MOO Utils Quick Reference

Ron Tapia
Hagbard@LambdaMOO
Ron@JaysHouse

May 6, 1993

Contents

Chapter 1

1.1 Intoduction

This is basically what you get when when you look at the utilities. I found a very similar document to be very handy when I was learning to program in MOO.

1.2 $string_utils

For a complete description of a given verb, do 'help $string_utils:verbname'

```
    Conversion routines:

:from_list    (list [,sep])                           => "foo1foo2foo3"
:english_list (str-list[,none-str[,and-str[, sep]]]) => "foo1, foo2, and foo3"
:title_list*c (obj-list[,none-str[,and-str[, sep]]]) => "foo1, foo2, and foo3"
                                                   or => "Foo1, foo2, and foo3"
:from_value   (value [,quoteflag [,maxlistdepth]])   => "{foo1, foo2, foo3}"
:print            (value)                             => "{foo1, foo2, foo3}"
:print_suspended  (value)                             => same, but may suspend
:print_truncating (value, length, suffix)             => "{foo1, fo.."

:english_number(42)          => "forty-two"
:english_ordinal(42)         => "forty-second"
:ordinal(42)                 => "42nd"
:group_number(42135 [,sep])  => "42,135"

    Type checking:

:is_numeric   (string) => return true if string is composed entirely of digits

    Parsing:

:explode (string,char) -- string => list of words delimited by char
:words   (string)      -- string => list of words (as with command line parser)
:word_start (string)   -- string => list of start-end pairs.

    Matching:

:match_string (string, pattern, options)       => * wildcard matching
:find_prefix  (prefix, string-list)=>list index of element starting with prefix
:index_delimited(string,target[,case]) =>index of delimited string occurrence
:match        (string, [obj-list, prop-name]+) => matching object
:match_player (string-list[,me-object])        => list of matching players
:match_object (string, location)               => default object match...

    Pretty printing:

:space         (n/string[,filler])     => n spaces
:left          (string,width[,filler]) => left justified string in field
:right         (string,width[,filler]) => right justified string in field
```

```
:center/re     (string,width[,filler]) => centered string in field
:columnize/se  (list,n[,width])        => list of strings in n columns

   Substitutions

:substitute (string,subst_list [,case])   -- general substitutions.
:pronoun_sub (string/list[,who[,thing[,location]]])
                                          -- pronoun substitutions.
:pronoun_sub_secure (string[,who[,thing[,location]]],default)
                                          -- substitute and check for names.
:pronoun_quote (string/list/subst_list)   -- quoting for pronoun substitutions.

   Miscellaneous string munging:

:trim         (string)       => string with outside whitespace removed.
:triml        (string)       => string with leading whitespace removed.
:trimr        (string)       => string with trailing whitespace removed.
:strip_chars  (string,chars) => string with all chars in `chars' removed.
:strip_all_but(string,chars) => string with all chars not in `chars' removed.
:capitalize/se(string)       => string with first letter capitalized.
:uppercase/lowercase(string) => string with all letters upper or lowercase.
:names_of     (list of OBJ)  => string with names and object numbers of items.

   Conversions to and from common string formats:

:character_to_ascii(char)    => integer ASCII rep of char
:character_to_hex_ascii(char) => two-digit hexadecimal string ASCII rep of char
:ascii_to_character(num)     => character represented in ASCII by num
:hex_ascii_to_character(hex) => ASCII character represented by two-character
 hexadecimal string hex

   A useful property:

.alphabet                    => "abcdefghijklmnopqrstuvwxyz"
```

1.3 $list_utils

Useful verbs for operating on lists.

```
append            (list,list,..) => result of concatenating the given lists
reverse           (list)         => reversed list
remove_duplicates (list)         => list with all duplicates removed
compress          (list)         => list with consecutive duplicates removed
setremove_all     (list,elt)     => list with all occurrences of elt removed
find_insert       (sortedlist,e) => index of first element > e in sortedlist
sort              (list[,keys])  => sorted list

make              (n[,e])        => list of n copies of e
range             (m,n)          => {m,m+1,...,n}

arrayset   (list,val,i[,j,k...]) => array modified so that list[i][j][k]==val

-- Mapping functions (take a list and do something to each element):

map_prop ({o...},prop)              => list of o.(prop)            for all o
map_verb ({o...},verb[,args)        => list of o:(verb)(@args)     for all o
map_arg  ([n,]obj,verb,{a...},args) => list of obj:(verb)(a,@args) for all a

-- Association list functions --

An association list (alist) is a list of pairs (2-element lists), though the
 following functions have been generalized for lists of n-tuples (n-element
 lists).  In each case i defaults to 1.

assoc        (targ,alist[,i]) => 1st tuple in alist whose i-th element is targ
iassoc       (targ,alist[,i]) => index of same.
assoc_prefix (targ,alist[,i]) => ... whose i-th element has targ as a prefix
iassoc_prefix(targ,alist[,i]) => index of same.
slice             (alist[,i]) => list of i-th elements
sort_alist        (alist[,i]) => alist sorted on i-th elements.
```

