intg whose value is the same as 'in'.

Errors:
_ **EINVAL** : if a NULL Void was passed as argument
_ **ENOMEM** : if no memory was available

Note:
XCLE_IntgCopy and XCLE_IntgClone do exactly the same thing.

Function XCLE_IntgFree

void XCLE_IntgFree(XCLE_Intg in) ;
Frees an Intg object.

Args:
_ **in** : Intg object to free

Function XCLE_IntgNew

XCLE_Intg XCLE_IntgNew(long nb) ;
Creates an Intg object from an integer value.

Args:
_ **nb** : value to give to the new object

Returns:
_ **NULL** : if no memory was available
else a new Intg whose value is 'nb'.

Errors:
_ **ENOMEM** : if no memory was available

Function XCLE_IntgEqual

unsigned char XCLE_IntgEqual(XCLE_Intg in, XCLE_Intg inw) ;
Compares two Intg.

Args:
_ **in** : Intg to compare
_ **inw** : reference Intg

Returns:
_ **-1** : if a NULL Intg was passed as argument
_ **0** : if the two Intg have distinct values
_ +1 " if the two Intg have the same value

Errors:
_ **EINVAL** : if a NULL Intg was passed as argument

Function XCLE_IntgValue

long XCLE_IntgValue(XCLE_Intg in) ;
Retrives the value of an Intg object.

Args:
_ **in** : Intg to query

Returns:
_ **0** : if a NULL Intg was passed as argument
else the integer value if the Intg object.

Errors:
_ **EINVAL** : if a NULL Intg was passed as argument

Fltp Object structure and methods

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

XCLE_Fltp object type

Function XCLE_FltpAlloc

XCLE_Fltp XCLE_FltpAlloc(void) ;
Allocates a new Fltp object with value zero.

Returns:
_ **NULL** : if no memory was available
else the allocated Fltp, with value zero.

Errors:
_ **ENOMEM** : if no memory was available

Note:
The Fltp object type is a double precision floating point number.

Function XCLE_FltpCopy

XCLE_Fltp XCLE_FltpCopy(XCLE_Fltp fp) ;
Copies a Fltp object.

Args:
_ **fp** : Fltp object to copy

Returns:
_ **NULL** : if no memory was available
else a new Fltp object with the same value as 'fp'.

Errors:
_ **EINVAL** : if a NULL was passed as argument
_ **ENOMEM** : if no memory was available

Note:
XCLE_FltpCopy and XCLE_FltpClone do exactly the same thing

Function XCLE_FltpClone

XCLE_Fltp XCLE_FltpClone(XCLE_Fltp fp) ;
Clones a Fltp object.

Args:
_ **fp** : Fltp object to clone

Returns:
_ **NULL** : if no memory was available
else a new Fltp object with the same value as 'fp'.

Errors:
_ **EINVAL** : if a NULL was passed as argument
_ **ENOMEM** : if no memory was available

Note:
XCLE_FltpCopy and XCLE_FltpClone do exactly the same thing

Function XCLE_FltpFree

void XCLE_FltpFree(XCLE_Fltp in) ;
Frees an Fltp object.

Args:
_ **in** : Fltp object to free

Function XCLE_FltpNew

XCLE_Fltp XCLE_FltpNew(double nb) ;
Creates a new Fltp object from a floating point value.

Args:
_ **nb** : value to give to the new object

Returns:
_ **NULL** : if no memory was available
else a new Fltp object with value 'nb'.

Errors:
- _ **EINVAL** : if a NULL was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_FltpEqual

unsigned char XCLE_FltpEqual(XCLE_Fltp fp, XCLE_Fltp fpw) ;
Compares two Fltp objects.

Args:
- _ **fp** : Fltp object to compare
- _ **fpw** : reference Fltp object

Returns:
- _ **-1** : if a NULL Fltp was passed as argument
- _ **0** : if the two Fltp objects had distinct values
- _ **+1** : if the two Fltp objects had the same value

Errors:
- _ **EINVAL** : if a NULL was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_FltpValue

double XCLE_FltpValue(XCLE_Fltp fp) ;
Retrives the value of a Fltp object.

Args:
- _ **in** : Fltp to query

Returns:
- _ **0** : if a NULL Fltp was passed as argument

else the floating point value if the Fltp object.

Errors:
- _ **EINVAL** : if a NULL Fltp was passed as argument

Strg Object structure and methods

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

XCLE_Strg object type

Function XCLE_StrgAlloc

XCLE_Strg XCLE_StrgAlloc(unsigned long size) ;
Allocates a new Strg object.

Args:
- _ **size** : length of allocated Strg

Returns:
- _ **NULL** : if no memory was available

else a new Strg object, with length 'size', and filled with '\0' chars.

Errors:
- _ **EINVAL** : if a NULL was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_StrgCopy

XCLE_Strg XCLE_StrgCopy(XCLE_Strg str) ;
Copies a Strg object.

Args:

_ **str** : Strg object to copy

Returns:

_ **NULL** : if no memory was available
else a complete copy of 'str'.

Errors:

_ **EINVAL** : if a NULL Strg was passed as argument
_ **ENOMEM** : if no memory was available

Note:

XCLE_StrgCopy and XCLE_StrgClone do exactly the same thing.

Function XCLE_StrgClone

XCLE_Strg XCLE_StrgClone(XCLE_Strg str) ;
Clones a Strg object.

Args:

_ **str** : Strg object to clone

Returns:

_ **NULL** : if no memory was available
else a complete copy of 'str'.

Errors:

_ **EINVAL** : if a NULL Strg was passed as argument
_ **ENOMEM** : if no memory was available

Note:

XCLE_StrgCopy and XCLE_StrgClone do exactly the same thing.

Function XCLE_StrgFree

void XCLE_StrgFree(XCLE_Strg str) ;
Frees a Strg object.

Args:

_ **str** : Strg object to free

Function XCLE_StrgNew

XCLE_Strg XCLE_StrgNew(char * chs) ;
Creates a Strg object from a character string (C convention, '\0'-terminated).

Args:

_ **chs** : character string to initialize the object from

Returns:

_ **NULL** : if no memory was available or 'chs' was NULL.

else a Strg object with the same characters as 'chs'.

Errors:
- _ **EINVAL** : if a NULL string was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_StrgLen

unsigned long XCLE_StrgLen(XCLE_Strg str) ;
Returns the length of a Strg object.

Args:
- _ **str** : Strg object to query

Returns:
-1 : if 'str' was NULL
else the Strg length.

Errors:
- _ **EINVAL** : if a NULL Strg was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_StrgCat

XCLE_Strg XCLE_StrgCat(XCLE_Strg str1, XCLE_Strg str2) ;
Concatenes two Strg objects, in the arguments' order.

Args:
- _ **str1** : first part of the concatenated string
- _ **str2** : last part of the concatenated string

Returns:
- _ **NULL** : if no memory was available

else the concatenation.

Errors:
- _ **EINVAL** : if a NULL Strg was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_StrgCut

XCLE_Strg XCLE_StrgCut(XCLE_Strg str, unsigned long beg, unsigned long end) ;
Extracts a portion of a Strg object.

Args:
- _ **str** : Strg object to cut from
- _ **beg** : index of first character of extracted part (beginning at 0)
- _ **end** : index of first character after extracted part

Returns:
- _ **NULL** : if no memory was available, or 'beg' > 'end'

a new Strg made form the extracted section.

Errors:

- _ **EINVAL** : if a NULL Strg was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_StrgEqual

unsigned char XCLE_StrgEqual(XCLE_Strg str, XCLE_Strg strw) ;
Compares two Strg object.

Args:

- _ **str** : Strg object to compare
- _ **strw** : reference Strg object

Returns:

- _ **-1** : if a NULL Strg was passed as argument
- _ **0** : if the two Strg were different
- _ **+1** : if the two Strg were equal

Errors:

- _ **EINVAL** : if a NULL Strg was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_StrgValue

unsigned long XCLE_StrgValue(XCLE_Strg str, char * out, unsigned long max) ;
Outputs a Strg characters into a character string.

