InfoLib

at your fingertips

A quick reference to the Inform Library

Version 1.5 (March 2002)

The road to brevity is via solecism and through imprecision – refer to the *Inform Designer's Manual* for the definitive story.

Library objects

`compass`

A `container` object holding the twelve direction objects `d_obj e_obj in_obj n_obj ne_obj nw_obj out_obj s_obj se_obj sw_obj u_obj w_obj`.

`LibraryMessages`

If defined (between Includes of `Parser` and `VerbLib`), changes standard library messages:

```
Object  LibraryMessages
  with  before [;
            action: "string";
            action: "string";
            action: switch (lm_n) {
                value: "string";
                value: "string",(a) lm_o,".";
                ...
                }
            ...
            ];
```

`selfobj`

The default player object. Avoid: use instead the `player` variable, which usually refers to `selfobj`.

`thedark`

A pseudo-room which becomes the `location` when there is no light (although the player object is not moved there).

Library constants

In addition to the standard constants `true` (1), `false` (0) and `nothing` (0), the Library defines `NULL` (–1) for an *`action`*, *`property`* or *`pronoun`* whose current value is undefined.

User-defined constants

Some constants control features rather than represent values.

`AMUSING_PROVIDED`

Activates the `Amusing` entry_point.

`DEATH_MENTION_UNDO`

Offers "UNDO the last move" when the game is over.

`DEBUG`

Activates the debug commands.

`Headline = "string"`

Mandatory: the game style, copyright information, etc.

`MANUAL_PRONOUNS`

Pronouns reflect only objects mentioned by the player.

`MAX_CARRIED = expr`

Maximum number of direct possessions that the player can carry (default 100).

`MAX_SCORE = expr`

Maximum game score (default 0).

`MAX_TIMERS = expr`

Maximum number of active timers/daemons (default 32).

`NO_PLACES`

The "`OBJECTS`" and "`PLACES`" verbs are not allowed.

`NUMBER_TASKS = expr`

Number of `scored` tasks to be performed (default 1).

`OBJECT_SCORE = expr`

For taking a `scored` object for the first time (default 4).

`ROOM_SCORE = expr`

For visiting a `scored` room for the first time (default 5).

`SACK_OBJECT = object`

A `container` object where the game places held objects.

`Story = "string"`

Mandatory: the name of the story.

`TASKS_PROVIDED`

Activates the task scoring system.

`USE_MODULES`

Activates linking with pre-compiled library modules.

`WITHOUT_DIRECTIONS`

De-activates standard compass directions (bar "`IN`" and "`OUT`"). Place alternative directions in the `compass`.

Library variables

`action`

The current *`action`*.

`actor`

The target of an instruction: the player, or an NPC.

`deadflag`

Normally 0: 1 indicates a regular death, 2 indicates that the player has won, 3 or more denotes a user-defined end.

`inventory_stage`

Used by `invent` and `list_together` properties.

`keep_silent`

Normally `false`; `true` makes most group 2 actions silent.

`location`

The player's current room; unless that's dark, when it contains `thedark`, `real_location` contains the room.

`notify_mode`

Normally `true`: `false` remains silent when score changes.

`noun`

The primary focus object for the current action.

`player`

The object acting on behalf of the human player.

`real_location`

The player's current room when in the dark.

`score`

The current score.

`second`

The secondary focus object for the current action.

`self`

The object which received a message .
(Note: a run-time variable, not a compile-time constant.)

`sender`

The object which sent a message (or `nothing`).

`task_scores`

A byte array holding scores for the task scoring system.

`the_time`

The game's clock, in minutes 0..1439 since midnight.

`turns`

The game's turn counter.

`wn`

The input stream word number, counting from 1.

1

Library routines

`Achieved(expr)`
A scored task has been achieved.

`AfterRoutines()`
In a group 2 action, controls output of 'after' messages.

`AllowPushDir()`
An object can be pushed from one location to another.

`Banner()`
Prints the game banner.

`ChangePlayer(object,flag)`
Player assumes the persona of the *object*. If the optional *flag* is `true`, room descriptions include "(as *object*)".

`CommonAncestor(object1,object2)`
Returns the nearest object which a parental relationship to both *objects*, or `nothing`.

`DictionaryLookup(byte_array,length)`
Returns address of word in dictionary, or 0 if not found.

