INFORM 7

The Program

Appendix A

Build 6F95 Graham Nelson

Inform is a natural-language design system for interactive fiction, first created in 1993. To most users it seems a single unified tool, but in fact is made up of core software, common to all platforms, combined with substantial user interfaces written independently for Mac OS X, Windows and Linux, and with documentation and examples. The core material is in turn divided into:

Chapters 1 to 14: the source code to the NI compiler, written in C

Appendix A: the Standard Rules, written in Inform 7

Appendix B: the template layer, written in Inform 6

Each of these chapter blocks is divided up into one or more named sections, which have both full names ("Grammar Lines") and abbreviations (`12/gl`, the 12 signifying Chapter 12). Finally, each section is divided into numbered "paragraphs", some named and others not. Code can thus be approximately located by "postal codes" such as `12/gl`.§7.

Of its nature, Inform must perform a computational task which is difficult to specify formally, particularly since part of its aim is to cope well with incorrect input from an inexperienced user. It has become a complex program some 120,000 lines in length, and like all such it must mitigate its complexity using internal stylistic conventions and principles of organisation. To this end it follows the "literate programming" dogma of Donald Knuth, an idea which had in any case influenced Inform's own design. In LP, a single source ("web") is both "tangled" into a functional form and also "woven" into a typeset form suitable for human readers. Inform uses its own LP tool, `inweb`, an adaptation of Knuth's `CWEB` which scales better to large multi-target projects.

The Inform project's main goal is to publish the entire core Inform code base, beginning in April 2008 with public drafts of Appendices A and B, approximately 600pp of material. These use only the simplest form of LP where tangling is minimal, and the reader needs no previous experience of the genre.

A The Standard Rules

`A/sr0`: *SR0 - Preamble.w* The titling line and rubric, use options and a few other technicalities before the Standard Rules get properly started.

`A/sr1`: *SR1 - Physical World Model.w* Verbal descriptions of spatial relationships; the hierarchy of kinds of object, and their properties.

`A/sr2`: *SR2 - Variables and Rulebooks.w* The global variables and those built-in rulebooks which do not belong either to specific actions or to specific activities.

`A/sr3`: *SR3 - Activities.w* The built-in activities and their default stock of rules; the locale description mechanism.

`A/sr4`: *SR4 - Actions.w* The standard stock of actions supplied with Inform, along with the rules which define them; and the Understand grammar which corresponds to them.

`A/sr5`: *SR5 - Phrase Definitions.w* The phrases making up the Inform language, and in terms of which all other phrases and rules are defined; and the final sign-off of the Standard Rules extension, including its minimal documentation.

SR0 - Preamble A/sr0

Purpose

The titling line and rubric, use options and a few other technicalities before the Standard Rules get properly started.

§**1.** The Standard Rules are like a boot program for a computer that is starting up: at the beginning, the process is delicate, and the computer needs a fairly exact sequence of things to be done; halfway through, the essential work is done, but the system is still too primitive to be much use, so we begin to create convenient intermediate-level code sitting on top of the basics; so that, by the end, we have a fully flexible machine ready to go in any number of directions. In this commentary, we try to distinguish between what must be done (or else NI will crash, or fail in some other way) and what is done simply as a design decision (to make the Inform language come out the way we want it). Quite interesting hybrid Informs could be built by making different decisions. Still, our design is not entirely free, since it interacts with the I6 template layer (the I7 equivalent of the old I6 library): a really radical alternate Inform would need a different template layer, too.

§**2. Title.** Every Inform 7 extension begins with a standard titling line and a rubric text, and the Standard Rules are no exception:

```
Version 2/090402 of the Standard Rules by Graham Nelson begins here.

"The Standard Rules, included in every project, define the basic framework
of kinds, actions and phrases which make Inform what it is."
```

§**3. Starting up.** The following block of declarations is actually written by `indoc` and modified each time we alter the documentation. It's a dictionary of symbolic names like `HEADINGS` to HTML page leafnames like `doc71`.

...and so on...

§**4.** The following has no effect, and exists only to be a default non-value for "use option" variables, should anyone ever create them:

```
Use ineffectual translates as (- ; -).
```

§**5.** Inform source text has a core of basic computational abilities, and then a whole set of additional elements to handle IF. We want all of those to be used, so:

```
Use interactive fiction language elements.
```

§6. Some Inform 7 projects are rather heavy-duty by the expectations of the Inform 6 compiler (which it uses as a code-generator): I6 was written fifteen years earlier, when computers were unimaginably smaller and slower. So many of its default memory settings need to be raised to higher maxima.

Note that the Z-machine cannot accommodate more than 255 verbs, so this is the highest `MAX_VERBS` setting we can safely make here.

```
Use MAX_ARRAYS of 1500.
Use MAX_CLASSES of 200.
Use MAX_VERBS of 255.
Use MAX_LABELS of 10000.
Use MAX_ZCODE_SIZE of 100000.
Use MAX_STATIC_DATA of 180000.
Use MAX_PROP_TABLE_SIZE of 200000.
Use MAX_INDIV_PROP_TABLE_SIZE of 20000.
Use MAX_STACK_SIZE of 65536.
Use MAX_SYMBOLS of 20000.
Use MAX_EXPRESSION_NODES of 256.
```

§7. These, on the other hand, are settings used by the dynamic memory management code, which runs in I6 as part of the template layer. Each setting translates to an I6 constant declaration, with the value chosen being substituted for `{N}`.

The "dynamic memory allocation" defined here is slightly misleading, in that the memory is only actually consumed in the event that any of the kinds needing to use the heap is actually employed in the source text being compiled. (8192 bytes may not sound much these days, but in the tight array space of the Z-machine it's quite a large commitment, and we want to avoid it whenever possible.)

```
Use dynamic memory allocation of at least 8192 translates as
    (- Constant DynamicMemoryAllocation = {N}; -).
Use maximum indexed text length of at least 1024 translates as
    (- Constant IT_MemoryBufferSize = {N}+3; -).

Use dynamic memory allocation of at least 8192.
```

§8. This setting is to do with the Inform parser's handling of multiple objects.

```
Use maximum things understood at once of at least 100 translates as
    (- Constant MATCH_LIST_WORDS = {N}; -).

Use maximum things understood at once of at least 100.
```

§**9.** Finally, some definitions of miscellaneous options: none are used by default, but all translate into I6 constant definitions if used. (These are constants whose values are used in the I6 library or in the template layer, which is how they have effect.)

```
Use American dialect translates as (- Constant DIALECT_US; -).
Use the serial comma translates as (- Constant SERIAL_COMMA; -).
Use full-length room descriptions translates as (- Constant I7_LOOKMODE = 2; -).
Use abbreviated room descriptions translates as (- Constant I7_LOOKMODE = 3; -).
Use memory economy translates as (- Constant MEMORY_ECONOMY; -).
Use authorial modesty translates as (- Constant AUTHORIAL_MODESTY; -).
Use no scoring translates as (- Constant NO_SCORING; -).
Use command line echoing translates as (- Constant ECHO_COMMANDS; -).
Use undo prevention translates as (- Constant PREVENT_UNDO; -).
Use predictable randomisation translates as (- Constant FIX_RNG; -).
Use fast route-finding translates as (- Constant FAST_ROUTE_FINDING; -).
Use slow route-finding translates as (- Constant SLOW_ROUTE_FINDING; -).
Use numbered rules translates as (- Constant NUMBERED_RULES; -).
Use telemetry recordings translates as (- Constant TELEMETRY_ON; -).
Use no deprecated features translates as (- Constant NO_DEPRECATED_FEATURES; -).
Use VERBOSE room descriptions translates as (- Constant DEFAULT_VERBOSE_DESCRIPTIONS; -).
Use BRIEF room descriptions translates as (- Constant DEFAULT_BRIEF_DESCRIPTIONS; -).
Use SUPERBRIEF room descriptions translates as (- Constant DEFAULT_SUPERBRIEF_DESCRIPTIONS; -).
```

SR1 - Physical World Model A/sr1

Purpose

Verbal descriptions of spatial relationships; the hierarchy of kinds of object, and their properties.

§**1.** Although at this point we can freely use the exceptional fixed-form command sentences – like "Use ⟨use-option⟩" – since their wording is built into NI, we are very limited in the general assertion sentences we can make. Only two verbs are built in, and this is because they require rather special handling: *to be*, since it is both fundamental and irregular; and *to have*, since although regular is it used also as an auxiliary coupled with *to be* (as in "if Daphne has been in the garden"). And no prepositions exist at all.

Well, we can't do much in talking about the world model without the fundamental spatial relationships of being inside, on top of and so forth. While the relations themselves are built-in to NI, and so are their one-word names (such as "containment"), the rest is down to us:

```
Part SR1 - The Physical World Model

Section SR1/0 - Language

The verb to relate (he relates, they relate, he related, it is related,
he is relating) implies the universal relation.

The verb to provide (he provides, they provide, he provided, it is provided,
he is providing) implies the provision relation.

The verb to be in implies the reversed containment relation.
The verb to be inside implies the reversed containment relation.
The verb to be within implies the reversed containment relation.
The verb to be held in implies the reversed containment relation.
The verb to be held inside implies the reversed containment relation.

The verb to contain (he contains, they contain, he contained, it is contained,
he is containing) implies the containment relation.
The verb to be contained in implies the reversed containment relation.

The verb to be on implies the reversed support relation.
The verb to be on top of implies the reversed support relation.

The verb to support (he supports, they support, he supported, it is supported,
he is supporting) implies the support relation.
The verb to be supported on implies the reversed support relation.

The verb to incorporate (he incorporates, they incorporate, he incorporated,
it is incorporated, he is incorporating) implies the incorporation relation.
The verb to be part of implies the reversed incorporation relation.
The verb to be a part of implies the reversed incorporation relation.
The verb to be parts of implies the reversed incorporation relation.
```

§**2.** The enclosure relation, indirectly defined in terms of the above more fundamental ones, has a verb but no prepositions (though of course "to be enclosed by" is in effect a prepositional expression of this).

```
The verb to enclose (he encloses, they enclose, he enclosed, it is enclosed,
he is enclosing) implies the enclosure relation.
```

§**3.** Those three relations expressed how the inanimate world is arranged, on the small scale: the relationships become a little more complicated once living beings are involved.

```
The verb to carry (he carries, they carry, he carried, it is carried, he is
carrying) implies the carrying relation.
The verb to hold (he holds, they hold, he held, it is held, he is holding)
implies the holding relation.
The verb to wear (he wears, they wear, he wore, it is worn, he is wearing)
implies the wearing relation.
```

§**4.** One living being is special to our language – the protagonist character, that is, the "player" – and so these three verbs all have adjectival forms which imply the player as the missing term.

```
Definition: a thing is worn if the player is wearing it.
Definition: a thing is carried if the player is carrying it.
Definition: a thing is held if the player is holding it.
```

§**5.** Animate beings also have the ability to see and touch their surroundings, but note that we only model the ability to do these things – we do not attempt to track what they actually do see or touch at any given moment, so there are no built-in verbs *to see* or *to touch*.

```
The verb to be able to see (he is seen) implies the visibility relation.
The verb to be able to touch (he is touched) implies the touchability relation.
```

§**6.** The special status of the player as the sensory focus, so to speak, is again shown in the adjectives defined here:

```
Definition: Something is visible rather than invisible if the player can see it.
Definition: Something is touchable rather than untouchable if the player can touch it.
```

§**7.** While many of the world-modelling assumptions in I7 are carried over from those tried and tested by I6, the idea of concealment is an exception. The old I6 attribute `concealed` simply marked some objects (which we would call "things") as being hidden from view in some way, but was never very satisfactory. What does hidden mean, exactly – to whose eyes, and in what way? Should you be able to take something which is hidden, if you happen to know it's there? And so on. It was the muddiest of all the attributes, and widely disused as a result. In I7, we instead took the view that concealment required an active agent continuously doing the concealing: it applies, for instance, to a dagger which someone intentionally hides beneath a cloak, but not to a key placed at the back of a shelf by somebody long gone.

```
The verb to conceal (he conceals, they conceal, he concealed, it is concealed,
he is concealing) implies the concealment relation.
Definition: Something is concealed rather than unconcealed if the holder of it conceals it.
```

§**8.** A final sort of pseudo-containment: does the entire world contain something, or not? (For things destroyed during play, or not yet created, the answer would be no.) These definitions are rather crudely made not in terms of a defined verb *to include* but instead use the phrase “the world model includes ...”, which will be defined later. (It doesn’t matter that it hasn’t been defined yet, because definitions, phrases and rules can be made in any order and still refer to each other.)

```
Definition: Something is on-stage rather than off-stage if I6 routine "OnStage"
    makes it so (it is indirectly in one of the rooms).
```

§**9. Verbal descriptions of numerical comparisons.** We might as well declare these now, too, though they’re not needed for any of the world-building work. (The verbal usages `<`, `>`, `<=` and `>=` are built into NI; those would be the same in any language, and are unlike other verbs since they have no inflected forms for non-present tenses.)

```
The verb to be greater than implies the numerically-greater-than relation.
The verb to be less than implies the numerically-less-than relation.
The verb to be at least implies the numerically-greater-than-or-equal-to relation.
The verb to be at most implies the numerically-less-than-or-equal-to relation.
```

§**10. Creating the world model.** The 0th kind, “kind”, is not created here but by NI itself. The first through to ninth kinds created now follow: they must not be reordered or moved. Note the two alterative plural definitions for the word “person”, with “people” being defined earlier to make it the default: “persons” is correct, but “people” is more idiomatically usual.

```
Section SR1/1 - Primitive Kinds

A room is a kind. [1]
A thing is a kind. [2]
A direction is a kind. [3]
A door is a kind of thing. [4]
A container is a kind of thing. [5]
A supporter is a kind of thing. [6]
A backdrop is a kind of thing. [7]
The plural of person is people. The plural of person is persons.
A person is a kind of thing. [8]
A region is a kind. [9]
```

§11. At this point, then, the hierarchy looks like so:

```
kind
    room [1]
    thing [2]
        door [4]
        container [5]
        supporter [6]
        backdrop [7]
        person [8]
    direction [3]
    region [9]
```

This framework is the minimum kit needed in order for NI to be able to manage the spatial relationships arising from its basic verbs. Room and thing are needed to distinguish places and objects; door and backdrop because they need to violate the basic rule that an object can only be in one place at once – a door is "in" both of the rooms it faces onto – and this requires special handling by NI; region because it violates the rule that rooms are not themselves subject to being contained in other objects, and again this requires special handling. That leaves "direction", "container", "supporter" and "person", and these are needed to express the concepts inherent in the sentences "A is east of B", "A is in B", "A is on B" and "A is carried by B". (We also need room and person in order to make sense of the words "somewhere" and "someone", for instance.)

Although further kinds will be created later ("vehicle", for instance), those are merely design choices, and NI would not be troubled by their absence.

§12. Naming properties. Certain properties used for names are held in common by all objects, of whatever kind. "Specification" is special: it isn't compiled, but holds text used to annotate the Kinds index. "Variable initial value" is likewise special; internally, knowledge about the initial value of a global variable is stored as knowledge about this property. It can't be used for anything else. "Indefinite appearance text" is also an internal property (it holds the text sometimes given in double-quotes immediately after an object is created).

```
An object has a text called specification.
An object has a text called indefinite appearance text.
An object has a value called variable initial value.
```

§13. These, on the other hand, are quite genuine:

```
An object has a text called printed name.
An object has a text called printed plural name.
An object has a text called an indefinite article.
An object can be plural-named or singular-named. An object is usually singular-named.
An object can be proper-named or improper-named. An object is usually improper-named.
```

§14. Rooms. We now detail each of the fundamental kinds in turn, in order of their declaration, and thus beginning with rooms.

```
Section SR1/2 - Rooms

The specification of room is "Represents geographical locations, both indoor
and outdoor, which are not necessarily areas in a building. A player in one
room is mostly unable to sense, or interact with, anything in a different room.
Rooms are arranged in a map."
```

§**15.** Rooms have rather few properties built in; this reflects their usual role in IF as ambient environments in which interesting things happen, rather than being direct participants.

```
A room can be privately-named or publically-named. A room is usually publically-named.
A room can be lighted or dark. A room is usually lighted.
A room can be visited or unvisited. A room is usually unvisited.

A room has a text called description.
```

§**16.** Note that the "map region" property here is created with the type "object", not "region", even though we think of it as always being a region. This is because of I7's type-checking rule: the type "object" can legally hold 0, meaning "nothing", but more specific object types – in this case "region" – cannot. That would make them illegal to use in a situation where no regions were created, because variables or properties of this kind couldn't be initialised. This is why the Standard Rules almost always declare object properties as "object" rather than anything more specific.

```
A room has an object called map region. The map region of a room is usually nothing.
```

§**17.** Rooms have two specialised spatial relationships of their own, which again we need verbal forms of:

```
The verb to be adjacent to implies the reversed adjacency relation.
Definition: A room is adjacent if it is adjacent to the location.

The verb to be regionally in implies the reversed regional-containment relation.
```

§**18.** There's no detailed writeup of regions, since they have no properties in the usual setup. So let's add this here for the Kinds index:

```
The specification of region is "Represents a broader area than a single
room, and allows rules to apply to a whole geographical territory. Each
region can contain many rooms, and regions can even be inside each other,
though they cannot otherwise overlap. For instance, the room Place d'Italie
might be inside the region 13th Arrondissement, which in turn is inside
the region Paris. Regions are useful mainly when the world is a large one,
and are optional."

A region can be privately-named or publically-named. A region is usually publically-named.
```

§**19. Things.** Things are ubiquitous:

```
Section SR1/3 - Things

The specification of thing is "Represents anything interactive in the model
world that is not a room. People, pieces of scenery, furniture, doors and
mislaid umbrellas might all be examples, and so might more surprising things
like the sound of birdsong or a shaft of sunlight."
```

§20. The large number of either/or properties things can have reflects the flexibility of the I6 world model, which we largely adopt for I7 too. That is, you can have any combination of lit/unlit, edible/inedible, fixed in place/portable, and so on. We can divide them into three broad categories: first, physical properties. Things come in $2^6 = 64$ physically different varieties, which is rather a lot, but although some combinations are very rare (edible lit pushable between rooms scenery is not met with often) this flexibility is helpful in mitigating the rigidity of the kinds structure, given that we have single inheritance of kinds. Note that, except for "lit", these are all really to do whether and how people can move things around – even edibility, which is the ability to be removed from the world model entirely.

```
A thing can be lit or unlit. A thing is usually unlit.
A thing can be edible or inedible. A thing is usually inedible.
A thing can be fixed in place or portable. A thing is usually portable.
A thing can be scenery.
A thing can be wearable.
A thing can be pushable between rooms.
```

§21. Second, status properties, which in effect refer to the past history of an item without our needing to use the past tenses, which can be tricky or inefficient. "Handled" means that the player has at some time carried the thing in question. (We used to have "initially carried" here, too, but that's now considered a part of the verb "to carry" rather than an adjective.)

```
A thing can be handled.
```

§22. Third, linguistic properties, influencing when and how the thing's name will be included in lists. ("Mentioned" goes here rather than as a status property because it refers only to the current room description, so it carries no long-term historic information. "Marked for listing", similarly, carries only short-term information and is used as workspace by the I6 library and also by some of the I7 template routines.)

```
A thing can be privately-named or publically-named. A thing is usually publically-named.
A thing can be described or undescribed. A thing is usually described.
A thing can be marked for listing or unmarked for listing. A thing is usually
unmarked for listing.
A thing can be mentioned or unmentioned. A thing is usually mentioned.
```

§23. We now have a mixed bag of value properties, all descriptive – it's an interesting reflection on how qualitative English text usually is that the world model so seldom needs quantitative properties (sizes, weights, distances, and so on).

```
A thing has a text called a description.
A thing has a text called an initial appearance.
```

§**24.** Lastly on things: an implication about scenery. The following sentence looks like an assertion much like others above ("A thing is usually inedible", for instance) – but this is misleading. What is different is that instead of reading $K(x) \Rightarrow Q(x)$, where K is a kind and Q is a property, this has the form $P(x) \Rightarrow Q(x)$: it says that an object having property P also probably has property Q. Such sentences are called implications, and the Standard Rules make only very sparing use of them. They can trip up the user (who may quite reasonably say that it is up to him what properties something has): but they are invaluable if they cause Inform to make deductions which any human reader would always make without thought.

They can of course be overruled by explicit sentences in the source text, just as every sentence qualified by "usually" can.

