

xl(standard) エージェント・リファレンス・マニュアル

森 洋久

joshua @ globalbase.org

2009-12-19 版

目次

第 1 章	はじめに	7
1.1	目的と概要	7
1.2	このマニュアルを読むために必要な知識	7
1.3	前提となるシステム要件	7
1.4	XL とは	7
1.5	構文	8
第 2 章	型	10
2.1	概要	10
2.2	型と値の標記	10
2.3	基本型	10
2.4	文字コードについて	11
2.5	構造型 (LISP の S 式に対応)	11
2.6	環境	13
第 3 章	インタプリタ実行モデル	14
3.1	概要	14
3.2	評価	14
3.3	関数	14
3.4	インタプリタのモデル	16
3.5	Remote と Local	17
3.6	エージェント	17
3.7	XL サーバ	17
第 4 章	その他	19
4.1	概要	19
4.2	プラグマ	19
4.3	組み込み関数インタフェース	19
第 5 章	XL データ型リファレンス	21
5.1	概要	21
5.2	XML 要素	23
5.2.1	XLT_NULL(ヌル型)	23
5.2.2	XLT_ERROR(エラー型)	24
5.2.3	XLT_PAIR(ペア型、リスト型)	25
5.2.4	XLT_SYMBOL(シンボル型)	26
5.2.5	XLT_STRING(文字列型)	27
5.2.6	XLT_INTEGER(整数型)	28
5.2.7	XLT_FLOAT(浮動小数点型)	29
5.2.8	XLT_FUNC(関数型)	30

5.2.9	XLT_DELAY(ディレイ型)	31
5.2.10	XLT_PTR(構造体ポインタ型)	32
5.2.11	XLT_RAW(RAW データ型)	33
5.2.12	XLT_ENV(環境型)	34
第 6 章	Listbase Environment 型リファレンス	35
6.1	概要	35
6.2	XML 要素	36
6.2.1	xlisp-env	36
6.2.2	type	37
6.2.3	parent	38
6.2.4	work	39
6.2.5	flags	40
6.2.6	hash	41
6.2.7	element	42
第 7 章	XL エラーリファレンス	43
7.1	概要	43
7.2	エラー	44
7.2.1	カテゴリ SYSTEM	44
7.2.2	カテゴリ SYNTAX	45
7.2.3	カテゴリ SEMANTIX	46
7.2.4	カテゴリ PROTOCOL	47
7.2.5	動詞分類 READ	48
7.2.6	動詞分類 EXEC	49
7.2.7	動詞分類 MANY	50
7.2.8	動詞分類 NOTHING	51
7.2.9	動詞分類 INVALID	52
7.2.10	動詞分類 UNSUPPORT	53
7.2.11	動詞分類 UNDEFINED	54
7.2.12	動詞分類 MISSMATCH	55
7.2.13	動詞分類 REQUIRED	56
7.2.14	動詞分類 OPEN	57
7.2.15	動詞分類 DUPLICATE	58
7.2.16	動詞分類 CLOSE	59
7.2.17	動詞分類 DENIED	60
7.2.18	動詞分類 UNSPECIFIED	61
7.2.19	動詞分類 ALREADY	62
7.2.20	動詞分類 UNSUTISFIED	63
7.2.21	動詞分類 RELAIED	64
7.2.22	動詞分類 ERROR UTILITY	65
7.2.23	動詞分類 CORRESPONDING	66
7.2.24	動詞分類 EXIT	67
7.2.25	動詞分類 MATHEMATICAL	68
7.2.26	動詞分類 LOOP	69
7.2.27	動詞分類 OVER	70

7.2.28	XLE_OK	71
7.2.29	XLE_SYSTEM_READ_FILE	72
7.2.30	XLE_SYSTEM_NETWORK	73
7.2.31	XLE_SYSTEM_NETWORK	74
7.2.32	XLE_SYSTEM_EXIT	75
7.2.33	XLE_SYSTEM_CANCEL	76
7.2.34	XLE_SYSTEM_INTERRUPT	77
7.2.35	XLE_SYSTEM_INTERNAL	78
7.2.36	XLE_SYSTEM_LOOP_BREAK	79
7.2.37	XLE_SYSTEM_LOOP_CONTINUE	80
7.2.38	XLE_SYSTEM_APPLICATION	81
7.2.39	XLE_SYNTAX_TOO_MANY_PUNC	82
7.2.40	XLE_SYNTAX_NO_PUNC	83
7.2.41	XLE_SYNTAX_ERROR_CODE	84
7.2.42	XLE_SYNTAX_UNSUTISFIED_FILE_END	85
7.2.43	XLE_SYNTAX_TOKEN_ERROR	86
7.2.44	XLE_SYNTAX_NOT_CORRESPOND_TAG	87
7.2.45	XLE_SYNTAX_UNTERMINATED_TEXT	88
7.2.46	XLE_SYNTAX_INVALID_ENTITY	89
7.2.47	XLE_SEMANTICS_EXEC_NO_FUNC	90
7.2.48	XLE_SEMANTICS_INV_PARAM_LENGTH	91
7.2.49	XLE_SEMANTICS_INV_FORMAT	92
7.2.50	XLE_SEMANTICS_UNUPPORT_FUNC	93
7.2.51	XLE_SEMANTICS_UNDEF_SYMBOL	94
7.2.52	XLE_SEMANTICS_TYPE_MISMATCH	95
7.2.53	XLE_SEMANTICS_REQ_SYMBOL	96
7.2.54	XLE_SEMANTICS_DIV_ZERO	97
7.2.55	XLE_SEMANTICS_UNIT_MISMATCH	98
7.2.56	XLE_PROTO_INV_OBJECT	99
7.2.57	XLE_PROTO_INV_RESOURCE	100
7.2.58	XLE_PROTO_INV_PARAM	101
7.2.59	XLE_PROTO_INV_FIELD_NAME	102
7.2.60	XLE_PROTO_INV_FILE_TYPE	103
7.2.61	XLE_PROTO_OPEN_FILE	104
7.2.62	XLE_PROTO_UNUPPORT_FILE_TYPE	105
7.2.63	XLE_PROTO_UNUPPORT_MODE	106
7.2.64	XLE_PROTO_UNUPPORT_PROTO	107
7.2.65	XLE_PROTO_UNUPPORT_TYPE	108
7.2.66	XLE_PROTO_DUP_DEFINITION	109
7.2.67	XLE_PROTO_UNDEF_NAME	110
7.2.68	XLE_PROTO_UNDEF_DATABASE	111
7.2.69	XLE_PROTO_UNDEF_RESOURCE	112
7.2.70	XLE_PROTO_REQ_PRESET	113
7.2.71	XLE_PROTO_PERMISSION_DENIED	114
7.2.72	XLE_PROTO_CLOSED	115
7.2.73	XLE_PROTO_ACCESS_STREAM	116

7.2.74	XLE_PROTO_UNSPC_RESOURCE	117
7.2.75	XLE_PROTO_ALREADY_OPEN	118
7.2.76	XLE_PROTO_UNSUTISFIED_INFO	119
7.2.77	XLE_PROTO_REQ_PUT	120
7.2.78	XLE_PROTO_EXEC_ERROR	121
7.2.79	XLE_PROTO_RELAY	122
7.2.80	XLE_PROTO_LAUNCH	123
7.2.81	XLE_PROTO_USER_ERROR	124
7.2.82	XLE_PROTO_UNKNOWN	125
7.2.83	XLE_PROTO_NO_ROUTE	126
7.2.84	XLE_PROTO_OVER_TTL	127
7.2.85	XLE_PROTO_INV_IID	128
7.2.86	XLE_PROTO_BUSY	129
第 8 章	XL 基本関数	130
8.1	概要	130
8.2	関数	131
8.2.1	ErrHandler	131
8.2.2	Env0	133
8.2.3	Env1	134
8.2.4	OpenInterpreter	135
8.2.5	Eval	137
8.2.6	Lambda	138
8.2.7	Define	140
8.2.8	If	141
8.2.9	Sequence	142
8.2.10	NewEnvironment	143
8.2.11	GetParentEnvironment	144
8.2.12	SetParentEnvironment	145
8.2.13	CurrentEnvironment	146
8.2.14	quote	147
8.2.15	GetElement	148
8.2.16	Next	149
8.2.17	cdr	150
8.2.18	car	151
8.2.19	CutOut	152
8.2.20	Append	153
8.2.21	List	154
8.2.22	mul,*	155
8.2.23	add,+	156
8.2.24	sub,-	157
8.2.25	div,/	158
8.2.26	rem,%	159
8.2.27	and	160
8.2.28	or	161
8.2.29	xor	162

8.2.30	not	163
8.2.31	And	164
8.2.32	Or	165
8.2.33	Xor	166
8.2.34	Not	167
8.2.35	Equ	168
8.2.36	Neq	169
8.2.37	Lt	170
8.2.38	Gt	171
8.2.39	Lteq	172
8.2.40	Gteq	173
8.2.41	Remote	174
8.2.42	Local	175
8.2.43	Result	176
8.2.44	Load	178
8.2.45	Save	180
8.2.46	Parse	182
8.2.47	Print	183
8.2.48	CutString	184
8.2.49	StringLength	185
8.2.50	SetChar	186
8.2.51	GetChar	187
8.2.52	Let	188
8.2.53	SetListbaseEnv	190
8.2.54	GetListbaseEnv	191
8.2.55	?xml,?xl	192
8.2.56	Arg	193
8.2.57	For	194
8.2.58	Break	196
8.2.59	Continue	197
8.2.60	Cast	198
8.2.61	DefineEval	202
8.2.62	Categorize	203
8.2.63	Chdir	204
第 9 章	XL 認証関数	205
9.1	概要	205
9.2	関数	207
9.2.1	authentication	207
9.2.2	service	208
9.2.3	target	210
9.2.4	auth-transaction	212

第 10 章 XL ストリーム・ルーティング関数	213
10.1 概要	213
10.2 関数	214
10.2.1 StreamRouting	214
10.2.2 ActivateStreamRouting	217

第1章 はじめに

1.1 目的と概要

xl エージェントは GLOBALBASE のエージェントの中で最も基本的なエージェントである。基本的な算術演算等が含まれている。これだけで簡単な言語系をなしており、簡単なスクリプトプログラムを作ることが出来る。本マニュアルはこの言語系の原理を説明し、また、言語体系のリファレンスを提供する。

なお、このマニュアルは 2007-05-07 現在まだ移行中なので、古いマニュアルのすべての情報を含んでいません。必要に応じて古いマニュアルを参照してください。

1.2 このマニュアルを読むために必要な知識

xl 言語を簡単に実行するためには、linux などの posix 系オペレーティングシステムで xl コマンド実行するのが早い。従って、posix 系のマシンについて基本的な操作が出来ることが望まれる。

1.3 前提となるシステム要件

xl エージェントが動作可能なマシンがあること。具体的には、posix 系オペレーティングシステムに LANDSCAPE GLOBALBASE SERVER [1] がインストールされているマシンがあれば良い。

1.4 XL とは

インターネットのブラウザなどでは、HTML、XML といったマークアップ言語をデータ表現の方法として使い、それを転送するプロトコルとして HTTP が一般的である。しかし、XML が発達するにつれて、データベースのクエリを XML 形式で記述する方法が提案されるなど、XML をデータ表現ではなく、プロトコルとしての性格の強い使われ方が成されるようになってきた。

XML がプロトコルとしての機能を充実すると、HTTP の役割は XML で置き換えることが出来る。たとえば、HTTP の GET 文は、... という XML の構文で置き換えることが出来る。このように全てが XML で置き換えられるようになると、パーザが二つから一つで十分になるなど、リソースの節約が期待される。しかし、単に構文を置き換えるだけでは、実行系は依然として二つ必要であるため、そのメリットはあまりない。

たとえば、XML でデータベースに検索クエリを出し、データを引き出したとする。HTTP を使用した場合、GET 文の中に XML で定義された検索クエリがネストした形でデータベースに送り込まれる。しかし、検索クエリのセマンティックスはデータを GET するという意味はすでに含まれており、冗長性は避けられない。

この問題の原因は、XML の構文が統一されたにも関わらず、依然としてプロトコルとその上で処理されるデータという二つの概念が存在しているからである。そこで、データとプロトコルという二つの概念が交換可能な言語 XL (XML LISP) を提案する。つまり、

```
<greeting>Hello World</greeting>
```

という XML 表現があったとする。この `greeting` というタグは、Hello World に付属した属性データであると解釈することもできるが、サーバがブラウザたいして「飾り立てて表示しろ」という指令するプロトコルとも取ることが出来、これらの解釈の違いを時と場合によって明示的にいろいろ変えることの出来る仕組みを提案する。

プロトコルをプログラムと置き換えれば、プログラムとデータが交換可能で、プログラムをデータとして処理したり、データをプログラムとして実行したりすることの出来る言語として、LISP, ML などの言語が存在する。XL では、XML の構文に LISP のセマンティックスを当てはめることによって XML 構文の解釈をいろいろプログラムする仕組みを提供する。具体的には、XML のタグを LISP の関数名、要素にネストした要素を LISP の関数の引数と解釈し、この関数の本体を定義する方法を導入する。たとえば、上述の例では、

```
(greeting "Hello World")
```

という LISP の S 式と同一であると解釈する。`greeting` という関数を定義すれば、この構文は実行可能となる。

ところが、XML の要素を LISP の関数と解釈したときに問題となるのは、関数の引数の評価方法の考え方が LISP でははっきりとしているが、XML では曖昧であるということである。この曖昧さには二種類ある。

まず最初の曖昧さは、評価順序の曖昧さである。XML 文書の要素の中の子要素は、アプリケーションの種類や、評価の状況などによって、必要な物だけが評価されるという Normal Order の評価が行われる一方で、全ての物をきちっと評価される Applicative Order の評価が行われる場合もある。

次の曖昧さは、評価方法そのものの曖昧さである。要素の評価の仕方は、どのような要素のなかにネストされるかによって評価のされ方が違うという曖昧さがある。

XML 文書を処理するためには、まず関数定義の時に、引数の評価順序を Normal Order か、Applicative Order か明示的に選択できるようにすることによって、最初の曖昧さに対応する。次に、関数定義の時に、引数を評価するための専用の環境を明示的に指定することによって、要素がネストした場合の評価の方法を切り替えることが出来るようにする。

1.5 構文

XL の構文は、XML 形式の構文と LISP 形式の構文 2 種類がある。本来、XML 形式のみで十分であるが、それでは煩雑になるので、LISP 形式も採用した。ただし、LISP 形式は XML 形式の要素の中に入れ子にすることが出来るが、その逆は出来ない。関数の仕様書などでは二つの形式を併記している。XL パーザには、XML モードと LISP モードの二種類がある。XML モードの時には、タグを認識するための構文解析が行われ、また、要素の中の `'(,;)'`、`'^'`、および数字を認識すると、LISP モードになる。LISP モードでは、`'<'` を認識すると、XML モードになる。二つのモードはネストしており、モードの変化点はパーザによって記憶されており、ネストが終わると自動的に基のモードへもどる。XML モードのときの要素は、空白や改行で区切られる。空白や改行は無視する化しないかは `?xml:option` で指定することが出来る。もし、空白を無視しないことにした場合、空白のところの区切り目に空白だけの文字列が挿入される。

```
<tag>
```

```

abc def
hhh
^sym
ghi
(+ a b)
jk 123
lm
(print stdout !
  <name data="eee"> hello </name>)
nop
</tag>

```

空白、改行を無視するモードでは、

```

(tag
  "abc" "def"
  "hhh"
  sym
  "ghi"
  (+ a b)
  "jk" 123
  "lm"
  (print stdout
    ([name data="eee"] "hello"))
  "nop"
</tag>

```

と等価である。空白、改行を無視しないモードでは、

```

(tag
  "abc" " " "def" "\n\t\t"
  "hhh"
  sym
  "ghi"
  (+ a b)
  "jk" 123
  "lm"
  (print stdout
    ([name data="eee"] "hello"))
  "nop"
</tag>

```

となる。タグのあと、`(` や `^` など、LISP モードへ変化する前後では空白や改行は無視されることに注意されたい。

第2章 型

2.1 概要

XL では、型は値に与えられている。ここでは、XL における型と、その実態、また型や変数の表記について述べる。

2.2 型と値の表記

XL で使用する値の種類は以下の通りである。

```
exp ::=
    fundamantal-exp      (基本型)
    structured-exp       (構造型)
    function-exp         (関数型)
    environment-exp      (環境型)
```

2.3 基本型

XL で定義される基本型は以下の通りである。

```
fundamental-exp ::=
    integer
        表記：XML 方式, LISP 方式とも同じ
    floating-point (浮動小数点)
        表記：XML 方式, LISP 方式とも同じ
    string
        表記：
            XML    解析文字列
            LISP   "... "
    symbol
        表記：
            XML    ^... (最後に空白が入る)
            LISP   以下参照。
    raw
        表記：
            XML 形式 LISP 形式とも同じ
            #byte 数#...RAW データ...
```

例：#5#!x\$%~

error

表記：

```
%E(  
  エラーシーケン番号 (行番号)  
  エラー発生 site の URL  
  エラー発生ファイル名  
  エラー発生関数名  
  エラーコード  
  exp (その他付加データ)  
)
```

environment

表記：なし

error 型は、エラーを処理するための特別な値で、エラーが発生した関数が戻り値として返す。一方、この値を受け取った関数は即座に同じエラーを返さなければならない。ただし、ErrHandler 関数は例外である。この関数はエラーを補足してエラーの後処理をするための関数である。

通常、インタプリタは、エラーが発生すると、それを Result 関数の引数として表示する。しかし、error 型のうち ErrDontPrint エラーは特殊なエラーで、インタプリタはこのエラーを受けるとなにも表示しない。このようにすることによって、ネットワークでエラーやデータがループする事態を回避している。

2.4 文字コードについて

XL の処理系内部ではあらゆる文字を 32bit 固定長で表現する。この文字コードを内部文字コードと呼ぶ。内部文字コードの割り当ては表 2.1 の通りである。

表 2.1: ゾーン

ゾーン	ゾーンの意味	割り当てられたコード
0000 ~ 7FFF FF FF	4 バイトコード	UCS
80000 ~ 800FF FF	2 バイトコード	JIS 第一水準、第二水準
80000 ~ BF FF FF FF	2 バイトコード	未使用
C0 000 ~ DF FF FF FF	3 バイトコード	未使用
E0 000 ~ FF FF FF FF	1 バイトコード	未使用

UCS および JIS の領域以外は、UCS や JIS と異なるコンセプトのコードを割り当てることが出来る。現在は EUC、SJIS は全てパーザによって JIS に変換される。

