PunyInform v4.0 quick reference

More information on last page.

Library variables

`action`
The current action, e.g. `##PutOn`.

`actor`
The creature that is currently being ordered to do something – usually the player.

`buffer`
The array where player input is kept.

`consult_from`
The first word number of a topic in player input.

`consult_words`
The number of words of a topic in player input.

`deadflag`
0 = game is on, 1 = player is dead, 2 = player has won, 3+ = game is over - `DeathMessage()` prints why.

`inp1`
Usually equals `noun`, but is `1` if `noun` is a number.

`inp2`
Usually equals `second`, but is `1` if `second` is a number.

`inventory_stage`
See `invent` property.

`inventory_style`
0 = tall, 1 = wide. Can be set by game. If `OPTIONAL_FLEXIBLE_INVENTORY` has been defined, player can change it ("inventory tall/wide").

`keep_silent`
Set to `true` to make most group 2 actions silent.

`location`
The room where the player is, or `thedark` if it's dark.

`lookmode`
1 = normal, 2 = long, 3 = short. Long mode shows the room description on every visit, short never. 1 is default. Many games set it to 2 in `Initialise`.

`meta`
Is `true` if `action` is a meta action, like `##Save`.

`no_implicit_actions`
Set this to `true` to disable all implicit actions.

`normal_directions_enabled` **PUNY++**
Is normally `true`. Set it to `false` to stop the library from recognizing the normal directions (N, S, UP, IN etc).

`noun`
The primary object of the current action, if any.

`num_words`
The number of words in player input.

`parse`
The array to keep track of the words in player input.

`parser_action`
In a `parse_name` routine, set this to `##PluralFound` if a match was made and it's in plural form.

`player`
Points to the object that represents the player.

`real_location`
The room where the player is, even if it's dark.

`receive_action`
The action (`##Insert` or `##PutOn`) that caused the fake action `##Receive`.

`scope_modified` **PUNY++**
If `OPTIONAL_MANUAL_SCOPE` has been defined, set this to `true` whenever (possibly) changing what's in scope.

`scope_stage`
Used in scope token routines referred to in the grammar.

`score`
The current score.

`second`
The secondary object of the current action, if any.

`selected_direction` **PUNY++**
The direction matched in input, if any, e.g. `n_to`.

`ship_directions_enabled` **PUNY++**
If `OPTIONAL_SHIP_DIRECTIONS` is defined, set this to `true`/`false` to turn ship directions, e.g. `'aft'`, on/off.

`task_done`
A byte array to note which tasks have been completed. `Achieved` updates this. Requires `TASKS_PROVIDED`.

`task_scores`
A byte array with scores for tasks. Requires `TASKS_PROVIDED`.

`the_time`
The current time, in minutes past midnight. Only used for games that show time on the statusline.

`turns`
The game's turn counter.

`update_moved` **PUNY++**
Set this to `true` whenever manually moving an object to the player's inventory, to ensure the `moved` attribute is updated and the score is increased when applicable.

`verb_word`
The verb in the current input, e.g. `'take'`.

`wn`
Word number in player input where `NextWord()` will read, from 1 to (`num_words` - 1).

Library constants

The Inform language defines `true` (1), `false` (0) and `nothing` (0). The library also defines `NULL` (-1), which is used for an action, property or pronoun which currently doesn't have a value. `DIRECTION_COUNT` holds the number of directions recognized: Normally 8, but 12 if `OPTIONAL_FULL_DIRECTIONS` is defined. `PUNYINFORM_MAJOR_VERSION` and `PUNYINFORM_MINOR_VERSION` hold the library version.

User-defined constants

`AMUSING_PROVIDED`
Offer the player to see amusing things to try after they complete the game. See `Amusing` entry point routine.

CUSTOM_PLAYER_OBJECT = object **PUNY++**

The player object to use instead of selfobj.

DEATH_MENTION_UNDO **~PUNY**

Print 'undo' option even when player wins.

DEBUG

Use debug mode, with debug verbs and more information when something goes wrong.

DEFAULT_CAPACITY = number **PUNY++**

The number of objects that can be held by a container or supporter that doesn't provide a value for capacity.

Headline = "^text^^"

Information about the game - genre, author, credits etc. Start with "^", end with "^^", separate sections with "^".

INITIAL_LOCATION_VALUE = object **PUNY++**

The location in which the player starts. A game must either define this, or set location in Initialise.