Args:

- _ **str** : Strg to copy
- _ **out** : output character string
- _ **max** : maximum number of characters to write

Returns:

- _ **-1** : if no memory was available, a NULL Strg or character string was passed else the number of character written.

Errors:

- _ **EINVAL** : if a NULL Strg or character string was passed as argument
- _ **ENOMEM** : if no memory was available

Note:

This method is distinct from XCLE_StrgOut in that it does not output a representation of the Strg as an object (i.e. with quotes, and special characters escaped) but the exact string buffer contents.

List Object structure and methods

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

XCLE_List object type

Function XCLE_ListAlloc

XCLE_List XCLE_ListAlloc(void) ;
Allocates a new, empty, list.

Returns:
_ **NULL** : if no memory was available
else a new List.

Errors:
_ **ENOMEM** : if no memory was available

Function XCLE_ListAllocBlock

XCLE_List XCLE_ListAllocBlock(unsigned long numb) ;
Allocates a new, empty, list, with space reserved.

Args:
_ **numb** : size of space to reservate

Returns:

_ **NULL** : if no memory was available
else a new list, with space already allocated for 'numb' objects.

Errors:

_ **ENOMEM** : if no memory was available

Function XCLE_ListCopy

XCLE_List XCLE_ListCopy(XCLE_List lst) ;
Copies a List object and its contents.

Args:

_ **lst** : List object to copy

Returns:

_ **NULL** : if no memory was available
else a complete (recursive) copy of the List.

Errors:

_ **EINVAL** : if a NULL List was passed as argument
_ **ENOMEM** : if no memory was available

Note:

This method copies the List given, and fills it with copies of the original's contents. Thus, this is a complete reallocation, and no memory buffer from the original List is reused. The new objects' reference counts are set to zero, while the originals' are unchanged.

Function XCLE_ListClone

XCLE_List XCLE_ListClone(XCLE_List lst) ;
Clones a List object.

Args:

_ lst

Returns:

_ **NULL** : if no memory was available
else a clone of the List object.

Errors:

_ **EINVAL** : if a NULL List was passed as argument
_ **ENOMEM** : if no memory was available

Note:

This method does only a "flat" copy of the List object, filling the copy with direct references to the original's objects. The new List reference count is set to zero, while the contained objects' reference counts are unchanged.

Function XCLE_ListDup

XCLE_List XCLE_ListUpRef(XCLE_List lst) ;

Increases the reference count of a List, and of all the objects it contains.

Args:
_ **lst** : the List to reference

Returns:
_ **NULL** : if a NULL List was passed as argument
else the original List object.

Errors:
_ **EINVAL** : if a NULL List was passed as argument

Function XCLE_ListDrop

XCLE_List XCLE_ListDrop(XCLE_List lst) ;
Decreases the reference count of a List, and of all the objects it contains.

Args:
_ **lst** : the List to dereference

Returns:
_ **NULL** : if a NULL List was passed as argument
else the original List object.

Errors:
_ **EINVAL** : if a NULL List was passed as argument

Function XCLE_ListFree

void XCLE_ListFree(XCLE_List lst) ;
Frees a List object, if its reference count is zero.

Args:
_ **lst** : List object to free

Function XCLE_ListLen

unsigned long XCLE_ListLen(XCLE_List lst) ;
Returns the length of a List.

Args:
_ **lst** : List to query

Returns:
_ **-1** : if a NULL List was passed as argument
else the List's length.

Errors:
_ **EINVAL** : if a NULL List was passed as argument

Function XCLE_ListGet

XCLE_Object XCLE_ListGet(XCLE_List lst, unsigned long pos) ;
Gets an object by its position in a List.

Args:
- _ **lst** : List to query
- _ **pos** : position of the object to retrieve

Returns:
- _ **-1** : if a NULL List was passed as argument, or the index was out of range
  else the retrieved object.

Errors:
- _ **EINVAL** : if a NULL List was passed as argument, or the index was out of range

Function XCLE_ListPut

XCLE_Object XCLE_ListPut(XCLE_List lst, unsigned long pos, XCLE_Object obj) ;
Puts an object by its position in a List.

Args:
- _ **lst** : List to update
- _ **pos** : position of the object to put
- _ **obj** : object to put into the List

Returns:
- _ **NULL** : if a NULL List was passed as argument, or the index was out of range
  else the original object that was at this position

Errors:
- _ **EINVAL** : if a NULL List was passed as argument, or the index was out of range

Function XCLE_ListDel

XCLE_List XCLE_ListDel(XCLE_List lst, unsigned long beg, unsigned long end) ;
Deletes the objects in positions [beg,end[in an List.

Args:
- _ **lst** : List to update
- _ **beg** : beginning of the section to cut, starting at 0
- _ **end** : end of this section

Returns:
- _ **NULL** : if a NULL List was passed, indexes were out of range, or a reallocation failed
  else the original List.

Errors:
- _ **EINVAL** : if a NULL List was passed as argument, or indexes were out of range
- _ **ENOMEM** : if no memory was available

Function XCLE_ListIns

XCLE_List XCLE_ListIns(XCLE_List lst, unsigned long pos, XCLE_List lsi) ;
Inserts the objects of the second List in the first, between the pos-th and the pos+1-th objects.

Args:

_ **lst** : List to insert into
_ **pos** : position of insertion point, starting at 0
_ **lsi** : inserted List

Returns:

_ **NULL** : if a NULL List was passed, indexes were out of range, or a reallocation failed
else the first List, with the second's objects inserted.

Errors:

_ **EINVAL** : if a NULL List was passed as argument, or indexes were out of range
_ **ENOMEM** : if no memory was available

Note:

To prevent internal structures and memory corruption, 'lsi' is emptied after successfull insertion. Using a null 'pos' means the second list is inserted at the very beginning of 'lst', while a 'pos' equal to the first List length means to insert at the very end.

Function XCLE_ListPush

XCLE_List XCLE_ListPush(XCLE_List lst, XCLE_Object obj) ;
Adds an object at the end of a List.

Args:

_ **lst** : List to update
_ **obj** : object to append

Returns:

_ **NULL** : if a NULL List or object was passed as argument, or a reallocation failed
else the updated List, with the object appended.

Errors:

_ **EINVAL** : if a NULL List or object was passed as argument
_ **ENOMEM** : if no memory was available

Function XCLE_ListPop

XCLE_Object XCLE_ListPop(XCLE_List lst) ;
Deletes an object from the end of a List.

Args:

_ **lst** : List to update

Returns:

_ **NULL** : if a NULL List was passed as argument, the List was empty, or a reallocation failed
else the object deleted from the ned of the List.

Errors:

_ **EINVAL** : if a NULL List was passed as argument or the List was empty
_ **ENOMEM** : if no memory was available

Function XCLE_ListUnshift

XCLE_List XCLE_ListUnshift(XCLE_List lst, XCLE_Object obj) ;
Adds an object at the beginning of a List.

Args:
- _ **lst** : List to update
- _ **obj** : object to prepend

Returns:
- _ **NULL** : if a NULL List was passed as argument or a reallocation failed

else the updated list, with the object prepended.

Errors:
- _ **EINVAL** : if a NULL List was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_ListShift

XCLE_Object XCLE_ListShift(XCLE_List lst) ;
Deletes an object from the beginning of a List.

Args:
- _ **lst** : List to update

Returns:
- _ **NULL** : if a NULL List was passed as argument, the List was empty, or a reallocation failed

else the object deleted from the beginning of the List.

Errors:
- _ **EINVAL** : if a NULL List was passed as argument or the List was empty
- _ **ENOMEM** : if no memory was available

Function XCLE_ListMap

unsigned long XCLE_ListMap(XCLE_List lst, , void * data) ;
Executes a function on all objects of a List, totalizing the return values.

OOPS:
Something was wrong here

Function XCLE_ListSort

XCLE_List XCLE_ListSort(XCLE_List lst,) ;
Sorts a List, using the provided object comparison function.