`DrawStatusLine()`
Refreshes the status line.

`GetGNAOfObject(object)`
Returns gender-number-animation 0..11 of the *object*.

`HasLightSource(object)`
Returns `true` if the *object* has light.

`IndirectlyContains(parent_object,object)`
Returns `true` if *object* is currently a child or grand-child or great-grand-child... of the *parent_object*.

`IsSeeThrough(object)`
Returns `true` if light can pass through the *object*.

`Locale(object,"string1","string2")`
Describes the contents of *object*, and returns their number. After objects with own paragraphs, the rest are listed preceded by *string1* or *string2*.

`LoopOverScope(routine,actor)`
Calls *routine*`(`*object*`)` for each *object* in scope. If the optional *actor* is supplied, that defines the scope.

`MoveFloatingObjects()`
Adjusts positions of game's `found_in` objects.

`NextWord()`
Returns the next dictionary word in the input stream, incrementing `wn` by one. Returns `false` if the word is not in the dictionary, or if the input stream is exhausted.

`NextWordStopped()`
Returns the next dictionary word in the input stream, incrementing `wn` by one. Returns `false` if the word is not in the dictionary, –1 if the input stream is exhausted.

`NounDomain(object1,object2,type)`
Performs object parsing; see also `ParseToken()`.

`ObjectIsUntouchable(object,flag)`
Tests if there is a barrier – a container object which is not open – between player and *object*. Unless the optional *flag* is `true`, outputs "You can't because ... is in the way". Returns `true` is a barrier is found, otherwise `false`.

`OffersLight(object)`
Returns `true` if the *object* offers light.

`ParseToken(type,value)`
Performs general parsing; see also `NounDomain()`.

`PlaceInScope(object)`
Used in an `add_to_scope` property or `scope=` token to put the *object* into scope for the parser.

`PlayerTo(object,flag)`
Moves the player to *object*. Prints its description unless optional *flag* is 1 (no description) or 2 (as if walked in).

`PrintOrRun(object,property,flag)`
If *object*`.`*property* is a string, output it (followed by a newline unless optional *flag* is `true`), and return `true`. If it's a routine, run it and return what the routine returns.

`PronounNotice(object)`
Associates an appropriate pronoun with the *object*.

`PronounValue('pronoun')`
Returns the object to which `'it'` (or `'him'`, `'her'`, `'them'`) currently refers, or `nothing`.

`ScopeWithin(object)`
Used in an `add_to_scope` property or `scope=` token to put the contents of the *object* in scope for the parser.

`SetPronoun('pronoun',object)`
Defines the *object* to which a given *pronoun* refers.

`SetTime(expr1,expr2)`
Sets `the_time` to *expr1* (in mins 0..1439 since midnight), running at *expr2* (+ve: *expr2* minutes pass each turn; –ve: `-`*expr2* turns take one minute; zero: time stands still).

`StartDaemon(object)`
Starts the *object*'s daemon.

`StartTimer(object,expr)`
Starts the *object*'s timer, initialising its `time_left` to *expr*. The object's `time_out` property will be called after that number of turns have elapsed.

`StopDaemon(object)`
Stops the *object*'s daemon.

`StopTimer(object)`
Stops the *object*'s timer.

`TestScope(object,actor)`
Returns `true` if the *object* is in scope, otherwise `false`. If the optional *actor* is supplied, that defines the scope.

`TryNumber(expr)`
Parses word *expr* in the input stream as a number, recognising decimals, also English words one..twenty. Returns the number 1..10000, or -1000 if the parse fails.

`UnsignedCompare(expr1,expr2)`
Returns –1 if *expr1* is less than *expr2*, 0 if *expr1* equals *expr2*, and 1 if *expr1* is greater than *expr2*. Both expressions are unsigned, in the range 0..65535.

`WordAddress(expr)`
Returns a byte array contains the raw text of word *expr* in the input stream.

`WordInProperty(word,object,property)`
Returns `true` if the dictionary *word* is listed in the *property* values for the *object*.

`WordLength(expr)`
Returns the length of word *expr* in the input stream.