2.5 構造型 (LISP の S 式に対応)

LISP 形式

```
structured-type ::=  
(
```

```
(exp exp ....)
([symbol 属性リスト])
([symbol 属性リスト] exp ....)
```

XML 形式

```
([symbol 属性リスト] ....)
```

は、

```
<symbol 同じ文字列 属性リスト> </symbol 同じ文字列>
```

と等価である。たとえば、

```
([greeting
  FirstName = "Hirohisa"
  LastName = "Mori"]
 "Hello World")
```

は、

```
<greeting
  FirstName="Hirohisa"
  LastName = "Mori">
  Hello World
</greeting>
```

と変換できる。この場合、Hello World は一続きの文字列として認識されるが、これを別々に分けたい場合は、間に `verb+|+|` を入れる。つまり、

```
<greeting
  FirstName="Hirohisa"
  LastName = "Mori">
  Hello \ World
</greeting>
```

は、

```
(greeting [
```

```
FirstName = "Hirohisa"  
LastName = "Mori"]  
"Hello" "World")
```

と等価である。

属性値にシンボルを定義したい場合は、

```
(greeting [FirstName = "^symbol" ]  
"Hello" "World")
```

```
<greeting FirstName = "^symbol"> Hello World </greeting>
```

というように、属性値の頭に ^ を付ければよい。^ を二つ続け、" ^^symbol " とすると、それはシンボルではなく、" ^symbol " という文字列と解釈される。

2.6 環境

変数と値の対応表。変数の名前は symbol で示される。つまり、symbol と値の対応表が環境である。

通常一つの環境には、一つの親環境がある。以下に述べる、評価においては、親環境の情報も検索対象となる。

しかし、関数の引数の評価時には、関数の引数評価用の環境を先に評価し、その後、関数のカレントの環境を評価する必要があるため、親環境を二つもった分岐環境が生成される。ただし、この環境にはシンボルや値をバインドすることは出来ない。

評価時の分岐環境の検索順序は、まず、第一番目の親環境、次に第二番目の親環境となる。したがって、親環境に定義されているシンボルが優先される。

第3章 インタプリタ実行モデル

3.1 概要

XL の関数の評価方法について述べる。

3.2 評価

プリミティブ関数の一つである評価関数 Eval を、次のように定義する。

(Eval environment-exp exp)

評価対象 (exp) としては、あらゆる型を受け付ける。そして以下に示す通り、評価した結果を返す。このとき、Eval に与えられた環境 (environment-exp) を評価対象のカレント環境と呼ぶ。

1. 評価対象が、symbol 以外の基本型の値の場合は、その値をそのまま返す。
2. 評価対象が、symbol であった場合は、与えられた環境および、親環境、そのまた親環境と検索し、symbol に相当する変数が初めて見つかった時点で、対応した値を返す。検索に失敗したらエラーを返す。エラーはシステム構造型として後述する。
3. 評価対象が、関数であった場合は、その値をそのまま返す。
4. 評価対象が構造型であった場合、その最初のシンボル、あるいはタグに相当するシンボルを評価し、
 - (a) その評価結果が関数でも環境でもでなかった場合は、エラーを返す。
 - (b) その評価結果が環境の場合、その環境を親環境とした環境を新しく生成し、これへカレント環境を変更する。この環境には、Local というシンボルにまへのカレント環境をバインドする。
 - (c) その評価結果が関数であった場合は、関数が normal-order か、 applicative-order かを判定、
 - (d) normal-order であれば、構造型の値、あるいは、要素をそのまま関数へ渡し、関数そのものを実行し、結果を返す。
 - (e) applicative-order であれば、構造型の値の二項目以降あるいは、要素の中身をすべて評価し、これらを引数として関数に渡し、実行結果を返す。

3.3 関数

引数を与えると、それに対し所定に計算をし値を返す。関数を呼び出すときには、

(func a b c)

または、

```
<func> a b c </func>
```

あるいは、属性リストをつかい

```
([func type="def"] a b c)
```

```
<func type="def"> a b c </func>
```

というように、構造型を使う。この構造型全体を関数 func への引数と呼び、func, a, b, c をそれぞれ、関数の 1 番目 2 番目..... の引数要素と呼ぶ。また、呼び出された側から、属性を参照する場合は、属性名に相当するシンボルを参照すればよい。

関数の引数の評価の仕方は、Normal と、Applicative との 2 種類がある。Normal では、引数は評価されずに、そのまま渡される。環境は渡される時に生成され、新しいカレント環境となる。引数を評価するかしないかは、関数を実行する中で明示的に決めたいときにこのオプションを選ぶ。

一方、Applicative では、引数は引数要素ごとに評価されてから渡される。引数評価のために、分岐環境が生成され、その一番目の親環境に関数定義時に定義した専用の環境がセットされ、第二番目の親環境に現在のカレント環境が設定される。評価する場合の環境の検索順序は関数定義時の環境、現在のカレント環境という順番になる。この仕組みは、

```
<SetAgent>
  "nichibun-agent"
  <User> hirohisa </User>
  <Passwd> abc </Passwd>
</SetAgent>
```

という呼び出しを考えたとき、User,Passwd を、SetAgent 外部で使用したときと、別の処理をしたいときに有効である。評価用の環境でこの特殊な処理の関数を定義すればよい。つまり、以下のように Sequence を使って新しい環境を定義してしようする。

```
<Sequece>
  <Define> ^Env <CurrentEnvironment/> </Define>
  <Define> ^User
    ()
    <Arguments> ^name </Arguments>
    ..... User の定義
  </Define>
  <Define> ^Passwd
    ()
    <Arguments> ^password </Arguments>
    ..... Passwd の定義
```

```

    </Define>
</Sequence>

<Define> ^SetAgent
  ^Env
  <Arguments> ^db ^user ^passwd </Arguments>
  .... SetAgent の定義
</Define>

....

<SetAgent>
  "nichibun-db"
  <User> hirohisa </User>
  <Passwd> abc </Passwd>
</SetAgent>

```

前段階が終わると、引数が評価され、関数本体へ実行が移る。おおかたの関数は、前段階なしの Applicative 関数であろう。また関数には、インタプリタにあらかじめ組み込まれたプリミティブ関数と、スクリプトによって定義できる、マクロ関数がある。マクロ関数の定義の仕方はプリミティブ関数 lambda および define で実現される。

3.4 インタプリタのモデル

一つのインタプリタが起動される時には、必ず一組の入力ストリームがある。たとえば、標準入出力から駆動されたり、またはファイルであったり、あるいはネットワークの外からコネクションがアクセプトされ、インタプリタが起動されるなどである。入力ストリームの先が、ファイルであったり、標準入出力であったりした場合は、このインタプリタをルート・インタプリタと呼ぶ。

インタプリタは、初期化時に、一つの環境を作り、この環境にいくつかのデフォルトの変数を定義し、入力ストリームから XL スクリプトが来るのを待つ。来たスクリプトを一つ一つの関数ごとに、先の環境と一緒に Eval 関数へ渡す。

駆動されたインタプリタは入力ストリームからルート要素が来るのを待つ。XL ストリームの場合にはいくつものルート要素があっても良い。つまり、いくつもの XML ファイルが送り込むことができる。

インタプリタは入力ストリームから送り込まれた要素を評価し、その結果を、もし対応する出力ストリームがあれば、そこへ返す。

インタプリタを生成する関数は、OpenInterpreter である。この関数によりインタプリタを生成すると、インタプリタ ID (IID) が返ってくる。IID は正の正数である。インタプリタを終了するときにはこの IID を使って CloseInterpreter を呼べばよい。

インタプリタの駆動方式には何種類かがある。ファイルや標準入出力を入力ストリームとして指定する方法、ネットワークのアクセプトポートを指定する方法。また、どこかの XL サーバのアクセプトポートを指定し、コネクションを貼る方法である。

ファイルや標準入出力を入力ストリームとして指定した場合はその入力ストリームがそのままインタプリタの入力ストリームとなる。

アクセプトポートを指定した場合は、そのアクセプトポートに接続要求が来るたびにソケットとインタプリタを生成し、ソケットの入力ストリームをインタプリタの入力ストリームとする。

コネクションを指定した場合は、指定するアクセプトポートに接続し、生成されたソケットの入力ストリームをインタプリタの入力ストリームとする。

このように、ネットワークのソケットの場合、ソケットは全二重なので、ソケットの両側にインタプリタが対称に生成されることになる。

3.5 Remote と Local

XL の特徴的なネットワーク操作関数に Remote と Local 関数がある。Remote は IID によって与えられたインタプリタの出力ストリームへ、与えられた命令群 (XML 要素群) を送り込み、その結果が入力ストリームより返ってくるのを待つ。ソケットの出力ストリームは反対側が入力ストリームとなっており、インタプリタがあるので、このインタプリタが送り込まれた XML 要素群を解釈し、結果を返す。返ってきた結果は、そのまま Remote の戻り値となる。

Remote をネストすると、次々に新しいインタプリタへソケットを通じてコネクトすることが出来る。複数のインタプリタをネットワークでカスケードし、情報をやりとりする並列実行が可能となる。

Local は、Remote の XML 要素を評価中のインタプリタ (リモートインタプリタ) から、ソケットの逆向きのストリームへ要素を送り込む仕組みである。この Local からの要素を評価するのは Remote を発行したインタプリタ (ローカルインタプリタ) ではないが、評価の時の環境を。ローカルインタプリタとそろえることにより、ローカルインタプリタのデータにアクセスすることが出来る。Remote が何段にもカスケードしている場合は、Local も同様にカスケードする事が出来る。

Local により送り込まれた要素は、Remote を呼び出したインタプリタの環境で評価をしなければならないので、まず、Remote 関数はあらかじめ、出力ストリームへ送り出す要素の先頭の行番号と、現在の環境をセットにしてデータベースに記憶する。一方、Local は直接、評価対象の要素を送り込むことはせず、もとの環境を検索し評価する機能をもった LocalEval 関数に評価対象の要素をネストして出力ストリームへ送り込む。

Local が二重にネストすると、二度目の Local は最後に Remote を発行したインタプリタの出力インタプリタへ評価要素を送り込まなければならない。そこで、上述の環境と行番号のセットに、さらに Remote を呼び出したインタプリタへのポインタを記憶しておく。この構造により次に出力すべき出力ストリームが認識できる。

3.6 エージェント

エージェントは環境を共有する一つまたは複数のインタプリタ群からなるプロセスのことを言う。これらインタプリタはこのプロセスのスレッドとして実現される。

エージェントはまず、プロセス駆動時に、その引数として渡されるファイルを入力ストリームとしたインタプリタを生成する。このファイルからはエージェントで実行可能な全ての関数が見える状態である。このファイルで OpenInterpreter を実行すると様々なインタプリタが生成され、実行可能になる。

3.7 XL サーバ

SetAgent とその他の Agent 検索機能、クライアントの認証を備えたエージェントのことである。

ユーザごと、クライアント側は駆動したいエージェントの名前と、接続モードを指定する。接続モードとは、エージェントの機能をどこまで使用可能かを定める物である。通常考えられる接続モードの決め方とし

ては、root モードと user モードの二種類を用意することである。接続モードはサーバの管理者が自由に決めることができる。二種類のみならず、Agent 要素を使って様々な種類を用意することができる。

また、ユーザのアカウントはエージェントひとつひとつに与えられるというよりは、いくつかのエージェントをまとめたグループを決め、このグループごとに定義する。これにより、一つの機関のエージェントは統一的に管理できるメリットがある。このグループも Agent 要素で定義できる。

認証は、クライアントの IP アドレスと、ユーザのパスワードによる認証の二種類がある

サーバはまず、accept タイプのインタプリタを駆動する。クライアントからのコネクション要求に対して、まず、クライアントの IP アドレスの認証を行う。

IP アドレスの認証は accept タイプの OpenInterpreter において Permission により指定される情報により行われる。

認証に失敗すると、クライアント側に Deny が送られ、接続はサーバから強制的に切断される。

IP が認証されると、対応する接続モードのリストが検索され、コネクションが維持され、インタプリタが割り当てられる。以降クライアントのユーザ認証が行われるが、このとき、ユーザは検索された接続モード以外では接続できない。この機能を利用し、たとえば、root モードではネットワークごとに接続することは出来ないなどの指定が出来る。

この時点で、クライアントからは SetAgent のみが実行可能になる。クライアントから SetAgent により新しいエージェントを駆動することができる。SetAgent は以下のように指定する。

```
<SetAgent>
  エージェント名
  接続モード
</SetAgent>
```

SetAgent 関数は、接続モードが IP の認証結果に違反していないか調べる。違反していれば、Permission Denied エラーを返す。

次に、エージェント名から、エージェントの情報を検索する。このエージェントを駆動するのにパスワードが必要と判断された場合、GetUserInfo 要素をクライアント側に送る。この関数の戻り値として、クライアントはユーザ名とパスワードを返し、サーバはこれによりユーザの認証を行う。認証に成功すれば、 が返され、エージェントが駆動される。認証に失敗すると、Permission Denied のエラーを返す。

GetUserInfo の引数は、グループ、エージェント、接続モードである。クライアント側は、グループと接続モードから必要なユーザ名とパスワードを検索できるような GetUserInfo 関数をあらかじめ用意しておく、結果を返す。クライアントが異なるサーバに接続する場合にはユーザ名、パスワードのテーブルを増やしていくことにより対応することができる。

もし、対応するグループ、接続モードがなかった場合は、エラーを返さず。すると、SetAgent は同じエラーを返す。

ここで、ユーザに必要な認証情報がないことを示し、ユーザ名パスワードを求めるダイアログを出すようなスクリプトを用意すればよい。

第4章 その他

4.1 概要

その他重要事項として、プラグマ、よおよび組み込み関数の呼び出しインタフェースについて述べる。

4.2 プラグマ

以下に挙げるタグは XL ではプラグマと呼ばれ、XL パーザに対する命令と解釈される。

```
<?xml version="1.0" .... ?>  
XML のバージョンと文字コード指定。
```

```
<?xl seq="数字" ?>  
文書の行数の固定
```

```
<?xl code="...." ?>  
文字コードの変更
```

```
<?xl appltag="ignore/remark" ?>  
文字コードの変更
```

そのほかの、`<? ... ?>` は、`appltag` の指定に従い、無視されるか、されないかが決まる。`appltag="remark"` のときは、`?` から始まるシンボルを一つもった、長さ 1 の構造型として翻訳される。

4.3 組み込み関数インタフェース

データベースのフロントエンドをするインタプリタには、データベース操作を行う固有の関数を組み込まなければならない。その組み込み関数は C で記述し、インタプリタのコードとリンクする。

まず以下のように、関数を定義する。

```
XLtype * pEmbeddedFunction(XLenv * e, XLtype * s) { s に関する処理。 return 結果 }
```

`s` は、`List (引数 0, 引数 1, 引数 2, ...)` という形を取っている。それぞれの引数は、以下に示す、`cLambda` におけるオプションが `Applicative` であれば、評価されてから渡される。

次に、インタプリタの初期の環境生成時に上記関数を環境に登録するコードを追加する。

```

XLenv * RootEnv;

main()
{

    Env0 = cNewEnvironment(0);
    ....
    cDefine(Env0,"EmbeddedFunction",
            cLambda( // 関数を得る。
                    "Applicative", // 関数の属性
                    0, // 引数評価のための環境
                    pEmbeddedFunction)); // 呼び出される関数名
    ....
}

```

以下に述べるプリミティブは、インタプリタのソースコード内では、

```

XLtype *
_プリミティブ名 (XLenv * e,XLtype * s)
{

}

```

という形で定義されている。新たなプリミティブを定義するときはこの関数を使うことができる。

また、このプリミティブをそのまま使うには、引数が *s* で示される構造型にすべてまとめられており、使いにくく、また、カレント環境を参照しない演算関数などでも、*e* が定義されている。従って、プリミティブを新たにコーディングするために使いやすく引数を整理した関数として、以下のような関数を定義している。

```

XLtype *
c プリミティブ名 (... )
{

}

```

上述 main 関数の例のなかで定義している `cDefine`,`cLambda` はこの関数である。引数の評価は `Eval` が行うので、`_プリミティブ名`、および、`c プリミティブ名`、のいずれの形式に置いても、引数の評価はされないことに注意しなければならない。

`XLtype` は、`XL` における基本型、構造型の内部形式であり、`XLenv` は環境の内部形式である。この実際の定義は別途インタプリタのソースコードのヘッダファイルにて明らかにする。

第5章 XL データ型リファレンス

5.1 概要

XL の 12 種類のデータ型について解説する。

データの記述形式はBNFによるが以下のトークンを前提としている。

```
XLT_SEXP_lisp ::= XLT_NULL
                | XLT_ERROR
                | XLT_PAIR
                | XLT_SYMBOL
                | XLT_STRING
                | XLT_INTEGER
                | XLT_FLOAT
                | XLT_FUNC
                | XLT_DELAY
                | XLT_PTR
                | XLT_RAW
                | XLT_ENV
                ;
```

また、

chars

は文字列。

digit_chars digit_chars_with_0

は、前者は頭に 0 のない数字、後者はあたまに 0 があってもかまわない数字の列をあらわす。

hex_digit_chars

は '0' ~ '9' の他に、'a' ~ 'f' および 'A' ~ 'F' を組み合わせた 16 進文字列である。

sp

は、スペース、タブ、改行といった空白文字の列である。

最後に XML 表現の中に、LISP 表現を埋め込む方法とその逆を述べる。前者は、'`' 文字または (を認識してパーザは XML 形式から LISP 形式に解釈を変更する。つまり、

```
LISP_in_XML ::= '`' XLT_SEXP_lisp
              | '^"' ('" XLT_SEXP_lisp )
              ;
```

となる。一方、XML への認識の変更は、'＜' の認識による。つまり、

```
XML_in_LISP ::= XLT_LIST_xml
```

である。

5.2 XML 要素

5.2.1 XLT_NULL(ヌル型)

プロトタイプ

LISP 形式

XLT_NULL_lisp ::= '(')'

XML 形式

なし。

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

LISP の NULL データ型に相当。値は一つしか持たない。

内部表現は、データ保存場所を持たない。

参考

バグ

5.2.2 XLT_ERROR(エラー型)

プロトタイプ

LISP 形式

```
XLT_ERROR_lisp ::= '% 'E' 'verb+|+|verb+|+|n'  
                '(' XLT_STRING_lisp [hostname] sp XLT_STRING_lisp [filename] sp XLT_INTEGER_lisp [line  
number] sp XLT_STRING_lisp [function] sp XLT_INTEGER [error code] 'verb+|+|verb+|+|n'  
                XLT_SEXP_lisp ')'
```