OOPS:
Something was wrong here

Function XCLE_ListEqual

unsigned char XCLE_ListEqual(XCLE_List lst, XCLE_List lstw) ;
Compares two Lists.

Args:

- **lst** : List object to compare
- **lstw** : reference List

Returns:

- **-1** : if a NULL List was passed as argument
- **0** : if the two Lists were distinct
- **+1** : if the two Lists were identical (i.e., same length, and equal objects)

Errors:

- **EINVAL** : if a NULL List was passed as argument

Code Object structure and methods

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

XCLE_Code memory management

Function XCLE_CodeAlloc

XCLE_Code XCLE_CodeAlloc(void) ;
Allocates a new Code object, with no execution handler nor data section.

Returns:
_ **NULL** : if no memory was available
else a new Code object.

Errors:
_ **EINVAL** : if a NULL was passed as argument
_ **ENOMEM** : if no memory was available

Note:
This constructor is only there as a utility, for the returned code object is totally useless: it has no name, no stack expectaion, no execution handler, and its data section is undefined. Using it would only result in a place filler, and executing it would do nothing.

Function XCLE_CodeCopy

XCLE_Code XCLE_CodeCopy(XCLE_Code cod) ;
Copies a Code object, along with its data segment.

Args:
_ **cod** : Code object to copy

Returns:
_ **NULL** : if a NULL Code was passed as argument or no memory was available
else a complete copy of the original Code object.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument
_ **ENOMEM** : if no memory was available

Note:
This is a complete copy: the Code's data segment is also replicated.

Function XCLE_CodeClone

XCLE_Code XCLE_CodeClone(XCLE_Code cod) ;
Clones a Code object.

Args:
_ **cod** : Code object to clone

Returns:
_ **NULL** : if a NULL Code was passed as argument or no memory was available
else a simple copy of the Code object.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument
_ **ENOMEM** : if no memory was available

Note:
This is a partial copy: the Code's data segment is the same as the original's.

Function XCLE_CodeUpRef

XCLE_Code XCLE_CodeDup(XCLE_Code cod) ;
Increases the reference count of a Code object and of its data segment.

Args:
_ **cod** : Code to reference

Returns:
_ **NULL** : if a NULL Code was passed as argument
else the referenced Code object.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument

Function XCLE_CodeDnRef

XCLE_Code XCLE_CodeDnRef(XCLE_Code cod) ;
Decreases the reference count of a Code object and of its data segment.

Args:
_ **cod** : Code to dereference

Returns:
_ **NULL** : if a NULL Code was passed as argument
else the dereferenced Code object.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument
_ **ENOMEM** : if no memory was available

Function XCLE_CodeFree

void XCLE_CodeFree(XCLE_Code cod) ;
Frees a Code object.

Args:
_ **cod** : Code to free

XCLE_Code name, handler, and data

Function XCLE_CodeGetName

char * XCLE_CodeGetName(XCLE_Code cod) ;
Returns the name field of a Code object.

Args:
_ **cod** : Code to query

Returns:
_ **NULL** : if a NULL Code was passed as argument, or no memory was available
else the name of this Code, as a \0-terminated string.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument
_ **ENOMEM** : if no memory was available

Note:
The returned char * buffer is allocated dynamically, with malloc. It is the responsibility of the programmer to free it when it is not used anymore.

Function XCLE_CodeSetName

XCLE_Code XCLE_CodeSetName(XCLE_Code cod, char * name) ;

Updates the name field of a Code objetc.

Args:
- _ **cod** : Code to update
- _ **name** : the new name (\0-terminated string)

Returns:
- _ **NULL** : if a NULL Code was passed as argument, or no memory was available

else the original 'cod' object.

Errors:
- _ **EINVAL** : if a NULL Code was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_CodeGetHandler

void * XCLE_CodeGetHandler(XCLE_Code cod) ;
Returns a pointer to the execution handler of a Code object.

Args:
- _ **cod** : Code to query

Returns:
- _ **NULL** : if a NULL Code was passed as argument

else the pointer to the execution handler.

Errors:
- _ **EINVAL** : if a NULL Code was passed as argument

Note:

If you do not know what is the Code's execution handler, avoid this function.
For instruction set programmers: this is the same address as the 'func' member in the CodeDef structure, that is, a CodeOperator function pointer.

Function XCLE_CodeGetHandler

XCLE_Code XCLE_CodeGetHandler(XCLE_Code cod, void * func) ;
Returns a pointer to the execution handler of a Code object.

Args:
- _ **cod** : Code to query
- _ **func** : the new execution handler

Returns:
- _ **NULL** : if a NULL Code was passed as argument

else the original 'cod' object.

Errors:
- _ **EINVAL** : if a NULL Code was passed as argument

Note:

If you do not know what is the Code's execution handler, avoid this function.
For instruction set programmers: execution handler are generally set through the parsing routines, that will look up a CodeDef structure from declared tables and set execution handlers accordingly. Setting it by hand should be done with care.

Function XCLE_CodeGetData

XCLE_Object XCLE_CodeGetData(XCLE_Code cod) ;
Returns the data segment of a Code object.

Args:
_ **cod** : Code to query

Returns:
_ **NULL** : if a NULL Code was passed as argument
else the data segment.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument

Note:
If you do not know what is the Code's data segment, avoid this function.
For instruction set programmers: instructions have generally empty (NULL) data segment. These are only used for parametrable instructions.

Function XCLE_CodeSetData

XCLE_Code XCLE_CodeSetData(XCLE_Code cod, XCLE_Object data) ;
Sets the data segment of a Code object.

Args:
_ **cod** : Code to update
_ **data** : data segment

Returns:
_ **NULL** : if a NULL Code was passed as argument
else the original 'cod' objetc.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument

Note:
If you do not know what is the Code's data segment, avoid this function.
For instruction set programmers: you will probably let the users determines data segments for parametrable instructions dynamically, through the parsing routines. However, executing a checking pass on data segments before execution can be useful. Combine the GetData and SetData methods with recursive ListMap calls to provide a data segment verification pass.

XCLE_Code signature and prototypes

Function XCLE_CodeSignatureNum

unsigned short XCLE_CodeSignatureNum(XCLE_Code cod) ;

Returns the number of distinct signatures for this Code.

Args:
_ **cod** : Code to query

Returns:
_ **-1** : if a NULL Code was passed as argument
else the number of distinct signatures.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument

Note:
A Code's signature is a set of expected arguments number/types, and the corresponding returned object number/types. Several can exist for a single Code, covering different instruction semantics. This function only counts the number of such semantics.

Function XCLE_CodeSignatureArgc

unsigned short XCLE_CodeSignatureArgc(XCLE_Code cod, unsigned short sign) ;
Returns the number of arguments required for a particular signature.

Args:
_ **cod** : Code to query
_ **sign** : index of signature

Returns:
_ **-1** : if a NULL Code was passed as argument
else the number of arguments required for the signature of index 'sign'.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument, or the signature index was invalid

Note:
A Code's signature is a set of expected arguments number/types, and the corresponding returned object number/types. Several can exist for a single Code, covering different instruction semantics. This function will return the number of arguments expected for the designated set.

Function XCLE_CodeSignatureRetc

unsigned short XCLE_CodeSignatureRetc(XCLE_Code cod, unsigned short sign) ;
Returns the number of results returned for a particular signature.

Args:
_ **cod** : Code to query
_ **sign** : index of signature

Returns:
_ **-1** : if a NULL Code was passed as argument
else the number of results returned for the signature of index 'sign'.

Errors:
_ **EINVAL** : if a NULL Code was passed as argument, or the signature index was invalid

Note:

A Code's signature is a set of expected arguments number/types, and the corresponding returned object number/types. Several can exist for a single Code, covering different instruction semantics. This function will return the number of results returned for the designated set.

