Enigma Level API II

Syntax sheet with examples

Compiled from Enigma 1.20 reference manual by Raoul

Position

Position Addition and Subtraction

```
result = pos <+|-> <pos | obj | cpos | polist>
result = <pos | obj | cpos | polist> <+|-> pos
```

```
newpos = po(3, 4) + {1, 2}              -- = po(4, 6)
newpos = myobject - po(1, 5)
newpolist = po(2, 3) + NEIGHBORS_4      -- po(1, 3) .. po(2, 4) .. po(3, 3) .. p
newpolist = po["myfloor#*"] - po(3, 0)
```

Position Multiplication and Division

```
result = pos <*|/> number
result = number * pos
```

```
newpos = 3 * po(3, 4)   -- = po(9, 12)
newpos = po(2, 3) / 2   -- = po(1, 1.5)
```

Position Sign

```
result = -pos
```

```
newpos = -po(3, 4)   -- = po(-3, -4)
```

Center

```
result = #pos
```

```
newpos = #po(3, 4)   -- = po(3.5, 4.5)
```

Position Comparison

```
result = pos1 <==|~=> pos2
```

```
bool = po(3, 4) == po({3, 4})  -- = true
bool = po(3, 4) == po(4, 3)    -- = false
bool = po(3, 4) ~= po(4, 3)    -- = true
```

Position Concatenation

```
result = pos1 .. <pos2 | polist>
result = <pos1 | polist> .. pos2
```

```
newpolist = po(3, 4) .. po(4, 4)
```

Position Coordinate Access

```
result = pos["x"]
result = pos["y"]
result1, result2 = pos:xy()
```

```
number = po(3, 4)["x"]            -- = 3
number = po(3, 4).x               -- = 3
number = po(3, 4)["y"]            -- = 4
number = po(3, 4).y               -- = 4
number1, number2 = po(3, 4):xy()  -- = 3, 4
```

Position Grid Rounding

```
result = pos:grid()
```

```
newpos = po(3.2, 4.7):grid()    -- = 3, 4
newpos = po(-2.4, -5.0):grid()  -- = -3, -5
```

Position Existence

```
result = pos:exists()
```

```
boolean = po(3.2, 4.7):exists()
```

Object

Object Attribute Access

```
result = obj["attributename"]
obj["attributename"] = value
obj:set({attributename1=value1, attributename2=value2, ...})
```

```
value = obj["color"]
value = obj.color
obj["color"] = BLACK
obj.color = BLACK
obj:set({target=mydoor, action="open"})
```

Object Messaging

```
result = obj:message("msg", value)
result = obj:msg(value)
```

```
value = obj:message("open")
value = obj:open()
value = obj:message("signal", 1)
value = obj:signal(1)
```

Object Comparison

```
result = obj1 <==|~=> obj2
```

```
bool = obj1 == obj1  -- = true
bool = obj1 == obj2  -- = false, if two different objects
bool = obj1 ~= obj2  -- = true, if two different objects
```

Object Existence

```
result = -obj
result = obj:exists()
```

```
bool = -obj
bool = obj:exists()
```

Object Kill

```
obj:kill()
```

```
obj:kill()
```

Object Kind Check

```
result = obj:is("kind")
result = obj:kind()
```

```
bool = obj:is("st_chess")
string = obj:kind()
```

Object Coordinate Access

```
result = obj["x"]
result = obj["y"]
result1, result2 = obj:xy()
```

```
number = obj["x"]
number = obj.x
number = obj["y"]
number = obj.y
number1, number2 = obj:xy()
```

Object Addition and Subtraction

```
result = obj <+|-> <pos | obj | cpos | polist>
result = <pos | obj | cpos | polist> <+|-> obj
```

```
newpos = obj + {1, 2}
newpos = myobject - obj
newpolist = obj + NEIGHBORS_4
newpolist = po["myfloor#*"] - obj
```

Object Center

```
result = #obj
```

```
newpos = #obj   -- e.g. po(3.5, 4.5)
```

Object Join

```
result = obj + group
result = group + obj
```

```
newgroup = obj1 + grp(obj2, obj3, obj1)   -- = grp(obj1, obj2, obj3)
newgroup = grp(obj2, obj3) + obj1         -- = grp(obj2, obj3, obj1)
```

Object Intersection

```
result = obj * group
result = group * obj
```

```
newgroup = obj1 * grp(obj1, obj2)  -- = grp(obj1)
newgroup = grp(obj2) * obj1         -- = grp()
```