MAX_CARRIED = number

Set the capacity of the player object (default 32).

MAX_FLOATING_OBJECTS = number **PUNY++**

The maximum number of objects that can have the found_in property (default 32).

MAX_SCOPE = number **PUNY++**

The maximum number of objects that can be in scope at one time (default 32).

MAX_SCORE = number

The maximum score the player can get.

MAX_TIMERS = number

The maximum number of objects which can have an active timer or daemon at the same time (default 32).

NO_PLACES

Don't define the verbs 'objects' and 'places'.

NO_SCORE = number **PUNY++**

Don't include a scoring mechanism. In z3 games, show the value of this constant as the score on the statusline.

NUMBER_TASKS = number

The number of scoring tasks (default 1). See TASKS_PROVIDED.

OBJECT_SCORE = number

The score for each object with the scored attribute the player picks up (default 4). Requires OPTIONAL_SCORED.

OPTIONAL_ALLOW_WRITTEN_NUMBERS **PUNY++**

Allow the player to use 'one' .. 'twenty' as numbers.

OPTIONAL_EXTENDED_METAVERBS **PUNY++**

Enable a set of nice-to-have metaverbs. See "Group 1 actions" in this document.

OPTIONAL_EXTENDED_VERBSET **PUNY++**

Enable a set of nice-to-have verbs.See "Group 2 actions" and "Group 3 actions" in this document.

OPTIONAL_FULL_DIRECTIONS **PUNY++**

Enable directions NE, NW, SE, SW.

OPTIONAL_FLEXIBLE_INVENTORY **PUNY++**

Allow player to change inventory style ("i tall/wide")

OPTIONAL_FULL_SCORE **PUNY++**

Enable the 'fullscore' verb, and optionally support for tasks (See TASKS_PROVIDED).

OPTIONAL_GUESS_MISSING_NOUN **PUNY++**

Make the parser fill in missing parts in player input.

OPTIONAL_MANUAL_REACTIVE **PUNY++**

The author takes responsibility for setting the reactive attribute on the right objects.

OPTIONAL_MANUAL_SCOPE **PUNY++**

The author takes responsibility for setting scope_modified = true when doing something in code which may affect scope.

OPTIONAL_NO_DARKNESS **PUNY++**

Assume there is always light. Don't define the light attribute.

OPTIONAL_ORDERED_TIMERS **PUNY++**

Timers/daemons are executed in order of their value for property timer_order.

OPTIONAL_PRINT_SCENERY_CONTENTS **PUNY++**

Make 'look' describe what's in/on containers/supporters which have the scenery attribute.

OPTIONAL_PROVIDE_UNDO **PUNY++**

Enable the 'undo' verb (z5 & z8 only!)

OPTIONAL_REACTIVE_PARSE_NAME **PUNY++**

Use the reactive attribute for all objects which have the parse_name property.

OPTIONAL_SCORED **PUNY++**

Enable support for the scored attribute.

OPTIONAL_SIMPLE_DOORS **PUNY++**

Lets you define property door_dir as an array and skip property door_to for simple two-way doors.

OPTIONAL_SHIP_DIRECTIONS **PUNY++**

Enable support for 'fore', 'aft' etc. Also see variable ship_directions_enabled.

OPTIONAL_SL_NO_MOVES **PUNY++**

Don't show moves on statusline (z5 & z8 only!).

OPTIONAL_SL_NO_SCORE **PUNY++**

Don't show score on statusline (z5 & z8 only!).

ROOM_SCORE = number

The score for each room with the scored attribute the player visits (default 5). Requires OPTIONAL_SCORED.

RUNTIME_ERRORS = number **PUNY++**

What to do when a runtime error occurs: 2 = Print error message, 1 = Print error code, 0 = Like 1 but also reduce checks for errors (default: 2 if DEBUG is defined, 1 if not)

SACK_OBJECT = object

The object the player will automatically try to put belongings into when the player's capacity has been reached.

STATUSLINE_SCORE **PUNY++**

Show score and moves, not time on the statusline.

STATUSLINE_TIME **PUNY++**

Show time, not score on the statusline.

Story = "text"

Mandatory: The name of the game.

TASKS_PROVIDED

Use tasks for scoring. Also define NUMBER_TASKS, byte array task_scores and entry point routine PrintTaskName.