Function XCLE_CodeSignatureMatch

unsigned short XCLE_CodeSignatureMatch(XCLE_Code cod, unsigned short argc, XCLE_Type * argt, unsigned short retc, XCLE_Type * rett) ;
Returns the number of signature matches for a given set of argument types and expected results.

Args:

- _ **cod** : Code to query
- _ **argc** : number of arguments
- _ **argt** : argument types (this is an XCLE_Type[argc] vector)
- _ **retc** : number of expected returns
- _ **rett** : types of expected returns (this is an XCLE_Type[retc] vector)

Returns:

- _ **-1** : if a NULL Code was passed as argument

else the number of signature matches.

Errors:

- _ **EINVAL** : if a NULL Code was passed as argument

Note:

This function computes the number of execution semantics supported by the Code object and consistent with the given argument types and expected returns. In most cases, nothing is expected of returned values, so setting 'retc' to 0 and 'rett' to NULL is appropriate.

Function XCLE_CodeSignatureAdd

XCLE_Code XCLE_CodeSignatureAdd(XCLE_Code cod, unsigned short argc, XCLE_Type * argt, unsigned short retc, XCLE_Type * rett) ;
Adds a new accepted signature for a Code object.

Args:

- _ **cod** : the Code to update
- _ **argc** : number of arguments
- _ **argt** : argument types (this is an XCLE_Type[argc] vector)
- _ **retc** : number of returns
- _ **rett** : returns types (this is an XCLE_Type[retc] vector)

Returns:

- _ **NULL** : if a NULL Code was passed as argument or an error occured

else the updated 'cod' object.

Errors:

- _ **ENOMEM** : if a reallocation failed
- _ **EINVAL** : if a NULL Code was passed as argument

Note:

A newly allocated Code object has no registered signature (XCLE_CodeSignatureNum would return 0). This is the method to be used to set the signature(s) before profiling or execution of objects using this Code.

Function XCLE_CodeSignatureDel

XCLE_Code XCLE_CodeSignatureDel(XCLE_Code cod, unsigned short argc, XCLE_Type * argt, unsigned short retc, XCLE_Type * rett) ;
Deletes a signature from a Code object.

Args:
- _ **cod** : the Code to update
- _ **argc** : number of arguments
- _ **argt** : argument types (this is an XCLE_Type[argc] vector)
- _ **retc** : number of returns
- _ **rett** : returns types (this is an XCLE_Type[retc] vector)

Returns:
- _ **NULL** : if a NULL Code was passed as argument

else the updated 'cod' object.

Errors:
- _ **EINVAL** : if a NULL Code was passed as argument

Note:
This method is provided for the sake of complteness, as deleting code signatures has no real use. The 'argc', 'argt', 'retc' and 'rett' must have the same values as those passed to XCLE_CodeSignatureAdd to add the targeted signature.

XCLE_Code object conversion and comparison

Function XCLE_ObjectToCode

XCLE_Code XCLE_ObjectToCode(XCLE_Object obj) ;
Converts a generic object into a Code object, if possible.

Args:
- _ **obj** : Object to convert

Returns:
- _ **NULL** : if a NULL object, or one with incompatible types, was passed as argument

else the actual Code object embedded into 'obj'.

Errors:
- _ **EINVAL** : if a NULL or invalid object was passed as argument

Function XCLE_CodeEqual

unsigned char XCLE_CodeEqual(XCLE_Code cod, XCLE_Code codw) ;
Compares two Code objects.

Args:

- _ **cod** : Code to compare
- _ **codw** : reference Code

Returns:

- _ **-1** : if a NULL Code was passed as argument
- _ **0** : if the two Codes were distinct
- _ **+1** : if the two Codes had the same name, argument and return expectations, and data segment

Errors:

- _ **EINVAL** : if a NULL Code was passed as argument

String conversion module

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

Canonical object string representation

Function XCLE_ObjectToString

unsigned long XCLE_ObjectToString(XCLE_Object obj, char * out, unsigned long len) ;
Writes a character string representation of a generic object.

Args:

- _ **obj** : object to write
- _ **out** : output string (allocated for at least len+1 characters)
- _ **len** : max number of characters to write

Returns:

- _ **-1** : if some error occurred
- _ the maximum number of characters needed to write completely the object if 'out' is NULL (len is ignored)
  else the total number of characters written.

Errors:

- _ **EINVAL** : if a NULL XCLE_Object was passed as argument

Function XCLE_VoidToString

unsigned long XCLE_VoidToString(XCLE_Void vd, char * out, unsigned long len) ;
Writes a Void object representation into a string: '¬'.

Args:
- _ **vd** : Void object to write
- _ **out** : output string
- _ **len** : max number of characters to write, trailing '\0' excluded

Returns:
- _ **-1** : if some error occured
- _ **1** : if 'out' was NULL or len>=1
- _ **0** : if 'out' was non-NULL and len was zero

Errors:
- _ **EINVAL** : if a NULL Void was passed as argument

Function XCLE_IntgToString

unsigned long XCLE_IntgToString(XCLE_Intg in, char * out, unsigned long len) ;
Writes an Intg object representation into a character string.

Args:
- _ **in** : Intg object to write
- _ **out** : output string
- _ **len** : maximum number of characters to write, trailing '\0' excluded

Returns:
- _ **NULL** : if no memory was available
- _ the number of characters needed to write an Intg, if 'out' was NULL
  else the number of character written (10/11, or up to len).

Errors:
- _ **EINVAL** : if a NULL Intg was passed as argument

Function XCLE_FltpToString

unsigned long XCLE_FltpToString(XCLE_Fltp fp, char * out, unsigned long len) ;
Writes a Fltp object representation into a character string.

Args:
- _ **fp** : Fltp object to write
- _ **out** : output string
- _ **len** : maximum number of characters to write, trailing '\0' excluded

Returns:
- _ **NULL** : if no memory was available
- _ the maximum number of characters needed, if 'out' was NULL
  else the number of characters written.

Errors:
- _ **EINVAL** : if a NULL was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_StrgToString

unsigned long XCLE_StrgToString(XCLE_Strg str, char * out, unsigned long len) ;
Writes a Strg object representation into a character string.

Args:
- _ str
- _ out
- _ len

Returns:
- _ **NULL** : if no memory was available

Errors:
- _ **EINVAL** : if a NULL Strg was passed as argument
- _ **ENOMEM** : if no memory was available

Function XCLE_ListToString

unsigned long XCLE_ListToString(XCLE_List lst, char * out, unsigned long len) ;
Writes a List object representation into a character string.

Args:
- _ **lst** : List object to write
- _ **out** : output character string (allocated for at least len+1 characters)
- _ **len** : maximum number of characters to write

Returns:
- _ **-1** : if a NULL List was passed as argument
- _ the maximum number of characters neede to write this List, if 'out' was NULL else the number of characters written.

Errors:
- _ **EINVAL** : if a NULL List was passed as argument

Function XCLE_CodeToString

unsigned long XCLE_CodeToString(XCLE_Code cod, char * out, unsigned long len) ;
Writes a Code object representation into a character string.

Args:
- _ **cod** : Code to write
- _ **out** : output string (allocated for at least len+1 characters)
- _ **len** : maximum number of character to write

Returns:
- _ **NULL** : if a NULL Code was passed as argument
- _ the number of characters neeede to write completely the Code, if 'out' was NULL else the total number of characters written.

Errors:
- _ **EINVAL** : if a NULL Code was passed as argument

Function XCLE_StackToString

unsigned long XCLE_StackToString(XCLE_Stack stk, char * out, unsigned long max, char * head, char * rowfmt, char * tail, unsigned long rows) ;
Prints the contents of an Stack, following the given formats.

Args:
- _ **stk** : Stack to print
- _ **out** : output character string
- _ **max** : maximum number of characters to print (final nil char EXCEPTED)
- _ **head** : head string
- _ **rowfmt** : row printing format
- _ **tail** : tail string
- _ **rows** : number of rows (height in chars)

Returns:
- _ **-1** : if the Stack was NULL
- _ the total number of characters that should be printed, if 'out' was NULL.
  else the number of characters printed.