The handful of implications in the Standard Rules are all commented as such.

```
Scenery is usually fixed in place. [An implication.]
```

§**25. Directions.** The first important point about directions is that they are not things and not rooms. They are not positions in the world, but imaginary arrows pointing in different ways one could go from those positions. In the language of geometry, we could call them tangent vectors which can be taken anywhere in space by parallel transport without altering them: that's to say, the "north" in one place is the same as the "north" anywhere else. (This is how we get away with having just one set of 12 direction objects, not 12 different ones for every location.) Implicit in that assumption is that the model world occupies a "flat" Euclidean space, to use further mathematical jargon: it doesn't wrap around on itself, and there are no bad positions where the directions fail. (Compare the Infocom game *Leather Goddesses of Phobos*, in which the South Pole of Mars is just such a singularity: there are three routes out of this location, all of them "north". This of course required special programming, and so it would in an Inform 7 work, too.) More concisely:

```
Section SR1/4 - Directions

The specification of direction is "Represents a direction of movement, such
as northeast or down. They always occur in opposite, matched pairs: northeast
and southwest, for instance; down and up."
```

§**26.** The only either/or property created for directions is used to allow them to be part of lists of objects:

```
A direction can be privately-named or publically-named. A direction is usually
publically-named.
A direction can be marked for listing or unmarked for listing. A direction is
usually unmarked for listing.
```

§**27.** The following value property expresses that all directions in I7 come in matched, diametrically opposing pairs – north/south, up/down and so on. This is a concept we need to provide so that I7 can apply its assumption that if room X is north of room Y, then probably room Y is also south of room X, and so on. (Geometrically, this is the operation of negation in the tangent bundle.) Note that the kind of value here is "direction", not "object": a value of 0, meaning "there's no opposite", is illegal.

```
A direction has a direction called an opposite.
```

§**28.** I6 historically began with no formal concept of “direction” and has no `direction` attribute marking some of its objects as directions (it looked instead for object-tree chidren of a pseudo-object called `compass`); by the time I6 did want such a formal concept, the use of attributes to encode what amounted to class membership was no longer thought to be good practice. So I6 directions are now expected to belong to a class called `CompassDirection`, and here we assert just that.

Our I7 directions will be created just like any other I7 objects, but we want them to emerge with the traditional names which I6 direction objects had: so, because the I6 object for north was always called `n_obj`, we want to ensure that the I7 direction “north” also comes out as `n_obj` in the compiled code. Special translates-into-I6-as sentences are used to force the I7 object compiler to use a given I6 identifier to represent the object, rather inventing something like `O12_north` as it otherwise would.

```
Include (-
    has scenery, ! class CompassDirection,
-) when defining a direction.
```

§**29.** The Standard Rules define only thirteen I7 objects, and here we go with twelve of them: the standard set of directions, which come in six pairs of opposites.

The following set – N/S, NE/SW, E/W, SE/NW, U/D, IN/OUT – is rooted in IF tradition. It seems unlikely that people would make IN/OUT a pair of directions today if starting from a clean slate: this is really a residue of the traditional implementation, in 70s and 80s IF, of commands which moved the player in unorthodox way. Outside the cave mouth, typing IN should take you inside; in the Y2 Rock Room, typing the magic word PLUGH should take you far away. The most convenient way to implement such commands in as few instructions as possible was to regard these as little-used compass directions rather than independent commands (some implementations of the original Adventure regarded XYZZY, PLUGH, PLOVER as all being directions, thus using 15 of the 16 possibilities which could be represented in a 4-bit field). In the 90s this was seen to be a little bogus, but since IN and OUT clearly applied in a variety of settings, they continued to be regarded as bona fide directions. In effect, they allow for one location to surround another: the canonical example would be a small white building in the middle of a field. Anyway, I7 accepts the current orthodoxy, so IN/OUT are allowed, even though they cause headaches for the interpretation of words like “inside” which might refer either to the “horizontal” or “vertical” spatial models as a result.

Of the rest, N/S, NE/SW, E/W, SE/NW and U/D, it’s noteworthy that this choice imposes a cubical grid on the world, simply because the compass directions are at 45 and 90 degree angles to each other: a hexagonal tessalation would be more faithful to distances (it would get rid of the awkward point that a NE move is $\sqrt{2}$ times the length of a N move), but in practice the world model doesn’t care much about distances, another example of its qualitative nature. A further point is that, in a three-dimensional cubic lattice, we ought to have another eight pairs of directions for “up and northeast”, “down and west” and so on – instead of which U/D are the only ways out of the horizontal plane. But natural language doesn’t work that way: it overwhelmingly provides words for horizontal travel, because that’s the plane in which our eyes normally see, and in which we normally walk. Linguistically, “north” genuinely means north, but “up” allows for any amount of lateral movement into the bargain. It’s a doctrine of I7 that linguistic bias is a good guide to what’s worth modelling and what is not, so we will now stop worrying about this and declare the actual objects.

The order of definition of the directions affects the way lists come out: the traditional order is N, NE, NW, S, SE, SW, E, W, U, D, IN, OUT.

```
The north is a direction.
The northeast is a direction.
The northwest is a direction.
The south is a direction.
The southeast is a direction.
The southwest is a direction.
The east is a direction.
The west is a direction.
The up is a direction.
The down is a direction.
The inside is a direction.
The outside is a direction.

The north has opposite south. Understand "n" as north.
The northeast has opposite southwest. Understand "ne" as northeast.
The northwest has opposite southeast. Understand "nw" as northwest.
The south has opposite north. Understand "s" as south.
The southeast has opposite northwest. Understand "se" as southeast.
The southwest has opposite northeast. Understand "sw" as southwest.
The east has opposite west. Understand "e" as east.
The west has opposite east. Understand "w" as west.
Up has opposite down. Understand "u" as up.
Down has opposite up. Understand "d" as down.
Inside has opposite outside. Understand "in" as inside.
Outside has opposite inside. Understand "out" as outside.

The inside object translates into I6 as "in_obj".
The outside object translates into I6 as "out_obj".

The verb to be above implies the mapping up relation.
The verb to be mapped above implies the mapping up relation.
The verb to be below implies the mapping down relation.
The verb to be mapped below implies the mapping down relation.
```

§**30. Doors.** Doors are, literally, a difficult edge case for the world model of IF, since they occupy the awkward junction between the two different ways of dividing up space: the "vertical" model of objects containing and supporting each other, all within a tree rooted by the room which represents, for the moment, the entire stage-set for the play; and the "horizontal" model of rooms stitched together at compass directions into a map. The difficulty arises because in order for a door to make sense in the horizontal model, it needs to be present in two different rooms at the same time, and then it doesn't make sense in the vertical model any more, because which object tree is it to be in?

```
Section SR1/5 - Doors

The specification of door is "Represents a conduit joining two rooms, most
often a door or gate but sometimes a plank bridge, a slide or a hatchway.
Usually visible and operable from both sides (for instance if you write
'The blue door is east of the Ballroom and west of the Garden.'), but
sometimes only one-way (for instance if you write 'East of the Ballroom is
the long slide. Through the long slide is the cellar.')."
```

§**31.** This is the first kind we have declared to be a kind of something else: a door is a kind of thing. That means a door inherits all of the properties of a thing, but in a way which allows us to change the normal expectations. So here we see the first case of assertions which contradict earlier ones, but in a narrower domain: a thing is usually portable, but a door is usually fixed in place.

Our difficulty with doors being multiply present would be enormously worse if we allowed anybody to move them around during play. So:

```
A door is always fixed in place.
A door is never pushable between rooms.
Include (- has door, -) when defining a door.
```

§**32.** "Every exit is an entrance somewhere else," as Stoppard's play *Rosencrantz and Guildenstern are Dead* puts it: and though not all I7 doors are present on both sides, they do nevertheless have two sides. The representation of this is quite tricky because, as Stoppard implies, it's all a matter of which side you look at it from. What we call the "other side", and whether or not we say that "the Ballroom is through the green door", depends entirely on which side of the green door we stand. The awkward truth is that these expressions are undefined unless the player is in one of the (possibly) two rooms in which the green door is present; and then they are defined relative to him.

The leading-through relation is built in to NI; the other side property, though, is merely a convenient name we give to the property in which the relation data is stored at run-time.

```
A door has an object called other side.
The other side property translates into I6 as "door_to".
Leading-through relates one room (called the other side) to various doors.
The verb to be through implies the leading-through relation.
```

§**33. Containers and supporters.** The carrying capacity property is the exception to the remarks above about the qualitative nature of the world model: here for the first and only time we have a value which can be meaningfully compared.

```
Section SR1/6 - Containers

The specification of container is "Represents something into which portable
things can be put, such as a teachest or a handbag. Something with a really
large immobile interior, such as the Albert Hall, had better be a room
instead."

A container can be enterable.
A container can be opaque or transparent. A container is usually opaque.
A container has a number called carrying capacity.
The carrying capacity of a container is usually 100.

Include (- has container, -) when defining a container.
```

§**34.** The most interesting thing to note here (and we will see it again in the definition of "people") is that "transparent" the I7 property is not a direct match onto `transparent` the I6 attribute. In I7, the term is applicable only to containers (a reform made in January 2008, but clarifying what was already de facto the case). In I6, the `transparent` attribute means that child-objects in the object tree are in scope whenever the parent object is: in the I7 world model that's always true for supporters, so we oblige all supporters to have the attribute `transparent` in their I6 compiled forms. The same will be true for people. That doesn't in practice mean that I7 never has high shelves or people with daggers concealed beneath cloaks – just that we no longer use I6's mechanism for hiding these things, and expect the user to write activity rules instead.

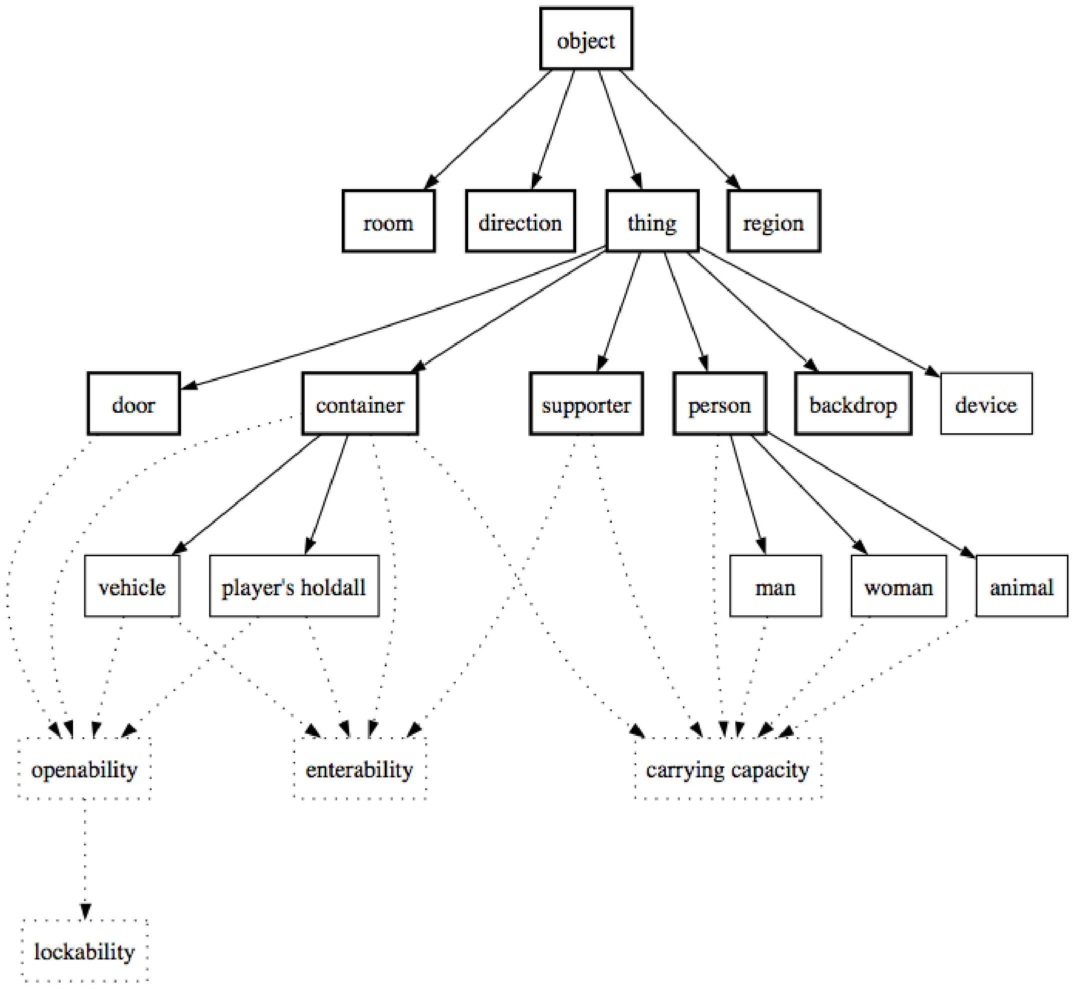
```
Section SR1/7 - Supporters

The specification of supporter is "Represents a surface on which things can be
placed, such as a table."

A supporter can be enterable.
A supporter has a number called carrying capacity.
The carrying capacity of a supporter is usually 100.

A supporter is usually fixed in place.

Include (-
    has transparent supporter
-) when defining a supporter.
```

§35. Kinds vs patterns. A problem faced by all object-oriented systems is "fear of the diamond", the problematic diagram of inheritance which results when we have two different subclasses B and C of a class A, which represent quite different ideas, but then we also turn out to want some behaviour D which is shared between some of the Bs and some of the Cs. For instance, we might have one class for people and another for buildings, but want to use the same code when it comes to (say) printing out top ten lists of basketball players (people) and skyscrapers (buildings) in height order: why not? But then again, what does D conceptually represent? Surely we aren't saying there's a natural concept of "basketball player/skyscraper"?

There are various responses, of which the most widely used now is probably that of `C++`'s notion of templates. We would define our top-ten business by writing a function applying to a list of objects of any class T such that T provided a height: there would then be no need for "basketball player/skyscraper" to be a class in its own right. Instead, we would define the behaviour as being available to anything for which it makes sense.

This is broadly what Inform 7 does, too, though not so formally. We use the term "pattern" for this, and have actually seen two patterns already – the way that containers and supporters share the "carrying capacity" limit, and also the notion of transparency – and it's by providing two patterns that we are able to deal with the likeness and also unlikeness of doors and containers. Their unlikeness is obvious; but their likeness is that they both grant or withhold access to some extent of space bordering on the current one. (Doors do this for the "horizontal" spatial model between rooms, whereas containers do it for the "vertical" spatial model of objects enclosing each other.)

The following diagram shows the kinds created by the Standard Rules (bold boxes for fundamental kinds, plain ones for discretionary additions) and the patterns of behaviour they share (dotted boxes).



§36. The openability pattern. To satisfy the openability pattern, a thing has to provide both of the either/or properties "open" and "openable". This entitles it to be opened and closed by the opening and closing actions, defined below. Note that I7 has no formal concept of patterns as part of its type-checking: instead, the rules for these actions explicitly check that they are being applied to things matching the pattern, as we shall see.

Doors and containers have, as it happens, exactly opposite conventions about the default values of these properties: but that doesn't mean they don't share the pattern.

```
Section SR1/8 - Openability

A door can be open or closed. A door is usually closed.
A door can be openable or unopenable. A door is usually openable.

A container can be open or closed. A container is usually open.
A container can be openable or unopenable. A container is usually unopenable.
```

§37. The lockability pattern. And similarly for lockability, because a principle of the world model is that any spatial barrier can be given a lock if the designer so chooses. To satisfy this pattern, a thing must

(i) satisfy the openability pattern, and
(ii) provide both the either/or properties "lockable" and "locked", and also the value property "matching key".

Both doors and containers make some implications so that the words "lockable" and "locked" carry the implied meanings which human readers expect, but this is not essential to the functioning of lockability: it's only a graceful addition.

```
Section SR1/9 - Lockability

A door can be lockable. A door is usually not lockable.
A door can be locked or unlocked. A door is usually unlocked.
A door has an object called a matching key.
A locked door is usually lockable. [An implication.]
A locked door is usually closed. [An implication.]
A lockable door is usually openable. [An implication.]

A container can be lockable. A container is usually not lockable.
A container can be locked or unlocked. A container is usually unlocked.
A container has an object called a matching key.
A locked container is usually lockable. [An implication.]
A locked container is usually closed. [An implication.]
A lockable container is usually openable. [An implication.]
```

§38. Note that the lock-fitting relation has, as its domains, "thing" and "thing". That means that compile-time typechecking will not reject an attempt to apply the relation to (say) two vehicles. At run time, evaluating "if X unlocks P" where P is a peculiar thing with no possibility of a lock will always come out false; but trying to force it with "now X unlocks P" will cause a run-time problem. In short, patterns are defended at run-time, not at compile-time.

```
Lock-fitting relates one thing (called the matching key) to various things.
The verb to unlock (it unlocks, they unlock, it unlocked, it is unlocked)
implies the lock-fitting relation.
```

§**39. Backdrops.** The true subtlety of backdrops is not visible in the brief description here: but they require careful handling both in NI and in the template layer code, because they can be in many rooms at once.

```
Section SR1/10 - Backdrops

The specification of backdrop is "Represents an aspect of the landscape
or architecture which extends across more than one room: for instance,
a stream, the sky or a long carpet."

A backdrop is usually scenery.
A backdrop is always fixed in place.
A backdrop is never pushable between rooms.
```

§**40. People.** From a compilation point of view, people are surprisingly easy to deal with. It may well be argued that this is because the I6 world model is so sketchy in modelling them, but that may actually be a good thing, because it's not at all obvious that any single model will be sensible for what different authors want to do with their characters.

On gender, see also the "man" and "woman" kinds below. Note that we have three gender choices available – male, female and neuter – but these are, for historical reasons to do with how gender is handled by the I6 library, managed using either/or properties rather than a single three-way value property. This doesn't in practice cause trouble. (Specifying something as neuter overrides the male/female choice, if anyone does both for the same object, but in practice nobody does.) When nothing is said about a person's gender, it is assumed male, though this is used only linguistically (for instance, the pronoun HIM can be used in commands about the object, rather than HER or IT). There has to be some convention here, and in a case where we don't know our linguistic ground, opting for the least surprising behaviour seems wisest.

```
Section SR1/11 - People

The specification of person is "Despite the name, not necessarily
a human being, but anything animate enough to envisage having a
conversation with, or bartering with."

A person can be female or male. A person is usually male.
A person can be neuter. A person is usually not neuter.

A person has a number called carrying capacity.
The carrying capacity of a person is usually 100.
```

§**41.** I6 has a concept approximately equivalent to I7's "person" – the I6 library attribute `animate` – but I6 allows only some of those objects to become the protagonist during play (using I6's `ChangePlayer` routine). To be eligible, an object must not only be `animate` but also provide a whole host of writeable properties.

But I7 provides those I6 properties for every "person", for the sake of a clean, uniform design, and accepting the cost that people therefore take more bytes of precious Z-machine array space than they necessarily would in I6. This is all part of the doctrine that in I7, all characters are equal in status: all can be the player, all can carry out actions. Anyway: here are all of those I6 properties, spatchcocked into the `Class` definition which NI will compile for "person" – see §21 of the DM4 for details of why these are needed and what they do.

```
Include (-
    has transparent animate
    with before NULL,
-) when defining a person.
```

§**42.** So to the thirteenth and final object created by the Standard Rules: the enigmatic default protagonist, whose name is not "player" but "yourself". (The I6 library requires this object to be created as `selfobj`, but that's not a name that is ever printed or parsed: it's a constant value used only in I6 source code.)

The `yourself` object has to be proper-named to prevent the I6 library from talking about "the yourself", as it otherwise might. "Undescribed" in this context means that "yourself" is not described as being present in room descriptions: this would be redundant and annoying.

The I6 property `saved_short_name` property is an implementation convenience for use if there is ever a change of player, in which case the printed name of the object will cease to be "yourself" and become "your former self" instead. When this happens, the previous printed name (or `short_name` in I6 terms) is stored in `saved_short_name` so that it can recovered later. (We can't assume it was necessarily "yourself" because the source text might have overridden this with a sentence like "The printed name of the player is "your dreary self".")

```
The yourself is an undescribed person. The yourself is proper-named.

The description of yourself is usually "As good-looking as ever."

The yourself object translates into I6 as "selfobj".
Include (-
    with saved_short_name "yourself",
-) when defining yourself.
```

§**43. Non-fundamental kinds.** We have now finished defining the nine fundamental kinds which NI requires in order for it to function. There are six more to define, but it's worth emphasising that none of these is required or assumed by either NI or its template layer of I6 code. So any of them could be changed drastically or got rid of entirely simply by amending the Standard Rules. (Like the "player-character" kind, born early 2003, died July 2007.)