Library routines

`Achieved(number)`

Complete scoring task `number`. Requires `TASKS_PROVIDED`.

`Banner()`

Print the game name, release, serial number etc. If `Initialise` returns `2`, this information isn't printed at game start, and you can call `Banner` later to print it.

`CommonAncestor(object1, object2)`

Return the nearest object that contains both `object1` and `object2` on some level, or `false`.

`DrawStatusLine()`

Print the statusline, in a z5 or z8 game.

`FastSpaces(number)` **PUNY++**

Print `number` spaces in an efficient manner.

`ImplicitDisrobeIfWorn(object)` **PUNY++**

Take off object if worn. Return false for success.

`ImplicitGrabIfNotHeld(object)` **PUNY++**

Take object if not held. Return false for success.

`IndirectlyContains(object1, object2)`

Return `true` if `object1` holds `object2`, on some level.

`LoopOverScope(routine, actor)`

Call `routine` once for each object in scope for `actor` (default: `player`), passing the object as a parameter.

`MoveFloatingObjects()`

Check which objects with `found_in` should be present.

`NextWord()`

Read word at position `wn` in player input and increment `wn`. If no word read or word not in dictionary, return false. Otherwise, return the word.

`NextWordStopped()`

Like `NextWord()` except it returns `-1` if `wn` points beyond the end of player input.

`NumberWord(numword)`

If `numword` is a word, e.g. `'six'`, representing a number 1-20, return the number. If not, return `false`. Requires `OPTIONAL_ALLOW_WRITTEN_NUMBERS`.

`NumberWords()`

Return the number of words in player input.

`ObjectCapacity(object)` **PUNY++**

Return the object's value for the `capacity` property, defaulting to `DEFAULT_CAPACITY` or 100 if not defined.

`ObjectIsUntouchable(object, flag)`

Return `true` if `player` can't touch `object`. If so, and `flag` is `false` or unspecified, also print a message.

`ParseToken(type, data)`

Use in general parsing routines. See DM4 index for examples.

`PlaceInScope(object)`

Use in user-supplied scope routines to put `object` in scope. Ignores `add_to_scope` property and children.

`PlayerTo(object, flag)`

Move player to `object`, which must be a room or an enterable object. If `flag` is `0` or omitted, print a long room description. If `flag` is `1`, keep quiet. If `flag` is `2`, print a room description based on the `lookmode` value.

`PrintContents(text, object, style)` **PUNY++**

Recursively list contents of `object`. Hide items that have `concealed` or `scenery`, unless `action` is `##Inv`.. Print or run `text` (unless 0) before first item. If `style` has `WORKFLAG_BIT` set, only print objects which have the `workflag` attribute. If style has `ISARE_BIT` set, print "is" or "are" before list. If it has `NEWLINE_BIT` set, print each object on a new line. Return `true` if any objects were listed. Call with `text==1` to not print anything but return `0` if `object` contains no printable objects, `1` if contents can be prefixed with "is", `2` for "are".

`PrintMsg(msg, arg1, arg2)` **PUNY++**

Print library message `msg` . Some messages need an argument or two, use `arg1` and `arg2` for this.

`PrintOrRun(object, property, flag)`

If `object.property` holds a routine or list of routines, run them using `RunRoutines(object, property)`. If it holds a string, print it and then a newline. If `flag` is `true`, skip the newline.

`PronounNotice(object)`

Make a pronoun (`'it'`, `'her'` etc) refer to `object`.

`RunRoutines(object, property, switch)`

If `object.property` holds a routine or list of routines, run each routine until one of them returns a non-zero value. Return the return value of the last routine run. If `switch` has a non-zero value, the routines can have switch-clauses to match this value, otherwise they can have switch-clauses to match `action`.

`RunTimeError(number)` **PUNY++**

Print a runtime error.

`ScopeWithin(object)`

Use in user-supplied scope routines. Put all items which are in `object` into scope, also recursively searching supporters, transparent objects and open containers, and check the `add_to_scope` property of all objects added.

`SetTime(number, step)`

Sets the time to `number` minutes after midnight. If `step` is a positive number, one turn takes `step` minutes. Otherwise, there are `-step` turns to a minute.

`TestScope(object, actor)`

Return `true` if `object` is in scope for `actor` (defaults to `player`).