Errors:
- _ **EINVAL** : if a NULL Stack was passed as argument

Note:
The row format string is printed for each stack row between 0 and rows-1.
Escape sequences take the form %fdd..dd?, where the 'f' char is an optionnal fill character (by dfault ' '), 'd' chars are digits and '?' denotes either the 'n' or 's' characters. They are replaced by the stack row number, for the 'n' format, and the corresponding object, for the 's' format, each written on the number of chars denoted by the digits.

Function XCLE_HashToString

unsigned long XCLE_HashToString(XCLE_Hash nms, char * out, unsigned long max, char * head, char * rowfmt, char * tail) ;
Prints a Hash keys and contents, acording to formats.

Args:
- _ **nms** : Hash to print
- _ **out** : output character string
- _ **max** : maximum number of characters to print (final nil char EXCEPTED)
- _ **head** : head string
- _ **rowfmt** : row printing format
- _ **tail** : tail string
- _ **rows** : number of rows (height in chars)

Returns:
- _ **-1** : if the Hash was NULL
- _ the total number of characters that should be printed, if 'out' was NULL.
  else the number of characters printed.

Errors:
- _ **EINVAL** : if a NULL Hash was passed as argument

Note:
The row format string is printed for each name entry in the Hash.

Escape sequences take the form %fdd..dd?, where the 'f' char is an optionnal fill character (by dfault ' '), 'd' chars are digits and '?' denotes either the 'n' or 's' characters. They are replaced by the entry name, for the 'n' format, and the corresponding object, for the 's' format, each written on the number of chars denoted by the digits.

Object Writing routines

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

Structures and typedefs

Typedef XCLE_WriteHandler

typedef unsigned long (* XCLE_WriteHandler) (void * dat, char * chr, unsigned long len) ;
Handler for the write operation.

Note:

The programmer has the responsability to keep in the 'dat' pointer any useful information. The handler is charged to effect the actual writing operation on whatever support is used, and must return the number of characters actually written, that may be less than 'len'.

Typedef XCLE_WriteFormat

typedef struct { ... } XCLE_WriteFormat ;
Data formatting information for the XCLE_*Write methods.

Members:

- **_ formflag** : Format flags
- **_ intgprec** : Precision (number of digits) for integers
- **_ intgbase** : Base for printing integers

_ **fltpprec** : Precision (number of digits) for floating point numbers
_ **fltpdeci** : Decimals (digits after separ.) for floating point numbers

Writer routines for object types

Function XCLE_ObjectWrite

unsigned long XCLE_ObjectWrite(XCLE_Object obj, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt) ;
Writing method for XCLE_Object.

Args:
_ **obj** : XCLE_Object to write
_ **wrt** : handler for the real write operation
_ **dat** : user-set data pointer to pass the handler
_ **fmt** : data formatting information

Returns:
_ **-1** : if some error occurred
_ the total number of character needed, if 'wrt' is NULL
else the total number of characters printed

Errors:
_ **EINVAL** : if a NULL XCLE_Object was passed as argument
or any error generated by the 'wrt' handler.

Function XCLE_VoidWrite

unsigned long XCLE_VoidWrite(XCLE_Void vd, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt) ;
Writing method for XCLE_Void.

Args:
_ **vd** : XCLE_Void to write
_ **wrt** : handler for the real write operation
_ **dat** : user-set data pointer to pass the handler
_ **fmt** : data formatting information

Returns:
_ **-1** : if some error occurred
_ the total number of character needed, if 'wrt' is NULL
else the total number of characters printed

Errors:
_ **EINVAL** : if a NULL XCLE_Void was passed as argument
or any error generated by the 'wrt' handler.

Function XCLE_IntgWrite

unsigned long XCLE_IntgWrite(XCLE_Intg in, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt) ;

Writing method for XCLE_Intg.

Args:
- _ **in** : XCLE_Intg to write
- _ **wrt** : handler for the real write operation
- _ **dat** : user-set data pointer to pass the handler
- _ **fmt** : data formatting information

Returns:
- _ **-1** : if some error occurred
- _ the total number of character needed, if 'wrt' is NULL else the total number of characters printed

Errors:
- _ **EINVAL** : if a NULL XCLE_Intg was passed as argument or any error generated by the 'wrt' handler.

Function XCLE_FltpWrite

unsigned long XCLE_FltpWrite(XCLE_Fltp fp, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt) ;

Writing method for XCLE_Fltp.

Args:
- _ **fp** : XCLE_Fltp to write
- _ **wrt** : handler for the real write operation
- _ **dat** : user-set data pointer to pass the handler
- _ **fmt** : data formatting information

Returns:
- _ **-1** : if some error occurred
- _ the total number of character needed, if 'wrt' is NULL else the total number of characters printed

Errors:
- _ **EINVAL** : if a NULL XCLE_Fltp was passed as argument or any error generated by the 'wrt' handler.

Function XCLE_StrgWrite

unsigned long XCLE_StrgWrite(XCLE_Strg str, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt) ;

Writing method for XCLE_Strg.

Args:
- _ **str** : XCLE_Strg to write
- _ **wrt** : handler for the real write operation
- _ **dat** : user-set data pointer to pass the handler
- _ **fmt** : data formatting information

Returns:
- _ **-1** : if some error occurred
- _ the total number of character needed, if 'wrt' is NULL else the total number of characters printed

Errors:

_ **EINVAL** : if a NULL XCLE_Strg was passed as argument
or any error generated by the 'wrt' handler.

Function XCLE_ListWrite

unsigned long XCLE_ListWrite(XCLE_List lst, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt) ;
Writing method for XCLE_List.

Args:

_ **lst** : XCLE_List to write
_ **wrt** : handler for the real write operation
_ **dat** : user-set data pointer to pass the handler
_ **fmt** : data formatting information

Returns:

_ **-1** : if some error occurred
_ the total number of character needed, if 'wrt' is NULL
else the total number of characters printed

Errors:

_ **EINVAL** : if a NULL XCLE_List was passed as argument
or any error generated by the 'wrt' handler.

Function XCLE_CodeWrite

unsigned long XCLE_CodeWrite(XCLE_Code cod, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt) ;
Writing method for XCLE_Code.

Args:

_ **cod** : XCLE_Code to write
_ **wrt** : handler for the real write operation
_ **dat** : user-set data pointer to pass the handler
_ **fmt** : data formatting information

Returns:

_ **-1** : if some error occurred
_ the total number of character needed, if 'wrt' is NULL
else the total number of characters printed

Errors:

_ **EINVAL** : if a NULL XCLE_Code was passed as argument
or any error generated by the 'wrt' handler.

Writer routines for execution context objects

Function XCLE_StackWrite

unsigned long XCLE_StackWrite(XCLE_Stack stk, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt, char * head, char * rowf, char * tail, unsigned long rown) ;

Writing method for XCLE_Code.

Args:

- _ **stk** : XCLE_Stack to write
- _ **wrt** : handler for the real write operation
- _ **dat** : user-set data pointer to pass the handler
- _ **fmt** : data formatting information
- _ **head** : header line
- _ **rowf** : row format line
- _ **tail** : tailer line
- _ **rown** : number of rows to write

Returns:

- _ **-1** : if some error occurred
- _ the total number of character needed, if 'wrt' is NULL else the total number of characters printed

Errors:

- _ **EINVAL** : if a NULL XCLE_Stack, or an invalid format string, was passed as argument or any error generated by the 'wrt' handler.

Note:

The row format string is printed for each stack row between 0 and rown-1. Escape sequences take the form %pdd..dd?, where the 'p' char is an optionnal padding character (by default ' '), 'd' chars are digits and '?' denotes either the 'n' or 's' characters. They are replaced by the stack row number, for the 'n' format specifier, and the corresponding object, for the 's' format, each written on the number of chars denoted by the digits.

Function XCLE_HashWrite

unsigned long XCLE_HashWrite(XCLE_Stack stk, XCLE_WriteHandler wrt, void * dat, XCLE_WriteFormat fmt, char * head, char * rowf, char * tail) ;

Writing method for XCLE_Code.