Object Difference

```
result = obj - group
result = group - obj
```

```
newgroup = obj1 - grp(obj2, obj1)  -- = grp()
newgroup = grp(obj1, obj2) - obj1  -- = grp(obj2)
```

Group

Group Messaging

```
result = group:message("msg", value)
result = group:msg(value)
```

```
value = group:message("open")
value = group:open()
value = group:message("signal", 1)
value = group:signal(1)
value = group:kill()
```

Group Attribute Write

```
group["attributename"] = value
group:set({attributename1=value1, attributename2=value2, ...})
```

```
group["color"] = BLACK
group.color = BLACK
group:set({target=mydoor, action="open"})
```

Group Comparison

```
result = group1 <==|~=> group2
```

```
bool = grp(obj1, obj2) == grp(obj2, obj1)  -- = true
bool = grp(obj1, obj2) == grp(obj1, obj3)  -- = false, if different object conte
bool = grp(obj1) ~= grp(obj2, obj1)        -- = true, if different object conten
```

Group Length

```
result = #group
```

```
number = #grp(obj1, obj2)         -- = 2
for i = 1, #group do obj = group[i] ... end
```

Group Member Access

```
result = group[index]
result = group[obj]
```

```
object = grp(obj1, obj2)[2]     -- = obj2
object = grp(obj1, obj2)[-1]    -- = obj2
object = grp(obj1, obj2)[0]     -- = NULL object
for i = 1, #group do obj = group[i] ... end
number = grp(obj1, obj2)[obj2]  -- = 2
number = grp(obj1, obj2)[obj3]  -- = nil
```

Group Loop

```
for obj in group do ... end
```

```
for obj in group do obj:toggle() end
```

Group Join

```
result = group + <obj | group>
result = <obj | group> + group
```

```
newgroup = obj1 + grp(obj2, obj3, obj1)   -- = grp(obj1, obj2, obj3)
newgroup = grp(obj2, obj3) + grp(obj1, obj3)   -- = grp(obj2, obj3, obj1)
```

Group Intersection

```
result = <obj | group> * group
result = group * <obj | group>
```

```
newgroup = obj1 * grp(obj2, obj1)  -- = grp(obj1)
newgroup = grp(obj1, obj2) * grp(obj2, obj1, obj3)  -- = grp(obj1, obj2)
```

Group Difference

```
result = <obj | group> - group
result = group - <obj | group>
```

```
newgroup = obj1 - grp(obj2, obj1)  -- = grp()
newgroup = grp(obj1, obj2, obj3) - grp(obj2, obj4)  -- = grp(obj1, obj3)
```

Group Shuffle

```
result = group:shuffle()
```

```
newgroup = grp(obj1, obj2)
```

Group Sorting

```
result = group:sort("circular")
result = group:sort("linear" <, direction>)
result = group:sort()
```

```
newgroup = grp(obj1, obj2, obj3):sort("linear", po(2,1))
newgroup = grp(obj1, obj2, obj3):sort("circular")
newgroup = grp(obj1, obj2, obj3):sort()
```

Group Subset

```
result = group:sub(number)
result = group:sub(start, end)
result = group:sub(start, -number)
```

```
newgroup = grp(obj1, obj2, obj3, obj4):sub(2)     -- = grp(obj1, obj2)
newgroup = grp(obj1, obj2, obj3, obj4):sub(-2)    -- = grp(obj3, obj4)
newgroup = grp(obj1, obj2, obj3, obj4):sub(2, 4)  -- = grp(obj2, obj3, obj4)
newgroup = grp(obj1, obj2, obj3, obj4):sub(2, -2) -- = grp(obj2, obj3)
```

Group Nearest Object

```
result = group:nearest(obj)
```

```
newobject = grp(obj1, obj2, obj3):nearest(obj4)
```

NamedObjects

NamedObjects Repository Request

```
result = no["name"]
```

```
obj = no["mydoor"]       -- exact name match
group = no["mydoors#*"]  -- any suffix
group = no["mydoor?"]    -- just one char suffix
group = no["mydoors?#*"] -- matches e.g. "mydoorsA#123435", "mydoorsB#1213"
```

NamedObjects Object Naming

```
no["name"] = obj
```

```
no["myobject"] = obj
```

PositionList

PositionList Comparison

```
result = polist1 <==|~=> polist2
```

```
bool = (po(2,3).. po(5,7)) == (po(2,3) .. po(5,7))  -- = true
bool = (po(2,3).. po(5,7)) == (po(4,0) .. po(5,7))  -- = false, different positi
bool = (po(2,3).. po(5,7)) == (po(5,7) .. po(2,3))  -- = false, different sequen
```