Equally, we could add others if we wanted. The judgement of what ought to be part of the basic hierarchy of kinds created by the Standard Rules isn't easy. The maximalist position is that users welcome a plethora of kinds to simulate many facets of real life, from canal-boats to candles. The minimalist position is that kinds are necessary only as the domain of relations (so that person is necessary as the domain of P in "P carries X", for instance), and that too many kinds confuses the picture and imposes what may be a constraining structure on the user, who should be free to decide for himself what concepts are most helpful to organise. These arguments are discussed further in the white paper, *Natural Language, Semantic Analysis and Interactive Fiction* (2005), but briefly: we are minimalist but not puritanically so.

§**44. Men, women and animals.** Of these discretionary kinds, so to speak, "man" and "woman" are perhaps the least challengeable. They are not obviously the domains of any natural relation (unless one takes a very old-fashioned idea of gender identity and supposes that, oh, "X keeps Y" implies that X is a wealthy man and Y a mistress). But they are so linguistically natural in story-telling: who would ever write "Jack is a person in the House. Jack is male." in preference to "Jack is a man in the House."

An awkward point here is that, of course, most people would simply say "Jack is in the House." and expect us to infer that Jack is a person from the fact that this is more often a human name than, say, a proprietary brand of microphone plug; and that Jack is male, because relatively few girls called Jacqueline are nicknamed Jack. As it happens the NI compiler doesn't allow for tentative statements about the kinds of objects (only about their property values), but it wouldn't be too hard to add such a system, with a little care. The trouble is that we would then need a large dictionary of boys' and girls' names, valid across American, Canadian, Australian and British English (together with a selection from foreign tongues), and this would always lead to puzzling omissions (why isn't "Glanville" recognised?) or ambiguities (why is "Pat" a man?). And similarly for titles: "Mr", "Mrs" and "Ms" are fairly indicative of gender, except in certain military contexts, but what about (say) "Admiral" or "Reverend", where there is a strong likelihood of masculinity but no more than that? So Inform 7's compromise position is that the user does have to specify gender explicitly, but that the kinds "man" and "woman" provide conveniently abbreviated ways to do so. (We consider male and female children to qualify in these categories.)

Anyway, we set out the Anglo-Saxon plurals, and then declare these kinds purely in terms of gender: they have no distinguishing behaviour.

```
Section SR1/12 - Animals, men and women

The plural of man is men. The plural of woman is women.

A man is a kind of person.
The specification of man is "Represents a man or boy."
A man is always male. A man is never neuter.

A woman is a kind of person.
The specification of woman is "Represents a woman or girl."
A woman is always female. A woman is never neuter.
```

§**45.** But what about "animal"? Animals turn up often in IF, and of course domestic animals have been part of human society since prehistoric times: but then again, the same can be said for stoves and larders, and we aren't declaring kinds for those.

The reason "animal" exists is mainly because it is almost always peculiar to write "P is a person". Now that we have "man" and "woman" taken care of, the remaining objects we might want to declare will almost always fall into this category: it's intended to be used for "people" who are animate but probably not intelligent, or anyway, not participants in human society. It seems unusual to write "The black Labrador is a person." because that sounds like an insistent assertion of rights and thus a quite different sort of statement. (Don't drown that Labrador! He's a *person.*)

As can be seen from the tiny definition of "animal", though, it's really nothing more than a name for a position in the kinds hierarchy. There is not even any implication for gender.

```
An animal is a kind of person.

The specification of animal is "Represents an animal, or at any rate a
non-human living creature reasonably large and possible to interact with: a
giant Venus fly-trap might qualify, but not a patch of lichen."
```

§**46. Devices.** The justification for providing a "device" kind is much thinner. It's done largely for traditional reasons – such a concept existed in the I6 library, which in turn followed Infocom conventions dating from the early 1980s. The inclusion is defensible as representing a common linguistic category found in everyday situations, where an inanimate object nevertheless does something while under direct or indirect human control: we can also imagine relations for which it could be a domain ("X is able to work D" meaning that person X understands how to use the controls of device D, say). It could equally be attacked as having a rather flimsy world model – it's just an on/off switch – and representing a pretty inchoate mass of concepts, from a mousetrap to a nuclear reactor.

```
Section SR1/13 - Devices

A device is a kind of thing.

A device can be switched on or switched off. A device is usually switched off.
Include (- has switchable, -) when defining a device.

The specification of device is "Represents a machine or contrivance of some
kind which can be switched on or off."
```

§**47. Vehicles.** Here again the justification boils down to tradition. Vehicles were a staple ingredient of the Infocom classics, largely because of code originally written for the inflatable boat in the 1978-79 mainframe version of *Zork*, which was then copied through into later titles. Unlike devices, though, vehicles are genuinely difficult to model, and the implementation provided by the Standard Rules would be quite a lot of work for a user to manage alone. (Consider, for instance, the case when the player is sitting in an open basket when Bill, driving a fork-lift truck, uses his vehicle to push the basket into another room.) There might perhaps be a case for moving all of the vehicles material into an extension, but it would have to be an extension supplied as part of the built-in set, and whenever it was used the result would be that the going action would rely on a pretty complicated interlacing of rules as between this extension and the Standard Rules.

Turning to implementation, I6 – surprisingly, perhaps – doesn't have a `vehicle` attribute: a vehicle is an object which is `enterable` and whose `before` rule for the I6 `##Go` action returns the magic value 1. A troublesome point here is that I6 makes no distinction between vehicles which contain and vehicles which support. But we do, because once we have decided to make "vehicle" a kind, it has to be either a kind of container or a kind of supporter: it can't be both. We get around this by providing for container-vehicles in the Standard Rules, as being the more commonly occurring case, while providing for the other with the extension Rideable Vehicles by Graham Nelson, which is in effect an offshoot of the Standard Rules and is built-in to every installation of Inform 7. This also provides for animals used as vehicles.

The alternative approach here would be to make "vehicle" not a kind but an either/or property of things, so as to provide a pattern of behaviour common to certain animals, containers and supporters. We could then move Rideable Vehicles back into the Standard Rules, but that would add a fair amount of code, and besides, it is unclear that "vehicleness" is something we want to come and go during play, or that it's appropriate as an either/or property of (for instance) a door or a person.

```
Section SR1/14 - Vehicles

A vehicle is a kind of container.

The specification of vehicle is "Represents a container large enough for
a person to enter, and which can then move between rooms at the driver's
instruction. (If a supporter is needed instead, try the extension
Rideable Vehicles by Graham Nelson.)"

A vehicle is always enterable.
```

§**48.** The part about vehicles not usually being portable is simply for realism's sake: generally speaking if something can hold human weight it's pretty large and heavy. (A bicycle is an edge case, and a skateboard is clearly an exception, but that's why the rule is only "usually".)

If all vehicles were wheeled, there would be a case for a rule such as "A vehicle is usually pushable between rooms." But this seems more likely to trip up the designer with a surprise discovery in beta-testing than to help him achieve realism. We don't want to be able to push hot-air balloons, boats or spacecraft between rooms.

```
A vehicle is usually not portable.
```

§**49. Player's holdalls.** This is the final kind created in the Standard Rules, and probably the most doubtful of all. It simply provides a hook to a cute and traditional feature of the I6 library whereby spare possessions are automatically cleared out of the player's way: it derives from the rucksack in the 1993 IF title *Curses*.

```
Section SR1/15 - Player's holdall

A player's holdall is a kind of container.

The specification of player's holdall is "Represents a container which the
player can carry around as a sort of rucksack, into which spare items are
automatically stowed away."

A player's holdall is always portable.
A player's holdall is usually openable.
```

§**50.** To enable the use of player's holdalls, we must declare a constant `RUCKSACK_CLASS` to tell some code in the template layer to use possessions with this I6 class as the rucksack pro tem. This is all a bit of a hack, to retrofit a degree of generality onto the original I6 library feature, and even then it isn't really fully general: only the player has the benefit of a "player's holdall" (hence the name), with other actors oblivious.

```
Include (-
    Constant RUCKSACK_CLASS = (+ player's holdall +);
-) after "Definitions.i6t".
```

§**51. Correspondence between I6 and I7 property and attribute names.** All of the kinds, objects and properties which make up the standard kit provided to every source text are now complete. We conclude Section SR1 by giving the NI compiler a dictionary to tell it how I7's names for properties – some value properties, some either/or – mesh with those in the I6 library.

Ordinarily, a new value property such as "astral significance" would be compiled by NI into an I6 property called something like

```
    P73_astral_significance
```

whereas a new either/or property might become either an I6 attribute or an I6 property holding only `true` or `false`, at the compiler's discretion. (It needs to use this discretion because I6 has a hard limit on the number of attributes, whereas there are no limits on the number of properties used in I7.) And if "astral significance" is a concept handled only by I7 source text, that's fine.

But we want our "printed name" property, for instance, to be the text which the I6 library prints out whenever it uses the `short_name` of an object: so we want the NI compiler to use the I6 identifier `short_name` for "printed name", not to invent a new one. NI therefore maintains a dictionary of equivalents, and here it is. (Any I7 property not named is handled purely by I7 source text in the remainder of the Standard Rules.)

§**52.** First, equivalents where I7 either/or properties map directly onto I6 attributes. Note the way “lit” (for things) and “lighted” (for rooms) both map onto the same I6 attribute, `light`. Attributes were in scarce supply in I6 (with a limit of 32 in the early days, later raised to 48) and this sort of reuse seemed sensible in the early 1990s, especially as the meanings were basically similar.

```
Section SR1/16 - Inform 6 equivalents

The wearable property translates into I6 as "clothing".
The undescribed property translates into I6 as "concealed".
The edible property translates into I6 as "edible".
The enterable property translates into I6 as "enterable".
The female property translates into I6 as "female".
The mentioned property translates into I6 as "mentioned".
The lit property translates into I6 as "light".
The lighted property translates into I6 as "light".
The lockable property translates into I6 as "lockable".
The locked property translates into I6 as "locked".
The handled property translates into I6 as "moved".
The neuter property translates into I6 as "neuter".
The switched on property translates into I6 as "on".
The open property translates into I6 as "open".
The openable property translates into I6 as "openable".
The privately-named property translates into I6 as "privately_named".
The plural-named property translates into I6 as "pluralname".
The proper-named property translates into I6 as "proper".
The pushable between rooms property translates into I6 as "pushable".
The scenery property translates into I6 as "scenery".
The fixed in place property translates into I6 as "static".
The transparent property translates into I6 as "transparent".
The visited property translates into I6 as "visited".
The marked for listing property translates into I6 as "workflag".
```

§**53.** Second, the I7 value properties mapping onto I6 properties. Again, `map_region` is a new I6 property of our own, while the rest are I6 staples. And see also “other side”, which is translated above for timing reasons.

```
The indefinite article property translates into I6 as "article".
The carrying capacity property translates into I6 as "capacity".
The description property translates into I6 as "description".
The initial appearance property translates into I6 as "initial".
The map region property translates into I6 as "map_region".
The printed plural name property translates into I6 as "plural".
The printed name property translates into I6 as "short_name".
The matching key property translates into I6 as "with_key".
```

SR2 - Variables and Rulebooks

Purpose

The global variables and those built-in rulebooks which do not belong either to specific actions or to specific activities.

§**1.** With the kinds, objects and properties all now made, we turn to more abstract constructions: rules, rulebooks, activities, global variables and so on. As at the beginning of Part SR1 above, we begin with an entirely clean slate – none of any of these things exists yet – but also with expectations on what we do, since NI and the I7 template layer need certain constructions to be made.

We begin with global variables. The order in which these are defined, and the section subheadings under which they are grouped, determine the way they are indexed in the Contents index of a project – that is, moving them around would reorder the Contents index, and rewording the subheadings SR2/1, SR2/2, ..., below would change the text of the subheadings in the Contents index accordingly.

Some of these variables are special to NI, and it knows which they are not by the order in which they're defined (as with kinds, above) but by their (I7) names. These are marked with the comment `[*]`, and cannot be renamed without altering NI to match. Those marked `[**]` are similarly named in the template layer.

§**2.** It's now the occasion for our first global variable definition, because although users often think of the protagonist object as a fixed object whose name is "player", it's in fact possible to change perspective during play and become somebody else – at which point the "player" variable will point to a different object.

Note that "player" is actually a variable belonging to the I6 library, as indeed most of those in the Standard Rules are. Without the explicit translation supplied below, it would probably be compiled as `Q1_player` or some such name; and might well be implemented by NI as an entry in an array (since the Z-machine supports only a limited number of global variables, in effect using them as a set of 240 registers, and we therefore don't want to create too many ourselves).

```
Part SR2 - Variables and Rulebooks

Section SR2/1 - Situation

The player is a person that varies. [*]
The player variable translates into I6 as "player".
```

§**3.** The I7 variable "location" corresponds to I6's `real_location`, not `location`. Its value is never equal to a pseudo-room representing darkness: it is always an actual room, and I7 has nothing corresponding to I6's `thedark` "room". Similarly, we use an I7 variable "darkness witnessed" for a flag which the I6 library would have stored as the `visited` attribute for the `thedark` object.

The "maximum score" is, rather cheekily, translated to an I6 constant: and this cannot be changed at run-time.

```
The location -- documented at var_location -- is an object that varies. [*]
The score -- documented at var_score -- is a number that varies.
The last notified score is a number that varies.
The maximum score is a number that varies. [*]
The turn count is a number that varies.
The time of day -- documented at var_time -- is a time that varies. [*]
The darkness witnessed is a truth state that varies.

The location variable translates into I6 as "real_location".
The score variable translates into I6 as "score".
The last notified score variable translates into I6 as "last_score".
The maximum score variable translates into I6 as "MAX_SCORE".
The turn count variable translates into I6 as "turns".
The time of day variable translates into I6 as "the_time".
```

§**4.** It is arguable that "noun", "second noun" and "person asked" ought to be rulebook variables belonging to the action-processing rules, so that they essentially did not exist outside of the context of an ongoing action. The reason this isn't done is partly historical (rulebook variables were a fairly late development, implemented in April 2007, though they had long been planned). But it is sometimes useful to get at the nouns in an action after it has finished, and making them global variables also makes them a little faster to look up (a good thing since they are used so much), and causes them to be indexed more prominently.

"Item described" is simply an I7 name for `self`. (In an object-oriented system such as I6, `self` is the natural way to refer to the object currently under discussion, within routines applying to all objects of its class.) In early drafts of I7, it was called "this object", but somehow this raised expectations too high about how often it would be meaningful: it looked like a pronoun running meanings from sentence to sentence.

```
Section SR2/2 - Current action

The noun -- documented at var_noun -- is an object that varies. [*]
The second noun is an object that varies. [*]
The person asked -- documented at var_person_asked -- is an object that varies. [*]
The reason the action failed -- documented at var_reason -- is an action name
based rule producing nothing that varies.
The item described is an object that varies.

The noun variable translates into I6 as "noun".
The second noun variable translates into I6 as "second".
The person asked variable translates into I6 as "actor".
The reason the action failed variable translates into I6 as "reason_the_action_failed".
The item described variable translates into I6 as "self".
```

§5. “Person reaching” turns out to have exactly the same meaning as “person asked” – they are both the `actor`, in I6 terms, but are used in different situations.

```
Section SR2/3 - Used when ruling on accessibility

The person reaching -- documented at var_person_reaching -- is an object that varies.
The container in question is an object that varies.
The supporter in question is an object that varies.
The particular possession -- documented at var_particular -- is a thing that varies.

The person reaching variable translates into I6 as "actor".
The container in question variable translates into I6 as "parameter_object".
The supporter in question variable translates into I6 as "parameter_object".
The particular possession variable translates into I6 as "particular_possession".
```

§6. Parsing variables follow. The I6 parser tends to put any data read as part of a command into the variable `parsed_number`, but then, I6 is typeless: we can’t have a single I7 variable for all these possibilities since it could then have no legal type. We solve this as follows. Whenever a kind of value K is created which can be parsed as part of a command, an I7 variable “the K understood” is also created, as a K that varies. All of these variables are translated into I6’s `parsed_number`, so in effect they provide aliases of each possible type for the same underlying memory location. The four exceptional kinds of value not parsed by the systematic approaches in NI for enumerated KOVs and units are “number”, “time”, “snippet” and “truth state”: because of their exceptional status, they don’t get “the K understood” variables created automatically for them, so we must construct those by hand. Hence “the number understood”, “the time understood”, “the topic understood” (for historical reasons this one is not called “the snippet understood”), “the truth state understood” but no others.

```
Section SR2/4 - Used when understanding typed commands

The player's command -- documented at var_command -- is a snippet that varies.
The matched text is a snippet that varies.
The number understood -- documented at var_understood -- is a number that varies. [*]
The time understood is a time that varies. [*]
The topic understood is a snippet that varies. [*]
The truth state understood is a truth state that varies. [*]
The current item from the multiple object list is an object that varies.

The player's command variable translates into I6 as "players_command".
The matched text variable translates into I6 as "matched_text".
The topic understood variable translates into I6 as "parsed_number".
The current item from the multiple object list variable translates into I6 as
    "multiple_object_item".
```

§7. The following are, unless the user creates global variables of his own, the last in the Contents index...

```
Section SR2/5 - Presentation on screen

The command prompt -- documented at var_prompt -- is a text that varies. [**]
The command prompt is ">".

The left hand status line -- documented at var_sl -- is a text that varies.
The right hand status line is a text that varies.

The left hand status line variable translates into I6 as "left_hand_status_line".
The right hand status line variable translates into I6 as "right_hand_status_line".

The listing group size is a number that varies.
The listing group size variable translates into I6 as "listing_size".
```

§**8.** ...but they are not the last global variables created by the Standard Rules.

These bibliographic data variables are concealed because they are under a heading which ends with the word "unindexed". There are two reasons why these variables are unindexed: first, they appear in a different guise only a little lower in the Contents index as part of the Library Card, and second, we don't want users to think of them as manipulable during play.

Rather sneakily, we also define a Figure here. This is done in order to make it legal to declare variables and properties of the kind of value "figure name" (because it ensures that such variables can always be initialised – there is always at least one Figure in the world). Of course plenty of Inform projects have no artwork at all: so the cover art figure is unique in that it might refer to nothing. That sounds a little arbitrary but in fact follows a convention used by the Blorb format for binding story files with their resources, which in turn follows Infocom conventions of 1987-89: the cover art always has resource ID number 1, whether it exists or not. NI creates figures and sound effects counting upwards from 1, giving them each unique resource ID numbers, so the first to be created gets ID number 1: by defining "figure of cover" here, we can be sure that we are first, so everything works.

```
Section SR2/6a - Unindexed Standard Rules variables - Unindexed

The story title, the story author, the story headline, the story genre
and the story description are text variables. [*****]
The release number and the story creation year are number variables. [**]

The story title variable translates into I6 as "Story".

Section SR2/6b - Unindexed Standard Rules variables - Unindexed (for figures language element only)

Figure of cover is the file of cover art.
```

§**9.** And we finish out the set with some "secret" variables used only by the Standard Rules or by NI, and which are therefore also unindexed. Their names are all hyphenated, to reduce the chance of anyone stumbling into them in a namespace clash.

The first set of three secret variables is used by the predicate calculus machinery in NI. This is the code which handles logical sentences such as "at least six doors are open" or descriptions such as "open doors", by reducing them to a logical notation which sometimes makes use of variables. For instance, "open doors" reduces to something like "all x such that door(x) and open(x)", with x being a variable. When NI works with such logical propositions, it often needs to substitute $x = c$, that is, to replace x with a given constant c. But it can only do this if c is an Inform 7 value. This is a problem if what it wants is to substitute in something which is only meaningful at the I6 level: say, it wants to substitute a value c which will eventually translate to `whatever`, but it can't find any I7 value c which will do this.

We solve this problem by constructing some unusual I7 variables whose only purpose is that NI can use them in substitutions. They should never be referred to in I7 source text anywhere else at all, not even elsewhere in the Standard Rules.

(1) The "substitution-variable" is used when NI needs to compile a proposition with one free variable, such as "door(x)", as a condition: it binds x with the substitution $x =$ *say-parameter* in order to turn this into a well-formed predicate calculus sentence with no free variables, then compiles it. This is a trick used in the definition of some of the "repeat" phrases below, as a result of the `{-bind-variable:...}` invocation command.

(2) The sentence "The i6-nothing-constant is an object that varies." is a rare example of a flat lie in the Standard Rules, as it is the I6 constant `nothing` and never varies at all. Again, it exists as a "variable" so that the substitution $x =$ `nothing` can be made.