Args:

- _ **stk** : XCLE_Stack to write
- _ **wrt** : handler for the real write operation
- _ **dat** : user-set data pointer to pass the handler
- _ **fmt** : data formatting information
- _ **head** : header line
- _ **rowf** : row format line
- _ **tail** : tailer line

Returns:

- _ **-1** : if some error occurred
- _ the total number of character needed, if 'wrt' is NULL else the total number of characters printed

Errors:

- _ **EINVAL** : if a NULL XCLE_Stack, or an invalid format string, was passed as argument or any error generated by the 'wrt' handler.

Note:

The row format string is printed for each key in the Hash. Escape sequences take the form %pdd..dd?, where the 'p' char is an optionnal

padding character (by default ' '), 'd' chars are digits and '?' denotes either the 'n' or 's' characters. They are replaced by the sh key, for the 'n' format specifier, and the corresponding object, for the 's' format, each written on the number of chars denoted by the digits.

Parsing module

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

Parsing context definition

Function XCLE_ParseCtxAlloc

XCLE_ParseCtx XCLE_ParseCtxAlloc(void) ;
Creates a new parsing context.

Returns:
- _ **NULL** : if memory allocation could not be performed
  else a new XCLE_ParseCtx object.

Function XCLE_ParseCtx_DescByName

const char * XCLE_ParseCtx_DescByName(XCLE_ParseCtx contx, char * str) ;
Finds a description for a Code name, looking for a definition in registered tables.

Args:
- _ **contx** : parsing context
- _ **str** : string to look up in the tables

Returns:
- _ **NULL** : if the string or parsing context were NULL, or the string was not found in the tables

else a Code of name 'str', built from the related definition.

NOTE:
Use XCLE_CodeRegister to register instructions before calling XCLE_CodeByName.

Function XCLE_ParseCtx_CodeByName

XCLE_Code XCLE_ParseCtx_CodeByName(XCLE_ParseCtx contx, char * str) ;
Creates a Code object by its name, looking for a definition in registered tables.

Args:
_ **contx** : parsing context
_ **str** : string to look up in the tables

Returns:
_ **NULL** : if the string or parsing context were NULL, or the string was not found in the tables
else a Code of name 'str', built from the related definition.

NOTE:
Use XCLE_CodeRegister to register instructions before calling XCLE_CodeByName.

Registry and Dynamic Loading

Function XCLE_CodeRegister

unsigned long XCLE_CodeRegister(XCLE_ParseCtx contx, char * name, XCLE_CodeDef * table, unsigned long tablesize) ;
Registers a primitives module.

Args:
_ **contx** : parsing context
_ **name** : module name
_ **table** : table of primitives definitions
_ **tablesize** : size of the definitions table

Returns:
_ **-1** : if the parsing context, the module name or the table were NULL
_ **0** : if the table was empty
else the number of modules registered.

Errors:
_ **ENOMEM** : if a reallocation failed
_ **EINVAL** : if the context was not properly defined, or the module name was NULL
_ **EINVAL** : if the table was NULL or empty

Function XCLE_CodeUnregister

unsigned long XCLE_CodeUnegister(XCLE_ParseCtx contx, char * name) ;
Unregisters a primitives module.

Args:

- _ **contx** : parsing context
- _ **name** : module name

Returns:

- _ **-1** : if the parsing context or the module name were NULL

else the number of modules registered.

Errors:

- _ **ENOMEM** : if a reallocation failed
- _ **EINVAL** : if the context was not properly defined
- _ **EINVAL** : if the module name was NULL or could not be found

Function XCLE_CodeLoad

unsigned long XCLE_CodeLoad(XCLE_ParseCtx contx, char * module) ;
Registers instruction definitions from a module file.

Args:

- _ **contx** : parsing context
- _ **module** : file name of the module

Returns:

- _ **-1** : if the parsing context was NULL, or the module was invalid

else the size of the table.

Errors:

- _ **ENOMEM** : if a reallocation failed
- _ **ENOENT** : if the module was not found
- _ **EINVAL** : if the context was not properly defined or the module structure invalid

NOTE:

This loading interface asks for two symbol names to be defined in modules (shared objects, or dynamically loaded libraries):

- _ **unsigned long XCL_Registry_Vers** : a version identifier produced by XCLE_MAKEVERSIONID
- _ **unsigned long XCL_Registry_Size** : the number of CodeDef objects defined by the module
- _ **XCLE_CodeDef XCL_Registry_Table** : a table of CodeDef objects defining the module contents

To prevent loading of modules compiled with incompatibles versions of XCLE, the module is loaded only if XCL_Registry_Vers matches XCLE_versionid() in their first two bytes.
Module developers should use the macro XCLE_MAKEVERSIONID with the major, minor and release version number of the XCLE library they use to produce XCL_Registry_Vers.

Parsing

Function XCLE_ObjectParse

XCLE_Object XCLE_ObjectParse(XCLE_ParseCtx contx, char * str, unsigned long * off) ;
Creates an object, from a character string object representation.

Args:

_ **contx** : parsing context
_ **str** : string to parse
_ **off** : offset pointer in the string indicating where to begin (or NULL)

Returns:
_ **NULL** : if a syntax error occurred or the end of the string was reached
else the next parsed XCLE_Object.

Errors:
_ **EINVAL** : if a syntax error occurred
_ **ENOMEM** : if no memory was available

Function XCLE_ListParse

XCLE_List XCLE_ListParse(XCLE_ParseCtx contx, char * str, unsigned long * off) ;
Parses a string into a list of objects.

Args:
_ **contx** : parsing context
_ **str** : string to parse
_ **off** : offset pointer in the string indicating where to begin (or NULL)

Returns:
_ **NULL** : if a syntax error occurred, or no object was found
else an XCLE_List of all recognised XCLE_Objects.

Errors:
_ **EINVAL** : if a syntax error occurred
_ **ENOMEM** : if no memory was available

Execution module

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

Return status management

Macro XCLE_EXCEPTION_OK

XCLE_Exception XCLE_EXCEPTION_OK ;
The Exception returned when execution was successful.

Note:
The comparison operator == can be used to test whether an XCLE_Exception is the same as XCLE_EXCEPTION_OK (since this is a static pointer value).

Function XCLE_ExceptionNew

XCLE_Exception XCLE_ExceptionNew(unsigned long ident, char * messg) ;
Creates a new XCLE_Exception with the specified error number, exception identifier, and message.

Args:
- **ident** : error identifier number
- **messg** : description of the error

Returns:
- **NULL** : if the memory could not be allocated

_ **XCLE_EXCEPTION_OK**: XCLE_ExceptionNew(0,NULL)
else a new XCLE_Exception.

Errors:
_ **ENOMEM** : if new memory could not be allocated

Function XCLE_ExceptionCopy

XCLE_Exception XCLE_ExceptionCopy(XCLE_Exception exp) ;
Creates a memory copy of the given exception.

Args:
_ **exp** : the original exception

Returns:
_ **NULL** : if the memory could not be allocated
_ **XCLE_EXCEPTION_OK**: if XCLE_EXCEPTION_OK was passed as argument
else a new XCLE_Exception.

Errors:
_ **ENOMEM** : if new memory could not be allocated
_ **EINVAL** : if a NULL XCLE_Exception was passed as argument

Function XCLE_ExceptionIdent

unsigned long XCLE_ExceptionIdent(XCLE_Exception exp) ;
Returns the error identifier of an exception.

Args:
_ **exp** : the exception

Returns:
_ **-1** : if a NULL XCLE_Exception was passed as argument
_ **0** : if XCLE_EXCEPTION_OK was passed as argument
else the exception's error identifier

Errors:
_ **EINVAL** : if a NULL XCLE_Exception was passed as argument

Function XCLE_ExceptionMessg

const char * XCLE_ExceptionMessg(XCLE_Exception exp) ;
Returns the message string of an exception.