1.4 $set_utils

This object is useful for operations that treat lists as sets (i.e., without concern about order and assuming no duplication).

```
 union(set, set, ...)        => union
 intersection(set, set, ...) => intersection

 diff*erence(set 1, set 2, ..., set n)
        => result of removing all elements of sets 2..n from set 1.
 exclusive_or(set, set, set, ...)
        => all elements that are contained in exactly one of the sets

  contains(set 1, set 2, ..., set n)
        => true if and only if all of sets 2..n are subsets of set 1
```

1.5 $object_utils

These routines are useful for finding out information about individual objects.

```
Examining everything an object has defined on it:
  all_verbs          (object) => like it says
  all_properties     (object) => likewise
  findable_properties(object) => tests to see if caller can "find" them
  owned_properties   (object[, owner]) => tests for ownership

Investigating inheritance:
  ancestors(object[,object...]) => all ancestors
  descendants      (object)     => all descendants
  ordered_descendants(object)   => descendants, in a different order
  leaves           (object)     => descendants with no children
  branches         (object)     => descendants with children
  isa        (object,class) => true iff object is a descendant of class (or ==)

Considering containment:
  contains      (obj1, obj2) => Does obj1 contain obj2 (nested)?
  all_contents      (object) => return all the (nested) contents of object

Verifying verbs and properties:
  has_property(object,pname) => false/true   according as object.(pname) exists
  has_verb    (object,vname) => false/{#obj} according as object:(vname) exists
  has_callable_verb          => same, but verb must be callable from a program
  match_verb  (object,vname) => false/{location, newvname}
                               (identify location and usable name of verb)
```

1.6 $command_utils

$command_utils is the repository for verbs that are of general usefulness to authors of all sorts of commands.

```
Detecting and Handling Failures in Matching
-------------------------------------------
:object_match_failed(match_result, name)
    Test whether or not a :match_object() call failed and print messages if so.
:player_match_failed(match_result, name)
    Test whether or not a :match_player() call failed and print messages if so.
:player_match_result(match_results, names)
    ...similar to :player_match_failed, but does a whole list at once.

Reading Input from the Player
-----------------------------
:read()         -- Read one line of input from the player and return it.
:yes_or_no([prompt])
                -- Prompt for and read a `yes' or `no' answer.
:read_lines()   -- Read zero or more lines of input from the player.
:dump_lines(lines)
                -- Return list of lines quoted so that feeding them to
                   :read_lines() will reproduce the original lines.

Utilities for Suspending
------------------------
:running_out_of_time()
                -- Return true if we're low on ticks or seconds.
:suspend_if_needed(time)
                -- Suspend (and return true) if we're running out of time.

Client Support for Lengthy Commands
-----------------------------------
:suspend(args)  -- Handle PREFIX and SUFFIX for clients in long commands.
```

1.7 $code_utils

Verbs for mucking around with MOO code.

```
parse_propref("foo.bar")  => {"foo","bar"} (or 0 if arg. isn't a property ref.)
parse_verbref("foo:bar")  => {"foo","bar"} (or 0 if arg. isn't a verb ref.)
parse_argspec("any","in","front","of","this","baz"...)
                          => {{"any", "in front of", "this"},{"baz"...}}
                                         (or string if args don't parse)

tonum(string)           => number (or E_TYPE if string is not a number)
toobj(string)           => object (or E_TYPE if string is not an object)
toerr(number or string) => error value (or 1 if out of range or unrecognized)
error_name(error value) => name of error (e.g., error_name(E_PERM) => "E_PERM")

verb_perms()    => the current task_perms (as set by set_task_perms()).
verb_location() => the object where the current verb is defined.
verb_documentation([object,verbname]) => documentation at beginning of
           verb code, if any -- default is the calling verb

   Preposition routines

prepositions()     => full list of prepostions
full_prep ("in")   => "in/inside/into"
short_prep("into") => "in"
short_prep("in/inside/into") => "in"
get_prep  ("off", "of", "the", "table") => {"off of", "the", "table"}

   Verb routines

verbname_match (fullname,name) => can `name' be used to call `fullname'
find_verb_named          (object,name[,n]) => verb number or -1 if not found
find_callable_verb_named (object,name[,n]) => verb number or -1 if not found
find_verbs_containing (pattern[,object|objlist])

   Verbs that do the actual dirty work for @show:

show_object  (object)
show_property(object,propname)
show_verbdef (object,verbname)

   Dirty work for explain_syntax

explain_verb_syntax(thisname,verbname,@verbargs)
```

```
   A random but useful verb

verb_or_property(object,name[,@args]) => result of verb or property call,
                                         or E_PROPNF
```