`WriteListFrom(object,expr)`
Outputs a list of *object* and its siblings, in the given style, an *expr* formed by adding any of: `ALWAYS_BIT`, `CONCEAL_BIT`, `DEFART_BIT`, `ENGLISH_BIT`, `FULLINV_BIT`, `INDENT_BIT`, `ISARE_BIT`, `NEWLINE_BIT`, `PARTINV_BIT`, `RECURSE_BIT`, `TERSE_BIT`, `WORKFLAG_BIT`.

`YesOrNo()`
Returns `true` if the player types "`YES`", `false` for "`NO`".

`ZRegion(arg)`
Returns the type of its *arg*: 3 for a string address, 2 for a routine address, 1 for an object number, or 0 otherwise.

2

•• 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.

4

Object attributes

`absent`
For a 'floating' object (one with a `found_in` property, which can appear in many rooms): is no longer there.

`animate`
For an object: is a living creature.

`clothing`
For an object: can be worn.

`concealed`
For an object: is present but hidden from view.

`container`
For an object: other objects can be put in (but not on) it.

`door`
For an object: is a door or bridge between rooms.

`edible`
For an object: can be eaten.

`enterable`
For an object: can be entered.

`female`
For an `animate` object: is female.

`general`
For an object or room: a general-purpose flag.

`light`
For an object or room: is giving off light.

`lockable`
For an object: can be locked; see the `with_key` property.

`locked`
For an object: can't be opened.

`male`
For an `animate` object: is male.

`moved`
For an object: is being, or has been, taken by the player.

`neuter`
For an `animate` object: is neither male nor female.

`on`
For a `switchable` object: is switched on.

`open`
For a `container` or `door` object: is open.

`openable`
For a `container` or `door` object: can be opened.

`pluralname`
For an object: is plural.

`proper`
For an object: the short name is a proper noun, therefore not to be preceded by "The" or "the".

`scenery`
For an object: can't be taken; is not listed in a room description.

`scored`
For an object: awards `OBJECT_SCORE` points when taken for the first time. For a room: awards `ROOM_SCORE` points when visited for the first time.

`static`
For an object: can't be taken.

`supporter`
For an object: other objects can be put on (but not in) it.

`switchable`
For an object: can be switched off or on.

`talkable`
For an object: can be addressed in "object, do this" style.

`transparent`
For a `container` object: objects inside it are visible.

`visited`
For a room: is being, or has been, visited by the player.

`workflag`
Temporary internal flag, also available to the program.

`worn`
For a `clothing` object: is being worn.

Optional entry points

These routines, if you supply them, are called when shown.

`AfterLife()`
The player has died. Setting `deadflag` to 0 resurrects her.

`AfterPrompt()`
The ">" prompt has been output.

`Amusing()`
The player has won and `AMUSING_PROVIDED` is defined.

`BeforeParsing()`
The parser has input some text, set up the buffer and parse tables, and initialised `wn` to 1, but done nothing else.

`ChooseObjects(`*`object`*`,`*`flag`*`)`
Parser has found "`ALL`" or an ambiguous noun phrase and decided that *`object`* should be excluded (`flag` is 0), or included (`flag` is 1). The routine should return: 0 to let this stand, 1 to force inclusion, or 2 to force exclusion. If `flag` is 2, the parser is undecided, and the routine should return an appropriate score 0..9.

`DarkToDark()`
The player has moved from one dark room to another.

`DeathMessage()`
The player has died and `deadflag` is 3 or more.

`GamePostRoutine()`
Called after all *`actions`*.

`GamePreRoutine()`
Called before all *`actions`*.

`Initialise()`
Mandatory; note British spelling: called at start. Must set `location`; can return 2 to suppress game banner.

`InScope()`
Called during parsing.

`LookRoutine()`
Called at the end of every `Look` description.

`NewRoom()`
Called when room changes, before description is output.

`ParseNoun(`*`object`*`)`
Called to parse the *`object`*'s name.

`ParseNumber(`*`byte_array`*`,`*`length`*`)`
Called to parse a number.

`ParserError(`*`number`*`)`
Called to handle an error.

`PrintRank()`
Completes the output of the score.

`PrintTaskName(`*`number`*`)`
Prints the name of the task.

`PrintVerb(`*`addr`*`)`
Called when an unusual verb is printed.

`TimePasses()`
Called after every turn.

`UnknownVerb()`
Called when an unusual verb is encountered.

5

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.

6