`TryNumber(wordnum)`

Try to parse word `wordnum` in player input as a number. If successful, return the number (0-10000, higher values are returned as 10000). If it's not a valid number, return -1000. If `OPTIONAL_ALLOW_WRITTEN_NUMBERS` is defined, also parse number words (`'one'` .. `'twenty'`).

`WordAddress(wordnum)`

Return the address in memory where the characters for word `wordnum` in player input are stored.

`WordLength(wordnum)`

Return the number of characters in word `wordnum` in player input.

`WordValue(wordnum)`

Return the dictionary word that word `wordnum` in player input matches, or `0` if no match was found.

`YesOrNo()`

Wait for the player to type something. Return `true` if they typed yes, `false` if they typed no, or ask again.

Printing rules

A printing rule is used to print something based on one argument, typically an object. Example of use:
`print "The pump is ", (OnOff) Pump, ".^";`

`CObjIs` **PUNY++**
Prints "The (object) ", and "is" or "are" (see `IsOrAre`).

`CTheyreorThats`
Prints "That's" or "They're" or "He's" etc.

`IsOrAre`
Prints "is" or "are", based on `pluralname` and if the object is the player object.

`ItorThem`
Prints "it", "them", "her" etc.

`OnOff` **PUNY++**
Prints "on" or "off", based on `on`.

`ThatOrThose`
Prints "that" or "those" based on `pluralname`.

Entry point routines

These routines, if defined by the game author, are run under the circumstances stated for each routine.

`AfterLife()`
When player has died. Can be used to revive player.

`AfterPrompt()`
After the input prompt has been printed.

`Amusing()`
When the player has won. Use it to print fun facts about the game. Requires `AMUSING_PROVIDED`.

`BeforeParsing()`
After player input, before parsing starts.

`ChooseObjects(obj, code)`
If `code` is 2, return a score 0-9 for how good a fit `obj` is for action `action_to_be`. Code 0 and 1 means player has typed ALL and parser means to exclude (0) or include (1) the object. Return 0 to don't interfere, 1 to force inclusion, 2 to force exclusion.

`DarkToDark()`
When player moves from one dark location to another.

`DeathMessage()`
When game ends and `deadflag` is set to 3 or higher. Should print a few words to say why the game ended.

`DebugParseNameObject(object)` **PUNY++**
When the parser checks for matching objects for a debug verb like `'purloin'`. Return true if `object` has a `parse_name` routine.

`DisallowTakeAnimate()` **PUNY++**
When the player tries to take `noun`, which has `animate`. Return `false` to allow this, or true to disallow it.

`GamePostRoutine()`
When `after` routines have been run.

`GamePreRoutine()`
Before `before` routines have been run.

`InScope(actor)`
When working out the scope for the actor. Call `ScopeWithin` and `PlaceInScope` to add objects to scope. Return true if no other objects should be in scope.

`Initialise` **~PUNY**
Mandatory: A (possibly empty) routine which is called when the game starts. May print an introduction. May return 2 to skip the game banner. Must set `location`, unless `INITIAL_LOCATION_VALUE` is defined.

`LibraryMessages(number, arg1, arg2)` **PUNY++**
When a library message is about to be printed. Use it to print your own complex library messages.

`LookRoutine`
After the room and everything in it has been described.

`NewRoom`
When the player has entered a new room, before the room is described.

`ParseNoun(object)` **~PUNY**
When checking if input matches `object`, before `parse_name` and `name` properties are checked. Can advance `wn` and return `-1` to consume words (`parse_name` + `name` will also be checked), just return `-1` to not interfere, or return how many words matched.

`ParseNumber(buffer, length)`
When the parser needs to check if the input word that starts at `buffer` and is `length` bytes long, is a number. Return the number, or `false` if no number was found.

`PrintRank()`
When the scoring message is printed. Prints the final part, typically giving the player a rank based on `score`.

`PrintTaskName(n)`
When listing a completed task. Print name of task `n`.

`PrintVerb(verb)`
When the parser needs to print a verb. Typically needed for long verbs. Return true if the routine printed the verb.

`TimePasses()`
After a game turn has ended, in which `turns` increased.

`UnknownVerb(word)`
When the parser doesn't recognize the verb. Return a dictionary word to use as the verb instead, or `false`.

Library objects

`Directions` **PUNY++**
Represents all directions. The `selected_direction` variable holds the matched direction (e.g. `n_to`) or 0.

`selfobj`
The default player object. It's better to use the `player` variable which normally refers to this.

