XCLE

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