XML 形式

なし。

内部要素

sp はスペースやタブ。

XLT_SEXP_lisp は LISP 形式で記述された任意の型のデータを示す。

5.2.5 節

5.2.6 節

属性

所属エージェント

xl

要素パス表現

なし

説明

エラーコード。XML 形式は存在しない。たとえば以下の通り。

%E

```
("localhost" "src/xml/ja-xl.xml" 634 "parser" 0x80021306  
("not correspond tag ( to" "prototype"))</Result>
```

XLT_STRING_lisp (5.2.5 節) [hostname] XLT_STRING_lisp (5.2.5 節) [filename] XLT_INTEGER_lisp (5.2.6 節) [line number] は、それぞれエラーの発生したホストのホスト名、ファイルのファイル名、その行番号を表している。

XLT_STRING_lisp (5.2.5 節) [function] はエラーを報告した関数の関数名である。

XLT_INTEGER (5.2.6 節) [error code] はエラーコード。

XLT_SEXP_lisp 部分はエラーに関する補足説明の情報が入る。

内部表現は、上記情報をメンバにもつ構造体である。

参考

バグ

5.2.3 XLT_PAIR(ペア型、リスト型)

プロトタイプ

LISP 形式

```
XLT_PAIR_lisp ::= '(' SP XLT_SEXP_lisp [car] sp '.' sp XLT_SEXP_lisp [cdr] sp ')'
```

```
XLT_LIST_lisp ::= '(' sp xlt_list sp ')'
```

```
xlt_list ::= XLT_SEXP_lisp  
          | XLT_SEXP_lisp SP xlt_list;
```

XML 形式

先頭がシンボルであるリストは XML の要素として表現可能

cdr 要素が XLT_NULL (5.2.1 節) でないペアは表現出来ない。

先頭がシンボルでないリストは表現出来ない。

```
XLT_LIST_xml ::= TAG XLT_SEXP_xml END_TAG
```

```
TAG ::= '<' XLT_SYMBOL_lisp sp ATTRIBUTES '>'
```

```
END_TAG ::= '<' '/' XLT_SYMBOL_lisp '>'
```

内部要素

sp はスペースやタブ。

XLT_SEXP_lisp は LISP 形式で記述された任意の型のデータを示す。

XLT_SEXP_xml は XML 形式で記述された任意の型のデータを示す。

属性

所属エージェント

xl

要素パス表現

なし

説明

XLT_PAIR は任意の型のデータへの 2 つのポインタ *car* と *cdr* からなるポインタのペアである。しかし、XL では純粋なペアとして使うことはない。ペアを組み合わせたリストとして利用する。リストとは、内部データの的には、

```
xlt_list_ ::= ^"(' SP XLT_SEXP_lisp[car] SP '.' SP xlt_list_ ')'  
          | XLT_NULL
```

という最後が XLT_NULL (5.2.1 節) のペアの連続と同値と考えることが出来る。

参考

バグ

5.2.4 XLT_SYMBOL(シンボル型)

プロトタイプ

LISP 形式

XLT_SYMBOL ::= *chars*

XML 形式

特定のシンボルを一つ記述することは出来ない。シンボルが先頭にあるリストを要素として定義することが出来る。

5.2.3 節を参照のこと。

内部要素

chars 文字列。

属性

所属エージェント

xl

要素パス表現

なし

説明

LISP 形式において、文字列はシンボルを表す。内部表現は可変長の文字列である。

参考

バグ

5.2.5 XLT_STRING(文字列型)

プロトタイプ

LISP 形式

XLT_STRING_lisp ::= ''' chars '''

XML 形式

XLT_STRING_xml ::= *chars*

内部要素

chars 文字列。

属性

所属エージェント

xl

要素パス表現

なし

説明

LISP では " で囲まれた文字列、XML では単純な文字列を文字列型としてあつかう。内部表現は可変長の文字列である。

参考

バグ

5.2.6 XLT_INTEGER(整数型)

プロトタイプ

LISP 形式

```
XLT_INTEGER_lisp ::= digit_chars
```

```
    | '0' digit_chars
```

```
    | '0' 'x' hex_digit_chars
```

```
    | '0' 'X' hex_digit_chars
```

```
    ;
```

XML 形式

```
XLT_INTEGER_xml ::= XLT_INTEGER_lisp
```

内部要素

digit_chars 数字の列。

hex_digit_chars 16進数字の列。

digit_chars_with_0 先頭に0があってもよい数字の列。

属性

所属エージェント

xl

要素パス表現

なし

説明

整数を表す。内部表現は64bit長の符号付き整数である。先頭が0から始まる整数は8進数として解釈され、0xまたは0Xで始まるものは、16進数と解釈される。Cの整数と同じである。

参考

バグ

5.2.7 XLT_FLOAT(浮動小数点型)

プロトタイプ

LISP 形式

```
XLT_FLOAT_lisp ::= digit_chars_with_0 '.' digit_chars_with_0  
                | digit_chars_with_0 '.' digit_chars_with_0 'e' sign digit_char_with_0  
                | digit_chars_with_0 '.' digit_chars_with_0 'E' sign digit_char_with_0 ;
```

XML 形式

```
XLT_INTEGER_xml ::= XLT_INTEGER_lisp
```

内部要素

digit_chars_with_0 先頭に 0 があってもよい数字の列。

属性

所属エージェント

xl

要素パス表現

なし

説明

浮動小数点を表す。内部表現は 64bit の倍精度浮動小数点である。C の浮動小数点 `double` と同じである。

参考

バグ

5.2.8 XLT_FUNC(関数型)

プロトタイプ

表現形式を持たない。

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

内部表現は、C 言語関数へのポインタ、または、`Lambda>` (8.2.6 節) によってユーザ定義された関数を表す、`XLT_SEXP_lisp` 形式の内部表現へのポインタである。

参考

バグ

5.2.9 XLT_DELAY(ディレイ型)

プロトタイプ

表現形式を持たない。

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

パーズングやデータアクセスの遅延を示すデータ型。ユーザが直接操作することは無い。

参考

バグ

5.2.10 XLT_PTR(構造体ポインタ型)

プロトタイプ

表現形式を持たない。

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

内部のデータ構造へのポインタ。システムが利用する型である。直接そのデータ構造をユーザが書き換えることは出来ない。

参考

バグ

5.2.11 XLT_RAW(RAW データ型)

プロトタイプ

LISP 形式

XLT_RAW ::= '#'*digit_chars*'#' *byte_stream*

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

digit_chars バイトからなる *byte_stream*。内部表現は、これをメモリ上に展開したもの。

参考

バグ

5.2.12 XLT_ENV(環境型)

プロトタイプ

表現形式を持たない。

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

XLT_SYMBOL (5.2.4 節) とデータの対応表。

参考

バグ

第6章 Listbase Environment 型リファレンス

6.1 概要

GetListbaseEnv (8.2.54 節) の戻り値、および、 SetListbaseEnv (8.2.53 節) の引数に与える Listbase Environment のフォーマットをここで定義する。

6.2 XML 要素

6.2.1 `xlisp-env`

プロトタイプ

```
<xlisp-env> <type> pair <type> pair-env1 pair-env2 </xlisp-env>  
<xlisp-env> <type> env <type> <parent> env1 <parent> <work> work <work> <flags> flags  
<flags> <hash> element.... <hash> </xlisp-env>
```

内部要素

<type> [1] 環境タイプ 内部要素は pair/env

<parent> [1] 親環境

<work> [1] ワークエリア

<flags> [1] フラグ

<hash> [1] バインド情報

pair-env1 pair-env2 [2] XLT_ENV (5.2.12 節), XLT_NULL (5.2.1 節) ペア環境

env1 [1] XLT_ENV (5.2.12 節), XLT_NULL (5.2.1 節) 親環境

work [1] XLT_PTR (5.2.10 節) ワークエリア

flags [1] XLT_INTEGER (5.2.6 節) フラグ

element [1] バインド情報

属性

所属エージェント

xl

要素パス表現

`xlisp-env`

`xlisp-env` / type (6.2.2 節)

`xlisp-env` / parent (6.2.3 節)

`xlisp-env` / work (6.2.4 節)

`xlisp-env` / flags (6.2.5 節)

`xlisp-env` / hash (6.2.6 節)

`xlisp-env` / hash (6.2.6 節)/ element (6.2.7 節)

説明

環境 (XLT_ENV (5.2.12 節)) の情報をリストの形で表現したもの。環境タイプは、環境が pair 型か env 型かを表し、その型によって、他の内部要素が異なる。

`pair-env1, pair-env2` はペア型のみにある内部要素でペアの各環境である。

他の内部要素は env 型の環境にある内部要素である。parent は親環境をあらわし、work はワークエリアを表す。flags は各種フラグ、hash 要素の中にはバインドされているデータの情報がリストされる。

内部表現は、データ保存場所を持たない。

参考

8.2.54 節, 8.2.53 節

バグ

6.2.2 type

プロトタイプ

<type> *type* <type>

内部要素

type [1] XLT_STRING (5.2.5 節) 環境タイプ pair/env

属性

所属エージェント

xl

要素パス表現

xlisp-env (6.2.1 節)

xlisp-env/ **type**

xlisp-env/ parent (6.2.3 節)

xlisp-env/ work (6.2.4 節)

xlisp-env/ flags (6.2.5 節)

xlisp-env/ hash (6.2.6 節)

xlisp-env/hash/ element (6.2.7 節)

説明

環境 (XLT_ENV (5.2.12 節)) の構造を決めるタイプを指定する。

参考

8.2.54 節, 8.2.53 節

バグ

6.2.3 parent

プロトタイプ

<parent> *parent* <parent>

内部要素

type [1] XLT_ENV (5.2.12 節) 親環境

属性

所属エージェント

xl

要素パス表現

xlisp-env (6.2.1 節)

xlisp-env/ *type* (6.2.2 節)

xlisp-env/ **parent**

xlisp-env/ *work* (6.2.4 節)

xlisp-env/ *flags* (6.2.5 節)

xlisp-env/ *hash* (6.2.6 節)

xlisp-env/ *hash* (6.2.6 節)/ *element* (6.2.7 節)

説明

環境 (XLT_ENV (5.2.12 節)) が *env* 型の際の親環境を示す。

参考

8.2.54 節, 8.2.53 節

バグ

6.2.4 work

プロトタイプ

<work> *work* <work>

内部要素

work [1] XLT_PTR (5.2.10 節) ワークエリア

属性

所属エージェント

xl

要素パス表現

xlisp-env (6.2.1 節)

xlisp-env/ type (6.2.2 節)

xlisp-env/ parent (6.2.3 節)

xlisp-env/ **work**

xlisp-env/ flags (6.2.5 節)

xlisp-env/ hash (6.2.6 節)

xlisp-env/ hash (6.2.6 節)/ element (6.2.7 節)

説明

環境 (XLT_ENV (5.2.12 節)) が env 型の際のワークエリアを示す。

参考

8.2.54 節, 8.2.53 節

バグ

6.2.5 flags

プロトタイプ

<flags> *work* <flags>

内部要素

flags [1] XLT_INTEGER (5.2.6 節) 各種フラグ

属性

所属エージェント

xl

要素パス表現

xlisp-env (6.2.1 節)

xlisp-env/ type (6.2.2 節)

xlisp-env/ parent (6.2.3 節)

xlisp-env/ work (6.2.4 節)

xlisp-env/ **flags**

xlisp-env/ hash (6.2.6 節)

xlisp-env/ hash (6.2.6 節)/ element (6.2.7 節)

説明

環境 (XLT_ENV (5.2.12 節)) が env 型の際のフラグを示す。

参考

8.2.54 節, 8.2.53 節

バグ

6.2.6 hash

プロトタイプ

<hash> <element> *symbol data* </element> ... <hash>

内部要素

<element> [0+] 各バインド情報

属性

所属エージェント

xl

要素パス表現

xlisp-env (6.2.1 節)

xlisp-env/ type (6.2.2 節)

xlisp-env/ parent (6.2.3 節)

xlisp-env/ work (6.2.4 節)

xlisp-env/ flags (6.2.5 節)

xlisp-env/ **hash**

xlisp-env/ **hash** / element (6.2.7 節)

説明

環境 (XLT_ENV (5.2.12 節)) が env 型のときのバインド情報ハッシュエリアのリスト。

参考

8.2.54 節, 8.2.53 節

バグ

6.2.7 element

プロトタイプ

<element> *symbol data* </element>

内部要素

symbol [1] XLT_SYMBOL (5.2.4 節) バインド先のシンボル名

data [1] 任意の型 バインドされているデータ

属性

所属エージェント

xl

要素パス表現

xlisp-env (6.2.1 節)

xlisp-env/ type (6.2.2 節)

xlisp-env/ parent (6.2.3 節)

xlisp-env/ work (6.2.4 節)

xlisp-env/ flags (6.2.5 節)

xlisp-env/ hash (6.2.6 節)

xlisp-env/hash/ **element**

説明

環境 (XLT_ENV (5.2.12 節)) が env 型のときのバインド情報本体を表す。

参考

8.2.54 節, 8.2.53 節

バグ

第7章 XLエラーリファレンス

7.1 概要

XL 関数の実行で発生するエラー `XLT_ERROR` のコードリファレンスである。エラーコードはまず4つのカテゴリに分類されており、これに、なにを行おうとしたときのエラーかを分類する動詞分類が添えられてエラーコードとなっている。

7.2 エラー

7.2.1 カテゴリ SYSTEM

プロトタイプ

XLE_SYS

コード

0x00010000

内部要素

属性

所属エージェント

x1

要素パス表現

なし

説明

システムエラーのカテゴリ。ファイル読み込み時に発生するエラーやインタラプト、あるいは、ユーザからは認識されないキャンセルの伝達や、Exit の実行、For などのループの終了や continue など、システムに関連したエラーのカテゴリ。

参考

バグ

7.2.2 カテゴリ SYNTAX

プロトタイプ

XLE_SYN

コード

0x00020000

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

シンタックスエラーのカテゴリ。XL 言語でかかれたファイルを読み込み中に発生する、レキシカルまたは文法的なエラーの分類。

参考

バグ

7.2.3 カテゴリ SEMANTIX

プロトタイプ

XLE_SEM

コード

0x00030000

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

セマンティックスエラーのカテゴリ。XL 関数を呼び出したときに、型が違うなど、統語的なエラーの分類。

参考

バグ

7.2.4 カテゴリ PROTOCOL

プロトタイプ

XLE_PRO

コード

0x00040000

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

プロトコルエラーのカテゴリ。アプリケーション依存のエラーや通信時に発生するエラーの分類。

参考

バグ

7.2.5 動詞分類 READ

プロトタイプ

XLE_READ

コード

0x00000100

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

ファイル等の読み込み時に発生するエラーの分類。

参考

バグ

7.2.6 動詞分類 EXEC

プロトタイプ

XLE_EXEC

コード

0x00000200

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

コードの実行時などに発生するエラーの分類。

参考

バグ

7.2.7 動詞分類 MANY

プロトタイプ

XLE_MANY

コード

0x00000300

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

引数の数など、必要数を数えた時に必要数が満たされなかった、あるいは、多すぎるといったエラーの分類。

参考

バグ

7.2.8 動詞分類 NOTHING

プロトタイプ

XLE_NOTH

コード

0x00000400

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

なにかの存在を確かめたときに発生する。存在しているべき物が無かったというエラーの分類。

参考

バグ

7.2.9 動詞分類 INVALID

プロトタイプ

XLE_INVA

コード

0x00000500

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

なんらかの条件を満たしてなければならないときに、それが満たされていないというエラー。

参考

バグ

7.2.10 動詞分類 UNSUPPORT

プロトタイプ

XLE_USUP

コード

0x00000600

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

将来的にはサポートされる予定ではあるが、現在はサポートされていない機能を利用した。

参考

バグ

7.2.11 動詞分類 UNDEFINED

プロトタイプ

XLE_UDEF

コード

0x00000700

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

定義されているべき物が定義されていなかった。

参考

バグ

7.2.12 動詞分類 MISSMATCH

プロトタイプ

XLE_MACH

コード

0x00000800

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

整合性がとれていない。

参考

バグ

7.2.13 動詞分類 REQUIRED

プロトタイプ

XLE_REQU

コード

0x00000900

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

必要なものが無かった。

参考

バグ

7.2.14 動詞分類 OPEN

プロトタイプ

XLE_OPEN

コード

0x00000a00

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

ファイルなどを開こうとしたが、開くことが出来なかった。

参考

バグ

7.2.15 動詞分類 DUPLICATE

プロトタイプ

XLE_DUPP

コード

0x00000b00

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

すでに定義、生成されている物をもう一度定義、生成しようとした。

参考

バグ

7.2.16 動詞分類 CLOSE

プロトタイプ

XLE_CLSE

コード

0x00000c00

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

すでに閉じられているもの、あるいは存在しない物を閉じようとした。

参考

バグ

7.2.17 動詞分類 DENIED

プロトタイプ

XLE_DENI

コード

0x0000d00

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

許可が与えられていないことをしようとした。

参考

バグ

7.2.18 動詞分類 UNSPECIFIED

プロトタイプ

XLE_UNSP

コード

0x00000e00

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

特定されていないものにアクセスしようとした。

参考

バグ

7.2.19 動詞分類 ALREADY

プロトタイプ

XLE_ALRD

コード

0x00000f00

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

すでに開かれているものを再度開こうとした。

参考

バグ

7.2.20 動詞分類 UNSUTISFIED

プロトタイプ

XLE_ALRD

コード

0x00001000

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

条件が満たされていない。

参考

バグ

7.2.21 動詞分類 RELAIED

プロトタイプ

XLE_RELAY

コード

0x00001100

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

インタプリターの RELAY エラー。現在は使われていない。

参考

バグ

7.2.22 動詞分類 ERROR UTILITY

プロトタイプ

XLE_UTIL

コード

0x00001200

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

システムエラーのユーティリティ用エラーコード。

参考

バグ

7.2.23 動詞分類 CORRESPONDING

プロトタイプ

XLE_CORR

コード

0x00001300

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

タグの対応がおかしい。

参考

バグ

7.2.24 動詞分類 EXIT

プロトタイプ

XLE_EXIT

コード

0x00001400

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

プログラム EXIT やキャンセルなど。

参考

バグ

7.2.25 動詞分類 MATHEMATICAL

プロトタイプ

XLE_MATH

コード

0x00001500

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

プログラム EXIT やキャンセルなど。

参考

バグ

7.2.26 動詞分類 LOOP

プロトタイプ

XLE_LOOP

コード

0x00001600

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

For (8.2.57 節) 分のための Break (8.2.58 節), Continue (8.2.59 節) のためのエラーコード

参考

バグ

7.2.27 動詞分類 OVER

プロトタイプ

XLE_OVER

コード

0x00001700

内部要素

属性

所属エージェント

xl

要素パス表現

なし

説明

OVER FLOW や、時間切れなどのエラー。

参考

バグ

7.2.28 XLE_OK

プロトタイプ

XLE_OK

コード

0

内部要素

属性

なし。

所属エージェント

x1

要素パス表現

なし

説明

エラーがなかったことを示す。

参考

バグ

7.2.29 XLE_SYSTEM_READ_FILE

プロトタイプ

XLE_SYSTEM_READ_FILE

コード

(XLE_ERR|XLE_SYS|XLE_READ|1)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