`thedark`
A fake room which `location` points to when it's dark in `real_location`. The player is never moved to `thedark`.

Group 1 actions

Group 1 actions are metaverbs that control gameplay, and debug verbs. They don't run `before` or `after` routines.

`Again`, `FullScore`, `LookModeLong`, `LookModeNormal`, `LookModeShort`, `NotifyOff`, `NotifyOn`, `Oops`, `OopsCorrection`, `Restart`, `Restore`, `Save`, `Score`, `Version`, `Quit`

`OPTIONAL_EXTENDED_METAVERBS` adds: `CommandsOff`, `CommandsOn`, `CommandsRead`, `Objects`, `Places`, `ScriptOff`, `ScriptOn`, `Verify`

`DEBUG` adds: `ActionsOff`, `ActionsOn`, `Debug`, `GoNear`, `Pronouns`, `RandomSeed`, `RoutinesOff`, `RoutinesOn`, `Scope`, `Purloin`, `TimersOff`, `TimersOn`, `Tree`

Group 2 actions

These are actions which the library knows how to perform. They change something in the game world or prints important information about it. They run both `before` and `after` routines.

`Close`	"CLOSE (noun)"
`Disrobe`	"TAKE OFF (noun)"
`Drop`	"DROP (noun)"
`Eat`	"EAT (noun)"
`Enter`	"ENTER (noun)"
`Examine`	"EXAMINE (noun)"
`Exit`	"EXIT (noun)"
`GetOff`	"GET OFF (noun)"
`Go`	"GO (direction)"
`Insert`	"INSERT (noun) INTO (second)"
`Inv`	"INVENTORY"
`Lock`	"LOCK (noun) with (second)"
`Look`	"LOOK"
`Open`	"OPEN (noun)"
`PutOn`	"PUT (noun) ON (second)"
`Remove`	"REMOVE (noun) FROM (second)"
`Search`	"SEARCH (noun)"
`SwitchOff`	"SWITCH OFF (noun)"
`SwitchOn`	"SWITCH ON (noun)"
`Take`	"TAKE (noun)"
`Transfer`	"TRANSFER (noun) TO (second)"
`Unlock`	"UNLOCK (noun) WITH (second)"
`Wait`	"WAIT"
`Wear`	"WEAR (noun)"

OPTIONAL_EXTENDED_VERBSET adds:

`Empty`	"EMPTY (noun)"
`EmptyT`	"EMPTY (noun) INTO (second)"
`GoIn`	"INSIDE"

Group 3 actions

These actions normally don't do anything, except print a standard message. They run `before` routines but not `after` routines.

`Answer`	"ANSWER (topic) TO (second)"
`Ask`	"ASK (noun) ABOUT (topic)"
`AskFor`	"ASK (noun) FOR (second)"
`AskTo`	"ASK (noun) TO (topic)"
`Attack`	"ATTACK (noun)"
`Climb`	"CLIMB (noun)"
`Consult`	"CONSULT (noun) ABOUT (topic)"
`Cut`	"CUT (noun)"
`Dig`	"DIG (noun)"
`Drink`	"DRINK (noun)"
`Fill`	"FILL (noun)"
`Give`	"GIVE (noun) to (second)"
`Jump`	"JUMP"
`JumpOver`	"JUMP OVER (noun)"
`Listen`	"LISTEN TO (noun)"
`Pull`	"PULL (noun)"
`Push`	"PUSH (noun)"
`PushDir`	"PUSH (noun) (direction)"
`Rub`	"RUB (noun)"
`Shout`	"SHOUT (topic)"
`ShoutAt`	"SHOUT (topic) AT (second)"
`Show`	"SHOW (noun) (second)"
`Smell`	"SMELL (noun)"
`Tell`	"TELL (noun) ABOUT (topic)"
`ThrowAt`	"THROW (noun) AT (second)"
`Tie`	"TIE (noun) TO (second)"
`Touch`	"TOUCH (noun)"
`Turn`	"TURN (noun)"

OPTIONAL_EXTENDED_VERBSET adds:

