

●● Object properties ●●●●●●●●●●●●●●●●●●●●

Where the *value* of a property can be a routine, several formats are possible (but remember: embedded “]” returns false, standalone “]” returns true,):

```
property [; statement; statement; ... ]
```

```
property [; return routine(); ]
```

```
property [; routine(); ]
```

```
property routine
```

“⊕” marks an additive property: such properties in an Object definition supplement, rather than supersede, the same properties in a Class definition (and are dealt with first).

add_to_scope

For an object: additional objects which follow it in and out of scope. The *value* can be: a space-separated list of *objects*, or a routine which invokes PlaceInScope() or ScopeWithin() to specify objects.

after ⊕

For an object: receives every *action* and *fake_action* for which this is the *noun*.

For a room: receives every *action* which occurs here.

The *value* is a routine of structure similar to a switch statement, having cases for the appropriate *actions* (and an optional default as well); it is invoked after the action has happened, but before the player has been informed. The routine should return: false to continue, telling the player what has happened, or true to stop processing the action and produce no further output.

article

For an object: the object’s indefinite article – the default is automatically “a”, “an” or “some”. The *value* can be: a string, or a routine which outputs a string.

articles

For a non-English object: its definite and indefinite articles. The *value* is an array of strings.

before ⊕

For an object: receives every *action* and *fake_action* for which this is the *noun*.

For a room: receives every *action* which occurs here.

The *value* is a routine invoked before the action has happened. See after.

cant_go

For a room: the message when the player attempts an impossible exit. The *value* can be: a string, or a routine which outputs a string.

capacity

For a container or supporter object: the number of objects which can be placed in or on it – the default is 100. For the player: the number which can be carried – selfobj has an initial capacity of MAX_CARRIED.

The *value* can be: a number, or a routine which returns a number.

d_to

For a room: a possible exit. The *value* can be:

- false (the default): not an exit;
- a string: output to explain why this is not an exit;
- a *room*: the exit leads to this room;
- a door object: the exit leads through this door;
- a routine which should return: false, a string, a *room*, a door object, or true to signify ‘not an exit’ and produce no further output.

daemon

The *value* is a routine which can be activated by StartDaemon(*object*) and which then runs once each turn until deactivated by StopDaemon(*object*).

describe ⊕

For an object: called before the object’s description is output. For a room: called before the room’s (long) description is output.

The *value* is a routine which should return: false to continue, outputting the usual description, or true to stop processing and produce no further output.

description

For an object: its description (output by Examine).

For a room: its long description (output by Look).

The *value* can be: a string, or a routine which outputs a string.

door_dir

For a compass object (d_obj, e_obj, ...): the direction in which an attempt to move to this object actually leads.

For a door object: the direction in which this door leads.

The *value* can be: a directional property (d_to, e_to, ...), or a routine which returns such a property.

door_to

For a door object: where it leads. The *value* can be:

- false (the default): leads nowhere;
- a string: output to explain why door leads nowhere;
- a *room*: the door leads to this room;
- a routine which should return: false, a string, a *room*, or true to signify ‘leads nowhere’ without producing any output.

e_to

See d_to.

each_turn ⊕

Invoked at the end of each turn (after all appropriate daemons and timers) whenever the object is in scope. The *value* can be: a string, or a routine.

found_in

For an object: the rooms where this object can be found, unless it has the absent attribute. The *value* can be:

- a space-separated list of *rooms* (where this object can be found) or *objects* (whose locations are tracked by this object);
- a routine which should return: true if this object can be found in the current location, otherwise false.

grammar

For an animate or talkable object: the *value* is a routine called when the parser knows that this object is being addressed, but has yet to test the grammar. The routine should return: false to continue, true to indicate that the routine has parsed the entire command, or a dictionary word (*'word'* or *-'word'*).

in_to

See d_to.

initial

For an object: its description before being picked up. For a room: its description when the player enters the room.

The *value* can be: a string, or a routine which outputs a string.

inside_description

For an enterable object: its description, output as part of the room description when the player is inside the object.

The *value* can be: a string, or a routine which outputs a string.

invent

For an object: the *value* is a routine for outputting the object's inventory listing, which is called twice. On the first call nothing has been output; *inventory_stage* has the value 1, and the routine should return: *false* to continue or *true* to stop processing and produce no further output. On the second call the object's indefinite article and short name have been output, but not any subsidiary information; *inventory_stage* has the value 2, and the routine should return: *false* to continue or *true* to stop processing and produce no further output.

life ⊕

For an animate object: receives person-to-person *actions* (Answer Ask Attack Give Kiss Order Show Tell ThrowAt WakeOther) for which this is the *noun*. The *value* is a routine of structure similar to a *switch* statement, having cases for the appropriate *actions* (and an optional default as well). The routine should return: *false* to continue, telling the player what has happened, or *true* to stop processing the action and produce no further output.

list_together

For an object: groups related objects when outputting an inventory or room contents list. The *value* can be:

- a *number*: all objects having this value are grouped;
- a *string*: all objects having this value are grouped as a count of the string;
- a routine which is called twice. On the first call nothing has been output; *inventory_stage* has the value 1, and the routine should return: *false* to continue, or *true* to stop processing and produce no further output. On the second call the list has been output; *inventory_stage* has the value 2, and there is no test on the return value.

n_to

See *d_to*.

name ⊕

Defines a space-separated list of words which are added to the Inform dictionary. Each word can be supplied in apostrophes '...' or quotes "..."; in all other cases only words in apostrophes update the dictionary.