ファイルの読み込み、および、レキシカルアナリシスの時に発生したエラー。

参考

バグ

7.2.30 XLE_SYSTEM_NETWORK

プロトタイプ

XLE_SYSTEM_NETWORK

コード

(XLE_ERR|XLE_SYS|XLE_READ|2)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

x1

要素パス表現

なし

説明

ネットワークアクセス時に問題が発生し、これ以上通信が出来ないというエラー。

参考

バグ

7.2.31 XLE_SYSTEM_NETWORK

プロトタイプ

XLE_SYSTEM_DONTPRINT

コード

(XLE_ERR | XLE_SYS | XLE_UTIL | 3)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

x1

要素パス表現

なし

説明

表示されないエラー。戻り値としても表示されない。

参考

バグ

7.2.32 XLE_SYSTEM_EXIT

プロトタイプ

XLE_SYSTEM_EXIT

コード

(XLE_ERR|XLE_SYS|XLE_EXIT|4)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

Exit 命令 [UNDEF REF (xl-Exit)] またはそれに等価な命令が実行された。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.33 XLE_SYSTEM_CANCEL

プロトタイプ

XLE_SYSTEM_CANCEL

コード

(XLE_ERR | XLE_SYS | XLE_EXIT | 5)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

実行がキャンセルされた。コンソールからの入力としては、ctrl+C 相当の実行が行われた。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.34 XLE_SYSTEM_INTERRUPT

プロトタイプ

XLE_SYSTEM_INTERRUPT

コード

(XLE_ERR | XLE_SYS | XLE_EXIT | 6)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

実行中に何らかの割り込みが発生され実行が中断された。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.35 XLE_SYSTEM_INTERNAL

プロトタイプ

XLE_SYSTEM_INTERNAL

コード

(XLE_ERR | XLE_SYS | XLE_INVA | 7)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

システム内部の復旧しがたいエラーが発生した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.36 XLE_SYSTEM_LOOP_BREAK

プロトタイプ

XLE_SYSTEM_LOOP_BREAK

コード

(XLE_ERR|XLE_SYS|XLE_LOOP|8)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

Break (8.2.58 節) が実行された。 For (8.2.57 節) 命令はこのエラーを受け取り、For 文自体を終了する。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.37 XLE_SYSTEM_LOOP_CONTINUE

プロトタイプ

XLE_SYSTEM_LOOP_CONTINUE

コード

(XLE_ERR|XLE_SYS|XLE_LOOP|9)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

Continue (8.2.59 節) が実行された。 For (8.2.57 節) 命令はこのエラーを受け取り、カウンタを 1 アップし、For 文自体を終了する。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.38 XLE_SYSTEM_APPLICATION

プロトタイプ

XLE_SYSTEM_APPLICATION

コード

(XLE_ERR|XLE_SYS|XLE_LOOP|10)

内部要素

各関数の仕様による。各関数の「エラー」の項目を参照。

属性

所属エージェント

xl

要素パス表現

なし

説明

アプリケーション依存のシステムエラーコード

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.39 XLE_SYNTAX_TOO_MANY_PUNC

プロトタイプ

XLE_SYNTAX_TOO_MANY_PUNC

コード

(XLE_ERR|XLE_SYN|XLE_MANY|1)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

パーズング中、終了の括弧”)”が開始の括弧より多い。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.40 XLE_SYNTAX_NO_PUNC

プロトタイプ

XLE_SYNTAX_NO_PUNC

コード

(XLE_ERR|XLE_SYN|XLE_NOTH|2)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

パーズング中、開始の括弧に対応する終了の括弧がない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.41 XLE_SYNTAX_ERROR_CODE

プロトタイプ

XLE_SYNTAX_ERROR_CODE

コード

(XLE_ERR|XLE_SYN|XLE_UNST|3)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

エラー型のデータをパーズングに文法上のエラーが発生した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.42 XLE_SYNTAX_UNSUTISFIED_FILE_END

プロトタイプ

XLE_SYNTAX_UNSUTISFIED_FILE_END

コード

(XLE_ERR|XLE_SYN|XLE_UNST|4)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

パーズング中、予期せぬスクリプトの位置でファイルの終端に達してしまった。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.43 XLE_SYNTAX_TOKEN_ERROR

プロトタイプ

XLE_SYNTAX_TOKEN_ERROR

コード

(XLE_ERR|XLE_SYN|XLE_UNST|5)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

パーズング中、トークンのエラーが発生した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.44 XLE_SYNTAX_NOT_CORRESPOND_TAG

プロトタイプ

XLE_SYNTAX_NOT_CORRESPOND_TAG

コード

(XLE_ERR|XLE_SYN|XLE_CORR|6)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

パーズング中、開始タグと終了タグが一致していない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.45 XLE_SYNTAX_UNTERMINATED_TEXT

プロトタイプ

XLE_SYNTAX_UNTERMINATED_TEXT

コード

(XLE_ERR|XLE_SYN|XLE_INVA|7)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

パーズング中、テキストの開始の”マークがあるものの、終了の”マークが存在しない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.46 XLE_SYNTAX_INVALID_ENTITY

プロトタイプ

XLE_SYNTAX_INVALID_ENTITY

コード

(XLE_ERR|XLE_SYN|XLE_INVA|8)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

パーズング中、文法上正しくないエンティティを発見した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.47 XLE_SEMANTICS_EXEC_NO_FUNC

プロトタイプ

XLE_SEMANTICS_EXEC_NO_FUNC

コード

(XLE_ERR|XLE_SEM|XLE_EXEC|1)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

関数型 (XLT_FUNC (5.2.8 節)) でないデータを実行しようとした。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.48 XLE_SEMANTICS_INV_PARAM_LENGTH

プロトタイプ

XLE_SEMANTICS_INV_PARAM_LENGTH

コード

(XLE_ERR|XLE_SEM|XLE_INVA|2)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

関数呼び出し時に、その関数の本来の引数の数とことなった数の引数で呼び出そうとしていた。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.49 XLE_SEMANTICS_INV_FORMAT

プロトタイプ

XLE_SEMANTICS_INV_FORMAT

コード

(XLE_ERR|XLE_SEM|XLE_INVA|3)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

関数に与えるデータの構造が合っていない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.50 XLE_SEMANTICS_UN SUPPORT_FUNC

プロトタイプ

XLE_SEMANTICS_UN SUPPORT_FUNC

コード

(XLE_ERR | XLE_SEM | XLE_USUP | 4)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

将来サポートする予定だが、現在は実装されていない機能を実行しようとした。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.51 XLE_SEMANTICS_UNDEF_SYMBOL

プロトタイプ

XLE_SEMANTICS_UNDEF_SYMBOL

コード

(XLE_ERR | XLE_SEM | XLE_UNDEF | 5)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

与えられたシンボル (XLT_SYMBOL (5.2.4 節)) を評価しようとしたが定義されていなかった。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.52 XLE_SEMANTICS_TYPE_MISMATCH

プロトタイプ

XLE_SEMANTICS_TYPE_MISMATCH

コード

(XLE_ERR | XLE_SEM | XLE_MACH | 6)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

与えられたデータの型が合っていない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.53 XLE_SEMANTICS_REQ_SYMBOL

プロトタイプ

XLE_SEMANTICS_REQ_SYMBOL

コード

(XLE_ERR|XLE_SEM|XLE_REQU|7)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

シンボル (XLT_SYMBOL (5.2.4 節)) が求められるところ、他の型のデータが使われている。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.54 XLE_SEMANTICS_DIV_ZERO

プロトタイプ

XLE_SEMANTICS_DIV_ZERO

コード

(XLE_ERR|XLE_SEM|XLE_MATH|8)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

ゼロで割り算を試みようとした。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.55 XLE_SEMANTICS_UNIT_MISMATCH

プロトタイプ

XLE_SEMANTICS_UNIT_MISMATCH

コード

(XLE_ERR|XLE_SEM|XLE_MACH|9)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

数値につけられた単位が合っていない。たとえば、

(+ 1km 1g)

のような計算をしようとした。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.56 XLE_PROTO_INV_OBJECT

プロトタイプ

XLE_PROTO_INV_OBJECT

コード

(XLE_ERR|XLE_PRO|XLE_INVA|1)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

与えられたオブジェクトが正しくない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.57 XLE_PROTO_INV_RESOURCE

プロトタイプ

XLE_PROTO_INV_RESOURCE

コード

(XLE_ERR|XLE_PRO|XLE_INVA|2)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

与えられたリソースが正しくない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.58 XLE_PROTO_INV_PARAM

プロトタイプ

XLE_PROTO_INV_PARAM

コード

(XLE_ERR|XLE_PRO|XLE_INVA|3)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

与えられたパラメータ、特に属性が無いが、正しくない値である。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.59 XLE_PROTO_INV_FIELD_NAME

プロトタイプ

XLE_PROTO_INV_FIELD_NAME

コード

(XLE_ERR|XLE_PRO|XLE_INVA|4)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

与えてはいけない属性が与えられた。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.60 XLE_PROTO_INV_FILE_TYPE

プロトタイプ

XLE_PROTO_INV_FILE_TYPE

コード

(XLE_ERR|XLE_PRO|XLE_INVA|5)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

ファイルタイプが間違っている。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.61 XLE_PROTO_OPEN_FILE

プロトタイプ

XLE_PROTO_OPEN_FILE

コード

(XLE_ERR|XLE_PRO|XLE_OPEN|6)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

ファイルやストリームがオープン出来なかった。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.62 XLE_PROTO_UNSUPPORT_FILE_TYPE

プロトタイプ

XLE_PROTO_UNSUPPORT_FILE_TYPE

コード

(XLE_ERR|XLE_PRO|XLE_USUP|7)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

サポートされていないファイルタイプを指定した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.63 XLE_PROTO_UNSUPPORTED_MODE

プロトタイプ

XLE_PROTO_UNSUPPORTED_MODE

コード

(XLE_ERR|XLE_PRO|XLE_USUP|8)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

サポートされていないモードを指定した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.64 XLE_PROTO_UNSUPPORT_PROTO

プロトタイプ

XLE_PROTO_UNSUPPORT_PROTO

コード

(XLE_ERR|XLE_PRO|XLE_USUP|9)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

サポートされていないプロトコルをを指定した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.65 XLE_PROTO_UNSUPPORT_TYPE

プロトタイプ

XLE_PROTO_UNSUPPORT_TYPE

コード

(XLE_ERR|XLE_PRO|XLE_USUP|10)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

サポートされていない型を指定した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.66 XLE_PROTO_DUP_DEFINITION

プロトタイプ

XLE_PROTO_DUP_DEFINITION

コード

(XLE_ERR|XLE_PRO|XLE_DUPP|11)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

二重定義しようとした。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.67 XLE_PROTO_UNDEF_NAME

プロトタイプ

XLE_PROTO_UNDEF_NAME

コード

(XLE_ERR|XLE_PRO|XLE_UNDEF|12)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

定義されていない名前を指定した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.68 XLE_PROTO_UNDEF_DATABASE

プロトタイプ

XLE_PROTO_UNDEF_DATABASE

コード

(XLE_ERR|XLE_PRO|XLE_UNDEF|13)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

定義されていないデータベースを指定した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.69 XLE_PROTO_UNDEF_RESOURCE

プロトタイプ

XLE_PROTO_UNDEF_RESOURCE

コード

(XLE_ERR|XLE_PRO|XLE_UNDEF|14)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

定義されていないデータベースを指定した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.70 XLE_PROTO_REQ_PRESET

プロトタイプ

XLE_PROTO_REQ_PRESET

コード

(XLE_ERR|XLE_PRO|XLE_REQU|15)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

初期設定の実行が求められる。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.71 XLE_PROTO_PERMISSION_DENIED

プロトタイプ

XLE_PROTO_PERMISSION_DENIED

コード

(XLE_ERR|XLE_PRO|XLE_DENI|16)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

許可が与えられていない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.72 XLE_PROTO_CLOSED

プロトタイプ

XLE_PROTO_CLOSED

コード

(XLE_ERR|XLE_PRO|XLE_CLSE|17)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

既に閉じられているファイルやストリームに対してオペレーションをしようとした。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.73 XLE_PROTO_ACCESS_STREAM

プロトタイプ

XLE_PROTO_ACCESS_STREAM

コード

(XLE_ERR|XLE_PRO|XLE_OPEN|18)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

ネットワークに何らかの問題が発生し、コネクションが切断された。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.74 XLE_PROTO_UNSPC_RESOURCE

プロトタイプ

XLE_PROTO_UNSPC_RESOURCE

コード

(XLE_ERR|XLE_PRO|XLE_UNSP|19)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

リソースが特定されていない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.75 XLE_PROTO_ALREADY_OPEN

プロトタイプ

XLE_PROTO_ALREADY_OPEN

コード

(XLE_ERR | XLE_PRO | XLE_ALRD | 20)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

既にオープンされているファイルやストリームなどをさらにオープンしようとした。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.76 XLE_PROTO_UNSUTISFIED_INFO

プロトタイプ

XLE_PROTO_UNSUTISFIED_INFO

コード

(XLE_ERR|XLE_PRO|XLE_UNST|21)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

必要な情報が与えられていない。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.77 XLE_PROTO_REQ_PUT

プロトタイプ

XLE_PROTO_REQ_PUT

コード

(XLE_ERR|XLE_PRO|XLE_REQU|22)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

PUT が求められている。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.78 XLE_PROTO_EXEC_ERROR

プロトタイプ

XLE_PROTO_EXEC_ERROR

コード

(XLE_ERR|XLE_PRO|XLE_EXEC|23)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

実行環境の整備においてエラーが発生した。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.79 XLE_PROTO_RELAY

プロトタイプ

XLE_PROTO_RELAY

コード

(XLE_ERR|XLE_PRO|XLE_RELY|24)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

インタプリターのリレー機能のエラー。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.80 XLE_PROTO_LAUNCH

プロトタイプ

XLE_PROTO_LAUNCH

コード

(XLE_ERR|XLE_PRO|XLE_OPEN|25)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

エージェントを実行しようとしたが出来なかった。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.81 XLE_PROTO_USER_ERROR

プロトタイプ

XLE_PROTO_USER_ERROR

コード

(XLE_ERR|XLE_PRO|XLE_EXEC|26)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

ユーザエラー。(他のエラーに移行の必要性あり。)

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.82 XLE_PROTO_UNKNOWN

プロトタイプ

XLE_PROTO_UNKNOWN

コード

(XLE_ERR | XLE_PRO | XLE_UNSP | 27)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

解析不能なエラーが発生した。特にネットワークルーティング上の問題で、通信し合う相手のエージェントのバージョンが低いために、相手に発生したエラーがなにか特定出来なかったなど。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.83 XLE_PROTO_NO_ROUTE

プロトタイプ

XLE_PROTO_NO_ROUTE

コード

(XLE_ERR|XLE_PRO|XLE_UNDEF|28)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

経路探索において、必要なリソースにたどり着く経路が発見出来なかった。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.84 XLE_PROTO_OVER_TTL

プロトタイプ

XLE_PROTO_OVER_TTL

コード

(XLE_ERR|XLE_PRO|XLE_OVER|28)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

経路探索において、TTL (Time To Live) の値がオーバーフローした。つまり、経路が長過ぎて探索出来なかった。考えられることとして、経路情報がループしているゴミの経路情報が発生しているということ。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.85 XLE_PROTO_INV_IID

プロトタイプ

XLE_PROTO_INV_IID

コード

(XLE_ERR|XLE_PRO|XLE_INVA|29)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

存在しないインタプリタ ID (IID) が指定された。

参考

[UNDEF REF (xl-Exit)]

バグ

7.2.86 XLE_PROTO_BUSY

プロトタイプ

XLE_PROTO_BUSY

コード

(XLE_ERR|XLE_PRO|XLE_OVER|30)

内部要素

属性

各関数の仕様による。各関数の「エラー」の項目を参照。

所属エージェント

xl

要素パス表現

なし

説明

エージェントの起動などで、サーバなどに負荷がかかっており、必要とする機能の起動が出来なかった。

参考

[UNDEF REF (xl-Exit)]

バグ

第8章 XL 基本関数

8.1 概要

インタプリタ初期化時に、環境にデフォルトで組み込まれる変数とその値をプリミティブと呼ぶ。そのうち値は固定的であり、C などのインタプリタを記述する言語で書かれている。そのうち、あらゆる XL でいつでも使える関数を汎用プリミティブとよぶ。また、データベース検索など何らかの高機能な目的のために拡張するために組み込まれるものを拡張プリミティブと呼ぶ。拡張プリミティブについては、この仕様では定義しない。

8.2 関数

8.2.1 ErrorHandler

プロトタイプ

LISP 形式

(ErrorHandler exp)

XML 形式

<ErrorHandler> exp </ErrorHandler>

引数

exp [1] すべての型

属性

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

エラーをとらえることの出来る特殊な Eval を使って、exp を評価する。エラーが発生している場合は、それを規定のリストにして返す。エラーが発生していない場合は、規定のリストの data のところに exp を埋め込みエラーコード 0 として返す。

```
(define c (ErrorHandler (/ a b)))  
<If> (= (GetElement c 1) (quote Error))  
  <Then> ゼロ割に関する処理 </Then>  
  <Else> 通常処理 </Else>  
</If>
```

といった処理を行うことにより、エラー処理が可能である。

戻り値

等関数の戻り値は以下の形式をとる。

LISP 形式

```
(Error シーケンス番号 エラー発生サイト URL エラー発生ファイル名 エラー発生関数  
エラーコード data)
```

XML 形式

```
<Error>  
  シーケンス番号  
  エラー発生サイト URL  
  エラー発生ファイル名
```

```
エラーコード
data
</Error>
```

exp が XLT_ERR の場合は、XLT_ERR の内容を上記リストに当てはめる。exp がエラーでない場合は、エラーコード=0 として、data のところに exp をそのまま保存する。