`Blow`	"BLOW (noun)"
`Mild`	"DARN"
`Burn`	"BURN (noun)"
`Buy`	"BUY (noun)"
`Kiss`	"KISS (noun)"
`No`	"NO"
`Set`	"SET (noun)"
`SetTo`	"SET (noun) TO (special)"
`Strong`	"SHIT"
`Sing`	"SING"
`Sleep`	"SLEEP"
`Sorry`	"SORRY"
`Squeeze`	"SQUEEZE (noun)"
`Swim`	"SWIM"
`Swing`	"SWING (noun)"
`Taste`	"TASTE (noun)"
`Think`	"THINK"
`Transfer`	"TRANSFER (noun) TO (second)"
`Wake`	"WAKE UP"
`WakeOther`	"WAKE UP (noun)"
`Wave`	"WAVE"
`Yes`	"YES"

Fake actions

These actions are not referred to anywhere in the grammar, and they don't have action routines, e.g. the fake action `Going` has no action routine `GoingSub`.

`Going`	Sent to the `before` routine for the room that the player is about to enter.
`LetGo`	Sent to the container/supporter from which the player takes something.
`NotUnderstood`	Sent to creature's `orders` when player issued an incomprehensible order to it.
`Order`	Sent to creature's `life` when player issued an order to it, and `orders` didn't handle it.
`PluralFound`	A `parse_name` routine can set `parser_action` to this value when a match is found and it's in plural.
`Receive`	Sent to the object the player tries to place something in/on. `receive_action` holds the original action.
`ThrownAt`	Sent by action `ThrowAt` to the object the player tries to throw something at.

Object attributes (flags)

An attribute is a flag which can be on or off.
[OBJ]means this is used for regular objects.
[ROOM] means this is used for rooms.

`absent [OBJ]`
For object with `found_in`: Removed from game for now.

`animate [OBJ]`
Is a living thing, can be talked to etc.

`clothing [OBJ]`
Can be worn.

`concealed [OBJ]`
Is visible but not easy to spot, like a secret door. Can be interacted with but is not printed in room description.

`container [OBJ]`
Objects can be put in it and removed from it, if it's `open`. Can't also have `supporter`. Can have `enterable`.

`door [OBJ]`
Is a portal between rooms. Use properties `door_to`, `door_dir` and, unless it's a one-way door, `found_in`.

`edible [OBJ]`
Can be eaten.

`enterable [OBJ]`
Can be entered. Must have `container` or `supporter`.

`female [OBJ]`
Can be referred to as she/her. Must have `animate`.

`general [OBJ] [ROOM]`
To be used by the game author for whatever they like.

`light [OBJ] [ROOM]`
Provides light. For room and container, lights up what's inside as well. Note: This attribute is not defined if `OPTIONAL_NO_DARKNESS` is defined.

`lockable [OBJ]`
Can be locked and unlocked, using the object specified by `with_key` property.

`locked [OBJ]`
Can't be opened.

`moved [OBJ]`
Is or has been held directly by the player.

`neuter [OBJ]`
Can be referred to as "it" (Mainly used for `animate` objects, as this is default behaviour for non-animates).

`on [OBJ]`
Is currently switched on. See `switchable` attribute.

`open [OBJ]`
For doors and containers: Is currently open.

`openable [OBJ]`
For doors and containers: Can be opened and closed.

`pluralname [OBJ]`
Can be referred to as they/them.

`proper [OBJ]`
Has a name which should never be preceded by an article, like "John".

`reactive [OBJ] [ROOM]` **PUNY++**
The object provides at least one of `add_to_scope`, `each_turn`, `react_before`, `react_after` (+ `parse_name` if `OPTIONAL_REACTIVE_PARSE_NAME` is defined). Note: unless `OPTIONAL_MANUAL_REACTIVE` is defined, the `reactive` attribute is set automatically.

`scenery [OBJ]`
Can't be taken, is not mentioned in room descriptions.

`scored [OBJ] [ROOM]`
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. Note: Only defined if `OPTIONAL_SCORED` is defined and `NO_SCORE` is not.

`static [OBJ]`
Can't be taken.

`supporter [OBJ]`
Is a supporter, meaning things can be placed on top of it. Can't also have `container`. Can have `enterable`.

`switchable [OBJ]`
Can be switched on and off. The `on` attribute tells its current state.

`talkable [OBJ]`
Can be talked to, even though it's not `animate`.

`transparent [OBJ]`
For a `container`: The contents are visible even if the container is closed. For an `animate` object: Held objects are visible to others. For other objects: Objects that are part of this objects (i.e. are *inside* this object) are visible.

`visited [ROOM]`
The player has seen this room.