Args:
_ **exp** : the exception

Returns:
_ **NULL** : if a NULL XCLE_Exception was passed as argument
_ **""** : if XCLE_EXCEPTION_OK was passed as argument
else the exception's message string

Errors:

_ **EINVAL** : if a NULL XCLE_Exception was passed as argument

Function XCLE_ExceptionFree

void XCLE_ExceptionFree(XCLE_Exception exp) ;
Frees an XCLE_Exception.

Args:
_ **exp** : XCLE_Exception to free

Note:
Freeing XCLE_EXCEPTION_OK using XCLE_ExceptionFree does nothing

Function XCLE_ExecCtxAlloc

XCLE_ExecCtx XCLE_ExecCtxAlloc(unsigned long chksz) ;
Allocates an execution context.

Args:
_ **chksz** : the chunk size used for the exception stack

Returns:
_ **NULL** : if the memory could not be allocated
else a new, empty, execution context.

Errors:
_ **ENOMEM** : if new memory could not be allocated

Function XCLE_ExecCtxFree

void XCLE_ExecCtxFree(XCLE_ExecCtx ctx) ;
Frees an execution context, along any eventual pending exception.

Args:
_ **ctx** : the execution context to free

Function XCLE_ExecCtxThrow

XCLE_ExecCtx XCLE_ExecCtxThrow(XCLE_ExecCtx ctx, char * src, XCLE_Exception exp) ;
Throws an exception in this context.

Args:
_ **ctx** : the execution context
_ **src** : the name of the calling code
_ **exp** : the exception to throw

Returns:
_ **NULL** : if 'ctx', 'src', or 'exp' were NULL
else the original context, with the exception added to the pending queue

Errors:

_ **ENOMEM** : if new memory could not be allocated
_ **EINVAL** : if 'ctx', 'src', or 'exp' were NULL

Note:
Nothing is done if exp is XCLE_EXCEPTION_OK.

Function XCLE_ExecCtxCatch

XCLE_Exception XCLE_ExecCtxCatch(XCLE_ExecCtx ctx) ;
Catches the last thrown exception.

Args:
_ **ctx** : the execution context

Returns:
_ **NULL** : if the execution context was NULL
_ **XCLE_EXCEPTION_OK** : if no exception was pending in this context
else the last thrown exception

Errors:
_ **EINVAL** : if the execution context was NULL

Function XCLE_ExecCtxClear

XCLE_ExecCtx XCLE_ExecCtxClear(XCLE_ExecCtx ctx) ;
Clears all pending exceptions in this context.

Args:
_ **ctx** : the execution context

Returns:
_ **NULL** : if the execution context was NULL
else the original execution context

Errors:
_ **EINVAL** : if the execution context was NULL

Function XCLE_ExecCtxMapExceptions

unsigned long XCLE_ExecCtxMapExceptions(XCLE_ExecCtx ctx, , void * dat) ;
Maps all pending exceptions in an execution context (e.g. for displaying).

Args:
_ **ctx** : the execution context
_ **map** : the mapping handler
_ **dat** : user-data pointer

Returns:
_ **NULL** : if the execution context or the map handler were NULL
else the sum of all values returned by 'map'

Errors:
_ **EINVAL** : if the execution context or the map handler were NULL

Note:

The map handler takes, in order, the informations pertaining to the exception's calling code, and the exception nature:

- _ **lev** : the depth of the currently mapped exception (0 for the root exception)
- _ **src** : the name of the calling code
- _ **idt** : the exception's identification number
- _ **msg** : the exception's description
- _ **dat** : user-defined data pointer (the last argument of XCLE_ExecCtxMapExceptions)

Typedef XCLE_CodeOperator

typedef XCLE_Exception (* XCLE_CodeOperator) (XCLE_ExecCtx, XCLE_Stack, XCLE_Hash, XCLE_Object) ;
The type of the handler segment of Code objects for them to be compatible with this module.

:

Object evaluation

Function XCLE_ObjectEval

XCLE_Exception XCLE_ObjectEval(XCLE_ExecCtx ctx, XCLE_Stack stk, XCLE_Hash hsh, XCLE_Object obj) ;
Evaluates an XCLE_Object.

Args:

- _ **ctx** : execution context
- _ **stk** : Stack environment
- _ **dom** : Variables environment
- _ **obj** : Object to execute

Returns:

- _ **NULL** : if a NULL Stack, Hash or Object was passed as argument
- _ **XCLE_EXCEPTION_OK** : if the execution terminated normally

else the last exception thrown in this context.

Errors:

- _ **EINVAL** : if a NULL Stack, Hash or Object was passed as argument

Function XCLE_ListEval

XCLE_Exception XCLE_ListEval(XCLE_ExecCtx ctx, XCLE_Stack stk, XCLE_Hash hsh, XCLE_List lst) ;
Evaluates an XCLE_List.

Args:

- _ **ctx** : execution context
- _ **stk** : Stack environment
- _ **dom** : Variables environment
- _ **lst** : list to execute

Returns:

_ **NULL** : if a NULL Stack, Hash or Object was passed as argument
_ **XCLE_EXCEPTION_OK**: if the execution terminated normally
else the last exception thrown in this context.

Errors:

_ **EINVAL** : if a NULL Stack, Hash or Object was passed as argument

Function XCLE_CodeEval

XCLE_Exception XCLE_CodeEval(XCLE_ExecCtx ctx, XCLE_Stack stk, XCLE_Hash hsh, XCLE_Code cod) ;
Evaluates an XCLE_Code object.

Args:

_ **ctx** : execution context
_ **stk** : Stack environment
_ **dom** : Variables environment
_ **cod** : code to execute

Returns:

_ **NULL** : if a NULL Stack, Hash or Object was passed as argument
_ **XCLE_EXCEPTION_OK**: if the execution terminated normally
else the last exception thrown in this context.

Errors:

_ **EINVAL** : if a NULL Stack, Hash or Object was passed as argument

---

File IO module

Instructions manual

Author: Yann LANDRIN-SCHWEITZER
Contact: varkhan@varkhan.net
Homepage: http://varkhan.varkhan.net/

Objects File-descriptor Printing

Function XCLE_ObjectPrint

unsigned long XCLE_ObjectPrint(XCLE_Object obj, int fd) ;
Prints an XCLE_Object into a file descriptor.

Args:
- _ **obj** : XCLE_Object to print
- _ **fd** : output file descriptor output file descriptor

Returns:
- _ **-1** : if some error occurred
  else the total number of characters printed

Errors:
- _ **EINVAL** : if a NULL XCLE_Object was passed as argument
  or any error generated by the 'write' system call on file descriptor 'fd'

Function XCLE_VoidPrint

unsigned long XCLE_VoidPrint(XCLE_Void vd, int fd) ;

Prints an XCLE_Void into a file descriptor.

Args:
- _ **vd** : XCLE_Void to print
- _ **fd** : output file descriptor output file descriptor

Returns:
- _ **NULL** : if no memory was available
else the total number of characters printed

Errors:
- _ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'write' system call on file descriptor 'fd'

Function XCLE_IntgPrint

unsigned long XCLE_IntgPrint(XCLE_Intg in, int fd) ;
Prints an XCLE_Intg into a file descriptor.

Args:
- _ **in** : XCLE_Intg to print
- _ **fd** : output file descriptor output file descriptor

Returns:
- _ **NULL** : if no memory was available
else the total number of characters printed

Errors:
- _ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'write' system call on file descriptor 'fd'

Function XCLE_FltpPrint

unsigned long XCLE_FltpPrint(XCLE_Fltp fp, int fd) ;
Prints an XCLE_Fltp into a file descriptor.

Args:
- _ **fp** : XCLE_Fltp to print
- _ **fd** : output file descriptor

Returns:
- _ **NULL** : if no memory was available

Errors:
- _ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'write' system call on file descriptor 'fd'

Function XCLE_StrgPrint

unsigned long XCLE_StrgPrint(XCLE_Strg str, int fd) ;
Prints an XCLE_Strg into a file descriptor.