(3) Well, once you start telling lies it's so hard to stop, and it's also a lie that the "I6-varying-global" translates to `nothing`. It actually translates to whatever the NI machinery for compiling propositions happens to want at the moment, so it has no permanent meaning at all. (It will always translate to an I6 global variable storing a value whose I7 kind is "object", so the type-checking machinery isn't

endangered by this chicanery. It will in fact never translate to `nothing`, but we make the translation sentence below in order to avoid allocating any storage at run-time for what is in the end only a label.)

```
Section SR2/6c - Unindexed Standard Rules variables - Unindexed

The substitution-variable is an object that varies. [*]
The substitution-variable variable translates into I6 as "subst__v".

The I6-nothing-constant is an object that varies. [*]
The I6-nothing-constant variable translates into I6 as "nothing".

The I6-varying-global is an object that varies. [*]
The I6-varying-global variable translates into I6 as "nothing".
```

§**10.** The remaining secret variables are:

(1) The "item-pushed-between-rooms" is needed to get the identity of an object being pushed by a command like PUSH ARMCHAIR NORTH out of I6 and into the action variable "thing gone with" of the going action.

(2) The "actor-location" is needed temporarily to store the room in which the actor of the current action is standing, and it wants to be an I6 global (rather than, say, a rulebook variable belonging to the action-processing rulebook) so that NI can use common code to handle this alongside `noun`, `second` and `actor` when compiling preambles to rules.

(3) The "parameter-object" is likewise needed in order to compile preambles to rules in object-based rulebooks.

```
The item-pushed-between-rooms is an object that varies.
The item-pushed-between-rooms variable translates into I6 as "move_pushing".

The actor-location is an object that varies. [*]
The actor-location variable translates into I6 as "actor_location".

The parameter-object is an object that varies. [*]
The parameter-object variable translates into I6 as "parameter_object".

The scene being changed is a scene that varies. [*]
The scene being changed variable translates into I6 as "parameter_object".
```

§**11.** And that completes the run through all the variables created in the Standard Rules.

§**12. Rulebooks.** There are 27 rulebooks which are, so to speak, "primitive" in Inform 7 – which are part of its workings and cannot safely be tampered with. NI requires them to be declared by the Standard Rules as the first 25 rulebooks to be created, and in the exact order below. (In fact, though, this is mostly so that it can prepare the index pages correctly: NI is not the part of I7 which decides what to use these rulebooks for. That is almost always the responsibility of the template I6 layer which, for instance, calls upon the action-processing rulebook when it wants an action processed.)

At the I6 level, a rulebook is referred to by its ID number, which counts upwards from 0 in order of creation. Any reordering of the constants below, therefore, is unsafe unless changes are made elsewhere so that the following three tallies always remain in synchrony:

(a) The sequence of declaration of these rulebooks in the Standard Rules.
(b) The inweb `@d` definitions in the form `TURN_SEQUENCE_RB` in the section Rulebooks of the NI source code.
(c) The I6 `Constant` definitions in the form `TURN_SEQUENCE_RB` in the file `Rulebooks.i6t` of the template I6 layer.

Anyway, we will declare the rulebooks and their variables or outcomes first, and come back to stock some of them with rules later. It seems appropriate to give first place to the procedural rules, the super-rulebook capable of controlling all others:

```
Section SR2/7 - The Standard Rulebooks

Procedural rules is a rulebook. [0]
```

§**13.** Every story file created by Inform 6 begins execution in a routine called `Main`, which is analogous to the `main()` function of a C program: it is as if the entire program is a call to this function.

In an I7 story file, the code in `Main` is the only code which is not executed in the context of a rulebook, and by design it does as little as possible. The definition is in part "Main" of "OrderOfPlay.i6t", and this is what it does:

(1) Consider the startup rules.
(2) Repeatedly follow the turn sequence rules until `deadflag` is set, which is an I6 variable used to indicate that the game has ended in one way or another.
(3) Follow the shutdown rules.

The startup rules are only considered, not followed, because we cannot risk a procedural rule being the first thing executed: starting up the virtual machine correctly has to be our first priority. (Except for not using the procedural rules, considering a rulebook is identical to following it.)

Briefly, the startup phase takes us to the end of the room description after the banner is printed. The turn sequence covers a complete turn, and runs through from prompting the player for a command to notifying him of any change in score which occurred. The shutdown rules then go from printing the obituary text, through final score, to the question about quitting or restarting.

```
Startup rules is a rulebook. [1]
Turn sequence rules is a rulebook. [2]
Shutdown rules is a rulebook. [3]
```

§**14.** Now a set of rulebooks to do with the passage of time.

```
Scene changing rules is a rulebook. [4]
When play begins is a rulebook. [5]
When play ends is a rulebook. [6]
When scene begins is a scene based rulebook. [7]
When scene ends is a scene based rulebook. [8]
Every turn rules is a rulebook. [9]
```

§**15.** The action machinery requires some 16 rulebooks to work, though that is the result of gradual simplification – in 2006 it required 25, for instance. The "action-processing" rulebook, like the turn sequence rulebook, is a master of ceremonies: it belongs to the Standard Rules and is only rarely if at all referred to by users.

As remarked above, it's something of a historical accident that "actor" is a rulebook variable belonging to the action-processing rules (and thus in scope for every rulebook it employs) while "noun" and "second noun" are global variables (and thus in scope everywhere).

The main action-processing rulebook delegates most of its detailed work to a subsidiary, the "specific action-processing" rulebook, at the point where what rulebooks we consult next depends on what the action is (hence "specific") – see below for more on how check/carry out/report rules are filed.

```
Action-processing rules is a rulebook. [10]
The action-processing rulebook has a person called the actor.
Setting action variables is a rulebook. [11]
The specific action-processing rules is a rulebook. [12]
The specific action-processing rulebook has a truth state called action in world.
The specific action-processing rulebook has a truth state called action keeping silent.
The specific action-processing rulebook has a rulebook called specific check rulebook.
The specific action-processing rulebook has a rulebook called specific carry out rulebook.
The specific action-processing rulebook has a rulebook called specific report rulebook.
The specific action-processing rulebook has a truth state called within the player's sight.
The player's action awareness rules is a rulebook. [13]
```

§**16.** The rules on accessibility and visibility, which control whether an action is physically possible, have named outcomes as a taste of syntactic sugar.

```
Accessibility rules is a rulebook. [14]
Reaching inside rules is an object-based rulebook. [15]
Reaching inside rules have outcomes allow access (success) and deny access (failure).
Reaching outside rules is an object-based rulebook. [16]
Reaching outside rules have outcomes allow access (success) and deny access (failure).
Visibility rules is a rulebook. [17]
Visibility rules have outcomes there is sufficient light (failure) and there is
insufficient light (success).
```

§**17.** Two rulebooks govern the processing of asking other people to carry out actions:

```
Persuasion rules is a rulebook. [18]
Persuasion rules have outcomes persuasion succeeds (success) and persuasion fails (failure).
Unsuccessful attempt by is a rulebook. [19]
```

§**18.** Next, the six classic rulebooks best known to users of Inform. It's perhaps an unfortunate point of the design that there are so many as six: that seems rather a lot of stages to go through, and indeed means that there is sometimes some ambiguity about which rulebook to use if one wants to achieve a given effect. There are really two reasons why things are done this way:

(a) To try to encourage a distinction between:

  (i) the general implementation of an action, made with carry out, check and report rules – say, a "photographing" action which could be used in any situation and could be copied and pasted into another project; and

  (ii) the contingent rules applying in particular situations in play, made with before, instead and after rules, such as that custodians at the Metropolitan Museum of Art forbid flash photography.

(b) To improve the efficiency of action-processing by forcing control to run only through those carry out, check and report rules which can possibly be relevant to the current action. Whereas all before, instead and after rules are all piled up together in their own rulebooks, check, carry out and report rules are divided up into specialised rulebooks tied to the particular action in progress. Thus on a taking action, the six stages followed are before, instead, check taking, carry out taking, after and report taking.

During play, then, the three rulebooks "check", "after" and "report" are completely empty. This is the result of a reform in April 2007 which wasn't altogether popular. Before then, NI rather cleverly filed rules like "Check doing something with the haddock" in the generic "check" rulebook and ran this rulebook as part of the action processing sequence. But this clearly broke principle (i) above, and meant that the six-stage process – already quite complicated enough – was actually a nine-stage process only pretending, by deceitful syntax, to be a six-stage one. Check rules sometimes appeared to be filed in the wrong order, breaking the ordinary precedence conventions, and this was not due to a bug but because they were only pretending all to be in the same rulebook. Still more clever indexing and rule-ordering tricks ameliorated this a little, but in the end it was just a bad design: withdrawing the ability to make check, carry out and report rules apply to multiple actions resulted in much cleaner code, and also a clearer conceptual definition of what these rulebooks were for. (But users still *didn't* like the change: actual functionality was withdrawn.)

So if they are always empty and never used, why are the three rulebooks called simply "check", "after" and "report" created in the first place? The answer is that this is a convenience for parsing rule preambles in NI: it provides a temporary home for such rules before they are divided up into their specific rulebooks, and it also makes it easier for NI to detect and give a helpful Problem message in response to rules like "Check taking or dropping the perch" which can't be filed anywhere in our scheme, and so have to be forbidden.

```
Before rules is a rulebook. [20]
Instead rules is a rulebook. [21]
Check rules is a rulebook. [22]
Carry out rules is a rulebook. [23]
After rules is a rulebook. [24]
Report rules is a rulebook. [25]
```

§**19.** The final rulebook in this roundup is one used for parsing ambiguous commands during play. This looks like the job of an activity rather than a rulebook, but (i) we want information in the form of a nicely-named five point scale of responses, rather than wanting something to be done, and (ii) it doesn't really make any sense to split that question into a before, for and after stage. So this is a named rulebook instead.

```
The does the player mean rules are a rulebook. [26]
The does the player mean rules have outcomes it is very likely, it is likely, it is possible,
it is unlikely and it is very unlikely.
```

§**20.** “Does the player mean” is essentially a front end for the I6 parser’s `ChooseObjects` entry point, which relies on numerical scores to assess the likelihood of possible choices. That makes it useful to have an I6 wrapper function which consults the “does the player mean” rulebook, translates the named outcomes into a score from 0 to 4, and returns that. Note that if the rulebook makes no decision and has no outcome, we return the middle-of-the-road value 2. (This wrapper routine looks as if it belongs in the template I6 layer, but having it here allows it to use the `(+` and `+)` escapes.)

```
Include (-
    [ CheckDPMR result sinp1 sinp2 rv;
        sinp1 = inp1; sinp2 = inp2; inp1 = noun; inp2 = second;
        rv = FollowRulebook( (+does the player mean rules+) );
        inp1 = sinp1; inp2 = sinp2;
        if ((rv) && RulebookSucceeded()) {
            result = ResultOfRule();
            if (result == (+ it is very likely outcome +) ) return 4;
            if (result == (+ it is likely outcome +) ) return 3;
            if (result == (+ it is possible outcome +) ) return 2;
            if (result == (+ it is unlikely outcome +) ) return 1;
            if (result == (+ it is very unlikely outcome +) ) return 0;
        }
        return 2;
    ];
-);
```

§**21.** And that’s it: all 25 of the named rulebooks now exist. There will, of course, be hundreds more rulebooks soon, created automatically as activities and actions are created – when we create the “dropping” action, for instance, we also create the “check dropping”, “carry out dropping” and “report dropping” rulebooks – but there are no more stand-alone rulebooks.

§**22. Rules.** At run-time, the value of a rule is the (packed) address of an I6 routine. In the case of rules created in I7 source text, the body of the rule definition is compiled by NI to a new I6 routine which carries it out. But there are also primitive rules which are implemented in the template I6 layer, and these need no I7 source text definitions: instead we simply tell NI the name of the I6 routine which will be handling this rule, and that it need not bother to create one for itself.

An example of this is provided by the first rule we shall create: the “little-used do nothing rule”. This is aptly named on both counts. We never follow it, we never put it into any rulebook. It exists only so that variables and properties with the kind of value “rule” can be initialised automatically to a safely neutral default value. It makes no decision.

```
Section SR2/8 - The Standard Rules

The little-used do nothing rule translates into I6 as "LITTLE_USED_DO_NOTHING_R".

Include (-
[ LITTLE_USED_DO_NOTHING_R; rfalse; ];
-);
```

§**23. Startup.** The startup rulebook is considered but not followed (see above) and so is immunised from the effect of procedural rules.

Every rulebook contains a (possibly empty) run of "first" rules, then a (possibly empty) run of miscellaneous rules, then a (possibly empty) run of "last" rules. It's unusual to have more than one rule anchored to either end as "first" or "last", but entirely legal, and we make use of that ability here.

The "first" rules here are the ones which get the basic machinery working to the point where it is safe to run arbitrary I7 source text. They necessarily do very low-level things, and it is not guaranteed that I7 phrases will behave to specification if executed before these early rules have finished. So it is hazardous to obstruct or alter them.

(a) The "initialise memory rule" starts up the memory allocation heap, if there is one, and sets some essential I6 variables. If there is any rule not to meddle with, this is it.

(b) The "virtual machine startup rule" carries out necessary steps to begin execution on the virtual machine in use: this entails relatively little on the Z-machine versions 5 or 8, but can involve extensive work to get the screen display working on Glulx or Z6. Before anything else happens, however, the "starting the virtual machine" activity (see below) is carried out: again, this is done in a careful way which avoids procedural rules firing.

(c) The "seed random number generator rule" seeds the RNG to a fixed value if NI has requested this (which it does in response to the `-rng` command line switch, which is in turn used by the `intest` testing utility: it's a way to make deterministic tests of programs which use random values).

(d) The "update chronological records rule" is described in further detail below, since it appears both here and also in the turn sequence rulebook. Here it's providing us with a baseline of initial truths from which we can later assess conditions such as "the marble door has been open". A subtle and questionable point of the design is that this rule is placed at a time when the object representing the player is not present in the model world. This is done to avoid regarding the initial situation as one of the turns for purposes of a rule preamble like "... when the player has been in the Dining Room for three turns". It's as if the player teleports into an already-existing world, like some Star Trek crewman, just in time for the first command.

(e) And the "position player in model world rule" completes the initial construction of the spatial model world.

(f) The "start in the correct scenes rule" ensures that we start out in the correct scenes. (This can't wait, because it's just conceivable that somebody has written a rule with a preamble like "When play begins during the Hunting Season...": it's also where the scene Entire Game begins.) That completes the necessary preliminaries before ordinary I7 rules can be run.

```
The start in the correct scenes rule is listed first in the startup rulebook. [6th.]
The position player in model world rule is listed first in the startup rulebook. [5th.]
The update chronological records rule is listed first in the startup rulebook. [4th.]
The seed random number generator rule is listed first in the startup rulebook. [3rd.]
The virtual machine startup rule is listed first in the startup rulebook. [2nd.]
The initialise memory rule is listed first in the startup rulebook. [1st.]

The virtual machine startup rule translates into I6 as "VIRTUAL_MACHINE_STARTUP_R".
The initialise memory rule translates into I6 as "INITIALISE_MEMORY_R".
The seed random number generator rule translates into I6 as "SEED_RANDOM_NUMBER_GENERATOR_R".
The update chronological records rule translates into I6 as "UPDATE_CHRONOLOGICAL_RECORDS_R".
The position player in model world rule translates into I6 as "POSITION_PLAYER_IN_MODEL_R".

This is the start in the correct scenes rule: consider the scene changing rules.
```

§**24.** The remaining rules, though, are fair game for alteration, and as if to prove the point they are all written in standard I7 source text. Note that the baseline score is set only after the when play begins rulebook has finished, because games starting with a non-zero score normally do this by changing the score in a when play begins rule: and we don't want such a change to be notified to the player as if it has happened through some action.

```
The when play begins stage rule is listed in the startup rulebook.
The fix baseline scoring rule is listed in the startup rulebook.
The display banner rule is listed in the startup rulebook.
The initial room description rule is listed in the startup rulebook.

This is the when play begins stage rule: follow the when play begins rulebook.

This is the fix baseline scoring rule: now the last notified score is the score.

This is the display banner rule: say "[banner text]".

This is the initial room description rule: try looking.
```

§**25. The turn sequence.** In each turn, a command is read and parsed from the keyboard, and any action(s) that requested is or are processed. (And may in turn cause other actions, which must also be processed.) There is then a fair amount of business needed to end one turn and get ready for another.

The turn sequences rulebook terminates early if `deadflag` becomes set at any point, so the last turn of play will be incomplete. Besides that consideration, it can also end early if the command for the turn was for an out-of-world action such as saving the game: in such cases, the "generate action rule" stops the rulebook once the action has been fully processed. All the same, play basically consists of running down this list of rules over and over again.

The turn sequence rulebook is the only one protected from being ignored by a procedural rule: ignoring it would lock the story file into an endless loop, so a run-time problem is issued instead if this is tried. (In pre-2008 drafts of Inform 7, ignoring what was then the turn sequence rulebook was a legitimate tactic, so the RTP message explains how to achieve the same effect by new methods.)

§**26.** The "first" rules in the turn sequence cover us up to the end of the events which take place in the model world during this turn's action(s).

(a) The "parse command rule" prints up the prompt, reads a command from the keyboard, parses it into dictionary words, deals with niceties such as UNDO or OOPS, and then runs it through the traditional I6 parser to turn it into a request for an action or list of actions. But see note below.

(b) The "generate action rule" then either sends a single action to the action-processing rules, or else runs through the list, printing the noun up with a colon each time and then sending the action to the action-processing rules. But see note below.

(c) We then run the scene changing rulebook, because people often write every turn rules which are predicated on particular scenes ("Every turn during the Grand Waltz: ..."), and such rules will fail if we haven't kept up with possible scene changes arising from something done in the action(s) just completed.

(d) The "every turn stage rule" follows the every turn rulebook. This earns its place among the "first" rules in order for it to have priority over all the other book-keeping rules at the end of a turn – including any which the user, or an extension included by the user, chooses to add to the turn sequence rules.

An unusual point here is that the "parse command rule" and the "generate action rule" are written such that they *do nothing unless the turn sequence rulebook is being followed at the top level* (by `Main`, that is). This prevents them from being used recursively, which would not work properly, and enables a popular trick from the time before the 2008 reform to keep working: we can simulate six turns going by in which the player does nothing by running "follow the turn sequence rules" six times in a row. Everything happens exactly as it should – the turn count, the time of day, timed events, and so on – except that no commands are read and no consequent actions generated.

```
A first turn sequence rule (this is the every turn stage rule): follow the every turn rules. [4th.]
A first turn sequence rule: consider the scene changing rules. [3rd.]
The generate action rule is listed first in the turn sequence rulebook. [2nd.]
The parse command rule is listed first in the turn sequence rulebook. [1st.]
```

§**27.** Three miscellaneous things then happen, all implemented by primitives in the template I6 layer:

(e) The "timed events rule" is the one which causes other rules, keyed to particular times of day, to fire.

(f) The "advance time rule" then causes the "time of day" global variable to advance by the duration of a single turn, which by default is 1 minute.

(g) The "update chronological records rule" tells the chronology machine that a slice of time has been completed. Inform can only decide past tense conditions like "the black door has been open" by continuously measuring the present to see if the black door is open now, and making a note for future reference if it is. But of course it's impossible for any computer to continuously do anything, and besides, Inform has other calls on it. What in fact happens is the Inform performs these measurements at "chronology points", which are strategic moments during play, and this is one of them.

```
The timed events rule is listed in the turn sequence rulebook.
The advance time rule is listed in the turn sequence rulebook.
The update chronological records rule is listed in the turn sequence rulebook.
```

§**28.** We now come to the rules anchored at the end, using "last". This part of the rulebook is reserved for book-keeping which has to happen positively at the end of the turn.

(h) First, we check for scene changes again. We did this only a short while ago, but scene changes might well have arisen as a result of rules which fired during the every turn rulebook, or from timed events, or in some other way, and it's important to start the next turn in the correct scene – so we check again to make sure.

(i) Then we run the "adjust light rule". Keeping track of light and darkness is quite difficult, and potentially also quite slow: it's not unlike a sort of discretised version of ray-tracing, with many light sources and barriers to think about (some transparent, some opaque). So we do this as little as possible: once per turn we calculate whether the player is in light or not, and act accordingly if so.

(j) The "note object acquisitions rule" does two things:

  (i) Gives the "handled" property to everything carried or worn by the player.

  (ii) Changes the current player's holdall in use, if necessary. (That's to say: if the player has dropped his previous player's holdall, we try to find a new one to use from his remaining possessions.)