For an object: identifies this object.

For a room: outputs "does not need to be referred to".

ne_to

See *d_to*.

number

For an object or room: the *value* is a general-purpose variable freely available for use by the program. A player object must provide (but not use) this variable.

nw_to

See *d_to*.

orders

For an animate or talkable object: the *value* is a routine called to carry out the player's orders. The routine should return: *false* to continue, or *true* to stop processing the action and produce no further output.

out_to

See *d_to*.

parse_name

For an object: the *value* is a routine called to parse an object's name. The routine should return: zero if the text makes no sense, -1 to cause the parser to resume, or the positive number of words matched.

plural

For an object: its plural form, when in the presence of others like it. The *value* can be: a string, or a routine which outputs a string.

react_after

For an object: detects nearby actions – those which take place when this object is in scope. The *value* is a routine invoked after the action has happened, but before the player has been informed. See *after*.

react_before

For an object: detects nearby actions – those which take place when this object is in scope. The *value* is a routine invoked before the action has happened. See *after*.

s_to

se_to

See *d_to*.

short_name

For an object: an alternative or extended short name. The *value* can be: a string, or a routine which outputs a string. The routine should return: *false* to continue by outputting the object's 'real' short name (from the head of the object definition), or *true* to stop processing the action and produce no further output.

short_name_indef

For a non-English object: the short name when preceded by an indefinite object. The *value* can be: a string, or a routine which outputs a string.

sw_to

See *d_to*.

time_left

For a timer object: the *value* is a variable to hold the number of turns left until this object's timer – activated and initialised by *StartTimer(object)* – counts down to zero and invokes the object's *time_out* property.

time_out

For a timer object: the *value* is a routine which is run when the object's *time_left* value – initialised by *StartTimer(object)*, and not in the meantime cancelled by *StopTimer(object)* – counts down to zero.

u_to

w_to

See *d_to*.

when_closed

when_open

For a container or door object: used when including this object in a room's long description. The *value* can be: a string, or a routine which outputs a string.

when_off

when_on

For a switchable object: used when including this object in a room's long description. The *value* can be: a string, or a routine which outputs a string.

with_key

For a lockable object: the 'key' *object* needed to lock and unlock the object, or nothing if no key fits.

●● Group 1 actions ●●●●●●●●●●●●●●●●●●●●

Group 1 actions support the 'meta' verbs and debug tools.

●● Group 2 actions ●●●●●●●●●●●●●●●●●●●●

Group 2 actions usually work, given the right circumstances. These are the standard actions and their triggering verbs.

Close	"CLOSE [UP]", "COVER [UP]", "SHUT [UP]"
Disrobe	"DISROBE", "DOFF", "REMOVE", "SHED", "TAKE OFF"
Drop	"DISCARD", "DROP", "PUT DOWN", "THROW"
Eat	"EAT"
Empty	"EMPTY [OUT]"
EmptyT	"EMPTY IN INTO ON ONTO TO"
Enter	"CROSS", "ENTER", "GET IN INTO ON ONTO", "GO IN INSIDE INTO THROUGH", "LEAVE IN INSIDE INTO THROUGH", "LIE IN INSIDE ON", "LIE ON TOP OF", "RUN IN INSIDE INTO THROUGH", "SIT IN INSIDE ON", "SIT ON TOP OF", "STAND ON", "WALK IN INSIDE INTO THROUGH"
Examine	"CHECK", "DESCRIBE", "EXAMINE", "L[OOK] AT", "READ", "WATCH", "X"
Exit	"EXIT", "GET OFF OUT UP", "LEAVE", "OUT[SIDE]", "STAND [UP]"
GetOff	"GET OFF"
Give	"FEED [TO]", "GIVE [TO]", "OFFER [TO]", "PAY [TO]"
Go	"GO", "LEAVE", "RUN", "WALK"
GoIn	"CROSS", "ENTER", "IN[SIDE]"
Insert	"DISCARD IN INTO", "DROP DOWN IN INTO", "INSERT IN INTO", "PUT IN INSIDE INTO", "THROW DOWN IN INTO"
Inv	"I[NV]", "INVENTORY", "TAKE INVENTORY"
InvTall	"I[NV] TALL", "INVENTORY TALL"
InvWide	"I[NV] WIDE", "INVENTORY WIDE"
Lock	"LOCK WITH"
Look	"L[OOK]"
Open	"OPEN", "UNCOVER", "UNDO", "UNWRAP"
PutOn	"DISCARD ON ONTO", "DROP ON ONTO", "PUT ON ONTO", "THROW ON ONTO"
Remove	"GET FROM", "REMOVE FROM", "TAKE FROM OFF"
Search	"L[OOK] IN INSIDE INTO THROUGH", "SEARCH"
Show	"DISPLAY [TO]", "PRESENT [TO]", "SHOW [TO]"
SwitchOff	"CLOSE OFF", "SCREW OFF", "SWITCH OFF", "TURN OFF", "TWIST OFF"