1.8 $math_utils

Various mathematical functions.

```
Trigonometric/Exponential functions:
  sin(a),cos(a),tan(a) -- returns 10000*(the value of the corresponding
       trigonometric function) angle a is in degrees.
  arctan([x,]y) -- returns arctan(y/x) in degrees in the range -179..180.
       x defaults to 10000.  Quadrant is that of (x,y).
  exp(x[,n]) -- calculates e^x with an nth order taylor polynomial

Statistical functions:
  combinations(n,r) -- returns the number of combinations given n objects
       taken r at a time.
  permutations(n,r) -- returns the number of permutations possible given
       n objects taken r at a time.

Number decomposition:
  div(n,d) -- correct version of / (handles negative numbers correctly)
  mod(n,d) -- correct version of % (handles negative numbers correctly)
  divmod(n,d) -- {div(n,d),mod(n,d)}
  parts(n,q[,i]) -- returns a list of two elements {integer,decimal fraction}

Other math functions:
  sqrt(x)      -- returns the largest integer n <= the square root of x
  pow(x,n)     -- returns x^n
  factorial(x) -- returns x!

Series:
  fibonacci(n) -- returns the 1st n fibonacci numbers in a list
  geometric(x,n) -- returns the value of the nth order geometric series at x

Integer Properties:
  gcd(a,b) -- find the greatest common divisor of the two numbers
  lcm(a,b) -- find the least common multiple of the two numbers
  are_relatively_prime(a,b) -- return 1 if a and b are relatively prime
  is_prime(n) -- returns 1 if the number is a prime and 0 otherwise

Miscellaneous:
  random(n) -- returns a random number from 0..n if n > 0 or n..0 if n < 0
  random_range(n[,mean]) -- returns a random number from mean - n..mean + n
       with mean defaulting to 0
  simpson({a,b},{f(a),f((a+b)/2),f(b)}) -- returns the numerical
      approximation of an integral using simpson's rule
```

```
Bitwise Arithmetic:
  AND(x,y) -- returns x AND y
  OR(x,y) -- returns x OR y
  XOR(x,y) -- returns x XOR y (XOR is the exclusive-or function)
  NOT(x) -- returns the complement of x
      All bitwise manipulation is of 32-bit values.
```

1.9 $time_utils

Useful verbs for dealing with time.

```
    Converting from seconds-since-1970
dhms          (time)                 => string ...DD:HH:MM:SS
english_time  (time[, reference time)=> string of y, m, d, m, s

    Converting to seconds
to_seconds    ("hh:mm:ss")           => seconds since 00:00:00
from_ctime    (ctime)                => corresponding time-since-1970
from_day      (day_of_week, which)   => time-since-1970 for the given day*
from_month    (month, which)         => time-since-1970 for the given month*
    (* the first midnight of that day/month)

    Converting to some standard English formats
day           ([c]time)              => what day it is
month         ([c]time)              => what month it is
ampm          ([c]time[, precision]) => what time it is, with am or pm

    Substitution
time_sub      (string, time)         => substitute time information

    Miscellaneous
sun           ([time])               => angle between sun and zenith
dst_midnight  (time)
```

1.10 $building_utils

Verbs useful for building. For a complete description of a given verb, do 'help $building_utils:verbname'.

```
make_exit(spec,source,dest[,don't-really-create]) => a new exit
          spec is an exit-spec as described in 'help @dig'

set_names(object, spec) - sets name and aliases for an object
parse_names(spec) => list of {name, aliases}
          in both of these, spec is of the form
            <name>[[,:]<alias>,<alias>,...]
          (as described in 'help @rename')

recreate(object, newparent) - effectively recycle and recreate object
          as a child of newparent
transfer_ownership(object, old-owner, new-owner) - just what it sounds like
```

1.11 $gender_utils

Defines the list of standard genders, the default pronouns for each, and routines for adding or setting pronoun properties on any gendered object.

```
Properties:
  .genders  -- list of standard genders
  .pronouns -- list of pronoun properties
  .ps .po .pp .pq .pr .psc .poc .ppc .pqc .prc
           -- lists of pronouns for each of the standard genders

  If foo is of gender this.gender[n],
  then the default pronoun foo.p is this.p[n]
  (where p is one of ps/po/pp/pq...)

Verbs:
  :set(object,newgender) -- changes pronoun properties to match new gender.
  :add(object[,perms[,owner]]) -- adds pronoun properties to object.

  :get_pronoun     (which,object) -- return pronoun for a given object
  :get_conj*ugation(verbspec,object) -- return appropriately conjugated verb
```

1.12 Others

There are at least two other utility objects: $perm_utils and $lock_utils.