(k) The "notify score changes rule" tells the player if the score has changed during the turn, or rather, since the last time either this rule or the startup "fix baseline scoring rule" ran. (If the score were to change in the course of an out-of-world action, it would be notified a turn late, but of course out-of-world actions are not supposed to do that sort of thing.)

```
A last turn sequence rule: consider the scene changing rules. [3rd from last.]
The adjust light rule is listed last in the turn sequence rulebook. [2nd from last.]
The note object acquisitions rule is listed last in the turn sequence rulebook. [Penultimate.]
The notify score changes rule is listed last in the turn sequence rulebook. [Last.]
This is the notify score changes rule:
    if the score is not the last notified score:
        issue score notification message;
        now the last notified score is the score;
```

§**29.** That's it, but we need to map these I7 rule names onto the names of their I6 primitives in the template layer.

```
The adjust light rule translates into I6 as "ADJUST_LIGHT_R".
The advance time rule translates into I6 as "ADVANCE_TIME_R".
The generate action rule translates into I6 as "GENERATE_ACTION_R".
The note object acquisitions rule translates into I6 as "NOTE_OBJECT_ACQUISITIONS_R".
The parse command rule translates into I6 as "PARSE_COMMAND_R".
The timed events rule translates into I6 as "TIMED_EVENTS_R".
```

§**30. Shutdown.** Goodbye is not the hardest word to say, but it does involve a little bit of work. It might not actually be goodbye, for one thing: if this rulebook ends in success, then we go back to repeating the turn sequence rulebook just as if nothing had happened.

(a) The "when play ends stage rule" follows the rulebook of the same name.
(b) The "resurrect player if asked rule" does nothing unless one of the "when play ends" rules ran the "resume the game" phrase, in which case it stops the rulebook with success (see above).
(c) The "print player's obituary rule" carries out the activity of nearly the same name (see below).
(d) The "ask the final question rule" asks the celebrated "Would you like to RESTART, RESTORE a saved game or QUIT?" question, and acts on the consequences. It can also cause an UNDO, and on a victorious ending may carry out the "amusing a victorious player" activity (see below). The rule only actually ends in the event of a QUIT: an UNDO, RESTART or RESTORE is performed at the hardware level by the virtual machine and destroys the current execution context entirely.

```
The when play ends stage rule is listed first in the shutdown rulebook.
The resurrect player if asked rule is listed last in the shutdown rulebook.
The print player's obituary rule is listed last in the shutdown rulebook.
The ask the final question rule is listed last in the shutdown rulebook.

This is the when play ends stage rule: follow the when play ends rulebook.
This is the print player's obituary rule:
    carry out the printing the player's obituary activity.

The resurrect player if asked rule translates into I6 as "RESURRECT_PLAYER_IF_ASKED_R".
The ask the final question rule translates into I6 as "ASK_FINAL_QUESTION_R".
```

§**31. Scene changing.** Scene changing is handled by a routine called `DetectSceneChange` which is compiled directly by NI: this is so primitive that it can't even be handled at the template layer. The rulebook is all a little elaborate given that it contains only one rule, but it's possible to imagine extensions which need to do book-keeping similar to scene changing and which want to make use of this opportunity.

```
The scene change machinery rule is listed last in the scene changing rulebook.

The scene change machinery rule translates into I6 as "DetectSceneChange".
```

§**32.** We couldn't do this earlier (because creating a scene automatically generates two rulebooks, and that would have thrown the rulebook numbering), so let's take this opportunity to define the "Entire Game" scene:

```
Section SR2/9 - The Entire Game scene

The Entire Game is a scene.
The Entire Game begins when the story has not ended.
The Entire Game ends when the story has ended.
```

§**33. Action-processing.** Action-processing happens on two levels: an upper level, handled by the main "action-processing" rulebook, and a lower level, "specific action-processing". This division clearly complicates matters, so why do we do it? It turns out be convenient for several reasons:

(a) Out-of-world actions like "saving the game" need to run through the lower level, or they won't do anything at all, but must not run through the upper level, or in-world rules (before or instead rules, for instance) might prevent them from happening.

(b) Requested actions such as generated by a command like "CLARK, BLOW WHISTLE" have the reverse behaviour, being handled at the upper level but not the lower. (If Clark should agree, a definite non-request action "Clark blowing the whistle" is generated afresh: that one does indeed get to the lower level, but the original request action doesn't.)

(c) Specific action-processing has a rather complicated range of outcomes: it must succeed or fail, according to whether the action either reaches the carry out rules or is converted into another action which does, but also ensure that in the event of failure, the exact rule causing the failure is recorded in the "reason the action failed" variable.

(d) The specific action-processing stage is where we have to split consideration into action-specific rulebooks (like "check taking") rather than general ones (like "instead"). To get this right, we want to use some rulebook variables, and these need to be set at exactly the correct moment. It's tricky to arrange for that in the middle of a big rulebook, but easy to do so at the beginning, so we want the start of the SA-P stage to happen at the start of a rulebook.

This does mean that an attempt by the user to move the before stage to just after the check stage (say) will fail – the before and check stages happen in different rulebooks, so no amount of rearranging will do this. (Procedural rules could still achieve this very doubtfully wise effect in any case.)

§**34.** The upper level of action-processing consists of seeing whether the actor's current situation forbids the action from being tried: does anybody or anything intervene to stop it? Are there are any basic reasons of physical realism why nobody could possibly try it? Does it require somebody else to cooperate who in fact chooses not to?

Doctrinally, the before stage must see actions before anything else can happen. (It needs the absolute freedom to start fresh actions and dispose of the one originally intended, sure in the knowledge that no other rules have been there first.) So before-ish material is anchored at the "first" end of the rulebook.

The other book-end on the shelf is provided by the instead stage, which doctrinally happens only when the action is now thought to be physically reasonable. So this is anchored at the "last" end. (In fact it is followed by several more rules also anchored there, but this is essentially just the clockwork machinery showing through: they aren't realism checks.)

Miscellaneous general rules to do with physical realism are placed between the two book-ends, then, and this is where any newly created action-processing rules created by the user or by extensions will go.

```
Section SR2/10 - Action processing
The before stage rule is listed first in the action-processing rules. [3rd.]
The set pronouns from items from multiple object lists rule is listed first in the
    action-processing rules. [2nd.]
The announce items from multiple object lists rule is listed first in the
    action-processing rules. [1st.]
The basic visibility rule is listed in the action-processing rules.
The basic accessibility rule is listed in the action-processing rules.
The carrying requirements rule is listed in the action-processing rules.
The instead stage rule is listed last in the action-processing rules. [4th from last.]
The requested actions require persuasion rule is listed last in the action-processing rules.
The carry out requested actions rule is listed last in the action-processing rules.
The descend to specific action-processing rule is listed last in the action-processing rules.
The end action-processing in success rule is listed last in the action-processing rules. [Last.]
```

§**35.** As we shall see, most of these rules are primitives implemented by the template I6 layer, but five are very straightforward.

```
This is the set pronouns from items from multiple object lists rule:
    if the current item from the multiple object list is not nothing,
        set pronouns from the current item from the multiple object list.
This is the announce items from multiple object lists rule:
    if the current item from the multiple object list is not nothing,
        say "[current item from the multiple object list]: [run paragraph on]".
This is the before stage rule: abide by the before rules.
This is the instead stage rule: abide by the instead rules.
```

§**36.** The final rule in the rulebook always succeeds: this ensures that action-processing always makes a decision. (I7's doctrine is that an action "succeeds" if and only if its carry-out stage is reached, so any action getting right to the end of this rulebook must have succeeded.)

```
This is the end action-processing in success rule: rule succeeds.
```

§**37.** The action-processing rulebook contains six primitives:

(1) The "basic visibility rule" checks the action to see if it requires light, and if so, and if the actor is the player and in darkness, asks the visibility rules (see below) whether the lack of light should stop the action. (It would be cleaner to apply this rule to all actors, but we would need much more extensive light calculations to do this.)

(2) The "basic accessibility rule" checks the action to see if it requires the noun to be touchable, and if so, asks the accessibility rulebook to adjudicate (see below); then repeats the process for the second noun.

(3) The "carrying requirements rule" checks the action to see if it requires the noun to be carried. If so, but the noun is not carried, it generates an implicit taking action (in effect, "try silently taking *N*"); and if that fails, then the rulebook is halted in failure. The process is then repeated for the second noun.

(4) If the action is one where the player requests somebody else to do something, the "requested actions require persuasion rule" asks the persuasion rulebook for permission.

(5) If the action is one where the player requests somebody else to do something, the "carry out requested actions rule" starts a new action by the person asked, and looks at the result: if it failed, the "unsuccessful attempt by" rulebook is run to tell the player what has (not) happened. (If that does nothing, a bland message is printed.) In all cases, the original action of requesting is ended in success: a success because, whatever happened, the request succeeded in making the actor try to do something.

(6) The "descend to specific action-processing rule" really only runs the specific action-processing rulebook, but it's implemented as a primitive in the template I6 layer because it must also find out which are the specific check, carry out and report rulebooks for the current action (for instance, "check taking" is the specific check rulebook for the "taking" action – which seems obvious from the names: but at run-time, the names aren't so visible).

```
The basic accessibility rule translates into I6 as "BASIC_ACCESSIBILITY_R".
The basic visibility rule translates into I6 as "BASIC_VISIBILITY_R".
The carrying requirements rule translates into I6 as "CARRYING_REQUIREMENTS_R".
The requested actions require persuasion rule translates into I6 as "REQUESTED_ACTIONS_REQUIRE_R".
The carry out requested actions rule translates into I6 as "CARRY_OUT_REQUESTED_ACTIONS_R".
The descend to specific action-processing rule translates into I6 as
"DESCEND_TO_SPECIFIC_ACTION_R".
```

§**38. Specific action-processing.** And now we descend to the lower level, which is much easier to understand.

```
The work out details of specific action rule is listed first in the specific
action-processing rules.

A specific action-processing rule
    (this is the investigate player's awareness before action rule):
    consider the player's action awareness rules;
    if rule succeeded, now within the player's sight is true;
    otherwise now within the player's sight is false.

A specific action-processing rule (this is the check stage rule):
    anonymously abide by the specific check rulebook.

A specific action-processing rule (this is the carry out stage rule):
    consider the specific carry out rulebook.

A specific action-processing rule (this is the after stage rule):
    if action in world is true, abide by the after rules.

A specific action-processing rule
    (this is the investigate player's awareness after action rule):
    if within the player's sight is false:
        consider the player's action awareness rules;
        if rule succeeded, now within the player's sight is true;

A specific action-processing rule (this is the report stage rule):
    if within the player's sight is true and action keeping silent is false,
        consider the specific report rulebook;

The last specific action-processing rule: rule succeeds.
```

§**39.** The unusual use of "anonymously abide by" above is a form of "abide by" which may be worth explaining. Suppose rule X consists of an instruction to abide by rulebook Y, and suppose that Y in fact fails when it reaches Y_k. If the ordinary "abide by" is used then the action will be deemed to have failed at rule X, but if "anonymously abide by" is used then it will be deemed to have failed at Y_k. (Thus X remains anonymous: it can never be the culprit.) We only use this at the check stage, because the carry out, report and after stages are not allowed to fail the action. (The after stage is allowed to end it, but not in failure.)

§**40.** The specific action-processing rulebook is probably more fruitful than the main one if we want to modify what happens. For instance:

> This is the sixth sense rule: if the player is not the actor, say "You sense that [the actor] is up to something."
>
> The sixth sense rule is listed before the carry out stage rule in the specific action-processing rules.

...produces the message at a time when the action is definitely possible and will succeed, but before anything has been done.

§**41.** The only rule not spelled out was the primitive "work out details of specific action rule", which initialises the rulebook variables so that they record the action's specific check, carry out and report rulebooks, and whether or not it is in world.

```
The work out details of specific action rule translates into I6 as
"WORK_OUT_DETAILS_OF_SPECIFIC_R".
```

§42. Player's action awareness. This rulebook decides whether or not an action by somebody should be routinely reported to the player: is he aware of it having taken place? If the rulebook positively succeeds then he is, and otherwise not.

```
A player's action awareness rule
    (this is the player aware of his own actions rule):
    if the player is the actor, rule succeeds.
A player's action awareness rule
    (this is the player aware of actions by visible actors rule):
    if the player is not the actor and the player can see the actor, rule succeeds.
A player's action awareness rule
    (this is the player aware of actions on visible nouns rule):
    if the noun is a thing and the player can see the noun, rule succeeds.
A player's action awareness rule
    (this is the player aware of actions on visible second nouns rule):
    if the second noun is a thing and the player can see the second noun, rule succeeds.
```

§43. Accessibility. The "accessibility" rulebook is not very visible to users: it's another behind-the-scenes rulebook for managing the decision as to whether the actor can touch any items which the intended action requires him to be able to reach.

In its default configuration, it contains only the "access through barriers" rule. This in all circumstances either succeeds or fails: in other words, it makes a definite decision, and this is why it is anchored as "last" in the rulebook. If users or extensions want to tweak accessibility at this general level, any new rules they add will get the chance to decide before the "access through barriers" rule settles the matter. But in practice we expect most users to work with one of the two reaching rulebooks instead.

```
Section SR2/11 - Accessibility

The access through barriers rule is listed last in the accessibility rules.

The access through barriers rule translates into I6 as "ACCESS_THROUGH_BARRIERS_R".
```

§44. Reaching inside. What the access through barriers rule does is to try to construct a path through the object containment tree from the actor to the item needed, and to apply the reaching inside or reaching outside rulebooks each time this path passes inside or outside of a container (or room). (Supporters never form barriers.)

```
The can't reach inside rooms rule is listed last in the reaching inside rules. [Penultimate.]
The can't reach inside closed containers rule is listed last in the reaching
inside rules. [Last.]

The can't reach inside closed containers rule translates into I6 as "CANT_REACH_INSIDE_CLOSED_R".
The can't reach inside rooms rule translates into I6 as "CANT_REACH_INSIDE_ROOMS_R".
```

§45. Reaching outside. And, not quite symmetrically since we don't need to block room-to-room reaching on both the inbound and outbound directions,

```
The can't reach outside closed containers rule is listed last in the reaching outside rules.

The can't reach outside closed containers rule translates into I6 as "CANT_REACH_OUTSIDE_CLOSED_R".
```

§**46. Visibility.** The visibility rulebook actually has the opposite sense to the one the name might suggest. It is applied only when the player (not any other actor) tries to perform an action which requires light to see by; the action is blocked if the rulebook succeeds. (Well, but we could hardly call it the invisibility rulebook. The name is supposed to suggest that the consideration of visibility is being applied: rather the way cricket matches end with a declaration that "light stopped play", meaning of course that darkness did.)

```
The can't act in the dark rule is listed last in the visibility rules.

The last visibility rule (this is the can't act in the dark rule): if in darkness, rule succeeds.
```

§**47. Does the player mean.** This rulebook is akin to a set of preferences about how to interpret the player's commands in cases of ambiguity. No two Inform users are likely to agree about the best way to decide these, so we are fairly hands-off, and make only one rule as standard:

```
Does the player mean taking something which is carried by the player
    (this is the very unlikely to mean taking what's already carried rule):
    it is very unlikely.
```

§**48.** And that completes the creation and stocking of the 25 rulebooks. More than half of them are initially empty, including before, instead and after – at the end of the day, these rulebooks are hooks allowing the user to change the ordinary behaviour of things, but ordinariness is exactly what the Standard Rules is all about.

§**49. Adjectives applied to values.** There is a small stock of built-in adjectives for values.

```
Section SR2/12 - Adjectival definitions

Definition: a number is even rather than odd if the remainder after dividing it by 2 is 0.
Definition: a number is positive if it is greater than zero.
Definition: a number is negative if it is less than zero.

Definition: a text is empty rather than non-empty if it is "".

Definition: an indexed text is empty rather than non-empty if I6 routine
    "INDEXED_TEXT_TY_Empty" says so (it contains no characters).

A scene can be recurring or non-recurring. A scene is usually non-recurring.
The Entire Game is recurring.

Definition: a scene is happening if I6 condition "scene_status-->(*1-1)==1"
    says so (it is currently taking place).

Definition: a table name is empty rather than non-empty if the number of filled rows in it is 0.
Definition: a table name is full rather than non-full if the number of blank rows in it is 0.

Definition: a rulebook is empty rather than non-empty if I6 routine "RulebookEmpty" says so (it
    contains no rules, so that following it does nothing and makes no decision).

Definition: an activity is empty rather than non-empty if I6 routine "ActivityEmpty" says so (its
    before, for and after rulebooks are all empty).
Definition: an activity is going on if I6 routine "TestActivity" says so (one
    of its three rulebooks is currently being run).

Definition: a list of values is empty rather than non-empty if I6 routine
    "LIST_OF_TY_Empty" says so (it contains no entries).

Definition: a use option is active rather than inactive if I6 routine
    "TestUseOption" says so (it has been requested in the source text).

Definition: a relation is equivalence if I6 routine
    "RELATION_TY_EquivalenceAdjective" makes it so (it is an equivalence
    relation, that is, it relates in groups).
Definition: a relation is symmetric if I6 routine
    "RELATION_TY_SymmetricAdjective" makes it so (it is a symmetric relation,
    that is, it's always true that X is related to Y if and only if Y is
    related to X).
Definition: a relation is one-to-one if I6 routine
    "RELATION_TY_OToOAdjective" makes it so (it is a one-to-one relation,
    that is, any given X can relate to only one Y, and vice versa).
Definition: a relation is one-to-various if I6 routine
    "RELATION_TY_OToVAdjective" makes it so (it is a one-to-various
    relation, that is, any given Y has only one X such that X relates to Y).
Definition: a relation is various-to-one if I6 routine
    "RELATION_TY_VToOAdjective" makes it so (it is a various-to-one
    relation, that is, any given X relates to only one Y).
Definition: a relation is various-to-various if I6 routine
    "RELATION_TY_VToVAdjective" makes it so (it is a
    various-to-various relation, that is, there are no limitations on how many
    X can relate to a given Y, or vice versa).
Definition: a relation is empty rather than non-empty if I6 routine
    "RELATION_TY_Empty" makes it so (it does not relate any values, that is,
    R(x,y) is false for all x and y).
```

§**50. Scene descriptions.** And there is one build-in value property for values.

```
Section SR2/13 - Scene descriptions

A scene has a text called description.
When a scene (called the event) begins (this is the scene description text rule):
    if the description of the event is not "",
        say "[the description of the event][paragraph break]".
```

§**51. Command parser errors.** This abstracts a set of return codes from the I6 parser, which are written there as constants with the notation `*_ETYPE`.

```
Section SR2/14 - Command parser errors

A command parser error is a kind of value. The command parser errors are
didn't understand error,
only understood as far as error,
didn't understand that number error,
can only do that to something animate error,
can't see any such thing error,
said too little error,
aren't holding that error,
can't use multiple objects error,
can only use multiple objects error,
not sure what it refers to error,
excepted something not included error,
not a verb I recognise error,
not something you need to refer to error,
can't see it at the moment error,
didn't understand the way that finished error,
not enough of those available error,
nothing to do error,
noun did not make sense in that context error,
referred to a determination of scope error, and
I beg your pardon error.
```

```
The latest parser error is a command parser error that varies.
The latest parser error variable translates into I6 as "etype".
```

SR3 - Activities A/sr3

Purpose

The built-in activities and their default stock of rules; the locale description mechanism.

§**1.** These must not be created until the basic rulebooks are in place, because creating any activity automatically creates three rulebooks as well.

Once again, there are no activities or assumptions about them built into NI, but the template I6 layer expects the following 26 activities to be created and in this order. (That is, the order here must exactly match that of the `*_ACT` constant definitions made in `Definitions.i6t`.) The activities are fairly completely described in the Inform documentation, so the only notes here concern implementation.

§**2.** We start with activities used in describing things and rooms.