PositionList Length

```
result = #polist
```

```
number = #(po(2,3) .. po(5,7)) -- = 2
for i = 1, #polist do pos = polist[i] ... end
```

PositionList Member Access

```
result = group[index]
```

```
pos = (po(2,3) .. po(5,7))[2]     -- = po(5,7)
pos = (po(2,3) .. po(5,7))[-1]    -- = po(5,7)
pos = (po(2,3) .. po(5,7))[0]     -- = nil
for i = 1, #polist do pos = polist[i] ... end
```

PositionList Concatenation

```
result = polist1 .. <pos | polist2>
result = <pos | polist1> .. polist2
```

```
newpolist = po(po(2,3), po(5,7)) .. po(4, 4) -- = (2,3),(5,7),(4,4)
```

PositionList Translation

```
result = polist <+|-> <pos | obj | cpos>
result = <pos | obj | cpos> <+|-> polist
```

```
newpolist = po(2, 3) + NEIGHBORS_4      -- po(1, 3) .. po(2, 4) .. po(3, 3) .. p
newpolist = po["myfloor#*"] - po(3, 0)
```

PositionList Stretching

```
result = polist * number
result = number * polist
```

```
newpolist = 2 * NEIGHBORS_4              -- = po(9, 12)
newpolist = (po(2,4) .. po(6,7)) * 1/2   -- = (1, 2), (3, 3.5)
```

Positions Repository

Positions Repository Request

```
result = po["name"]
```

```
pos = po["mydoor"]        -- exact name match
polist = po["mydoors#*"]  -- any suffix
polist = po["mydoor?"]    -- just one char suffix
polist = po["mydoors?#*"] -- matches e.g. "mydoorsA#123435", "mydoorsB#1213"
```

Positions Repository Storage

```
po["name"] = obj
```

```
po["mypos"] = pos
```

Position Convertion

```
result = po(<obj | pos | {x,y} | x,y>)
```

```
pos = po(pos2)
pos = po(obj)
pos = po({2, 4})
pos = po(3, 7)
```

PositionList Convertion

```
result = po(group | {pos1, pos2, pos3})
```

```
polist = po(group)
polist = po({po(3, 7), po(2, 6)})
```

Tile and Object Declaration

Tile concat

```
result = tile .. <tile | odecl>
result = <tile | odecl> .. tile
```

```
newtile = ti{st_chess"} .. {"fl_sahara"}
newtile = ti{st_chess"} .. {"fl_sahara"} .. {"it_cherry"}   -- Lua error due to
newtile = (ti{st_chess"} .. {"fl_sahara"}) .. {"it_cherry"} -- evaluation order
newtile = ti{st_chess"} .. {"fl_sahara"} .. ti{"it_cherry"} -- converted one of
```

Tiles Repository

Tiles Storage

```
ti["key"] = <tile | odecl>
```

```
ti["#"] = tile
ti["$"] = {st_chess"}
ti["$"] = {st_switch"}   -- error of key reassignment
ti["anykey"] = {st_chess"}
```

Tiles Request

```
result = ti["key"]
```

```
tile = ti["#"]
```

Tile Convertion

```
result = ti(odecl)
```

```
tile = ti({"st_chess"})
```

World

World Creation

```
width, height = wo(topresolver, defaultkey, map)
width, height = wo(topresolver, libmap)
width, height = wo(topresolver, defaultkey, width, height)
```

```
w, h = wo(ti, "  ", 20, 13)
w, h = wo(resolver, " ", {
       "                    ",
       ...
       "                    "})
w, h = wo(ti, mylibmap)
```

add

```
wo:add(tile_declarations)
wo:add(target, tile_declarations)
```

```
wo:add({"ot_rubberband", anchor1="a1", anchor2="w", length=2, strength=80, thres
wo:add(ti["r"] .. {"ot_wire", anchor1="w1", anchor2="w2"})
wo:add(YIN, {"it_magicwand})
wo:add(no["mybag"], {"it_magicwand} .. ti["h"] .. ti["c"])
```

World Tile Set

```
wo[<object | position | table | group | polist>] = tile_declarations
```

```
wo[no["myobjectname"]] = {"st_chess"}
wo[po(3, 4)] = ti["x"]
wo[{2, 5}] = ti["x"] .. ti["y"]
wo[no["floorgroup#*"]] = {"it_burnable_oil"}
wo[no["myobjectname"] + NEIGHBORS_4] = ti["x"]
```