SwitchOn	"SCREW ON", "SWITCH ON", "TURN ON", "TWIST ON"
Take	"CARRY", "GET", "HOLD", "PEEL [OFF]", "PICK UP", "REMOVE", "TAKE"
Transfer	"CLEAR TO", "MOVE TO", "PRESS TO", "PUSH TO", "SHIFT TO", "TRANSFER TO"
Unlock	"OPEN WITH", "UNDO WITH", "UNLOCK WITH"
VagueGo	"GO", "LEAVE", "RUN", "WALK"
Wear	"DON", "PUT ON", "WEAR"

●● Group 3 actions ●●●●●●●●●●●●●●●●●●●●

Group 3 actions are by default stubs which output a message and stop at the 'before' stage (so there is no 'after' stage).

Answer	"ANSWER TO", "SAY TO", "SHOUT TO", "SPEAK TO"
Ask	"ASK ABOUT"
AskFor	"ASK FOR"
Attack	"ATTACK", "BREAK", "CRACK", "DESTROY", "FIGHT", "HIT", "KILL", "MURDER", "PUNCH", "SMASH", "THUMP", "TORTURE", "WRECK"
Blow	"BLOW"
Burn	"BURN [WITH]", "LIGHT [WITH]"
Buy	"BUY" "PURCHASE"
Climb	"CLIMB [OVER UP]", "SCALE"
Consult	"CONSULT ABOUT ON", "LOOK UP IN", "READ ABOUT IN", "READ IN"
Cut	"CHOP", "CUT", "PRUNE", "SLICE"
Dig	"DIG [WITH]"
Drink	"DRINK", "SIP", "SWALLOW"
Fill	"FILL"
Jump	"HOP", "JUMP", "SKIP"
JumpOver	"HOP OVER", "JUMP OVER", "SKIP OVER"
Kiss	"EMBRACE", "HUG", "KISS"
Listen	"HEAR", "LISTEN [TO]"
LookUnder	"LOOK UNDER"
Mild	Various mild swearwords.
No	"NO"
Pray	"PRAY"
Pull	"DRAG" "PULL"
Push	"CLEAR", "MOVE", "PRESS", "PUSH", "SHIFT"
PushDir	"CLEAR", "MOVE", "PRESS", "PUSH", "SHIFT"
Rub	"CLEAN", "DUST", "POLISH", "RUB", "SCRUB", "SHINE", "SWEEP", "WIPE"
Set	"ADJUST", "SET"
SetTo	"ADJUST TO", "SET TO"

Sing	"SING"
Sleep	"NAP", "SLEEP"
Smell	"SMELL", "SNIFF"
Sorry	"SORRY"
Squeeze	"SQUASH", "SQUEEZE"
Strong	Various strong swearwords.
Swim	"DIVE", "SWIM"
Swing	"SWING [ON]"
Taste	"TASTE"
Tell	"TELL ABOUT"
Think	"THINK"
ThrowAt	"THROW AGAINST AT ON ONTO"
Tie	"ATTACH [TO]", "FASTEN [TO]", "FIX [TO]", "TIE [TO]"
Touch	"FEEL", "FONDLE", "GROPE", "TOUCH"
Turn	"ROTATE", "SCREW", "TURN", "TWIST", "UNSCREW"
Wait	"WAIT" "Z"
Wake	"AWAKE[N]", "WAKE [UP]"
WakeOther	"AWAKE[N]", "WAKE [UP]"
Wave	"WAVE"
WaveHands	"WAVE"
Yes	"Y[ES]"

●● Fake actions ●●●●●●●●●●●●●●●●●●●●

LetGo	Generated by Remove.
ListMiscellany	Outputs a range of inventory messages.
Miscellany	Outputs a range of utility messages.
NotUnderstood	Generated when the parser fails to interpret some orders.
Order	Receives things not handled by orders.
PluralFound	Tells the parser that parse_name() has identified a plural object.
Prompt	Outputs the prompt, normally ">".
Receive	Generated by Insert and PutOn.
TheSame	Generated when the parser can't distinguish between two objects.
ThrownAt	Generated by ThrowAt.