Most of the activities begin with all three rulebooks empty, since by default they do not intervene. This one, however, has a "last" for-rule, because it's required to do something definite: it must actually print a name, and the last for-rule is the default way to do this. (The assumption is that any other for-rule added by the user will halt the rulebook before the default implementation is reached.) We also include a before rule which marks any item whose name is being printed with the "mentioned" property, for reasons to be found below.

```
Part SR3 - Activities

Section SR3/1 - Definitions

Printing the name of something (documented at act_pn) is an activity. [0]

Before printing the name of a thing (called the item being printed)
    (this is the make named things mentioned rule):
    now the item being printed is mentioned.

The standard name printing rule is listed last in the for printing the name rulebook.
The standard name printing rule translates into I6 as "STANDARD_NAME_PRINTING_R".

Printing the plural name of something (documented at act_ppn) is an activity. [1]

Rule for printing the plural name of something (called the item) (this is the standard
    printing the plural name rule):
    say the printed plural name of the item.
The standard printing the plural name rule is listed last in the for printing
the plural name rulebook.

Printing a number of something (documented at act_pan) is an activity. [2]

Rule for printing a number of something (called the item) (this is the standard
    printing a number of something rule):
    say "[listing group size in words] ";
    carry out the printing the plural name activity with the item.
The standard printing a number of something rule is listed last in the for printing
a number rulebook.
```

§**3.** When they occur in room descriptions, names of things are sometimes supplemented by details:

```
Printing room description details of something (documented at act_details) is an activity. [3]
```

§**4.** Names of things are often formed up into lists, in which they are sometimes grouped together:

```
Listing contents of something (documented at act_lc) is an activity. [4]
The standard contents listing rule is listed last in the for listing contents rulebook.
The standard contents listing rule translates into I6 as "STANDARD_CONTENTS_LISTING_R".
Grouping together something (documented at act_gt) is an activity. [5]
```

§**5.** And such lists of names are formed up in turn into room descriptions. Something which is visible in a room can either have a paragraph of its own or can be relegated to the list of "nondescript" items at the end.

```
Writing a paragraph about something (documented at act_wpa) is an activity. [6]
```

§**6.** When these paragraphs have all gone by, the nondescript items left over are more briefly listed: the following activity gets the chance to change how this is done.

```
Listing nondescript items of something (documented at act_lni) is an activity. [7]
```

§**7.** Darkness behaves, for room description purposes, a little as if it were a room in its own right. Until the 1990s that was almost always how darkness was implemented in IF programs: this persists in I6, but not I7, where the existence of a room-which-is-not-a-room would break type safety.

```
Printing the name of a dark room (documented at act_darkname) is an activity. [8]
Printing the description of a dark room (documented at act_darkdesc) is an activity. [9]
Printing the announcement of darkness (documented at act_nowdark) is an activity. [10]
Printing the announcement of light (documented at act_nowlight) is an activity. [11]
Printing a refusal to act in the dark (documented at act_toodark) is an activity. [12]

The look around once light available rule is listed last in for printing the
announcement of light.

This is the look around once light available rule:
    try looking.
```

§**8.** Two special forms of printing: the status line at the top of the screen, refreshed every turn during play, and the banner which appears at or close to the start of play:

```
Constructing the status line (documented at act_csl) is an activity. [13]
Printing the banner text (documented at act_banner) is an activity. [14]
```

§9. Now a brace of activities to intervene in how the Inform parser does its parsing, arranged roughly in chronological order of their typical use during a single turn of a typed command, its parsing, and final conversion into an action.

The unusual notation "(future action)" here allows NI to parse rule preambles for these activities in a way which would refer to the action which might, at some point in the future, be generated – during parsing we don't of course yet know what that action is, but there is always a current guess at what it might be.

```
Reading a command (documented at act_reading) is an activity. [15]
Deciding the scope of something (future action) (documented at act_ds) is an activity. [16]
Deciding the concealed possessions of something (documented at act_con) is an activity. [17]
Deciding whether all includes something (future action) (documented at act_all)
    is an activity. [18]
The for deciding whether all includes rules have outcomes it does not (failure) and
    it does (success).
Clarifying the parser's choice of something (future action) (documented at act_clarify)
    is an activity. [19]
Asking which do you mean (future action) (documented at act_which) is an activity. [20]
Printing a parser error (documented at act_parsererror) is an activity. [21]
Supplying a missing noun (documented at act_smn) is an activity. [22]
Supplying a missing second noun (documented at act_smn) is an activity. [23]
Implicitly taking something (documented at act_implicitly) is an activity. [24]
```

§10. Here are the default rules for the behaviour of ALL:

```
Rule for deciding whether all includes scenery while taking (this is the
    exclude scenery from take all rule): rule fails.
Rule for deciding whether all includes people while taking (this is the
    exclude people from take all rule): rule fails.
Rule for deciding whether all includes fixed in place things while taking (this
    is the exclude fixed in place things from take all rule): rule fails.
```

§11. The supplying activities are linguistically interesting, for reasons gone into in the paper *Interactive Fiction, Natural Language and Semantic Analysis*: English verbs do not naturally seem to feature optional nouns. Indeed, we say "it rained on Tuesday" where "it" refers to nothing at all, merely because we can't bring ourselves to leave a gap and say just "rained on Tuesday". A better example here would be "it sounded like rain", because we do the same to convey the idea of listening ambiently rather than to any single thing: listening appears to be rare among actions in that it can equally well take a noun as not. Just as English handles this problem by supplying a spurious "it" which appears to mean "the world at large", so Inform handles it by supplying the current location, with the same idea in mind. And the same applies to the sense of smell, which can be similarly defocused.

```
Rule for supplying a missing noun while an actor smelling (this is the ambient odour rule):
    now the noun is the location.

Rule for supplying a missing noun while an actor listening (this is the ambient sound rule):
    now the noun is the location.
```

§12. The following rule is something of a dodge to provide a better parser response to commands like GO or BRETT, GO. (Putting the rule here, and giving it a name, allows the user to override it and thus accept the idea of vague going after all.)

```
Rule for supplying a missing noun while an actor going (this is the block vaguely going rule):
    issue library message going action number 7.
```

§13. The very first rules: those invoked at the earliest possible moment when the virtual machine is starting up. We need this hook, really, for the Glulx VM, which requires various styles to be created.

```
Starting the virtual machine (documented at act_startvm) is an activity. [25]
The enable Glulx acceleration rule is listed first in for starting the virtual machine.
The enable Glulx acceleration rule translates into I6 as "ENABLE_GLULX_ACCEL_R".
```

§14. And the very last rules of all. The obituary is a rare example of a sequence of events in the I6 library having been rolled up into an activity, partly because it's one of the few clear-cut moments where several unconnected things happen in succession.

```
Amusing a victorious player (documented at act_amuse) is an activity. [26]
Printing the player's obituary (documented at act_obit) is an activity. [27]
The print obituary headline rule is listed last in for printing the player's obituary.
The print final score rule is listed last in for printing the player's obituary.
The display final status line rule is listed last in for printing the player's obituary.
The print obituary headline rule translates into I6 as "PRINT_OBITUARY_HEADLINE_R".
The print final score rule translates into I6 as "PRINT_FINAL_SCORE_R".
The display final status line rule translates into I6 as "DISPLAY_FINAL_STATUS_LINE_R".
```

§15. There is one last question: the one which usually reads "Would you like to RESTART, RESTORE a saved game, or QUIT?", but which sometimes provides other options too. The "ask the final question rule" handles this, and does so by repeatedly calling the following activity:

```
Handling the final question is an activity. [28]
```

§16. It follows that this activity *must* at least sometimes do something dramatic to the execution state: perform a quit, for instance. Four primitive rules are available for the drastic things which the activity might wish to do, but these are not placed in any rulebook: instead they are available for anyone who wants to call them. (In the default implementation below, we put references to them into a table.)

```
The immediately restart the VM rule translates into I6 as "IMMEDIATELY_RESTART_VM_R".
The immediately restore saved game rule translates into I6 as "IMMEDIATELY_RESTORE_SAVED_R".
The immediately quit rule translates into I6 as "IMMEDIATELY_QUIT_R".
The immediately undo rule translates into I6 as "IMMEDIATELY_UNDO_R".
```

§**17.** We structure the activity so that the printing of the question and typing of the answer take place at the "before" stage, and then the parsing and acting upon this answer take place at the "for" stage. Reading the keyboard is the last rule in "before". With the "for" stage, the idea is that any extra rule slipped in by the user can take precedence over the default implementation, so the latter is the last there, too.

```
The print the final question rule is listed in before handling the final question.
The print the final prompt rule is listed in before handling the final question.
The read the final answer rule is listed last in before handling the final question.
The standard respond to final question rule is listed last in for handling the final question.

This is the print the final prompt rule: say "> [run paragraph on]".

The read the final answer rule translates into I6 as "READ_FINAL_ANSWER_R".
```

§**18.** That clears away the underbrush and reduces us to two matching tasks: (i) to print the question, (ii) to parse the answer, given that we want to be able to vary the set of choices available.

We do this by reading the options from the Table of Final Question Options. (See below for its default contents.) Each row is an option, whose wording must be placed in the topic column. The final question wording entry can either be text describing the option – e.g., "perform a RESTART" – or can be left blank, making the option a secret one, omitted from the question but still recognised as an answer. The only if victorious entry can be set to make the option available only after a victorious ending, not after a loss; Infocom's traditional AMUSING option behaved thus. Finally, the table specifies what to do if the option is taken: either it provides a rule, or an activity to carry out. (If it provides only an activity, but that activity is empty, then the option is omitted from the question and not recognised as an answer.)

```
Section SR3/2 - Final Question

This is the print the final question rule:
    let named options count be 0;
    repeat through the Table of Final Question Options:
        if the only if victorious entry is false or the story has ended finally:
            if there is a final response rule entry
                or the final response activity entry [activity] is not empty:
                if there is a final question wording entry, increase named options count by 1;
    if the named options count is less than 1, abide by the immediately quit rule;
    say "Would you like to ";
    repeat through the Table of Final Question Options:
        if the only if victorious entry is false or the story has ended finally:
            if there is a final response rule entry
                or the final response activity entry [activity] is not empty:
                if there is a final question wording entry:
                    say final question wording entry;
                    decrease named options count by 1;
                    if the named options count is 0:
                        say "?[line break]";
                    otherwise if the named options count is 1:
                        if the serial comma option is active, say ",";
                        say " or ";
                    otherwise:
                        say ", ";
```

§**19.** And the matching rule to parse and respond to the answer:

```
This is the standard respond to final question rule:
    repeat through the Table of Final Question Options:
        if the only if victorious entry is false or the story has ended finally:
            if there is a final response rule entry
                or the final response activity entry [activity] is not empty:
                if the player's command matches the topic entry:
                    if there is a final response rule entry, abide by final response rule entry;
                    otherwise carry out the final response activity entry activity;
                    rule succeeds;
    issue miscellaneous library message number 8.
```

§**20.** The table of final options is the only material under the heading "Section SR/Q", to make it easy for users to replace with entirely different tables.

These settings are the traditional ones used by Inform since 1995 or so. The UNDO option has customarily been a "secret", though not much of one, since it somewhat cheapens the announcement of a calamity to be immediately offered the chance to reverse it: death, where is thy sting?

```
Section SR3/3 - Final question options

Table of Final Question Options
final question wording only if victorious topic final response rule final response activity
"RESTART" false "restart" immediately restart the VM rule --
"RESTORE a saved game" false "restore" immediately restore saved game rule --
"see some suggestions for AMUSING things to do" true "amusing" -- amusing a victorious player
"QUIT" false "quit" immediately quit rule --
"UNDO the last command" false "undo" immediately undo rule --
```

§21. Locale activities. A "locale description" is a segment of the text produced by LOOK: the "locale" is a clutch of objects at a given level in the object tree. Most room descriptions consist of a top line, a description of the place, and then a single (though often, as here, multi-paragraph) locale:

```
Sentier Le Corbusier
A coastal walk along the rocky shore between Nice and Menton.
```
now the locale for the room Sentier Le Corbusier:
```
A translucent jellyfish has been washed up by the waves.

You can also see a bucket and a spade here.
```

A locale typically contains a run of paragraphs specific to interesting items, especially those not yet picked up, followed by a paragraph which lists the "nondescript" items – those not given paragraphs of their own, such as the bucket and spade. (Some items, though – typically scenery, but also for instance the player – are not even nondescript and do not appear at all.) A locale can contain no interesting paragraphs, or no list of nondescript items, or can even contain neither: that is, it can be entirely empty.

When the player is in or on top of something, multiple locales are described:

```
Sentier Le Corbusier (in the golf cart)
A coastal walk along the rocky shore between Nice and Menton.
```
now the locale for the room Sentier Le Corbusier:
```
A translucent jellyfish has been washed up by the waves.

You can also see a bucket and a spade here.
```
now the locale for the golf cart:
```
In the golf cart you can see a map of Villefranche-sur-Mer.
```

To sum up, the text produced by LOOK consists of a header (produced by the carry out looking rules) followed by one or more locale descriptions (produced by the activity below).

§22. Locale Implementation. When describing a locale, we keep a Table of interesting objects, each associated with a priority – a number indicating how important, and therefore how near to the top of the description, the object is. A special syntax allows us to create the Table with exactly the same number of rows as there are things in the model world: thus, in the worst case where all things are in a single locale, we still will not run out of table rows. (We do this rather than creating a large but fixed-size table because memory is very short in some Z-machine I7 works, so we want to take only what we might actually need. The table structure is not as wasteful as it might look: an experiment with using a number property of things instead showed that this table was actually more efficient, because of the property numbering overhead in the Z-machine memory representation of objects.)

```
Section SR3/4 - Locale descriptions - Unindexed

Table of Locale Priorities
notable-object (an object) locale description priority (a number)
-- --
with blank rows for each thing.

To describe locale for (O - object):
    carry out the printing the locale description activity with O.

To set the/-- locale priority of (O - an object) to (N - a number):
    if O is a thing:
        if N <= 0, now O is mentioned;
        if there is a notable-object of O in the Table of Locale Priorities:
            choose row with a notable-object of O in the Table of Locale Priorities;
            if N <= 0, blank out the whole row;
            otherwise now the locale description priority entry is N;
        otherwise:
```

```
        if N is greater than 0:
            choose a blank row in the Table of Locale Priorities;
            now the notable-object entry is 0;
            now the locale description priority entry is N;
```

§23. Printing the Locale Description. This is handled by the "printing the locale description" activity. The before stage works out which objects might be of interest; the for stage actually prints paragraphs; the after stage is initially empty, but can be used to insert all kinds of interesting information into a room description.

We count the paragraphs printed in a global variable, not an activity variable, since it needs to be consulted in sub-activities whose rules are outside what would be its scope; that doesn't matter, though, since locale descriptions are not nested. (If they were, the above table would fail in any case.)

(1) Disaster would ensue if the user tampered with the "initialise locale description rule", but nobody is likely to do this other than intentionally.
(2) The "find notable locale objects rule" in fact only runs a further activity, the "choosing notable locale objects" activity. The task here is to identify the objects which might by virtue of their location appear in the locale, and to assign each of them a priority number.
(3) The "interesting locale paragraphs rule" goes through all of the notable objects chosen at stage (2), in order of priority, and offers each to yet another activity: the "printing a locale paragraph about" activity. This can either print a paragraph related to the item in question, or demote it as being not even nondescript (by changing its priority to 0). The default is to do nothing, in which case the item becomes nondescript.
(4) The "you-can-also-see rule" prints what is, ordinarily, the final paragraph of the locale description, listing the nondescript items. It goes to some trouble to find out whether these all have a common object tree parent, listing them with "list contents of" if they do: this is so that people who have written rules such as "Rule for printing the name of the blur while listing contents..." will take effect, because the "listing contents" activity will be going on. Provided that the notable objects chosen in (2) are all children of the locale domain, this will always happen. If the user should add rules to make quite different objects also notable, then the "you-can-also-see rule" has to resort to listing in a way which doesn't use the "listing contents" activity – since the list is not in fact a list of the contents of anything.

```
Printing the locale description of something (documented at act_pld) is an activity.

The locale paragraph count is a number that varies.

Before printing the locale description (this is the initialise locale description rule):
    now the locale paragraph count is 0;
    repeat with item running through things:
        now the item is not mentioned;
    repeat through the Table of Locale Priorities:
        blank out the whole row.

Before printing the locale description (this is the find notable locale objects rule):
    let the domain be the parameter-object;
    carry out the choosing notable locale objects activity with the domain;
    continue the activity.

For printing the locale description (this is the interesting locale paragraphs rule):
    let the domain be the parameter-object;
    sort the Table of Locale Priorities in locale description priority order;
    repeat through the Table of Locale Priorities:
        [say "[notable-object entry]...";]
        carry out the printing a locale paragraph about activity with the notable-object entry;
    continue the activity.

For printing the locale description (this is the you-can-also-see rule):
```

```
let the domain be the parameter-object;
let the mentionable count be 0;
repeat with item running through things:
    now the item is not marked for listing;
repeat through the Table of Locale Priorities:
    [say "[notable-object entry] - [locale description priority entry].";]
    if the locale description priority entry is greater than 0,
        now the notable-object entry is marked for listing;
    increase the mentionable count by 1;
if the mentionable count is greater than 0:
    repeat with item running through things:
        if the item is mentioned:
            now the item is not marked for listing;
    begin the listing nondescript items activity with the domain;
    if the number of marked for listing things is 0:
        abandon the listing nondescript items activity with the domain;
    otherwise:
        if handling the listing nondescript items activity:
            if the domain is a room:
                if the domain is the location, say "You ";
                otherwise say "In [the domain] you ";
            otherwise if the domain is a supporter:
                say "On [the domain] you ";
            otherwise if the domain is an animal:
                say "On [the domain] you ";
            otherwise:
                say "In [the domain] you ";
            say "can [if the locale paragraph count is greater than 0]also [end if]see ";
            let the common holder be nothing;
            let contents form of list be true;
            repeat with list item running through marked for listing things:
                if the holder of the list item is not the common holder:
                    if the common holder is nothing,
                        now the common holder is the holder of the list item;
                    otherwise now contents form of list is false;
                if the list item is mentioned, now the list item is not marked for listing;
            filter list recursion to unmentioned things;
            if contents form of list is true and the common holder is not nothing,
                list the contents of the common holder, as a sentence, including contents,
                    giving brief inventory information, tersely, not listing
                    concealed items, listing marked items only;
            otherwise say "[a list of marked for listing things including contents]";
            if the domain is the location, say " here";
            say ".[paragraph break]";
            unfilter list recursion;
        end the listing nondescript items activity with the domain;
continue the activity.
```

§24. Choosing Notable Locale Objects. By default, the notable objects are exactly the children of the domain, and they all have equal priority (1). Since table sorting is stable, and thus preserves the row order of rows with equal priority, the eventual order of listing is by default the same as the order in which things are added to the table, which in turn is the object-tree traversal order.

```
Choosing notable locale objects of something (documented at act_cnlo) is an activity.

For choosing notable locale objects (this is the standard notable locale objects rule):
    let the domain be the parameter-object;
    let the held item be the first thing held by the domain;
    while the held item is a thing:
        set the locale priority of the held item to 5;
        now the held item is the next thing held after the held item;
    continue the activity.
```

§25. Printing a Locale Paragraph. By default there are four kinds of "interesting" locale paragraph, and the following setup is fairly complicated because it implements conventions gradually built up between 1978 and 2008.

To recap, this activity is run on each notable thing in turn, in priority order. (It is only run on notable things for efficiency reasons.)

The basic principle is that, at every stage, we should consider an item only if it is not "mentioned" already. This will happen if it has been named by a previous paragraph, but also if it has been explicitly marked as such to get rid of it. In considering an item, we have three basic options:

(a) Print a paragraph about the item and mark it as mentioned – this is good for interesting items deserving a paragraph of their own.
(b) Print a paragraph, but do not mark it as mentioned – this is only likely to be useful if we want to print information related to the item without mentioning the thing itself. (For instance, if the presence of a mysterious parcel resulted in a ticking noise, we could print a paragraph about the ticking noise without mentioning the parcel, which would then appear later.)
(c) Mark the item as mentioned but print nothing – this gets rid of the item, ensuring that it will not appear in the final "you can also see" sentence, and will not be considered by subsequent rules.
(d) Do nothing at all – the item then becomes "nondescript" and appears in the final "you can also see" sentence, unless somebody else mentions it in the mean time.