Args:
- _ **str** : XCLE_Strg to print

_ **fd** : output file descriptor

Returns:
_ **NULL** : if no memory was available

Errors:
_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'write' system call on file descriptor 'fd'

Function XCLE_ListPrint

unsigned long XCLE_ListPrint(XCLE_List lst, int fd) ;
Prints an XCLE_List into a file descriptor.

Args:
_ **lst** : XCLE_List to print
_ **fd** : output file descriptor

Returns:
_ **NULL** : if no memory was available

Errors:
_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'write' system call on file descriptor 'fd'

Function XCLE_CodePrint

unsigned long XCLE_CodePrint(XCLE_Code cod, int fd) ;
Prints an XCLE_Code into a file descriptor.

Args:
_ **cod** : XCLE_Code to print
_ **fd** : output file descriptor

Returns:
_ **NULL** : if no memory was available

Errors:
_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'write' system call on file descriptor 'fd'

Objects Stream Printing

Function XCLE_ObjectFPrint

unsigned long XCLE_ObjectFPrint(XCLE_Object obj, FILE * os) ;
Prints an XCLE_Object into an output stream.

Args:
_ **obj** : XCLE_Object to print
_ **os** : output stream output file descriptor

Returns:

_ **-1** : if some error occurred
else the total number of characters printed

Errors:

_ **EINVAL** : if a NULL XCLE_Object was passed as argument
or any error generated by the 'fwrite' system call on stream 'os'

Function XCLE_VoidFPrint

unsigned long XCLE_VoidFPrint(XCLE_Void vd, FILE * os) ;
Prints an XCLE_Void into an output stream.

Args:

_ **vd** : XCLE_Void to print
_ **os** : output stream output file descriptor

Returns:

_ **NULL** : if no memory was available
else the total number of characters printed

Errors:

_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'fwrite' system call on stream 'os'

Function XCLE_IntgFPrint

unsigned long XCLE_IntgFPrint(XCLE_Intg in, FILE * os) ;
Prints an XCLE_Intg into an output stream.

Args:

_ **in** : XCLE_Intg to print
_ **os** : output stream output file descriptor

Returns:

_ **NULL** : if no memory was available
else the total number of characters printed

Errors:

_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'fwrite' system call on stream 'os'

Function XCLE_FltpFPrint

unsigned long XCLE_FltpFPrint(XCLE_Fltp fp, FILE * os) ;
Prints an XCLE_Fltp into an output stream.

Args:

_ **fp** : XCLE_Fltp to print
_ **os** : output stream

Returns:

_ **NULL** : if no memory was available

Errors:

_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'fwrite' system call on stream 'os'

Function XCLE_StrgFPrint

unsigned long XCLE_StrgFPrint(XCLE_Strg str, FILE * os) ;
Prints an XCLE_Strg into an output stream.

Args:

_ **str** : XCLE_Strg to print
_ **os** : output stream

Returns:

_ **NULL** : if no memory was available

Errors:

_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'fwrite' system call on stream 'os'

Function XCLE_ListFPrint

unsigned long XCLE_ListFPrint(XCLE_List lst, FILE * os) ;
Prints an XCLE_List into an output stream.

Args:

_ **lst** : XCLE_List to print
_ **os** : output stream

Returns:

_ **NULL** : if no memory was available

Errors:

_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'fwrite' system call on stream 'os'

Function XCLE_CodeFPrint

unsigned long XCLE_CodeFPrint(XCLE_Code cod, FILE * os) ;
Prints an XCLE_Code into an output stream.

Args:

_ **cod** : XCLE_Code to print
_ **os** : output stream

Returns:

_ **NULL** : if no memory was available

Errors:

_ **EINVAL** : if a NULL was passed as argument
or any error generated by the 'fwrite' system call on stream 'os'

Stack and Hash Printing

Function XCLE_StackPrintF

unsigned long XCLE_StackPrintF(XCLE_Stack stk, int fd, char * head, char * rowfmt, char * tail, unsigned long rows) ;
Prints the contents of an XCLE_Stack in a file descriptor, following the given formats.

Args:
- _ **stk** : XCLE_Stack
- _ **fd** : file descriptor
- _ **head** : head string
- _ **rowfmt** : row printing format
- _ **tail** : tail string
- _ **rows** : number of rows (height in chars)

Returns:
- _ **-1** : if some error occurred

else the number of characters printed.

Errors:
- _ **EINVAL** : if a NULL XCLE_Stack was passed as argument

Function XCLE_HashPrintF

unsigned long XCLE_HashPrintF(XCLE_Hash hsh, int fd, char * head, char * rowfmt, char * tail) ;
Prints the contents of an XCLE_Hash in a file descriptor, following the given formats.

Args:
- _ **hsh** : XCLE_Hash
- _ **fd** : file descriptor
- _ **head** : head string
- _ **rowfmt** : row printing format
- _ **tail** : tail string
- _ **rows** : number of rows (height in chars)

Returns:
- _ **-1** : if some error occurred

else the number of characters printed.

Errors:
- _ **EINVAL** : if a NULL XCLE_Hash was passed as argument

Function XCLE_StackFPrintF

unsigned long XCLE_StackFPrintF(XCLE_Stack stk, FILE * os, char * head, char * rowfmt, char * tail, unsigned long rows) ;
Prints the contents of an XCLE_Stack in an output stream, following the given formats.

Args:
- _ **stk** : XCLE_Stack

_ **os** : output stream
_ **head** : head string
_ **rowfmt** : row printing format
_ **tail** : tail string
_ **rows** : number of rows (height in chars)

Returns:
_ **-1** : if some error occurred
else the number of characters printed.

Errors:
_ **EINVAL** : if a NULL XCLE_Stack was passed as argument

Function XCLE_HashFPrintF

unsigned long XCLE_HashFPrintF(XCLE_Hash hsh, int fd, char * head, char * rowfmt, char * tail) ;
Prints the contents of an XCLE_Hash in an output stream, following the given formats.

Args:
_ **hsh** : XCLE_Hash
_ **os** : output stream
_ **head** : head string
_ **rowfmt** : row printing format
_ **tail** : tail string
_ **rows** : number of rows (height in chars)

Returns:
_ **-1** : if some error occurred
else the number of characters printed.

Errors:
_ **EINVAL** : if a NULL XCLE_Hash was passed as argument

File-descriptor Parsing

Function XCLE_ObjectScan

XCLE_Object XCLE_ObjectScan(XCLE_ParseCtx contx, int fd, unsigned long * ln) ;
Parses an object from an input stream.

Args:
_ **contx** : parsing context
_ **fd** : file descriptor to read from
_ **ln** : line number

Returns:
_ **NULL** : if an IO error or a syntax error occured
else one parsed object from the stream.

Errors:
_ **EBADF** : an invalid file descriptor war specified

_ **EIO** : a read error occured
_ **ENOMEM** : no memory was available
_ **EINVAL** : a syntax error occured

Note:
Do not rely on the file descriptor offset to be anywhere usable or in relation to the data read. The buffering scheme used does not allow this. This is a BUG, and will be corrected in the future by repositioning the stream offset on file streams. Pipe streams will remain bugged until FILE* reading is implemented.

Function XCLE_ListScan

XCLE_List XCLE_ListScan(XCLE_ParseCtx contx, int fd, unsigned long * ln) ;
Parses a list from an input flux.

Args:
_ **contx** : parsing context
_ **fd** : file descriptor to read from
_ **ln** : line number

Returns:
_ **NULL** : if an IO error or a syntax error occured
else the parsed List.

Errors:
_ **EBADF** : an invalid file descriptor war specified
_ **EIO** : a read error occured
_ **ENOMEM** : no memory was available
_ **EINVAL** : a syntax error occured

Note:
Do not rely on the file descriptor offset to be anywhere usable or in relation to the data read. The buffering scheme used does not allow this. This is a BUG, and will be corrected in the future by repositioning the stream offset on file streams. Pipe streams will remain bugged until FILE* reading is implemented.