Global Attribute Set

```
wo["attritbutename"] = value
```

```
wo["ConserveLevel"] = true
```

Global Attribute Get

```
var = wo["attritbutename"]
```

```
var = wo["IsDifficult"]
```

drawBorder

```
wo:drawBorder(upperleft_edge, lowerright_edge, <tile | key, resolver>)
wo:drawBorder(upperleft_edge, width, height, <tile | key, resolver>)
```

```
wo:drawBorder(po(0, 0), wo["Width"], wo["Height"], ti["#"])
wo:drawBorder(no["myRectUL"], no["myRectLR"], {"st_grate1"})
wo:drawBorder(no["myRectUL"], no["myRectLR"], {"fl_water"} .. ti["X"])
wo:drawBorder(no["myRectUL"], no["myRectLR"], "x", myresolver)
```

drawMap

```
wo:drawMap(resolver, anchor, ignore, map, [readdir])
wo:drawMap(resolver, anchor, libmap-map, [readdir])
```

```
wo:drawMap(ti, po(5, 7), "-", {"abcabc"})
wo:drawMap(ti, anchor_object, "--", {"--##--##","##--##"})
wo:drawMap(ti, {12, 5}, " ", {"122  221"}, MAP_ROT_CW)
```

drawRect

```
wo:drawRect(upperleft_edge, lowerright_edge, <tile | key, resolver>)
wo:drawRect(upperleft_edge, width, height, <tile | key, resolver>)
```

```
wo:drawRect(po(0, 0), wo["Width"], wo["Height"], ti[" "])
wo:drawRect(no["myRectUL"], no["myRectLR"], {"fl_water"})
wo:drawRect(no["myRectUL"], no["myRectLR"], {"fl_water"} .. ti["#"])
wo:drawRect(no["myRectUL"], no["myRectLR"], "x", myresolver)
```

world floor

```
result = wo:fl(<pos| {x,y} | x,y | obj | group | polist>)
```

```
use fl(...) instead
```

world item

```
result = wo:it(<pos | {x,y} | x,y | obj | group | polist>)
```

```
use it(...) instead
```

shuffleOxyd

```
wo:shuffleOxyd(rules)
```

```
wo:shuffleOxyd()
wo:shuffleOxyd({no["borderoxyds#*"]:sort("circular"), circular=true})
wo:shuffleOxyd({"leftoxyds#*","rightoxyds#*", min=3}, {"islandoxyds#*", max=0})
```

world stone

```
result = wo:st(<pos | {x,y} | x,y | obj | group | polist>)
```

```
use st(...) instead
```

Functions

cond

```
cond(condition, iftrue, iffalse)
```

```
ti["x"] = cond(wo["IsDifficult"], {"st_death"}, ti["#"])
ti["D"] = cond(wo["IsDifficult"], {"st_death"}, {"nil"})
```

fl

```
result = fl(<pos | {x,y} | x,y | obj | group| polist>)
```

```
floor = fl(po(3, 5))
floor = fl({3, 5})
floor = fl(3, 5)
floor = fl(mystone)
group = fl(no["door#*"])
group = fl(po(3, 5)..po(4, 2))
```

grp

```
grp(<{obj1,obj2,...} | obj1,obj2,... | group>)
```

```
newgroup = grp(obj1, obj2, obj3)
newgroup = grp({obj1,obj2})
newgroup = grp{}            -- empty group
newgroup = grp(group)       -- a copy of group cleaned of invalid ‘NULL’ objects
```

it

```
result = it(<pos | {x,y} | x,y | obj | group | polist>)
```

```
item = it(po(3, 5))
item = it({3, 5})
item = it(3, 5)
item = it(mystone)
group = it(no["door#*"])
group = it(po(3, 5)..po(4, 2))
```

ORI2DIR

```
result = ORI2DIR[orientation]
```

```
direction = ORI2DIR[NORTH]      -- N  = po(0, -1)
direction = ORI2DIR[SOUTHEAST]  -- SE = po(1,  1)
direction = ORI2DIR[NODIR]      --      po(0,  0)
```

random

```
result = random(< | n | l,u>)
```

```
float = random()          -- e.g. 0.402834
integer = random(20)      -- e.g. 13
integer = random(5, 10)   -- e.g. 5
```

st

```
result = st(<pos | {x,y} | x,y | obj | group | polist>)
```

```
stone = st(po(3, 5))
stone = st({3, 5})
stone = st(3, 5)
stone = st(myfloor)
group = st(no["cherry#*"])
group = st(po(3, 5)..po(4, 2))
```