エラー
なし。
参考
バグ

8.2.2 Env0

プロトタイプ

Env0

引数

各関数の「環境」の項目に指示されている。

属性

評価形式

environment

所属エージェント

xl

所属環境

Env0

説明

レベル0の関数、シンボルが定義されている環境が Env0 である。レベル1の関数、シンボルが定義されている環境が Env1 (8.2.3 節) である。Env1 の親環境は Env0 である。また エージェント立ち上げ時に動くインタプリタは、Env1 がエントリ環境となる。

戻り値

なし。

エラー

なし。

参考

バグ

8.2.3 Env1

プロトタイプ

Env1

引数

各関数の「環境」の項目に指示されている。

属性

評価形式

environment

所属エージェント

x1

所属環境

Env0

説明

レベル0の関数、シンボルが定義されている環境が Env0 (8.2.2 節) である。レベル1の関数、シンボルが定義されている環境が Env1 である。Env1 の親環境は Env0 である。また エージェント立ち上げ時に動くインタプリタは、Env1 がエントリ環境となる。

戻り値

なし。

エラー

なし。

参考

バグ

8.2.4 OpenInterpreter

プロトタイプ

LISP 形式

(OpenInterpreter connection-type env-exp option)

XML 形式

<OpenInterpreter> connection-type env-exp option </OpenInterpreter>

引数

connection-type [1] XLT_STRING (5.2.5 節)stdio/socket/ipc/file

env-exp [1] XLT_ENV (5.2.12 節)

option [1] 任意の型 オプション

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env1

説明

インタプリタの受け口を定義する。受け口の接続元を、stdio,socket,ipc の中から指定する。また、最初のエントリ環境を env-exp に指定する。option は以下の通り。

表 8.1: OpenInterpreter

LISP 形式	XML 形式	説明
(Port integer-exp)	<Port> integer-exp </Port>	ソケットまたは
(MaxConnection integer-exp)	<MaxConnection> integer-exp </MaxConnection>	最大ソケット/I
(InputFileName string-exp)	<InputFileName> string-exp </InputFileName>	接続元の入カス
(OutputFileName string-exp)	<OutputFileName> string-exp </FileName>	接続元の出カス
(MaxConnectionTime integer-exp)	<MaxConnectionTime> integer-exp </MaxConnectionTime>	最大コネクショ
(MaxSilentTime integer-exp)	<MaxSilentTime> integer-exp </MaxSilentTime>	最大停止時間の
(Environment "new/thisone")	<Environment> new/thisone </Environment>	"thisone" を指定

表 8.2: OpenInterpreter(2)

命令	対象となる受け口の種類	デフォルト値
Port	socket/ipc	9090
MaxConnection	socket/ipc	10
InputFileName	file	なし。指定しないとエラー
OutputFileName	file	なし。指定しないと出力されない。
MaxConnectionTime	stdio/socket/ipc/file	無限大 (-1)
MaxSilentTime	stdio/socket/ipc/file	無限大 (-1)
Environment	stdio/socket/ipc/file	"thisone"

戻り値

成功時 IID (インタプリタ ID) 失敗すると所定のエラーコード。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

(a) "first argument (interpreter type)"

(b) "first argument (interpreter environment)"

2. XLE_PROTO_INV_PARAM (7.2.58 節)

"interpreter accept type"

3. XLE_PROTO_ACCESS_STREAM (7.2.73 節)

参考

バグ

8.2.5 Eval

プロトタイプ

LISP 形式

(Eval env exp)

XML 形式

<Eval> env exp </Eval>

引数

env [1] XLT_ENV (5.2.12 節)

exp [1] すべての型

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

環境変数と、もう一つ値を受け取り、その値を評価し、結果を返す。

戻り値

評価結果。

エラー

1. XLE.SEMANTICS_TYPE_MISMATCH (7.2.52 節)
2. XLE.SYSTEM_CANCEL (7.2.33 節)
3. XLE.SEMANTICS_UNSUPPORTED_FUNC (7.2.50 節) "eval:unsupported lambda"
4. XLE.SEMANTICS_INV_PARAM_LENGTH (7.2.48 節)
 - (a) "invalid parameters length"
applicative 関数の引数の長さが関数定義と違う。
 - (b) "invalid parameters length (2)"
applicative 以外の関数の引数の長さが関数定義と違う。
5. XLE.SEMANTICS_EXEC_NO_FUNC (7.2.47 節) "execute no function"
6. XLE.SEMANTICS_TYPE_MISMATCH (7.2.52 節) "eval (attribute symbol)" "eval:type mismatch"
属性値を評価したところ、属性値としては使えない k 型であった。
7. XLE.SEMANTICS_UNDEF_SYMBOL (7.2.51 節)
 - (a) "eval (attribute symbol)" "eval:undefined symbol-1"
属性値を評価しようとしたところ、属性に与えられているシンボルは環境に無かった。
 - (b) "eval (symbol)" "eval:undefined symbol"
評価しようとしたシンボルが環境に無かった。

参考

バグ

8.2.6 Lambda

プロトタイプ

LISP 形式

1. (Lambda [Order="Normal"] (Arguments symbol) exp)
2. (Lambda [Order="Applicative"] env-exp (Arguments symbol) exp)
3. (Lambda [Order="Normal"] symbol exp)
4. (Lambda [Order="Applicative"] env-exp symbol exp)

XML 形式

1. <Lambda Order="Normal"> <Arguments> ^symbol ^symbol </Arguments> exp </Lambda>
2. <Lambda Order="Applicative"> env-exp <Arguments> ^symbol ^symbol </Arguments> exp </Lambda>
3. <Lambda Order="Normal"> ^symbol exp </Lambda>
4. <Lambda Order="Applicative"> env-exp ^symbol exp </Lambda>

引数

Arguments [0-1] XLT_SYMBOL (5.2.4 節) のリスト

exp [1] 任意の型

env-exp [0-1] XLT_ENV (5.2.12 節)

属性

Order Applicative / Normal

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

評価順序 (Order) には、引数の評価方法として Normal が Applicative のいずれかを指定する。Normal の場合は引数はすべて評価されずに、そのまま関数に渡される。環境は新たに生成され、そこに引数、属性情報などが記憶される。Applicative の場合は、引数はすべて評価され関数に渡される。関数は *exp* を評価し、その戻り値を関数の戻り値とする。引数の評価の時に使用する環境は、*env-exp* で指定される。*env-exp* = () とすると評価時のカレント環境で引数を評価する。

Lambda の戻り値は関数である。この戻り値を Eval によって評価すると、まず、与えられた環境で、この関数の属性に従って引数を評価する。次に、与えられた環境を親に持つ環境を生成し、その環境に、シンボルリストで定義されたシンボルと先の値の対応を記憶し、この環境をもって、*exp* を評価する。評価結果を、この関数の戻り値とする。

(1) の場合、引数の指定された順番と、Argument のシンボルの順番を照らし合わせ、対応するシンボルに引数を対応させる。一方、(2) は関数呼び出し時の構造型全体をシンボル *symbol* に対応させる。

戻り値

評価結果。

エラー

1. XLE_SEMANTICS_INV_FORMAT (7.2.49 節) "format error in lambda argment"

参考
バグ

8.2.7 Define

プロトタイプ

LISP 形式

1. (Define symbol exp)
2. (Define [Order="Normal"] symbol1 (Arguments symbol2) exp)
3. (Define [Order="Applicative"] symbol1 env-exp (Arguments symbol2) exp)
4. (Define [Order="Normal"] symbol1 symbol2 exp)
5. (Define [Order="Applicative"] symbol1 env-exp symbol2 exp)

XML 形式

1. <Define> ^symbol1 exp </Define>
2. <Define Order="Normal"> ^symbol1 <Arguments> ^symbol2 </Arguments> exp </Define>
3. <Define Order="Applicative"> ^symbol1 env-exp <Arguments> ^symbol2 </Arguments> exp </Define>
4. <Define Order="Normal"> ^symbol1 ^symbol2 exp </Define>
5. <Define Order="Applicative"> ^symbol1 ^symbol2 env-exp exp </Define>

引数

symbol1 [1] XLT_SYMBOL (5.2.4 節) バインドするシンボル

symbol2 [0-1] XLT_SYMBOL (5.2.4 節) 引数リストを入れるシンボル

exp [1] バインド対象。

Arguments [0-1] XLT_SYMBOL (5.2.4 節) のリスト 引数。

属性

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

(1) はカレント環境において、*symbol1* と、*exp* の評価結果を対応させる。もし、*symbol* のエントリがすでにあった場合は、その値を書き換える。

(2)(3) は (Define *symbol1* (Lambda (Arguments) *exp*) の略記形。

(4)(5) は (Define *symbol1* (Lambda *symbol* *exp*) の略記形。

戻り値

エラーがなければ、バインドされたシンボル。

エラー

1. XLE_SEMANTICS_INV_FORMAT (7.2.49 節)

参考

バグ

8.2.8 If

プロトタイプ

LISP 形式

```
( If exp1 ( Then exp2 ) ( Else exp3 ) )
```

XML 形式

```
<If> exp1 <Then> exp2 </Then> <Else> exp3 </Else> </If>
```

引数

exp1 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

Then [0-1] 評価式

Else [0-1] 評価式

属性

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

最初の *exp* を評価し、0 だったら、<Then></Then> の中の *exp* を評価し、その結果を、戻り値とする。0 以外だったら、<Else></Else> の中の *exp* を評価し、その結果を戻り値とする。

戻り値

exp2 の評価結果、または、*exp3* の評価結果。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)
2. XLE_PROTO_INV_PARAM (7.2.58 節) "then else parameter error"

Then,Else いずれも存在しなかった。

参考

バグ

8.2.9 Sequence

プロトタイプ

LISP 形式

(Sequence env exp

XML 形式

<Sequence> env exp </Sequence>

引数

env [1] XLT_ENV (5.2.12 節), XLT_NULL (5.2.1 節)

exp [1+] 任意の型

属性

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

env= () であるとき、現在のカレント環境を親環境とする新しい環境を生成し、それをカレント環境とする。そのカレント環境のもとで、*exp* を順番に評価し、最後の *exp* の戻り値を Sequence の戻り値とする。途中でエラーが発生した場合は、その時点でそのエラーを戻り値とする。

env が環境型 (XLT_ENV (5.2.12 節)) である場合はその環境上で同様の実行をする。

戻り値

最後の式の戻り値。あるいはエラーを起こした式のエラー

エラー

1. XLE.SEMANTICS_TYPE_MISMATCH (7.2.52 節)

env が XLT_ENV (5.2.12 節) または XLT_NULL (5.2.1 節) でない。

参考

バグ

8.2.10 NewEnvironment

プロトタイプ

LISP 形式

(NewEnvironment env-exp)

XML 形式

<NewEnvironment> env-exp</NewEnvironment>

引数

env-exp [1] XLT_ENV (5.2.12 節), XLT_NULL (5.2.1 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

env-exp を親環境とした環境を生成する。*env-exp*= () である場合は、親環境がない環境が生成される。

戻り値

生成された環境。

エラー

1. XLE.SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.11 GetParentEnvironment

プロトタイプ

LISP 形式

(GetParentEnvironment env-exp)

XML 形式

<GetParentEnvironment> env-exp </GetParentEnvironment>

引数

env-exp [1] XLT_ENV (5.2.12 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

env-exp の親環境を返す。*env-exp* に親環境がない場合 () を返す。

戻り値

親環境

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.12 SetParentEnvironment

プロトタイプ

LISP 形式

(SetParentEnvironment env-exp1 env-exp2)

XML 形式

<SetParentEnvironment> env-exp1 env-exp2 </SetParentEnvironment>

引数

env-exp1 [1] XLT_ENV (5.2.12 節)

env-exp2 [1] XLT_ENV (5.2.12 節), XLT_NULL (5.2.1 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

env-exp2 を *env-exp1* の親環境としてセットする。戻り値は ()。 *env-exp2* を XLT_NULL (5.2.1 節) とすると、親環境はなしとなる。

戻り値

XLT_NULL (5.2.1 節)

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.13 CurrentEnvironment

プロトタイプ

LISP 形式

(CurrentEnvironment)

XML 形式

<CurrentEnvironment/>

引数

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

現在のカレント環境を返す。

戻り値

現在のカレント環境を返す。

エラー

参考

バグ

8.2.14 quote

プロトタイプ

LISP 形式

(quote exp)

'exp

XML 形式

<quote> exp </quote>

引数

exp [1] 任意の型

属性

type 任意 normal/direct default="normal"

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

exp を評価しないでそのまま返す。ただし、*type*="normal" の時には、*exp* の内部に、(\$ *exp2*) という S 式がある場合は、*exp2* を評価し、そこに埋め込む。

戻り値

exp

エラー

参考

バグ

8.2.15 GetElement

プロトタイプ

LISP 形式

1. (GetElement exp integer)
2. (GetElement exp symbol)
3. (GE exp integer)
4. (GE exp symbol)

XML 形式

1. <GetElement> exp integer </GetElement>
2. <GetElement> exp ^symbol <GetElement>
3. <GE> exp integer <GE>
4. <GE> exp symbol <GE>

引数

exp [1] XLT_PAIR (5.2.3 節)

integer [0-1] XLT_INTEGER (5.2.6 節)

symbol [0-1] XLT_SYMBOL (5.2.4 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

(1) は、*exp* で与えられた値が構造型であった場合、*integer* 番目の要素を返す。(2) は *symbol* で示される要素を検索し返す。

戻り値

対応するリスト。

エラー

参考

バグ

8.2.16 Next

プロトタイプ

LISP 形式

(Next exp)

XML 形式

<Next> exp </Next>

引数

exp [1] XLT_PAIR (5.2.3 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp で与えられた値が XLT_PAIR (5.2.3 節) であった場合、*cdr* の要素を返す。 *cdr* (8.2.17 節) と同じ。
の値を返す。

戻り値

対応するリスト。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

8.2.18 節, [UNDEF REF (*cdr*)]

バグ

8.2.17 cdr

プロトタイプ

LISP 形式

(cdr exp)

XML 形式

<cdr> exp </cdr>

引数

exp [1] XLT_PAIR (5.2.3 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp で与えられた値が XLT_PAIR (5.2.3 節) であった場合、cdr の要素を返す。Next (8.2.17 節) と同じ。
の値を返す。

戻り値

対応するリスト。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

8.2.18 節, 8.2.17 節

バグ

8.2.18 car

プロトタイプ

LISP 形式

(car exp)

XML 形式

<car> exp </car>

引数

exp [1] XLT_PAIR (5.2.3 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp で与えられた値が XLT_PAIR (5.2.3 節) であった場合、*car* の要素を返す。 の値を返す。

戻り値

対応するリスト。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

8.2.18 節, 8.2.17 節

バグ

8.2.19 CutOut

プロトタイプ

LISP 形式

(CutOut exp integer1 integer2)

XML 形式

<CutOut> exp verb+|+| integer1 verb+|+| integer2 </CutOut>

引数

exp [1] XLT_PAIR (5.2.3 節)

integer1 [1]

integer2 [1]

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp で与えられた値が XLT_PAIR (5.2.3 節)(リスト型) であった場合、その *integer1* から *integer2* 番目の部分を切り出したリストを返す。

戻り値

対応するリスト。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.20 Append

プロトタイプ

LISP 形式

(Append exp1 exp2)

XML 形式

<Append> exp1 exp2 </Append>

引数

exp1 [1] XLT_PAIR (5.2.3 節)

exp2 [1] XLT_PAIR (5.2.3 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 *exp2* で与えられた値が構造型であった場合、*exp1* と *exp2* をつなげた構造型 の値を返す。

exp1 と *exp2* の最初の要素がシンボルで、*exp2* のシンボルも属性情報を持っていたと すると、*exp2* の属性情報は、*exp1* の属性情報へ上書きされる。*exp2* のシンボルの属 性情報は消える。

exp1 の最初がシンボルでない場合は、*exp2* のシンボルの属性情報が消えるのみ。

exp1 *exp2* が構造型で無かった場合は type mismatch エラーを

戻り値

リスト。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.21 List

プロトタイプ

LISP 形式

(List exp)

XML 形式

<List>exp</List>

引数

exp [1+] 任意の型

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp ... をこの順番でリストに含めた値を返す。

戻り値

リスト。

エラー

1. XLE.SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.22 mul,*

プロトタイプ

LISP 形式

(* exp exp)

XML 形式

<mul> exp exp </mul>

引数

exp [2+] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

*は XML のタグの中で使うことが出来ないなので、mul を全く同じ関数として定義している。

かけ算を引数要素の *exp* に対して施し、結果を返す。 *exp* として、取ることの出来る型は、XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節) である。

XLT_INTEGER (5.2.6 節) と XLT_FLOAT (5.2.7 節) が混在していると、まずは、XLT_INTEGER (5.2.6 節) を XLT_FLOAT (5.2.7 節) にキャストしてから演算を行う。

戻り値

積の結果

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)
- 2.