`workflag [OBJ] [ROOM]`
Temporary internal flag. Can be used by game code too.

`worn [OBJ]`
For object that has `clothing`: Is currently being worn.

Object properties

A property is a 16-bit value or a list of values.
[OBJ] means this is used for regular objects.
[ROOM] means this is used for rooms.
(+) means "additive" - if an object which defines the property inherits from a class which also defines the property, it gets both values.

`add_to_scope [OBJ]`
A list of objects that should be added to scope when this object is in scope, or a routine which puts objects in scope using `ScopeWithin` and `PlaceInScope`.

`after [OBJ] [ROOM] (+)`
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 property value is a routine, which usually has sections like switch-clauses, each listing one or more actions, a colon and the code to run. There can be a `default` clause which runs if nothing else was matched. There can also be code before the first clause, which will run regardless of action. 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 [OBJ]` **~PUNY**
A string or a routine to print the indefinite article for the object name.The default article is "some" for objects that have `pluralname`, nothing for objects that have `proper`, and "a" for all others.

`before [OBJ] [ROOM] (+)`
Like `after`, but is run before the action happens. Returning true stops the default action from happening at all.

`cant_go [ROOM]`
A string or a routine to print a message, when the player tries to go in a direction where there's no exit.

`capacity [OBJ]` **~PUNY**
The maximum number of items that can be in this container, on this supporter or held by this actor. To read the capacity of an object, taking the default capacity into consideration, you must call `ObjectCapacity(object)`.

`d_to [ROOM]`
Holds a possible exit. The value can be any of:
* `false` – not an exit
* a room where the exits leads
* a door object – the exit leads through this door
* a string saying why the player can't go there
* a routine which either returns `false`, a room, a door object, or prints its own message and returns `true`.

`daemon [OBJ] [ROOM]`
A routine that is executed every turn once it is started. Use `StartDaemon` and `StopDaemon` to start/stop it.

`describe [OBJ] (+)` **~PUNY**
A string or a routine to print a paragraph of text for an object in a room description. If it's a string or it's a routine which returns `true`, the object won't be further described. Not supported for rooms as in DM4. Note: Start and end the description with a newline ("^").

`description [OBJ] [ROOM]`
For an object: A string or a routine to print the text the player should get when examining the object.
For a room: A string or a routine to print the room description.

`door_dir [OBJ]` **~PUNY**
For a door: A direction (e.g. `n_to`) or a routine returning a direction. This says in which direction the door lies in `location`. If `OPTIONAL_SIMPLE_DOORS` is defined and `found_in` holds a list with two rooms, `door_dir` can be a list of two directions.

`door_to [OBJ]` **~PUNY**
For a door: A room or a routine returning a room. This says where the door leads, when the player is in `location`. If `OPTIONAL_SIMPLE_DOORS` is defined and `found_in` holds a list with two rooms, `door_to` can be omitted.

`e_to [ROOM]`
An exit property. See `d_to`.

`each_turn [OBJ] [ROOM] (+)`
A routine which is executed every turn when the object is in scope.

`found_in [OBJ]`
A list of rooms where the object is present, or a routine which returns `true` if the object is present in `location`. If the object has `absent`, it's not present anywhere.

`grammar [OBJ]`
For `animate` or `talkable` objects: Called when object is spoken to. Can advance `verb_wordnum`. Return `true` if routine has parsed all input and set up `action`, `noun` and `second`. Return 'verb' to use this verb's grammar instead, or -'verb' to use this verb's grammar but fall back to the verb in player input if parsing fails. Return `false` to parse as usual.

`in_to [ROOM]`
An exit property. See `d_to`.

`initial [OBJ] [ROOM]`
For an object: A string or a routine to describe the object before it's been picked up. Note: For doors/containers and switchable objects, use `when_open` + `when_closed` and `when_on` + `when_off` respectively.
For a room: A string or a routine to print a text when the player enters the room.

`inside_description [OBJ]`
For an enterable object: A string or a routine that will printed/run when the player is in/on the object.

`invent [OBJ]`
A routine to print the object in a list (typically in player inventory or a room description). First the routine is called before the object name has been printed, with `inventory_stage` set to 1. Then it's called again when the object name has been printed but no additional information (e.g. "(providing light)"), with `inventory_stage` set to 2.
For both calls, the routine should return `false` to continue or `true` to stop all further output.