Briefly, then, the following is the standard method:

(1) The "don't mention player's supporter in room descriptions rule" excludes anything the player is directly or indirectly standing on or, less frequently, in. The header of the room description has probably already said something like "Boudoir (on the four-poster bed)", so the player can't be unaware of this item.
(2) The "don't mention scenery in room descriptions rule" excludes scenery.
(3) The "don't mention undescribed items in room descriptions rule" excludes the player object. (It's redundant to say "You can also see yourself here.") At present nothing else in I7 is "undescribed" in this sense.
(4) The "set pronouns from items in room descriptions rule" adjusts the meaning of pronouns like IT and HER to pick up items mentioned. Thus if a room description ends "Mme Tourmalet glares at you.", then HER would be adjusted to mean Mme Tourmalet.
(5) The "offer items to writing a paragraph about rule" gives the "printing a paragraph about" activity a chance to intervene. We detect whether it does intervene or not by looking to see if it has printed any text.
(6) The "use initial appearance in room descriptions rule" uses the initial appearance property of an object which has never been handled as a paragraph.
(7) The "describe what's on scenery supporters in room descriptions rule" is a somewhat controversial feature: whereas the rest of Inform's room description conventions are generally consensus, this one is much disliked by some users for its occasional inappropriateness. It prints text such as "On the

mantelpiece is a piece of chalk." for items which, like the mantelpiece, are scenery mentioned (we assume) in the main room description. (It is assumed that scenery supporters make their contents more prominently visible than scenery containers, which we do not announce the contents of.) The ability to modify, replace or abolish this rule was one of the main motivations to break room description up into activities in March 2008.

```
Printing a locale paragraph about something (documented at act_plp) is an activity.

For printing a locale paragraph about a thing (called the item)
    (this is the don't mention player's supporter in room descriptions rule):
    if the item encloses the player, set the locale priority of the item to 0;
    continue the activity.

For printing a locale paragraph about a thing (called the item)
    (this is the don't mention scenery in room descriptions rule):
    if the item is scenery, set the locale priority of the item to 0;
    continue the activity.

For printing a locale paragraph about a thing (called the item)
    (this is the don't mention undescribed items in room descriptions rule):
    if the item is undescribed:
        set the locale priority of the item to 0;
    continue the activity.

For printing a locale paragraph about a thing (called the item)
    (this is the set pronouns from items in room descriptions rule):
    if the item is not mentioned, set pronouns from the item;
    continue the activity.

For printing a locale paragraph about a thing (called the item)
    (this is the offer items to writing a paragraph about rule):
    if the item is not mentioned:
        if a paragraph break is pending, say "[conditional paragraph break]";
        carry out the writing a paragraph about activity with the item;
        if a paragraph break is pending:
            increase the locale paragraph count by 1;
            now the item is mentioned;
            say "[command clarification break]";
    continue the activity.

For printing a locale paragraph about a thing (called the item)
    (this is the use initial appearance in room descriptions rule):
    if the item is not mentioned:
        if the item provides the property initial appearance and the
            item is not handled and the initial appearance of the item is
            not "":
            increase the locale paragraph count by 1;
            say "[initial appearance of the item]";
            say "[paragraph break]";
            if a locale-supportable thing is on the item:
                repeat with possibility running through things on the item:
                    now the possibility is marked for listing;
                    if the possibility is mentioned:
                        now the possibility is not marked for listing;
                say "On [the item] ";
                list the contents of the item, as a sentence, including contents,
                    giving brief inventory information, tersely, not listing
                    concealed items, prefacing with is/are, listing marked items only;
                say ".[paragraph break]";
            now the item is mentioned;
    continue the activity.
Definition: a thing (called the item) is locale-supportable if the item is not
scenery and the item is not mentioned and the item is not undescribed.
For printing a locale paragraph about a thing (called the item)
    (this is the describe what's on scenery supporters in room descriptions rule):
    if the item is [not undescribed and the item is] scenery and
        the item does not enclose the player:
        if a locale-supportable thing is on the item:
            set pronouns from the item;
            repeat with possibility running through things on the item:
                now the possibility is marked for listing;
                if the possibility is mentioned:
                    now the possibility is not marked for listing;
            increase the locale paragraph count by 1;
            say "On [the item] ";
            list the contents of the item, as a sentence, including contents,
                giving brief inventory information, tersely, not listing
                concealed items, prefacing with is/are, listing marked items only;
            say ".[paragraph break]";
    continue the activity.
```

SR4 - Actions A/sr4

Purpose

The standard stock of actions supplied with Inform, along with the rules which define them; and the Understand grammar which corresponds to them.

§**1.** NI comes with no actions built in, and only one (perhaps unexpected) assumption: that the 8th action defined is "going".

The order, and the subheadings, here are responsible for the order and subheadings used in the Actions page of the Index.

```
Part SR4 - Actions

Section SR4/1 - Generic action patterns

Section SR4/2 - Standard actions concerning the actor's possessions
```

§2. Taking inventory.

```
Taking inventory is an action applying to nothing.
The taking inventory action translates into I6 as "Inv".
```

```
The specification of the taking inventory action is "Taking an inventory of
one's immediate possessions: the things being carried, either directly or in
any containers being carried. When the player performs this action, either
the inventory listing, or else a special message if nothing is being carried
or worn, is printed during the carry out rules: nothing happens at the report
stage. The opposite happens for other people performing the action: nothing
happens during carry out, but a report such as 'Mr X looks through his
possessions.' is produced (provided Mr X is visible)."
```

§3. There used to be a rule, documented here, to do with pronouns, and this was explained in terms of Missee Lee, a black and white cat living in North Oxford; named for a Cambridge-educated pirate queen in the South China seas who is the heroine – or villainess – of the tenth in Arthur Ransome's Swallows and Amazons series of children's books, *Missee Lee* (1941). The rule was then removed, but it seemed sad to delete the only mention of Missee, and all the more so since she died (at a grand old age and in mid-spring) in 2008.

§4. Carry out.

```
Carry out taking inventory (this is the print empty inventory rule):
    if the first thing held by the player is nothing, stop the action with
        library message taking inventory action number 1.
```

```
Carry out taking inventory (this is the print standard inventory rule):
    issue library message taking inventory action number 2;
    say ":[line break]";
    list the contents of the player, with newlines, indented, including contents,
        giving inventory information, with extra indentation.
```

§5. Report.

```
Report an actor taking inventory (this is the report other people taking
    inventory rule):
    if the actor is not the player,
        issue actor-based library message taking inventory action number 5 for the actor.
```

§6. Taking.

```
Taking is an action applying to one thing.
The taking action translates into I6 as "Take".
```

```
The specification of the taking action is "The taking action is the only way
an action in the Standard Rules can cause something to be carried by an actor.
It is very simple in operation (the entire carry out stage consists only of
'now the actor carries the noun') but many checks must be performed before it
can be allowed to happen."
```

§7. Check.

```
Check an actor taking (this is the can't take yourself rule):
    if the actor is the noun, stop the action with library message taking
        action number 2 for the noun.
Check an actor taking (this is the can't take other people rule):
    if the noun is a person, stop the action with library message taking
        action number 3 for the noun.
Check an actor taking (this is the can't take component parts rule):
    if the noun is part of something (called the whole), stop the action
        with library message taking action number 7 for the whole.
Check an actor taking (this is the can't take people's possessions rule):
    let the local ceiling be the common ancestor of the actor with the noun;
    let H be the not-counting-parts holder of the noun;
    while H is not nothing and H is not the local ceiling:
        if H is a person, stop the action with library message taking action
            number 6 for H;
        let H be the not-counting-parts holder of H;
Check an actor taking (this is the can't take items out of play rule):
    let H be the noun;
    while H is not nothing and H is not a room:
        let H be the not-counting-parts holder of H;
    if H is nothing, stop the action with library message taking action
        number 8 for the noun.
Check an actor taking (this is the can't take what you're inside rule):
    let the local ceiling be the common ancestor of the actor with the noun;
    if the local ceiling is the noun, stop the action with library message
        taking action number 4 for the noun.
Check an actor taking (this is the can't take what's already taken rule):
    if the actor is carrying the noun, stop the action with library message
        taking action number 5 for the noun;
    if the actor is wearing the noun, stop the action with library message
        taking action number 5 for the noun.
Check an actor taking (this is the can't take scenery rule):
    if the noun is scenery, stop the action with library message taking
        action number 10 for the noun.
Check an actor taking (this is the can only take things rule):
    if the noun is not a thing, stop the action with library message taking
        action number 15 for the noun.
Check an actor taking (this is the can't take what's fixed in place rule):
    if the noun is fixed in place, stop the action with library message taking
        action number 11 for the noun.
Check an actor taking (this is the use player's holdall to avoid exceeding
    carrying capacity rule):
    if the number of things carried by the actor is at least the
        carrying capacity of the actor:
        if the actor is holding a player's holdall (called the current working sack):
            let the transferred item be nothing;
            repeat with the possible item running through things carried by
                the actor:
                if the possible item is not lit and the possible item is not
                    the current working sack, let the transferred item be the possible item;
            if the transferred item is not nothing:
                issue library message taking action number 13 for the
                    transferred item and the current working sack;
                silently try the actor trying inserting the transferred item
                    into the current working sack;
                if the transferred item is not in the current working sack, stop the action;
Check an actor taking (this is the can't exceed carrying capacity rule):
    if the number of things carried by the actor is at least the
        carrying capacity of the actor, stop the action with library
        message taking action number 12 for the actor.
```

§**8.** Carry out.

```
Carry out an actor taking (this is the standard taking rule):
    now the actor carries the noun.
```

§**9.** Report.

```
Report an actor taking (this is the standard report taking rule):
    if the actor is the player, issue library message taking action number 1
        for the noun;
    otherwise issue actor-based library message taking action number 16 for the noun.
```

§10. Removing it from.

```
Removing it from is an action applying to two things.
The removing it from action translates into I6 as "Remove".
The specification of the removing it from action is "Removing is not really
an action in its own right. Whereas there are many ways to put something down
(on the floor, on top of something, inside something else, giving it to
somebody else, and so on), Inform has only one way to take something: the
taking action. Removing exists only to provide some nicely worded replies
to impossible requests, and in all sensible cases is converted into taking.
Because of this, it's usually a bad idea to write rules about removing:
if you write a rule such as 'Instead of removing the key, ...' then it
won't apply if the player simply types TAKE KEY instead. The safe way to
do this is to write a rule about taking, which covers all possibilities."
```

§11. Check.

```
Check an actor removing something from (this is the can't remove what's not inside rule):
    if the holder of the noun is not the second noun, stop the action with
        library message removing it from action number 2 for the noun.
Check an actor removing something from (this is the can't remove from people rule):
    let the owner be the holder of the noun;
    if the owner is a person:
        if the owner is the actor, convert to the taking off action on the noun;
        stop the action with library message taking action number 6 for the owner.
Check an actor removing something from (this is the convert remove to take rule):
    convert to the taking action on the noun.
The can't take component parts rule is listed before the can't remove what's not
inside rule in the check removing it from rules.
```

§12. Dropping.

```
Dropping is an action applying to one thing.
The dropping action translates into I6 as "Drop".
```

```
The specification of the dropping action is "Dropping is one of five actions
by which an actor can get rid of something carried: the others are inserting
(into a container), putting (onto a supporter), giving (to someone else) and
eating. Dropping means dropping onto the actor's current floor, which is
usually the floor of a room - but might be the inside of a box if the actor
is also inside that box, and so on.
```

```
The can't drop clothes being worn rule silently tries the taking off action
on any clothing being dropped: unlisting this rule removes both this behaviour
and also the requirement that clothes cannot simply be dropped."
```

§13. Check.

```
Check an actor dropping (this is the can't drop yourself rule):
    if the noun is the actor, stop the action with library message putting
        it on action number 4.
```

```
Check an actor dropping (this is the can't drop what's already dropped rule):
    if the noun is in the holder of the actor, stop the action with library
        message dropping action number 1 for the noun.
```

```
Check an actor dropping (this is the can't drop what's not held rule):
    if the actor is carrying the noun, continue the action;
    if the actor is wearing the noun, continue the action;
    stop the action with library message dropping action number 2 for the noun.
```

```
Check an actor dropping (this is the can't drop clothes being worn rule):
    if the actor is wearing the noun:
        issue library message dropping action number 3 for the noun;
        silently try the actor trying taking off the noun;
        if the actor is wearing the noun, stop the action;
```

```
Check an actor dropping (this is the can't drop if this exceeds carrying
    capacity rule):
    let H be the holder of the actor;
    if H is a room, continue the action; [room floors have infinite capacity]
    if H provides the property carrying capacity:
        if H is a supporter:
            if the number of things on H is at least the carrying capacity of H:
                if the actor is the player,
                    issue library message dropping action number 5 for H;
                stop the action;
        otherwise if H is a container:
            if the number of things in H is at least the carrying capacity of H:
                if the actor is the player,
                    issue library message dropping action number 6 for H;
                stop the action;
```

§**14.** Carry out.

```
Carry out an actor dropping (this is the standard dropping rule):
    now the noun is in the holder of the actor.
```

§**15.** Report.

```
Report an actor dropping (this is the standard report dropping rule):
    if the actor is the player, issue library message dropping action number 4
        for the noun;
    otherwise issue actor-based library message dropping action number 7 for the noun.
```

§16. Putting it on.

```
Putting it on is an action applying to two things.
The putting it on action translates into I6 as "PutOn".
```

```
The specification of the putting it on action is "By this action, an actor puts
something he is holding on top of a supporter: for instance, putting an apple
on a table."
```

§17. Check.

```
Check an actor putting something on (this is the convert put to drop where possible rule):
    if the second noun is down or the actor is on the second noun,
        convert to the dropping action on the noun.
Check an actor putting something on (this is the can't put what's not held rule):
    if the actor is carrying the noun, continue the action;
    if the actor is wearing the noun, continue the action;
    issue miscellaneous library message number 26 for the noun;
    silently try the actor taking the noun;
    if the actor is carrying the noun, continue the action;
    stop the action with library message putting it on action number 1 for the noun.
Check an actor putting something on (this is the can't put something on itself rule):
    let the noun-CPC be the component parts core of the noun;
    let the second-CPC be the component parts core of the second noun;
    let the transfer ceiling be the common ancestor of the noun-CPC with the second-CPC;
    if the transfer ceiling is the noun-CPC,
        stop the action with library message putting it on action number 2 for
            the noun.
Check an actor putting something on (this is the can't put onto what's not a supporter rule):
    if the second noun is not a supporter,
        stop the action with library message putting it on action number 3 for
            the second noun.
Check an actor putting something on (this is the can't put onto something being carried rule):
    if the actor encloses the second noun,
        stop the action with library message putting it on action number 4 for
            the second noun.
Check an actor putting something on (this is the can't put clothes being worn rule):
    if the actor is wearing the noun:
        issue library message putting it on action number 5 for the noun;
        silently try the actor trying taking off the noun;
        if the actor is wearing the noun, stop the action;
Check an actor putting something on (this is the can't put if this exceeds
    carrying capacity rule):
    if the second noun provides the property carrying capacity:
        if the number of things on the second noun is at least the carrying capacity
            of the second noun,
            stop the action with library message putting it on action number 6 for
            the second noun;
```

§**18.** Carry out.

```
Carry out an actor putting something on (this is the standard putting rule):
    now the noun is on the second noun.
```

§**19.** Report.

```
Report an actor putting something on (this is the concise report putting rule):
    if the actor is the player and the I6 parser is running multiple actions,
        stop the action with library message putting it on action number 7
        for the noun;
    otherwise continue the action.
Report an actor putting something on (this is the standard report putting rule):
    if the actor is the player, issue library message putting it on action
        number 8 for the noun;
    otherwise issue actor-based library message putting it on action
        number 9 for the noun.
```

§20. Inserting it into.

```
Inserting it into is an action applying to two things.
The inserting it into action translates into I6 as "Insert".
The specification of the inserting it into action is "By this action, an actor puts
something he is holding into a container: for instance, putting a coin into a
collection box."
```

§21. Check.

```
Check an actor inserting something into (this is the convert insert to drop where
    possible rule):
    if the second noun is down or the actor is in the second noun,
        convert to the dropping action on the noun.
Check an actor inserting something into (this is the can't insert what's not held rule):
    if the actor is carrying the noun, continue the action;
    if the actor is wearing the noun, continue the action;
    issue miscellaneous library message number 26 for the noun;
    silently try the actor taking the noun;
    if the actor is carrying the noun, continue the action;
    stop the action.
Check an actor inserting something into (this is the can't insert something into itself rule):
    let the noun-CPC be the component parts core of the noun;
    let the second-CPC be the component parts core of the second noun;
    let the transfer ceiling be the common ancestor of the noun-CPC with the second-CPC;
    if the transfer ceiling is the noun-CPC,
        stop the action with library message inserting it into action number 5 for
            the noun.
Check an actor inserting something into (this is the can't insert into closed containers rule):
    if the second noun is a closed container,
        stop the action with library message inserting it into action number 3 for
            the second noun.
Check an actor inserting something into (this is the can't insert into what's not a
    container rule):
    if the second noun is not a container,
        stop the action with library message inserting it into action number 2 for
            the second noun.
Check an actor inserting something into (this is the can't insert clothes being worn rule):
    if the actor is wearing the noun:
        issue library message inserting it into action number 6 for the noun;
        silently try the actor trying taking off the noun;
        if the actor is wearing the noun, stop the action;
Check an actor inserting something into (this is the can't insert if this exceeds
    carrying capacity rule):
    if the second noun provides the property carrying capacity:
        if the number of things in the second noun is at least the carrying capacity
        of the second noun,
            stop the action with library message inserting it into action number 7 for
                the second noun;
```

§**22.** Carry out.

```
Carry out an actor inserting something into (this is the standard inserting rule):
    now the noun is in the second noun.
```

§**23.** Report.

```
Report an actor inserting something into (this is the concise report inserting rule):
    if the actor is the player and the I6 parser is running multiple actions,
        stop the action with library message inserting it into action number 8
        for the noun;
    otherwise continue the action.
Report an actor inserting something into (this is the standard report inserting rule):
    if the actor is the player, issue library message inserting it into action
        number 9 for the noun;
    otherwise issue actor-based library message inserting it into action number 10 for the noun.
```

§24. Eating.

```
Eating is an action applying to one carried thing.
The eating action translates into I6 as "Eat".
```

```
The specification of the eating action is "Eating is the only one of the
built-in actions which can, in effect, destroy something: the carry out
rule removes what's being eaten from play, and nothing in the Standard
Rules can then get at it again.
```

```
Note that, uncontroversially, one can only eat things with the 'edible'
either/or property, and also that, more controversially, one can only
eat things currently being held. This means that a player standing next
to a bush with berries who types EAT BERRIES will force a '(first taking
the berries)' action."
```

§25. Check.

```
Check an actor eating (this is the can't eat unless edible rule):
    if the noun is not a thing or the noun is not edible,
        stop the action with library message eating action number 1 for the noun.
```

```
Check an actor eating (this is the can't eat clothing without removing it first rule):
    if the actor is wearing the noun:
        issue library message dropping action number 3 for the noun;
        try the actor trying taking off the noun;
        if the actor is wearing the noun, stop the action;
```

§26. Carry out.

```
Carry out an actor eating (this is the standard eating rule):
    remove the noun from play.
```

§27. Report.

```
Report an actor eating (this is the standard report eating rule):
    if the actor is the player, issue library message eating action number 2
        for the noun;
    otherwise issue actor-based library message eating action number 3 for the noun.
```

§28. Going.

```
Section SR4/3 - Standard actions which move the actor

Going is an action applying to one visible thing.
The going action translates into I6 as "Go".

The specification of the going action is "This is the action which allows people
to move from one room to another, using whatever map connections and doors are
to hand. The Standard Rules are written so that the noun can be either a
direction or a door in the location of the actor: while the player's commands
only lead to going actions with directions as nouns, going actions can also
happen as a result of entering actions, and then the noun can indeed be
a door."

The going action has a room called the room gone from (matched as "from").
The going action has an object called the room gone to (matched as "to").
The going action has an object called the door gone through (matched as "through").
The going action has an object called the vehicle gone by (matched as "by").
The going action has an object called the thing gone with (matched as "with").

Rule for setting action variables for going (this is the standard set going variables rule):
    now the thing gone with is the item-pushed-between-rooms;
    now the room gone from is the location of the actor;
    if the actor is in an enterable vehicle (called the carriage),
        now the vehicle gone by is the carriage;
    let the target be nothing;
    if the noun is a direction:
        let direction D be the noun;
        let the target be the room-or-door direction D from the room gone from;
    otherwise:
        if the noun is a door, let the target be the noun;
    if the target is a door:
        now the door gone through is the target;
        now the target is the other side of the target from the room gone from;
    now the room gone to is the target.
```

§29. Check.

```
Check an actor going when the actor is on a supporter (called the chaise)
    (this is the stand up before going rule):
    issue library message going action number 27 for the chaise;
    silently try the actor exiting.

Check an actor going (this is the can't travel in what's not a vehicle rule):
    let H be the holder of the actor;
    if H is the room gone from, continue the action;
    if H is the vehicle gone by, continue the action;
    stop the action with library message going action number 1 for H.

Check an actor going (this is the can't go through undescribed doors rule):
    if the door gone through is not nothing and the door gone through is undescribed,
        stop the action with library message going action number 2 for the room gone from.

Check an actor going (this is the can't go through closed doors rule):
    if the door gone through is not nothing and the door gone through is closed:
        issue library message going action number 28 for the door gone through;
        silently try the actor opening the door gone through;
        if the door gone through is open, continue the action;
        stop the action.
Check an actor going (this is the determine map connection rule):
    let the target be nothing;
    if the noun is a direction:
        let direction D be the noun;
        let the target be the room-or-door direction D from the room gone from;
    otherwise:
        if the noun is a door, let the target be the noun;
    if the target is a door:
        now the target is the other side of the target from the room gone from;
    now the room gone to is the target.
Check an actor going (this is the can’t go that way rule):
    if the room gone to is nothing:
        if the door gone through is nothing, stop the action with library
            message going action number 2 for the room gone from;
        stop the action with library message going action number 6 for the door gone through;
```

§**30.** Carry out.

```
Carry out an actor going (this is the move player and vehicle rule):
    if the vehicle gone by is nothing,
        surreptitiously move the actor to the room gone to during going;
    otherwise
        surreptitiously move the vehicle gone by to the room gone to during going.
Carry out an actor going (this is the move floating objects rule):
    if the actor is the player,
        update backdrop positions.
Carry out an actor going (this is the check light in new location rule):
    if the actor is the player,
        surreptitiously reckon darkness.
```