参考

バグ

8.2.23 add, +

プロトタイプ

LISP 形式

(+ exp exp)

XML 形式

<add> exp exp </add>

引数

exp [2+] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節), XLT_STRING (5.2.5 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

*,+は XML のタグの中で使うことが出来ないので、add を全く同じ関数として定義している。

足し算を引数要素の *exp* に対して施し、結果を返す。*exp* として、取ることの出来る型は、XLT_STRING (5.2.5 節), XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節) である。

XLT_INTEGER (5.2.6 節) と XLT_FLOAT (5.2.7 節) が混在していると、まずは、XLT_INTEGER (5.2.6 節) を XLT_FLOAT (5.2.7 節) にキャストしてから演算を行う。全てが String であれば、それを引数の順番 でつなぎ合わせた物を戻り値とする。

戻り値

加算結果

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.24 sub,-

プロトタイプ

LISP 形式

(- exp1 exp2)

XML 形式

_{exp1 exp2}

引数

exp1 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

exp2 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

引き算。exp1 から exp2 を引く。

戻り値

減算結果

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.25 div,/

プロトタイプ

LISP 形式

(/ exp1 exp2)

XML 形式

<div> exp1 exp2 </div>

引数

exp1 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

exp2 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

わり算。exp1 を exp2 で割る。

戻り値

除算結果

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.26 rem,%

プロトタイプ

LISP 形式

(% exp1 exp2)

XML 形式

<rem> exp1 exp2 </rem>

引数

exp1 [1] XLT_INTEGER (5.2.6 節)

exp2 [1] XLT_INTEGER (5.2.6 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

わり算の余り。exp1 を exp2 で割った余りを求める。

戻り値

剰余結果

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.27 and

プロトタイプ

LISP 形式

(and exp1 exp2)

XML 形式

<and> exp1 exp2</and>

引数

exp1, exp2 [2+] XLT_INTEGER (5.2.6 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 exp2 は整数で、これらのビットごとの論理積を取って戻り値とする。

戻り値

ビットの論理積

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.28 or

プロトタイプ

LISP 形式

(or exp1 exp2)

XML 形式

<or> exp1 exp2</or>

引数

exp1,exp2 [2+] XLT_INTEGER (5.2.6 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 exp2 は整数で、これらのビットごとの論理和を取って戻り値とする。

戻り値

ビットの論理和

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.29 xor

プロトタイプ

LISP 形式

(xor exp1 exp2)

XML 形式

<xor> exp1 exp2</xor>

引数

exp1,exp2 [2+] XLT_INTEGER (5.2.6 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 exp2 は整数で、これらのビットごとの排他的論理和を取って戻り値とする。

戻り値

ビットの排他的論理和

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.30 not

プロトタイプ

LISP 形式

(not exp)

XML 形式

<not> exp </not>

引数

exp [1] XLT_INTEGER (5.2.6 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp が整数のとき、このビットごとの not を取って戻り値とする。

戻り値

ビットの否定 (反転)

エラー

1. XLT_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.31 And

プロトタイプ

LISP 形式

(And exp1 exp2)

XML 形式

<And> exp1 exp2</And>

引数

exp1, exp2 [2+] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 exp2 は 0 のとき「偽」、0 以外るとき「真」という解釈である。この解釈で論理積をとり、結果を 0「偽」、1「真」で返す。C 言語の論理演算の考え方と同一である。

戻り値

論理積

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.32 Or

プロトタイプ

LISP 形式

(Or exp1 exp2)

XML 形式

<Or> exp1 exp2</Or>

引数

exp1, exp2 [2+] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 exp2 は 0 のとき「偽」、0 以外るとき「真」という解釈である。この解釈で論理和をとり、結果を 0「偽」、1「真」で返す。C 言語の論理演算の考え方と同一である。

戻り値

論理和

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.33 Xor

プロトタイプ

LISP 形式

(Xor exp1 exp2)

XML 形式

<Xor> exp1 exp2</Xor>

引数

exp1,exp2 [2+] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 exp2 は 0 のとき「偽」、0 以外るとき「真」という解釈である。この解釈で排他的論理和をとり、結果を 0 「偽」、1 「真」で返す。C 言語の論理演算の考え方と同一である。

戻り値

排他的論理和

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.34 Not

プロトタイプ

LISP 形式

(Not exp)

XML 形式

<Not> exp </Not>

引数

exp [1] XLT_INTEGER (5.2.6 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp は 0 のとき「偽」、0 以外るとき「真」という解釈である。この解釈で否定をとり、結果を 0 「偽」、1 「真」で返す。C 言語の論理演算の考え方と同一である。

戻り値

否定

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.35 Equ

プロトタイプ

LISP 形式

(= exp1 exp2)

(Equ exp1 exp2)

XML 形式

<Equ> exp1 exp2</Equ>

引数

exp1, exp2 [1] 任意の型

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 と *exp2* が等しいとき 1「真」、等しくないときは 0「偽」を返す。

戻り値

XLT_INTEGER (5.2.6 節)0 か 1。

エラー

参考

バグ

8.2.36 Neq

プロトタイプ

LISP 形式

(!= exp1 exp2)

(Neq exp1 exp2)

XML 形式

<Neq> exp1 exp2</Neq>

引数

exp1, exp2 [1] 任意の型

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 と *exp2* が等しくないとき 1「真」、等しくないときは 0「偽」を返す。

(Not (Equ *exp1* *exp2*)) と等価。

戻り値

XLT_INTEGER (5.2.6 節)0 か 1。

エラー

参考

バグ

8.2.37 Lt

プロトタイプ

LISP 形式

(Lt exp1 exp2)

(< exp1 exp2) ただし、< はエスケープしないとシンタックスエラーになるので、

(< exp1 exp2) と書く必要がある。

XML 形式

<Lt> exp1 exp2</Lt>

引数

exp1, exp2 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節), XLT_STRING (5.2.5 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 < exp2 のとき 1「真」、等しくないときは 0「偽」を返す。

戻り値

XLT_INTEGER (5.2.6 節) 0 か 1。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.38 Gt

プロトタイプ

LISP 形式

(Gt exp1 exp2)

(> exp1 exp2)

XML 形式

<Gt> exp1 exp2</Gt>

引数

exp1, exp2 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節), XLT_STRING (5.2.5 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 > *exp2* のとき 1「真」、等しくないときは 0「偽」を返す。

戻り値

XLT_INTEGER (5.2.6 節)0 か 1。

エラー

参考

バグ

8.2.39 Lteq

プロトタイプ

LISP 形式

(Lteq exp1 exp2)

(<= exp1 exp2) ただし、< はエスケープしないとシンタックスエラーになるので、

(<= exp1 exp2) と書く必要がある。

XML 形式

<Lteq> exp1 exp2</Lteq>

引数

exp1, exp2 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節), XLT_STRING (5.2.5 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 <= *exp2* のとき 1「真」、等しくないときは 0「偽」を返す。

戻り値

XLT_INTEGER (5.2.6 節)0 か 1。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.40 Gteq

プロトタイプ

LISP 形式

(Gteq exp1 exp2)

(>= exp1 exp2)

XML 形式

<Gteq> exp1 exp2</Gteq>

引数

exp1, exp2 [1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節), XLT_STRING (5.2.5 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp1 >= *exp2* のとき 1「真」、等しくないときは 0「偽」を返す。

戻り値

XLT_INTEGER (5.2.6 節)0 か 1。

エラー

参考

バグ

8.2.41 Remote

プロトタイプ

LISP 形式

(Remote integer-exp exp)

XML 形式

<Remote> integer-exp exp </Remote>

引数

integer-exp [1] XLT_INTEGER (5.2.6 節)

exp [1+] 任意の型

属性

評価形式

normal

所属エージェント

xl

所属環境

Env1

説明

integer-exp は OpenInterpreter (8.2.4 節) で得られた識別子。 *integer-exp* は先に評価される。識別子の示すコネクション先に、 *exp ...* を順番に送信する。 *exp* は先方のインタプリタで評価され、結果が先方のインタプリタによって <Result> (8.2.43 節) 要素で返される。その結果値を Remote の戻り値とする。

戻り値

<Result> (8.2.43 節) の第二変数。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)
2. XLE_PROTO_ACCESS_STREAM (7.2.73 節)
コネクションが戻り値を得る前に切断された。
3. XLE_PROTO_INV_ID

integer-exp で示される ID のインタプリタは存在しなかった。

参考

バグ

8.2.42 Local

プロトタイプ

LISP 形式

(Local exp)

XML 形式

<Local> exp </Local>

引数

exp [1] 任意の型

属性

評価形式

normal

所属エージェント

xl

所属環境

Env1

説明

どこからかのコネクション先でこの関数が評価されたとき、*exp* をコネクション もとのインタープリタへ送り、コネクションもとの環境で評価する。評価結果 はコネクション先へ <Result> (8.2.43 節) を使って送り返される。Result の結果値が Local の 戻り値となる。

戻り値

<Result> (8.2.43 節) の第二変数。

エラー

1. XLE.PROTO_ACCESS_STREAM (7.2.73 節)

コネクションが戻り値を得る前に切断された。

2. XLE.PROTO_INV_ID

integer-exp で示される ID のインタープリタは存在しなかった。

参考

バグ

8.2.43 Result

プロトタイプ

LISP 形式

(Result integer-exp exp)

XML 形式

<Result> integer-exp exp</Result>

引数

integer-exp [1] XLT_INTEGER (5.2.6 節)

exp [1] 任意の型

属性

評価形式

interpreter

所属エージェント

xl

所属環境

Env1

説明

標準入力、あるいは、ネットワークから入力など入力ストリームから入力された 関数の評価結果を Result で返す。integer-exp は seq 値 (行番号)、exp は評価結果。

標準入力、あるいは、ネットワークから入力など入力ストリームから入力された 関数の評価結果を Result で返す。integer-exp は seq 値、exp は評価結果、これを 結果値と呼ぶ。そして、type-exp はその型を表す。入力ストリームから 15 行目に、

```
<If> (= a b)
  <Then>    .... </Then>
  <Else>    .... </Else>
</If>
```

と送り込んだとする。もし、強制的に 15 行目に固定したい場合は、

```
<seq no="15">
```

を送ってから、上記 If を送る。

これに対応する Result の第 2 引数の integer-exp は 15 となる。exp は関数の評価結果である。たとえば、上述 If の戻り値は次のように表現される。

```
<Result>15
  (1.2 3)
</Result>
```

二つのインタプリタがつながっており、一方のストリーム出力が片方のストリーム入力となっている場合は、<Result> を受けたインタプリタはこれを評価しなければならない。<Result> を受けるインター

プリタは、おそらくその前に、<Remote> (8.2.41 節) や <Local> (8.2.42 節) を発行していると思われる。Remote や Local は seq をつけて関数を送り出して、その結果待ちをしている。Result は integer-exp (評価しない) に対応する Remote/Local を探しだし、その Remote/Local へ結果値を受け渡す。

<Result> の評価結果は常に ErrDontPrint。なぜならば、戻り値を出力すると、二つのインタプリタの間で結果が永遠にループすることになるからである。

戻り値

なし。インタプリタのフロントエンドスレッドが値を横取りする。エラーも同様にしてなし。

エラー

参考

バグ

8.2.44 Load

プロトタイプ

LISP 形式

(Load string-exp)

XML 形式

<Load> string-exp </Load>

引数

string-exp [1] XLT_STRING (5.2.5 節)

属性

option

format.mode どちらか必ず一つ必要 同じ意味の属性 文字列型 raw/xml/html/text/text.escape/icalendar/exec

format.text 文字列型 on/off

encoding 任意 文字列 テキストのエンコーディング default=システムのデフォルト

terminate 任意 文字列 *format.mode*=”text.escape”を選択した場合の分離文字列

評価形式

applicative

所属エージェント

xl

所属環境

Env1

説明

string-exp で指定されたファイル名のファイルを開き、読み込み、その内容を S 式で返す。そのファイルの中身のフォーマットを各種属性で指定する。

option または *format.mode*

この属性はファイルのフォーマットを指定する。属性名はどちらを使っても良い。省略することは出来ない。

- raw

ファイルを単なるバイト列と解釈し、4096 バイトの XLT_RAW (5.2.11 節) 型のデータの連続リストとして返す。

- xl

XL (XML) フォーマットとして読み込む。

- html

ファイルを HTML ファイルと解釈する。

- text

テキストデータと解釈する。ただし—続きの XLT_STRING (5.2.5 節) として読み込む。

- text.escape

テキストデータと解釈する。ただし、属性 *terminate* で指定された文字列を認識するとそこで、文字列を分割し、文字列のリストを返す。

- icalendar

iCalendar のフォーマットとして読み込みます。読み込んだフォーマットは S 式に変換され、戻り値として返される。

- **exec**

XL の実行形式として読み込み、実行する。

format.text

XL は、文字列を数字は XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節) というように各型に分離して読み込み、S 式に変換する。format.text="on" にすると、これを XML のタグ以外はすべて文字列として解釈するようにする。format.text="off" では、もとの XL の解釈で文字列をパースする。

encoding

format.mode="raw" でない場合、ファイルのテキストエンコードの指定。

戻り値

エラーでない場合。ファイルの内容。format.mode="exec" の場合、読み込んだ内容を実行し、その結果を返す。

エラー

1. XLE.SEMANTICS_TYPE_MISMATCH (7.2.52 節)
2. XLE.PROTO_INV_PARAM (7.2.58 節)
3. XLE.PROTO_OPEN_FILE (7.2.61 節)
4. XLE.SYSTEM_READ_FILE (7.2.29 節)
5. XLE.SYNTAX_TOKEN_ERROR (7.2.43 節)
6. XLE.SYNTAX_INVALID_ENTITY (7.2.46 節)
7. XLE.SYNTAX_UNSUTISFIED_FILE_END (7.2.42 節)
8. XLE.SYSTEM_EXIT (7.2.32 節)
9. XLE.SYNTAX_ERROR_CODE (7.2.41 節)
10. XLE.SYNTAX_NO_PUNC (7.2.40 節)
11. XLE.SYSTEM_INTERRUPT (7.2.34 節)
12. XLE.SYNTAX_TOO_MANY_PUNC (7.2.39 節)
13. XLE.SYNTAX_NOT_CORRESPOND_TAG (7.2.44 節)

参考

バグ

8.2.45 Save

プロトタイプ

LISP 形式

(Save string-exp exp)

XML 形式

<save option="xl/raw"> string-exp verb+|+| exp </Save>

引数

string-exp [1] XLT.STRING (5.2.5 節)

exp [1] 任意の型

属性

option 必須 文字列 raw/xl/csv

oflag 必須 オープン時のフラグ creat:trunc:append の組み合わせ

mode 任意 ファイルを生成するときのファイルモードの指定。C ライブラリ open の mode に同じ。

format.mode 任意 文字列 lisp/xl/html

format.indent 任意 文字列 on/off default="off"

format.multiroot 任意 文字列 on/off default="on"

format.text 任意 文字列 on/off default="off"

encoding 任意 文字列 出力文字エンコード default はシステムのコード。

評価形式

applicative

所属エージェント

xl

所属環境

Env1

説明

string-exp で示されたファイルに、exp の内容をセーブする。option="xl"を指定すると、exp をそのままの形でセーブする。option="raw"を指定すると、exp は raw データのリストと解釈され、最初の raw データから、ファイルに書き出されていく。option="csv"である場合は、exp のリストを CSV 形式で出力する。CSV 形式の場合、以下に挙げるように exp はそれに合ったフォーマットでなければならない。

```
( (element1-1 element1-2 ....)
  (element2-1 element2-2 ....)
  ....)
```

elementX-Y は XLT.INTEGER (5.2.6 節), XLT.FLOAT (5.2.7 節), XLT.STRING (5.2.5 節) のいずれかでなければならない。elementX-Y はそのまま対応する CSV の行列に出力される。

format.mode

option="xl"の場合、exp を lisp 形式、xl (XML) 形式、HTML 形式に変換し出力する。

format.indent

option="xl"の場合、format.indent="on"で出力にインデントをつける。

format.multiroot

option="xl"の場合、マルチルートにするかどうか。マルチルートとは、一つのファイルの中に複数のルート要素を認める書き方である。しかし、通常 XML ファイルは、本文の要素以外に <?xml ...?> という要素

も入るので、マルチルートに指定するのが一般的である。マルチルートの場合、複数のルート要素をリストとして与える。

`format.text`

`option="xl"` かつ、`format.mode="XML"`、`format.mode="HTML"` に指定されている場合、要素として解釈出来ないリストの (や) を省略する。たとえば、`option="xl" format.mode="html" format.text="on"` とすると、

```
("今日は" ("とても" ([A HREF="http://www.globalbase.org/"] "暖かい日")) "です")
```

というリストが合った場合、

```
今日はとても <A HREF="http://www.globalbase.org/">暖かい日</A>です
```

と出力される。 `format.text="off"` では、

```
今日は ("とても" <A HREF="http://www.globalbase.org/">暖かい日</A>) です
```

と出力される。

戻り値

XLT_NULL (5.2.1 節)

エラー

1. XLE_PROTO_INV_PARAM (7.2.58 節)
2. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)
3. XLE_PROTO_INV_PARAM (7.2.58 節)

参考

バグ

8.2.46 Parse

プロトタイプ

LISP 形式

(Parse string-exp)]] > XML 形式

<Parse> string-exp </Parse>]] >

引数

string-exp [1] XLT.STRING (5.2.5 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

string-exp で与えられた文字列を S 式に置き換える。

戻り値

S 式

エラー

基本的にパーシングを行うときに発生するエラー。

1. XLE.SEMANTICS_TYPE_MISMATCH (7.2.52 節)
2. XLE.SYSTEM_READ_FILE (7.2.29 節)
3. XLE.SYNTAX_TOKEN_ERROR (7.2.43 節)
4. XLE.SYNTAX_INVALID_ENTITY (7.2.46 節)
5. XLE.SYNTAX_UNSUTISFIED_FILE_END (7.2.42 節)
6. XLE.SYSTEM_EXIT (7.2.32 節)
7. XLE.SYNTAX_ERROR_CODE (7.2.41 節)
8. XLE.SYNTAX_NO_PUNC (7.2.40 節)
9. XLE.SYSTEM_INTERRUPT (7.2.34 節)
10. XLE.SYNTAX_TOO_MANY_PUNC (7.2.39 節)
11. XLE.SYNTAX_NOT_CORRESPOND_TAG (7.2.44 節)

参考

バグ

8.2.47 Print

プロトタイプ

LISP 形式

(Print exp)

XML 形式

<Print> exp </Print>

引数

string-exp [1] 任意の型

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp で与えられる S 式を文字列に変換する。

戻り値

S 式

エラー

参考

バグ

8.2.48 CutString

プロトタイプ

LISP 形式

(CutString string-exp integer-exp1 integer-exp2)

XML 形式

<CutString> string-exp </CutString>

引数

string-exp [1] XLT_STRING (5.2.5 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

string-exp で与えられる文字列の *integer-exp1* 文字目から *integer-exp2* 文字を切り出す。ただし、文字列の先頭の文字は 0 文字目と数える。*integer-exp1* + *integer-exp2* が文字列の長さより長い場合は、存在している文字数のところまでの文字列を返す。

戻り値

XLT_STRING (5.2.5 節)

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.49 StringLength

プロトタイプ

LISP 形式

(StringLength string-exp)

XML 形式

<StringLength> string-exp </StringLength>

引数

string-exp [1] XLT_STRING (5.2.5 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

string-exp で与えられた文字列の長さを返す。

戻り値

XLT_INTEGER (5.2.6 節)

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.50 SetChar

プロトタイプ

LISP 形式

(SetChar integer-exp)

XML 形式

<SetChar> integer-exp </SetChar>

引数

integer-exp [1] XLT_INTEGER (5.2.6 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

integer-exp で与えられた文字コードの文字一字からなる文字列を返す。

戻り値

XLT_STRING (5.2.5 節)

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.51 GetChar

プロトタイプ

LISP 形式

(GetChar string-exp integer-exp)

XML 形式

<GetChar> string-exp integer-exp </GetChar>

引数

string-exp [1] XLT_STRING (5.2.5 節)

integer-exp [1] XLT_INTEGER (5.2.6 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

string-exp で与えられた文字列の *integer-exp* 番目の内部文字コードを返す。

戻り値

XLT_INTEGER (5.2.6 節)

