ERLANG QUICK REFERENCE

By Karl Voelker

DATA

Chars	`$A, $\n`
Binary	`2#101`
Atom	`foo, 'Bar_'`
Ref (unique)	`make_ref()`
Anon. fun.	`fun (X)->X; ... end`
Tuple	`{foo,42}, {}`
List	`[], [a,b], [a|[]]`
Decl. record	`record(tag,{a,b}).`
New record	`#tag{a=1,b=2}`
Get rec. field	`expr#tag.field`
Get field index	`#tag.field`
Update record	`expr#tag{field=term}`
Bool atoms	`true, false`
List comp.	`[X || X<-[1,2], X > 1]`

MISCELLANEOUS

Comparison	`/= == =< < >= >`
Exact equality	`=/= =:=`
Arithmetic	`+ - * / div rem abs`
Boolean	`not and or xor`
Short-circuit	`orelse andalso`
Lists	`++ --`
Tuple accessor	`element(n,tup)`
Tuple mutator	`setelement(n,tup,term)`
Tuple size	`tuple_size(tup)`
Atom->String	`atom_to_list(atom)`
String->Atom	`list_to_atom(string)`
Integer->String	`integer_to_list(int)`
String->Integer	`list_to_integer(string)`
Tuple->List	`tuple_to_list(tuple)`
List->Tuple	`list_to_tuple(list)`
Type tests	`is_TYPE(term)`
Record type test	`is_record(term,tag)`
List length	`length(list)`

RUNNING ERLANG

REPL	`erl`
Quit Erlang	`halt()`
Compiler	`erlc foo.erl bar.erl`
Make	`erl -make`

MANUAL PAGES

File I/O	`man 3erl file`

MODULES and TOP-LEVEL FUNCTIONS

Decl. module	`module(name).`
Export	`export([f/2,g/3]).`
Import	`import(module,[h/4]).`
Fun. decl.	`FunClause; ... .`
Fun. clause	`Name(Pat,...) -> Body`
Clause w/ guard	`f(X) when Cond -> Body`
Seq. in clause	`f(X) -> X + Y, Z;`

PROCESSES

New process	`spawn(Module,Fn,ArgList)`
Name a process	`register(Name, PID)`
List names	`registered()`
Get PID by name	`whereis(Name)`
Send	`PID ! Msg`
Receive	`receive P->B; ... end`
Timeout	`... after n -> Body end`
Watch other proc.	`link(P), spawn_link(P)`
Stop watching	`unlink(P)`
Send exit signal	`exit(PID,Reason)`
Catch exit signals	`process_flag( trap_exit, true)`

SPECIAL EXPRESSIONS

If	`if G -> Body; ... end`
Case	`case E of Pat -> Body; ... end`
Case guard	`Pat when G -> Body;`
Expr. Block	`begin e1, ... end`

VARIABLES and PATTERN MATCHING

Variable	`Foo, Bar, _X, X@Y`
Bind pattern	`[X|Y] = foo()`
Anonymous	`_`
String prefix	`"foo" ++ Str`

PER-PROCESS DICTIONARY

Accessors	`get(K), get_keys(V)`
	`get()`
Mutators	`put(K,V)`
	`erase(K), erase()`