`life [OBJ] (+)`
For `animate` objects: Works like a `before` routine, but receives only person-to-person actions (`Answer`, `Ask`, `Attack`, `Give`, `Kiss`, `Order`, `Show`, `Tell`, `ThrowAt`, `WakeOther`). Can be a string instead of a routine.

`n_to [ROOM]`
An exit property. See `d_to`.

`name [OBJ] [ROOM]`

A list of dictionary words. For an object, these are the words that can be used to refer to the object. For a room, these are words which should yield a reply like "You don't need to refer to that.". For an object, but not a room, the `name` property can be overridden by the `parse_name` property.

`ne_to [ROOM]`

An exit property. See `d_to`. Requires `OPTIONAL_FULL_DIRECTIONS` to work.

`nw_to [ROOM]`

An exit property. See `d_to`. Requires `OPTIONAL_FULL_DIRECTIONS` to work.

`orders [OBJ]`

For `animate` or `talkable` objects: A routine to carry out the player's orders or decline to do so. The routine should either return `false`, or print a message and return `true` to stop further processing. The player object's `orders` routine is called first, and then the addressed object's `orders` routine is called.

`out_to [ROOM]`

An exit property. See `d_to`.

`parse_name [OBJ]`

A routine to parse player input and decide if it matches this object. The routine calls `NextWord()` and/or `NextWordStopped()` to read words and returns the number of words that match, `0` for no match or `-1` to say it chooses not to decide (i.e. the `name` property will be consulted, if provided).

`react_after [OBJ]`

Like `after`, but receives all actions taking place when this object is in scope.

`react_before [OBJ]`

Like `before`, but receives all actions taking place when this object is in scope.

`s_to [ROOM]`

An exit property. See `d_to`.

`se_to [ROOM]`

An exit property. See `d_to`. Requires `OPTIONAL_FULL_DIRECTIONS` to work.

`short_name [OBJ]`

A string or routine to print the short name of the object, overriding the name provided in the object's name string. The routine should return `true` to signal that it has printed the name, or `false` to say that the library should still print the object's name string. Sometimes it's useful to print a prefix (e.g. an adjective) and return `false`.

`sw_to [ROOM]`

An exit property. See `d_to`. Requires `OPTIONAL_FULL_DIRECTIONS` to work.

`time_left [OBJ] [ROOM]`

For an object which has a `time_out` property: After `StartTimer(object, turns)` has been called, `time_left` holds the number of turns left before `time_out` will be called.
For other objects/rooms: Use it as a general variable.

`time_out [OBJ] [ROOM] (+)`

A routine to be called when a timer times out. Start the countdown with `StartTimer(object, turns)`.

`timer_order [OBJ] [ROOM]` **PUNY++**

A number that determines when this object's `timer/daemon` executes relative to other timers/daemons. The lower the earlier. Objects that don't provide it have the value 100. Requires `OPTIONAL_ORDERED_TIMERS`.

`u_to [ROOM]`

An exit property. See `d_to`.

`w_to [ROOM]`

An exit property. See `d_to`.

`when_closed [OBJ]`

For doors and containers: A string or routine to describe the object when it's closed.

`when_off [OBJ]`

For a switchable object: A string or routine to describe the object when it's off.

`when_on [OBJ]`

Like `when_off`, but for when it's on. Not used if the object has `moved`.

`when_open [OBJ]`

Like `when_closed`, but for when it's open. Not used if the object has `moved`.

`with_key [OBJ]` **~PUNY**

For `lockable` objects: The object which works as a key, or a routine which returns `true` if the object held in `second` works as a key.

About this document

This is meant to be printed out and serve as a quick index to all the functionality that the PunyInform library provides. A similar document for the Inform 6 language is Inform in Four Minutes, available at http://www.firthworks.com/roger/

This is not meant to be a document from which to learn PunyInform, a replacement for The Inform Designer's Manual, Fourth Edition (DM4) or the PunyInform manual and tutorials. To keep it short, this document leaves out finer details. Always consult DM4 and/or the PunyInform manual to get the full picture (See Legend below).

Created and maintained by Fredrik Ramsberg. Improvement suggestions by Garry Francis, Johan Berntsson and Nick Moffitt.

Based on InfoLib at Your Fingertips by Roger Firth.

Legend

Items marked **PUNY++** aren't described in DM4. Items marked **~PUNY** don't work exactly as described in DM4. See PunyInform manual for details on these items.