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)
2. XLE_PROTO_INV_PARAM (7.2.58 節)

integer-exp が文字列の長さの範囲外。

参考

バグ

8.2.52 Let

プロトタイプ

XML 形式

```
<Let>
<Env> environment... </Env>...
<Sub> substitution.... </Sub>...
</Let>
```

引数

```
<Env> environemnt... </Env> [0+] 多重継承用環境のリスト
environemnt [1+] XLT_ENV (5.2.12 節)
<Sub> substitution.... </Sub> [0+] ローカル変数リスト
substitution [1+] XLT_PAIR (5.2.3 節) 変数名をシンボルとし、バインドする値を内部要素とする要素。
```

属性

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

環境 (5.2.12 節) を一つ生成します。そのとき、Env 要素を指定しない場合は、その直前のカレント環境を親環境とします。もし、Env 要素を指定した場合、その中で定義される環境をカレント環境のペア環境を生成しこれを親とした子環境生成します。この子環境の中で、Sub で定義したシンボルにデータをバインドします。Env,Sub 量要素とは異なる内部要素を発見すると、その時点から、先の要素を評価して行きます。最後に評価した値を戻り値とします。書式例としては以下の通り。

```
<Let>
<Env> ^TargetEnv ^SomeEnv</Env>
<Sub>
<t> (+ 1 1)</t>
<u> (* 2 3)</u>
</Sub>
(SomeFunction t u)
</Let>
```

Env で実現する子環境は、図 8.1 に示す通りである。

また、変数 t,u には、それぞれ、2,6 がバインドされる。SomeFunction にはそれらの値が渡される。SomeFunction は、先に示した、図 8.1 における next-environment を起点として評価され、その戻り値が、Let の戻り値ともなる。

戻り値

最後に評価した式の戻り値

エラー

1. XLE_SEMANTICS_INV_FORMAT (7.2.49 節)

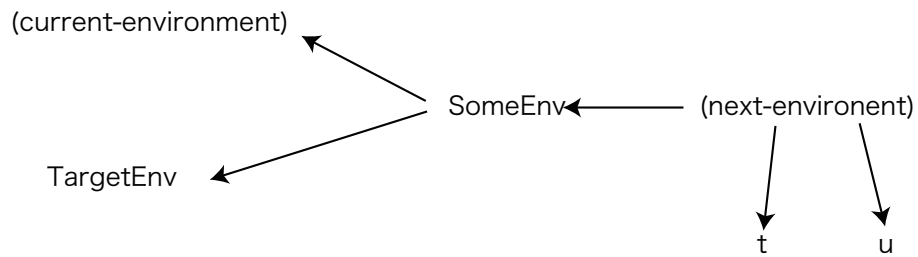


図 8.1: 環境の継承関係

(a) "symbol is required"

最初にあたった リスト (5.2.3 節) の先頭が シンボル (5.2.4 節) でなかった。

(b) "environment type is required"

Env 要素の内部要素が、XLT_ENV (5.2.12 節) でなかった。

(c) "substitution pair is required"

"symbol is required"

Sub 要素の中身が、シンボル名から始まる要素で無かった。

参考

バグ

8.2.53 SetListbaseEnv

プロトタイプ

XML 形式

```
<SetListbaseEnv>
```

```
envXML environment </SetListbaseEnv>
```

```
<SetListbaseEnv>
```

```
envXML </SetListbaseEnv>
```

引数

envXML [1] XLT_PAIR (5.2.3 節) 環境情報を表す XML

environment [0-1] XLT_ENV (5.2.12 節) 環境情報 *envXML* をセットする環境

属性

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

環境 *environment* に環境情報 *envXML* をセットする。 *envXML* の構造に関しては、6 節を参照されたい。 *environment* が省略された場合は、新しい環境が生成され、そこにセットアップされる。

戻り値

セットアップされた XLT_ENV (5.2.12 節)。 *environment* が指定されている場合は、 *environment* 。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.49 節)

envXML,environment の型が異なる。

2. XLE_SEMANTICS_INV_FORMAT (7.2.49 節)

environment のフォーマットがあていない。

参考

8.2.54 節, 6 節

バグ

8.2.54 GetListbaseEnv

プロトタイプ

XML 形式

```
<GetListbaseEnv>
```

```
environment </GetListbaseEnv>
```

引数

environment [1] XLT_ENV (5.2.12 節) 対象となる環境

属性

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

環境 *environment* より、これをリスト化した環境情報としてとりだす。環境情報の構造は、6 節を参照。

戻り値

6 節

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.49 節)

参考

8.2.53 節, 6 節

バグ

8.2.55 ?xml,?xl

プロトタイプ

XML 形式

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<?xl version="1.0" encoding="UTF-8" ?>
```

引数

属性

version [必須] XLT_STRING (5.2.5 節)XML または XL のバージョン *encoding* [必須] XLT_STRING (5.2.5 節) ファイルおよびストリームのエンコーディング

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

XML または XL ファイルのヘッダ部分につける要素。XL の関数として解釈可能である。Load (8.2.44 節) などの関数においては、このヘッダを読み、以降のコーディングをあわせる。

戻り値

6 節

エラー

参考

バグ

8.2.56 Arg

プロトタイプ

XML 形式

```
<Arg>
```

```
arg-no </Arg>
```

```
<Arg/>
```

ただし、引数なしは ver.B.b15 以降のサポート

LISP 形式

```
( Arg arg-no )
```

```
( Arg )
```

ただし、引数なしは ver.B.b15 以降のサポート

引数

arg-no [1] XLT_INTEGER (5.2.6 節) エージェント引数の番号

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

エージェントが呼び出されるときに、渡された引数を参照する関数。 *arg-no* がその引数の番号となる。ver.B.b15 以降のインプリメントでは、 *arg-no* が省略されると、引数の数を返す。以前のバージョンでは、エラーとなる。エージェントは POSIX 系のアーキテクチャの場合プロセスと等価である。エージェントに渡される引数は POSIX 系プロセスとしては以下の通りである。

(1) agent-name script-file [stdout [errout]] / [arg1 arg2 arg3]

(2) agent-name [+.....] script-file [- [-]] / [arg1 arg2 arg3]

なお、[] は省略可能であることを示す。(1) のパターンでは、agent-name が起動し XL で記述された script-file を実行する。stdout,errout はそれぞれプリント出力およびエラー出力のファイル名を与える。"-”を指定するとそのままそれぞれ、POSIX 系ファイルの標準出力、エラー出力へ送り出す。"/”の後にはエージェントに手渡す引数を並べる。当該関数は、これらの引数を参照するための物である。

(2) の書式は最初にシステム情報を載せたパターンである。xslv [?] エージェントがなどがエージェントを起動するさいにつけるデータである。

POSIX 系でなくとも、類似のインタフェースが用意されている。

戻り値

XLT_STRING (5.2.5 節)

エラー

1. XLE.SEMANTICS_TYPE_MISMATCH (7.2.49 節)

2. XLE.SYSTEM_INTERNAL (7.2.35 節)

参考

バグ

8.2.57 For

プロトタイプ

XML 形式

```
<For> condition exp.... </For>
```

LISP 形式

```
( For condition exp... )
```

引数

condition [0-1] XLT_INTEGER (5.2.6 節), XLT_FLOAT (5.2.7 節) 終了条件文

exp [0+] 任意の型 実行文

属性

dec [任意] 文字列 カウントダウン変数。

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

属性 *dec* が存在するときは、*condition* は存在しない。一方、*dec* が存在しないときは、*condition* は存在し有効である。

dec が存在しないとき

condition が有効となる。For 文はまず *condition* を評価する。この時点の評価前を ループ初期時点 と呼ぶ。その結果が、0 の場合、そのまま終了する。結果が 0 以外である場合、*exp* を順番に評価し、*condition* の評価へもどる。

exp の評価が終了した時点 を ループ終了時点 と呼ぶ。

dec が存在するとき

condition は存在しない。For 文は、*dec* 属性に示される変数を評価する。この時点の評価前を ループ初期時点 と呼ぶ。結果が、0 または、XLT_NULL (5.2.1 節) の場合、For 文は終了する。それ以外の場合、*exp* を順番に評価実行し、最後に、*dec* の値を以下のように変化させる。

exp の評価が終了した時点 を ループ終了時点 と呼ぶ。

- *dec* の変数が XLT_INTEGER (5.2.6 節) または、XLT_FLOAT (5.2.7 節) である場合、1 減らした値を、*dec* へセットしなおす。
- *dec* の変数が XLT_PAIR (5.2.3 節) である場合、(cdr (8.2.17 節)*dec*) の値を *dec* にセットしなおす。

その後、For 初期時点へ戻る。

For 文に関連した関数として Break 文 (8.2.58 節) および、Continue 文 (8.2.59 節) がある。Break 文は *exp* およびその内部で呼ばれたときに、For 文を即終了する。引数として終了時点の戻り値を与える関数である。Continue 文は、やはり *exp* およびその内部で使用可能であり、評価時点で、For 文のループ終了時点へ飛ぶ。

戻り値

exp が全く評価されなかった場合、0 (XLT_INTEGER (5.2.6 節)) が返る。 *exp* が一つ以上評価された場合、最後に評価された *exp* の値が返る。

Break (8.2.58 節) が実行された場合、その第一引数が戻り値となる。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.49 節)

condition の型違反。 *dec* であらわされる変数の型違反。

2. その他、 *dec* が評価される時点で発生するエラー。 Eval (8.2.5 節) が発生する可能性のあるエラーと同等のエラーである。

参考

8.2.58 節, 8.2.59 節

バグ

8.2.58 Break

プロトタイプ

XML 形式

<Break> *exp* </Break>

LISP 形式

(Break *exp*)

引数

exp [0-1] 任意の型 ループ文戻り値

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

For 文 (8.2.57 節) の内部の評価関数の一つとして使い、この Break 文が評価された時点で For 文を強制的に終了させる。終了時点の For 文の戻り値を、Break の *exp* で指定する。*exp* を指定しない場合は、XLT_NULL (5.2.1 節) が For 文の戻り値となる。

戻り値

XLE_SYSTEM_LOOP_BREAK エラーのみ。For 文で補足され、当該エラーの data を For 文の戻り値として返す。

エラー

1. XLE_SYSTEM_LOOP_BREAK (7.2.36 節)

もし、これが、ユーザの認識可能な状態のエラーとして返った場合。For 文の外部で Break 文を使っている可能性がある。

参考

8.2.57 節, 8.2.59 節

バグ

8.2.59 Continue

プロトタイプ

XML 形式

<Continue/>

LISP 形式

(Continue)

引数

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

For 文 (8.2.57 節) の内部の評価関数の一つとして使い、この Continue 文が評価された時点で For 文のループ終了時点へ移動する。ループ終了時点については 8.2.57 節を参照のこと。

戻り値

XLE_SYSTEM_LOOP_CONTINUE エラーのみ。For 文で補足され、ループ終了時点への移動処理が行われる。

エラー

1. XLE_SYSTEM_LOOP_CONTINUE (7.2.37 節)

もし、これが、ユーザの認識可能な状態のエラーとして返った場合。For 文の外部で Continue 文を使っている可能性がある。

参考

8.2.57 節, 8.2.58 節

バグ

8.2.60 Cast

プロトタイプ

XML 形式

<Cast> *exp* </Cast>

LISP 形式

(Cast *exp*)

引数

exp [1] 任意の型 キャストする対象のデータ

属性

type [必須] 文字列 型キャストする型

option [任意] 文字列 キャスト方法のオプション

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

exp の型を、属性 *type* で示される型に変換します。 *option* でキャスト方法のオプションを与えることが出来ます。

type の属性値として以下のものがとれます。

- "null"

XLT_NULL (5.2.1 節) 型

変換元

- XLT_NULL (5.2.1 節)
そのままの値を返す。
- それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

- "error"

XLT_ERROR (5.2.2 節) 型

そのままの値 (エラー) を返す。

- "list"

XLT_PAIR (5.2.1 節) 型

変換元

- XLT_NULL (5.2.1 節)
そのままの値を返す。
- XLT_PAIR (5.2.3 節)
そのままの値を返す。
- それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

- "symbol"

XLT_SYMBOL (5.2.4 節) 型

変換元

- XLT_SYMBOL (5.2.5 節)
そのままの値を返す。
option の値
 - * small
すべて小文字のシンボルに変換
 - * capital
すべて大文字のシンボルに変換
- XLT_STRING (5.2.5 節)
文字列をシンボルに変換する。
- それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

- "string"

XLT_STRING (5.2.5 節) 型

変換元

- XLT_SYMBOL (5.2.5 節)
シンボルを文字列に変換する。
- XLT_STRING (5.2.5 節)
option の値
 - * small
すべて小文字のシンボルに変換
 - * capital
すべて大文字のシンボルに変換
- XLT_INTEGER (5.2.5 節)
整数を文字列に変換する。
option の値
 - * char
整数を L_CHAR 文字コードと解釈し、この文字コード 1 文字からなる文字列に変換する。
- XLT_FLOAT (5.2.5 節)
浮動小数点型を文字列に変換する。
- それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

- "integer"

XLT_INTEGER (5.2.6 節) 型

変換元

- XLT_STRING (5.2.5 節)
option の値

* zeroIgnore

文字列冒頭に 0 がある場合、そのゼロを削除した物を変換する。連続する 0 も同様削除する。

– XLT_INTEGER (5.2.5 節)

そのままの値を返す。

– XLT_FLOAT (5.2.5 節)

浮動小数点型を整数に変換する。小数点以下は切り捨てる。

– それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

● ”floating”

XLT_FLOAT (5.2.7 節) 型

変換元

– XLT_STRING (5.2.5 節)

option の値

* zeroIgnore

文字列冒頭に 0 がある場合、そのゼロを削除した物を変換する。連続する 0 も同様削除する。

– XLT_INTEGER (5.2.5 節)

整数を浮動小数点に変換する。

– XLT_FLOAT (5.2.5 節)

そのままの値を返す。

– それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

● ”function”

XLT_FUNC (5.2.8 節) 型

変換元

– XLT_FUNC (5.2.8 節)

そのままの値を返す。

– それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

● ”delay”

XLT_DELAY (5.2.9 節) 型

XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

● ”pointer”

XLT_PTR (5.2.10 節) 型

変換元

– XLT_PTR (5.2.10 節)

そのままの値を返す。

– それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

● ”raw”

XLT_RAW (5.2.11 節) 型

変換元

- XLT_RAW (5.2.11 節)
そのままの値を返す。
- それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

- ”environment”

XLT_ENV (5.2.12 節) 型

変換元

- XLT_ENV (5.2.12 節)
そのままの値を返す。
- それ以外は XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

戻り値

変換結果が返る。

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

type 属性と実際に与えられた値 *exp* が合っていない。

2. XLE_PROTO_INV_PARAM (7.2.58 節)

type 属性に与えられた文字列が正しくない。

参考

バグ

8.2.61 DefineEval

プロトタイプ

XML 形式

```
<DefineEval> env symbol data </DefineEval>
```

```
<DE> env symbol data </DE>
```

LISP 形式

```
( DefineEval Env symbol data )
```

```
( DE Env symbol data )
```

引数

env [1] XLT_ENV (5.2.12 節), XLT_NULL (5.2.1 節) データが登録環境

symbol [1] XLT_SYMBOL (5.2.4 節) 登録データのシンボル

data [1] 任意の型 代用値

属性

評価形式

normal

所属エージェント

xl

所属環境

Env0

説明

DefineEval は、まず、*env* 上で *symbol* を評価する。この評価がエラーとなった場合、*data* の評価値を返す。もし、*symbol* の評価がエラーとはならなかった場合は、この評価結果をさらに評価した値を返す。

env が XLT_NULL の場合は、カレント環境に強制的に *data* をバインドする。そのときに、*data* は評価しない。

戻り値

symbol の評価結果の評価、または、*data* の評価結果

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

参考

バグ

8.2.62 Categorize

プロトタイプ

XML 形式

```
<Categorize> cmpfunc target </Categorize>
```

LISP 形式

```
( Categorize cmpfunc target )
```

引数

cmpfunc [1] XLT_FUNC (5.2.8 節) 比較用関数

target [1] XLT_PAIR (5.2.3 節) 分類対象データ

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

target で与えられるリストを頭からスキャンし、隣同士の要素を *cmpfunc* によって比較する。一致すると同じカテゴリと判断し、一つのリストへまとめる。カテゴリごとに分類されたリストのリストを返す。たとえば、

```
( Categorize = ' ( 11133345661
```

といったリストを与えると、比較関数 '=' によって、次のような結果を返す。

```
( ( 11( 33( ( ( 6( 1
```

ここで、Categorize はデータのソートをしないので、最初の三つの 1 と最後の二つの 1 は、別のカテゴリとしてリストかされることに注意が必要である。同一のカテゴリにまとめようとする場合は、Sort [UNDEF REF (xl-Sort)] などによってあらかじめソートしておく必要がある。

cmpfunc は *target* にリストされる要素の型と同一な引数を二つとる関数であり、戻り値は整数である必要がある。戻り値が 0 である場合、Categorize は二つの要素がことなっていると判断する。その他の場合、二つの要素を同一と判断し、同じリストにまとめる。

戻り値

二重リスト (5.2.3 節)

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

(a) "type mismatch"

Categorize 関数の引数 *cmpfunc*, *target* の型が異なっている。

(b) "type mismatch the return of cmp function"

cmpfunc を実行したところ、その戻り値が整数でない。

2. その他 *cmpfunc* に指定した関数によって発生するエラーもある。

参考

バグ

8.2.63 Chdir

プロトタイプ

XML 形式

<Chdir> *dir* </Chdir>

LISP 形式

(Chdir *dir*)

引数

dir [1] XLT_STRING (5.2.5 節) ディレクトリパス

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env1

説明

カレントディレクトリを与えられたディレクトリパスに変更する。

戻り値

XLT_NULL (5.2.1 節)

エラー

参考

バグ

第9章 XL 認証関数

9.1 概要

ver.B.b16以降においてLANDSCAPEサーバにおいて認証機能を実装した。そのための関数をここで解説する。LANDSCAPEサーバにおいて、xldocs配下の各ディレクトリにあるコンテンツを認証によって保護作成するためには、保護したいディレクトリに、「.xlaccess」という不可視ファイルをおき、ここに認証命令を設定することによって実現する。いわばapacheの.htaccessのようなファイルである。その一例を以下に挙げる。

```
<?xl version="0.1" encoding="UTF-8"?>
<authentication>
  <service
    name="access-admin"
    type="fix-user-passwd"
    user="Etsuransha"
    passwd="etsu1523"
    description="Aero Photograph"
    description-url="http://www.globalbase.org/"
  />
  <service
    name="root-admin"
    type="localhost-root"
  />

  <target prefix=".mtx" operation="Get">
    access-admin
  </target>
  <target prefix=".mtx" operation="Get">
    root-admin
  </target>

  <target prefix=".lst" operation="Get">
    access-admin
  </target>
  <target prefix=".lst" operation="Get">
    root-admin
  </target>

</authentication>
```

基本的には、authentication 環境の中において、service により認証の単位となる「COSMOS リファレンス・マニュアル」[2]の「サービス(認証)」を指定する。そのサービス名を、各アクセス状況に対応させる(9.2.3節)。このコンテンツをアクセスした場合、どのようにCOSMOS上に表示されるかは、「COSMOS リファレンス・マニュアル」[2]の「認証ダイアログウィンドウ」を参照されたい。

9.2 関数

9.2.1 authentication

プロトタイプ

LISP 形式

(authentication exp)

XML 形式

<authentication> exp </authentication>

引数

exp [1] すべての型

属性

評価形式

environment

所属エージェント

xl

所属環境

Env0

説明

authentication 環境。この章で説明する関数が定義されている。

戻り値

エラー

なし。

参考

バグ

9.2.2 service

プロトタイプ

LISP 形式

(service)

XML 形式

<service/>

引数

なし。

属性

- 基本属性

type [必須] 文字列 サービス・タイプ

name [必須] 文字列 サービス名

description [任意] 文字列 サービスの説明またはタイトル。

description-url [任意] 文字列 サービスの詳しい説明への URL (HTTP)

- **fix-user-passwd**

user [必須] 文字列 ユーザ名

passwd [必須] 文字列 パスワード

- **localhost-root**

特になし。

評価形式

applicative

所属エージェント

xl

所属環境

authentication

説明

サービスを定義する。まず *name* および、サービスタイプ *type* を指定する。サービス名はサーバ上で一意である必要がある。複数存在するとクライアントが混乱する可能性がある。ここで指定するサービス名にサーバのドメイン名を足した URI 形式、SERVICE:// [サーバ名]:0/ [name] であらわされる文字列が実際のサービス名となる。[サーバ名]が入っているのでクライアント側でサービス名が重複することはない。

type に指定出来る値としては、ver.B.b16 時点では、

- fix-user-passwd

- localhost-root

の二種類がある。fix-user-passwd は、固定したユーザ名とパスワードを指定するタイプの認証方式である。したがって、*user* と *passwd* という新たな二つの属性を必要とする。localhost-root はクライアントがサーバと同じサーバ上にあり、しかも root パーミッションで動いている場合に認証される。localhost-root の場合は新たな属性は必要ない。

戻り値

XLT_NULL (5.2.1 節)

エラー

1. XLE_PROTO_INV_PARAM (7.2.58 節)

指定すべき属性が指定されていない。

参考
バグ

9.2.3 target

プロトタイプ

LISP 形式

(target str ...)

XML 形式

```
<target> str... </target>
```

引数

str [1+] XLT.STRING (5.2.5 節)

属性

prefix [必須] 文字列 対象ファイルの拡張子

operation [必須] 文字列 対象アクセス命令 Get,Set

評価形式

applicative

所属エージェント

xl

所属環境

authentication

説明

`target` は実際にサービスを施すコンテンツを指定する。`.xlaccess` の置かれているディレクトリまたは、より下のディレクトリにあるコンテンツであり、かつ、`prefix` 拡張子をもっているコンテンツが対象となる。このコンテンツに対して、`operation` で指定されるアクセスが来た場合に引数 `str` で指定されたサービスを適用する。

サービスを複数指定すると、クライアントは対象となるコンテンツにアクセスするためには、それらのサービスにより提供される認証を全部満たさなければならない。AND 条件である。一方、同じ、`prefix, operation` を指定した `target` ほ複数回ならべると、それらが OR 条件となり、どれか一つのサービスの認証を満たしていればコンテンツにアクセス出来ることになる。

たとえば以下の例であれば、

```
<?xl version="0.1" encoding="UTF-8"?>
<authentication>
  <service
    name="access-admin"
    type="fix-user-passwd"
    user="Etsuransha"
    passwd="etsu1523"
    description="Aero Photograph"
    description-url="http://www.globalbase.org/"
  />
  <service
    name="root-admin"
    type="localhost-root"
  />

  <target prefix=".mtx" operation="Get">
    access-admin
```

```

</target>
<target prefix=".mtx" operation="Get">
    root-admin
</target>

<target prefix=".lst" operation="Get">
    access-admin root-admin
</target>

</authentication>

```

.mtx 拡張子のファイルにアクセスする場合には、access-admin サービスまたは、root-admin サービスのどちらかの認証に成功すればアクセス可能となる。一方、.lst 拡張子のファイルは、access-admin と root-admin の両方のサービスの認証をクリアしなければならない。

LANDSCAPE サーバはシステムのトラックバック機能により、コンテンツ相互に参照を行うことにより検索インデックス等を作成している。少なくとも一つのサーバ内では相互のアクセスを可能とするために、localhost-root 認証を各条件に OR 条件に入れることをお進めする。

戻り値

XLT_NULL (5.2.1 節)

エラー

1. XLE.PROTO_INV_OBJECT (7.2.56 節)

undefined service name 指定されたサービスが定義されていない。

2. XLE.SYSTEM_INTERNAL (7.2.35 節)

authentication target internal error (AUTH_DIR) 予期せぬシステム上のエラー。 .xlaccess 以外のファイルに target を記述したりすると発生する。

3. XLE.SEMANTICS_TYPE_MISMATCH (7.2.52 節)

str 等の型が一致していない。

参考

バグ

9.2.4 auth-transaction

プロトタイプ

LISP 形式

```
( auth-transaction code path service-list )
```

XML 形式

```
<auth-transaction> code path service-list </auth-transaction>
```

引数

code [1] XLT_INTEGER (5.2.6 節)

path [1] XLT_STRING (5.2.5 節)

service-list [1] XLT_PAIR (5.2.3 節)

属性

評価形式

applicative

所属エージェント

xl

所属環境

Env0

説明

クライアント側の認証を受け付ける関数。特にこの関数をユーザまたは管理者が直接利用することはない。*service-list* に認証対象となるサービスをリストする。これを受けたクライアントは手持ちのサービスリストと比較し認証の準備が出来ているものについて、認証情報を付加し、戻り値とする。

path は認証対象となったコンテンツへのパスである。*code* は現在 1 に固定。

戻り値

XLT_PAIR (5.2.3 節) *service-list*

エラー

なし。

参考

バグ

第10章 XLストリーム・ルーティング関数

10.1 概要

ver.B.b17.03以降において、ストリーム・ルーティングがサポートされました。ストリーム・ルーティングとは、一種のプロキシであって、ファイヤーウォールやフィルター等のために、外部のサーバへのアクセス制限がある場合に利用します。特に、LANDSCAPEサーバがあるエリアから外部への接続が制限されている場合、サーバ同士のコミュニケーションが制限されるため、正常なデータの公開が出来なくなります。そのような場合、すべての外部への接続をある、外部接続を許可された一つのLANDSCAPEサーバ経由で外部へ送り出すように設定する必要があります。このような場合にストリーム・ルーティングを使います。

すべてのストリームはクライアント（参照元）から `connection` 命令によるアクティブ・オープンによって、どこかのサーバで既にパッシブ・オープンされているポート（参照先）へ接続されます。しかし参照元から直接参照先へ接続できないとき、途中の中継サーバをたよって接続するように設定します。これがストリーム・ルーティングです。

ストリーム・ルーティングを実現するためには、参照元が直接参照先へ接続せず、中継サーバ（proxy）へ接続するよう促す必要があるため、参照元自体にストリーム・ルーティングの設定を行う必要があります。また、中継サーバにもストリーム・ルーティングを設定することにより、最終的に参照先へストリームを導きます。

参照元、および中継サーバにおける、ストリーム・ルーティングの設定は、以下の2種類あります。

1. 中継サーバに接続してくるストリームの一つ前のノードのアドレスと参照先のペアを条件として、このストリームをどこへ接続するかを決めます。
2. アクティブ・オープンされたストリームの参照先を条件として、このストリームをどこへ接続するかを決めます。

2番目の条件は、`me`（自分）から接続されてくるストリームと考えると、記述上は1番目の条件と同一に扱うことができます。

一方どこへ接続するかのパターンは以下の通りです。

1. `direct` 参照先へ直接接続します。
2. `reject` 接続を拒絶します。
3. `proxy` 中継サーバへ接続します。`gateway` 属性によって中継サーバを指定します。

10.2 関数

10.2.1 StreamRouting

プロトタイプ

LISP 形式

```
( [ StreamRouting src="" dest="" type="" ]  
( [ StreamRouting src="" dest="" type="" gateway="" ])
```

XML 形式

```
<StreamRouting/>
```

引数

属性

src [必須] 文字列

アドレス書式に基づく、一つ前のノードのアドレス

dest [必須] 文字列

アドレス書式に基づく、参照先アドレス

type [必須] 文字列

アクション・タイプ direct, proxy, reject

gateway [任意] 文字列

中継タイプが proxy のときに、中継サーバをアドレス書式に基づいて指定する。

評価形式

Applicative

所属エージェント

xl

所属環境

Env1

説明

ストリーム・ルーティングの一つのテーブルを指定する。指定されたテーブルは、ストリーム・ルーティング・テーブルの末尾に追加される。

アドレス書式

アドレス書式は以下の形式をとる。

```
addr-format ::=  
    'me'  
    | 'all'  
    | addr ':' port ':' id  
    | addr ':' port  
    | addr  
    ;  
addr ::= domain-name  
    | ip-addr  
    | ip-addr '/' NUM  
    ;  
ip-addr ::=  
    NUM '.' NUM '.' NUM '.' NUM  
    ;
```

```

domain-name ::=
    STRING
    ;
port ::= NUM
    |
    ;
id ::= NUM
    |
    ;

```

addr-format には、me, all といった文字列を指定することが可能であり、me を指定した場合は、自己に割り振られた、アドレスを示し、また、localhost, 127.0.0.1 といった自己アドレスを指し示す。一方、all を指定する場合、すべてのアドレスを指し示すことが出来る。addr は ip アドレスまたは、ドメイン名を取ることができ、ip アドレスを指定する場合は、/を入れて、ネットマスクビット数を指定することも可能である。

port はポート番号である。id は現在は使用していない。省略または、id=0 とする。
例えば以下のような書式となる。

```

160.193.15.2:8080:
160.193.15.0/24:8080:0
gbs.ur-plaza.osaka-cu.ac.jp:80:

```

src

アクティブオープン、または、中継サーバに対して、中継エージェント xlsr を指定してのストリームの接続があった場合、このストリームの 一つ前のノードのアドレスと、参照先アドレスを抽出し、これと、src,dest を比較する。ストリーム・ルーティング・テーブルを頭からチェックしていき、初めてストリームの条件にあったテーブルが選ばれる。src は一つ前のノードのアドレスの条件を与える。一方、src="me" とすると、アクティブオープンされたストリームが条件にマッチする。src="all" とすると一つのまへのノードはなんでもよいということになる。

dest

dest は参照先のアドレスの条件を与える。dest="me" とすると、参照先として自分のアドレスを指定しているストリームが選択される。dest="all" とすると、参照先は無条件となる。

type

条件がマッチしたときのアクションの種類を指定する。type= direct を指定すると、参照先へ直接接続を試みる。type= reject を指定すると、コネクションは切断される。type= proxy を指定すると、コネクションは、属性 gateway で指定された中継サーバ (proxy) へ接続される。

gateway

type= proxy の時のみに有効な属性で、中継サーバのアドレス、ポートを前述アドレス書式にて与える。type がその他の属性であるときには、この属性は無視される。

戻り値

XLT_NULL (5.2.1 節)

エラー

1. XLE_PROTO_INV_PARAM (7.2.58 節)

参考
バゲ

10.2.2 ActivateStreamRouting

プロトタイプ

LISP 形式

(ActivateStreamRouting info)

XML 形式

<ActivateStreamRouting> info </ActivateStreamRouting>

引数

なし。

属性

なし。

評価形式

applicative

所属エージェント

xl

所属環境

Env1

説明

エージェント xl 標準入出力を入力ストリームとして解釈し、これをストリームルーティングする。以降 xl エージェントの標準入出力に入ってくる情報はすべて、ルーティング先へ転送され、xl エージェントでは処理されなくなる。

標準入出力、または転送先へのストリームが切断された場合 xl エージェント全体が終了する。

info によって設定されたタイムアウトを迎えた場合も xl エージェント全体が終了する。

info の設定方法

info の内容は、次のような書式となっている。

- (1) [参照先], [TimeToLive 値], [タイムアウト値]
- (2) [参照先], [TimeToLive 値]
- (3) [参照先]

(2) (3) は (1) の省略形で、(2) の場合は、[タイムアウト値]= 3600 と同一である。また、(3) では、TimeToLive 値= 1 と同値である。タイムアウト値は秒で指定する。たとえば、

```
gbs.ur-plaza.osaka-cu.ac.jp:8080: ,5,120
```

となる。

ActiveStreamRouting は、info を解析し、参照先を得ると同時に、標準入出力の接続先を調べ、一つ前のノードのアドレスを得る。これらの情報を総合し、あらかじめ設定されているストリーム・ルーティング・テーブルを参照し、アクションを決定する。アクションが中継サーバへの接続であった場合、[TimeToLive 値] を一つカウントダウンし、これが 0 になった場合は、接続を断念する。

戻り値

XLT_NULL (5.2.1 節)

エラー

1. XLE_SEMANTICS_TYPE_MISMATCH (7.2.52 節)

info が文字列でない。

参考

1. /usr/local/xl-gbs/xlconf/std/sr.xl

ストリーム・ルーティング設定ファイル

2. /usr/local/xl-gbs/xlscript/gbs/a_xlsr.xl

xlsr エージェント。xl エージェントを使用。

バグ

関連図書

- [1] 森洋久. LANDSCAPE スタートアップ・マニュアル. GLOBALBASE PROJECT, 2006.
- [2] 森洋久. COSMOS リファレンス・マニュアル. GLOBALBASE PROJECT, 2006.

履歴

1. 日時: 2009-12-19
マニュアル生成。(2009-12-19 版)
2. 日時: 2000-09-30
著者: 森 洋久 反映されたバージョン: ver.A.
環境に分岐環境の記述を追加。それに伴い、関数の評価のところも変更。
3. 日時: 2000-10-31
著者: 森 洋久 反映されたバージョン: ver.A.
「構文」の章の構文解析についての記述について更新。LISP 形式の場合の属性リストの記述法について変更。インタプリタのモデルを変更。それにともない Remote, Local の解釈をはっきりさせた。また、多少エージェント、サーバの用語を変更した。
4. 日時: 2000-11-05
著者: 森 洋久 反映されたバージョン: ver.A.
インタプリタのモデルの変更にともない、OpenInterpreter, CloseInterpreter のセマンティックスを変更。OpenInterpreter でソケットをオープンできるようにし、CloseInterpreter で閉じることが出来るようにした。Connect は OpenInterpreter で代用できるので標準サポートしないことにした。Cancel の追加。
5. 日時: 2000-11-11
著者: 森 洋久 反映されたバージョン: ver.A.
SetMainInterpreter, Switch, Arg, GetId, GetUserInfo, Permission, Account, Agent, GetVariable, Launch の追加。しかし、Permission の実装は不完全。Account の最初の要素は group が妥当。まだ変更していない。GetEnvVar は、GetVariable に変更した方がよい。MainInterpreter は SetMainInterpreter に変更した方がよい。OpenInterpreter のオプションに Permission を追加。SetAgent の仕様の変更
6. 日時: 2000-11-15
著者: 森 洋久 反映されたバージョン: ver.A.
WaitInterpreter の追加。
7. 日時: 2000-12-24
著者: 森 洋久 反映されたバージョン: ver.A.
ErrHandler の仕様を変更。While, Break, Continue, ListLength の追加。OpenInterpreter に Result オプションを追加。
8. 日時: 2000-01-05
著者: 森 洋久 反映されたバージョン: ver.A.
出力フォーマットに関するオプションを OpenInterpreter, Save に追加。Arg の引数の数え方の変更。

9. 日時: 2000-01-14
著者: 森 洋久 反映されたバージョン: ver.A.
DivideString の追加、Sort の追加。quote の仕様変更。
10. 日時: 2001-01-24
著者: 森 洋久 反映されたバージョン: ver.A.
環境を関数と見なす新しいシンタックスの追加。(関数の評価のところを参照のこと) ThisAgent-Name, DatabasePath, SetPrefixMode, GetPrefixMode, Get, Set, GetLocalHostName, GetLocal-HostIP, OpenSession, RemoteSession, CloseSession, Sync, Lock, Unlock の追加
11. 日時: 2001-01-25
著者: 森 洋久 反映されたバージョン: ver.A.
InvokeError,DeleteSymbol,MoveSymbol を追加。
12. 日時: 2001-02-03
著者: 森 洋久 反映されたバージョン: ver.A.
DeleteSymbol,MoveSymbol を DeleteDefine,MoveDefine と改称。また、属性値に、シンボルを入れることが出来るようにした。LISP のリストの中で!を使うと、XML 形式へ戻るというシンタックスをやめた。<を受信したところで XML 形式へ戻る。従って、"<"を S 式のシンボルなどとして使いたい場合は、< を使うこと。
13. 日時: 2006-08-02
著者: 森 洋久 反映されたバージョン: ver.B.b11.02
新しい、マニュアルフォーマットへ移行開始。
14. 日時: 2006-08-08
著者: 森 洋久 反映されたバージョン: ver.B.b11.02
新しい、マニュアルフォーマットへ移行終了。型リファレンスとエラーコードリファレンスの追加。
15. 日時: 2006-08-15
著者: 森 洋久 反映されたバージョン: ver.B.b11.02
エラーコードの参照を行った。Let の記述を追加。これから新しい関数の記述を徐々に追加して行く。
16. 日時: 2006-08-15
著者: 森 洋久 反映されたバージョン: ver.B.b12
[UNDEF REF (xl-DefileEval)] を追加。
17. 日時: 2007-04-08
著者: 森 洋久 反映されたバージョン: ver.B.b15
8.2.56 節の機能追加対応。
18. 日時: 2007-05-06
著者: 森 洋久 反映されたバージョン: ver.B.b16
9 節の項目を追加。

19. 日時: 2008-05-19

著者: 森 洋久 反映されたバージョン: ver.B.b17.03

10 節の項目を追加。