§**31.** Report.

```
Report an actor going (this is the describe room gone into rule):
    if the player is the actor:
        produce a room description with going spacing conventions;
    otherwise:
        if the noun is a direction:
            if the location is the room gone from:
                if the location is the room gone to:
                    continue the action;
                otherwise:
                    if the noun is up :
                        issue actor-based library message going action number 8;
                    otherwise if the noun is down:
                        issue actor-based library message going action number 9;
                    otherwise:
                        issue actor-based library message going action number 10 for the noun;
            otherwise:
                let the back way be the opposite of the noun;
                if the location is the room gone to:
                    let the room back the other way be the room back way from the
                        location;
                    let the room normally this way be the room noun from the
                        room gone from;
                    if the room back the other way is the room gone from or
                        the room back the other way is the room normally this way:
                        if the back way is up:
                            issue actor-based library message going action number 11;
                        otherwise if the back way is down:
                            issue actor-based library message going action number 12;
                        otherwise:
                            issue actor-based library message going action number 13
                                for the back way;
                    otherwise:
                        issue actor-based library message going action number 14;
                otherwise:
                    if the back way is up :
                        issue actor-based library message going action number 15
                            for the room gone to;
                    otherwise if the back way is down:
                        issue actor-based library message going action number 16
                            for the room gone to;
                    otherwise:
                        issue actor-based library message going action number 17
                            for the room gone to and the back way;
        otherwise if the location is the room gone from:
            issue actor-based library message going action number 18 for the noun;
        otherwise:
            issue actor-based library message going action number 19 for the noun;
        if the vehicle gone by is not nothing:
            say " ";
            if the vehicle gone by is a supporter, issue actor-based library message
                going action number 20 for the vehicle gone by;
            otherwise issue actor-based library message going action number 21
                for the vehicle gone by;
        if the thing gone with is not nothing:
            if the player is within the thing gone with:
                issue actor-based library message going action number 22 for the thing gone with;
            otherwise if the player is within the vehicle gone by:
                issue actor-based library message going action number 23 for the thing gone with;
            otherwise if the location is the room gone from:
                issue actor-based library message going action number 24 for the thing gone with;
            otherwise:
                issue actor-based library message going action number 25 for the thing gone with;
        if the player is within the vehicle gone by and the player is not
            within the thing gone with:
            issue actor-based library message going action number 26;
            say ".";
            try looking;
            continue the action;
        say ".";
```

§32. Entering.

```
Entering is an action applying to one thing.
The entering action translates into I6 as "Enter".
```

```
The specification of the entering action is "Whereas the going action allows
people to move from one location to another in the model world, the entering
action is for movement inside a location: for instance, climbing into a cage
or sitting on a couch. (Entering is not allowed to change location, so any
attempt to enter a door is converted into a going action.) What makes
entering trickier than it looks is that the player may try to enter an
object which is itself inside, or part of, something else, which might in
turn be... and so on. To preserve realism, the implicitly pass through other
barriers rule automatically generates entering and exiting actions needed
to pass between anything which might be in the way: for instance, in a
room with two open cages, an actor in cage A who tries entering cage B first
has to perform an exiting action."
```

```
Rule for supplying a missing noun while entering (this is the find what to enter
rule):
    if something enterable (called the box) is in the location,
        now the noun is the box;
    otherwise continue the activity.
```

```
The find what to enter rule is listed last in the for supplying a missing noun
rulebook.
```

§33. Check.

```
Check an actor entering (this is the convert enter door into go rule):
    if the noun is a door, convert to the going action on the noun.
```

```
Check an actor entering (this is the convert enter compass direction into go rule):
    if the noun is a direction, convert to the going action on the noun.
```

```
Check an actor entering (this is the can't enter what's already entered rule):
    let the local ceiling be the common ancestor of the actor with the noun;
    if the local ceiling is the noun, stop the action with library message
        entering action number 1 for the noun.
```

```
Check an actor entering (this is the can't enter what's not enterable rule):
    if the noun is not enterable, stop the action with library message
        entering action number 2 for the noun.
```

```
Check an actor entering (this is the can't enter closed containers rule):
    if the noun is a closed container, stop the action with library message
        entering action number 3 for the noun.
```

```
Check an actor entering (this is the can't enter something carried rule):
    let the local ceiling be the common ancestor of the actor with the noun;
    if the local ceiling is the actor, stop the action with library message
        entering action number 4 for the noun.
```

```
Check an actor entering (this is the implicitly pass through other barriers rule):
    if the holder of the actor is the holder of the noun, continue the action;
    let the local ceiling be the common ancestor of the actor with the noun;
    while the holder of the actor is not the local ceiling:
        let the target be the holder of the actor;
        issue library message entering action number 6 for the target;
        silently try the actor trying exiting;
        if the holder of the actor is the target, stop the action;
    if the holder of the actor is the noun, stop the action;
    if the holder of the actor is the holder of the noun, continue the action;
    let the target be the holder of the noun;
    if the noun is part of the target, let the target be the holder of the target;
    while the target is a thing:
        if the holder of the target is the local ceiling:
            issue library message entering action number 7 for the target;
            silently try the actor trying entering the target;
            if the holder of the actor is not the target, stop the action;
            convert to the entering action on the noun;
            continue the action;
        let the target be the holder of the target;
```

§**34.** Carry out.

```
Carry out an actor entering (this is the standard entering rule):
    surreptitiously move the actor to the noun.
```

§**35.** Report.

```
Report an actor entering (this is the standard report entering rule):
    if the actor is the player:
        issue library message entering action number 5 for the noun;
    otherwise if the noun is a container:
        issue actor-based library message entering action number 8 for the noun;
    otherwise:
        issue actor-based library message entering action number 9 for the noun;
    continue the action.
Report an actor entering (this is the describe contents entered into rule):
    if the actor is the player, describe locale for the noun.
```

§36. Exiting.

```
Exiting is an action applying to nothing.
The exiting action translates into I6 as "Exit".
The exiting action has an object called the container exited from (matched as "from").
The specification of the exiting action is "Whereas the going action allows
people to move from one location to another in the model world, and the
entering action is for movement deeper inside the objects in a location,
the exiting action is for movement back out towards the main floor area.
Climbing out of a cupboard, for instance, is an exiting action. Exiting
when already in the main floor area of a room with a map connection to
the outside is converted to a going action. Finally, note that whereas
entering works for either containers or supporters, exiting is purely for
getting oneself out of containers: if the actor is on top of a supporter
instead, an exiting action is converted to the getting off action."
Setting action variables for exiting:
    now the container exited from is the holder of the actor.
```

§37. Check.

```
Check an actor exiting (this is the convert exit into go out rule):
    let the local room be the location of the actor;
    if the container exited from is the local room:
        if the room-or-door outside from the local room is not nothing,
            convert to the going action on the outside;
Check an actor exiting (this is the can't exit when not inside anything rule):
    let the local room be the location of the actor;
    if the container exited from is the local room, stop the action with
        library message exiting action number 1 for the actor.
Check an actor exiting (this is the can't exit closed containers rule):
    if the actor is in a closed container (called the cage), stop the action
        with library message exiting action number 2 for the cage.
Check an actor exiting (this is the convert exit into get off rule):
    if the actor is on a supporter (called the platform),
        convert to the getting off action on the platform.
```

§38. Carry out.

```
Carry out an actor exiting (this is the standard exiting rule):
    let the former exterior be the not-counting-parts holder of the container exited from;
    surreptitiously move the actor to the former exterior.
```

§39. Report.

```
Report an actor exiting (this is the standard report exiting rule):
    if the actor is the player:
        issue library message exiting action number 3 for the container exited from;
    otherwise:
        issue actor-based library message exiting action number 6 for the container exited from;
    continue the action.
Report an actor exiting (this is the describe room emerged into rule):
    if the actor is the player,
        produce a room description with going spacing conventions.
```

§40. Getting off.

```
Getting off is an action applying to one thing.
The getting off action translates into I6 as "GetOff".
```

```
The specification of the getting off action is "The getting off action is for
actors who are currently on top of a supporter: perhaps standing on a platform,
but maybe only sitting on a chair or even lying down in bed. Unlike the similar
exiting action, getting off takes a noun: the platform, chair, bed or what
have you."
```

§41. Check.

```
Check an actor getting off (this is the can’t get off things rule):
    if the actor is on the noun, continue the action;
    if the actor is carried by the noun, continue the action;
    stop the action with library message getting off action number 1 for the noun.
```

§42. Carry out.

```
Carry out an actor getting off (this is the standard getting off rule):
    let the former exterior be the not-counting-parts holder of the noun;
    surreptitiously move the actor to the former exterior.
```

§43. Report.

```
Report an actor getting off (this is the standard report getting off rule):
    if the actor is the player:
        issue library message exiting action number 3 for the noun;
    otherwise:
        issue actor-based library message exiting action number 5 for the noun;
    continue the action.
```

```
Report an actor getting off (this is the describe room stood up into rule):
    if the actor is the player,
        produce a room description with going spacing conventions.
```

§44. Looking.

```
Section SR4/4 - Standard actions concerning the actor's vision

Looking is an action applying to nothing.
The looking action translates into I6 as "Look".

The specification of the looking action is "The looking action describes the
player's current room and any visible items, but is made more complicated
by the problem of visibility. Inform calculates this by dividing the room
into visibility levels. For an actor on the floor of a room, there is only
one such level: the room itself. But an actor sitting on a chair inside
a packing case which is itself on a gantry would have four visibility levels:
chair, case, gantry, room. The looking rules use a special phrase, 'the
visibility-holder of X', to go up from one level to the next: thus the
visibility-holder of the case is the gantry.

The 'visibility level count' is the number of levels which the player can
actually see, and the 'visibility ceiling' is the uppermost visible level.
For a player standing on the floor of a lighted room, this will be a count
of 1 with the ceiling set to the room. But a player sitting on a chair in
a closed opaque packing case would have visibility level count 2, and
visibility ceiling equal to the case. Moreover, light has to be available
in order to see anything at all: if the player is in darkness, the level
count is 0 and the ceiling is nothing.

Finally, note that several actions other than looking also produce room
descriptions in some cases. The most familiar is going, but exiting a
container or getting off a supporter will also generate a room description.
(The phrase used by the relevant rules is 'produce a room description with
going spacing conventions' and carry out or report rules for newly written
actions are welcome to use this too if they would like to. The spacing
conventions affect paragraph division, and note that the main description
paragraph may be omitted for a place not newly visited, depending on the
VERBOSE settings.) Room descriptions like this are produced by running the
check, carry out and report rules for looking, but are not subject to
before, instead or after rules: so they do not count as a new action. The
looking variable 'room-describing action' holds the action name of the
reason a room description is currently being made: if the player typed
LOOK, this will indeed be set to the looking action, but if we're
describing a room just reached by GO EAST, say, it will be set to the going
action. This can be used to customise carry out looking rules so that
different forms of description are used on going to a room as compared with
looking around while already there."

The looking action has an action name called the room-describing action.
The looking action has a truth state called abbreviated form allowed.
The looking action has a number called the visibility level count.
The looking action has an object called the visibility ceiling.

Setting action variables for looking (this is the determine visibility ceiling
    rule):
    if the actor is the player, calculate visibility ceiling at low level;
    now the visibility level count is the visibility ceiling count calculated;
    now the visibility ceiling is the visibility ceiling calculated;
    now the room-describing action is the looking action.
```

§45. Carry out.

```
Carry out looking (this is the room description heading rule):
    say bold type;
    if the visibility level count is 0:
        begin the printing the name of a dark room activity;
        if handling the printing the name of a dark room activity,
            issue miscellaneous library message number 71;
        end the printing the name of a dark room activity;
    otherwise if the visibility ceiling is the location:
        say "[visibility ceiling]";
    otherwise:
        say "[The visibility ceiling]";
    say roman type;
    let intermediate level be the visibility-holder of the actor;
    repeat with intermediate level count running from 2 to the visibility level count:
        issue library message looking action number 8 for the intermediate level;
        let the intermediate level be the visibility-holder of the intermediate level;
    say line break;
    say run paragraph on with special look spacing.
Carry out looking (this is the room description body text rule):
    if the visibility level count is 0:
        if set to abbreviated room descriptions, continue the action;
        if set to sometimes abbreviated room descriptions and
            abbreviated form allowed is true and
            darkness witnessed is true,
            continue the action;
        begin the printing the description of a dark room activity;
        if handling the printing the description of a dark room activity,
            issue miscellaneous library message number 17;
        end the printing the description of a dark room activity;
    otherwise if the visibility ceiling is the location:
        if set to abbreviated room descriptions, continue the action;
        if set to sometimes abbreviated room descriptions and abbreviated form
            allowed is true and the location is visited, continue the action;
        print the location's description;
Carry out looking (this is the room description paragraphs about objects rule):
    if the visibility level count is greater than 0:
        let the intermediate position be the actor;
        let the IP count be the visibility level count;
        while the IP count is greater than 0:
            now the intermediate position is marked for listing;
            let the intermediate position be the visibility-holder of the
                intermediate position;
            decrease the IP count by 1;
        let the top-down IP count be the visibility level count;
        while the top-down IP count is greater than 0:
            let the intermediate position be the actor;
            let the IP count be 0;
            while the IP count is less than the top-down IP count:
                let the intermediate position be the visibility-holder of the
                    intermediate position;
                increase the IP count by 1;
            [if we ever support I6-style inside descriptions, here's where]
            describe locale for the intermediate position;
            decrease the top-down IP count by 1;
    continue the action;
Carry out looking (this is the check new arrival rule):
    if in darkness:
        now the darkness witnessed is true;
    otherwise:
        if the location is a room, now the location is visited;
```

§**46.** Report.

```
Report an actor looking (this is the other people looking rule):
    if the actor is not the player,
        issue actor-based library message looking action number 9.
```

§47. Examining.

```
Examining is an action applying to one visible thing and requiring light.
The examining action translates into I6 as "Examine".
The specification of the examining action is "The act of looking closely at
something. Note that the noun could be either a direction or a thing, which
is why the Standard Rules include the 'examine directions rule' to deal with
directions: it simply says 'You see nothing unexpected in that direction.'
and stops the action. (If you would like to handle directions differently,
list another rule instead of this one in the carry out examining rules.)
Some things have no description property, or rather, have only a blank text
as one. It's possible that something interesting may be said anyway (see
the rules for directions, containers, supporters and devices), but if not,
we give up with a bland response. This is done by the examine undescribed
things rule."
The examining action has a truth state called examine text printed.
```

§48. Carry out.

```
Carry out examining (this is the standard examining rule):
    if the noun provides the property description and the description of the noun is not "":
        say "[the description of the noun][line break]";
        now examine text printed is true.
Carry out examining (this is the examine directions rule):
    if the noun is a direction:
        issue library message examining action number 5 for the noun;
        now examine text printed is true.
Carry out examining (this is the examine containers rule):
    if the noun is a container:
        if the noun is open or the noun is transparent:
            if something described which is not scenery is in the noun and something which
                is not the player is in the noun:
                issue library message searching action number 7 for the noun;
                now examine text printed is true;
            otherwise if examine text printed is false:
                if the player is in the noun:
                    make no decision;
                issue library message searching action number 6 for the noun;
                now examine text printed is true;
Carry out examining (this is the examine supporters rule):
    if the noun is a supporter:
        if something described which is not scenery is on the noun and something which is
            not the player is on the noun:
            issue library message looking action number 4 for the noun;
            now examine text printed is true.
Carry out examining (this is the examine devices rule):
    if the noun is a device:
        issue library message examining action number 3 for the noun;
        now examine text printed is true.
Carry out examining (this is the examine undescribed things rule):
    if examine text printed is false:
        issue library message examining action number 2 for the noun.
```

§49. Report.

```
Report an actor examining (this is the report other people examining rule):
    if the actor is not the player,
        issue actor-based library message examining action number 4 for the noun.
```

§50. Looking under.

```
Looking under is an action applying to one visible thing and requiring light.
The looking under action translates into I6 as "LookUnder".
The specification of the looking under action is "The standard Inform world
model does not have a concept of things being under other things, so this
action is only minimally provided by the Standard Rules, but it exists here
for traditional reasons (and because, after all, LOOK UNDER TABLE is the
sort of command which ought to be recognised even if it does nothing useful).
The action ordinarily either tells the player he finds nothing of interest,
or reports that somebody else has looked under something.
The usual way to make this action do something useful is to write a rule
like 'Instead of looking under the cabinet for the first time: now the
player has the silver key; say ...' and so on."
```

§51. Carry out.

```
Carry out an actor looking under (this is the standard looking under rule):
    stop the action with library message looking under action number 2 for
        the noun.
```

§52. Report.

```
Report an actor looking under (this is the report other people looking under rule):
    if the actor is not the player,
        issue actor-based library message looking under action number 3 for the noun.
```

§53. Searching.

```
Searching is an action applying to one thing and requiring light.
The searching action translates into I6 as "Search".
```

```
The specification of the searching action is "Searching looks at the contents
of an open or transparent container, or at the items on top of a supporter.
These are often mentioned in room descriptions already, and then the action
is unnecessary, but that wouldn't be true for something like a kitchen
cupboard which is scenery - mentioned in passing in a room description, but
not made a fuss of. Searching such a cupboard would then, by listing its
contents, give the player more information than the ordinary room description
shows.
```

```
The usual check rules restrict searching to containers and supporters: so
the Standard Rules do not allow the searching of people, for instance. But
it is easy to add instead rules ('Instead of searching Dr Watson: ...') or
even a new carry out rule ('Check searching someone (called the suspect): ...')
to extend the way searching normally works."
```

§54. Check.

```
Check an actor searching (this is the can't search unless container or supporter rule):
    if the noun is not a container and the noun is not a supporter,
        stop the action with library message searching action number 4 for
            the noun.
```

```
Check an actor searching (this is the can't search closed opaque containers rule):
    if the noun is a closed opaque container,
        stop the action with library message searching action number 5 for
            the noun.
```

§55. Report.

```
Report searching a container (this is the standard search containers rule):
    if the noun contains a described thing which is not scenery,
        issue library message searching action number 7 for the noun;
    otherwise
        issue library message searching action number 6 for the noun.
```

```
Report searching a supporter (this is the standard search supporters rule):
    if the noun supports a described thing which is not scenery,
        issue library message searching action number 3 for the noun;
    otherwise
        issue library message searching action number 2 for the noun.
```

```
Report an actor searching (this is the report other people searching rule):
    if the actor is not the player,
        issue actor-based library message searching action number 8 for the noun.
```

§56. Consulting it about.

```
Consulting it about is an action applying to one thing and one topic.
The consulting it about action translates into I6 as "Consult".
```

```
The specification of the consulting it about action is "Consulting is a very
flexible and potentially powerful action, but only because it leaves almost
all of the work to the author to deal with directly. The idea is for it to
respond to commands such as LOOK UP HENRY FITZROY IN HISTORY BOOK, where
the topic would be the snippet of command HENRY FITZROY and the thing would
be the book.
```

```
The Standard Rules simply parry such requests by saying that the player finds
nothing of interest. All interesting responses must be provided by the author,
using rules like 'Instead of consulting the history book about...'"
```

§57. Report.

```
Report an actor consulting something about (this is the block consulting rule):
    if the actor is the player,
        issue library message consulting it about action number 1 for the noun;
    otherwise
        issue actor-based library message consulting it about action number 2 for the noun.
```

§58. Locking it with.

```
Section SR4/5 - Standard actions which change the state of things
Locking it with is an action applying to one thing and one carried thing.
The locking it with action translates into I6 as "Lock".
The specification of the locking it with action is "Locking is the act of
using an object such as a key to ensure that something such as a door or
container cannot be opened unless first unlocked. (Only closed things can be
locked.)
Locking can be performed on any kind of thing which provides the either/or
properties lockable, locked, openable and open. The 'can't lock without a lock
rule' tests to see if the noun both provides the lockable property, and if
it is in fact lockable: it is then assumed that the other properties can
safely be checked. In the Standard Rules, the container and door kinds both
satisfy these requirements.
We can create a new kind on which opening, closing, locking and unlocking
will work thus: 'A briefcase is a kind of thing. A briefcase can be openable.
A briefcase can be open. A briefcase can be lockable. A briefcase can be
locked. A briefcase is usually openable, lockable, open and unlocked.'
Inform checks whether the key fits using the 'can't lock without the correct
key rule'. To satisfy this, the actor must be directly holding the second
noun, and it must be the current value of the 'matching key' property for
the noun. (This property is seldom referred to directly because it is
automatically set by assertions like 'The silver key unlocks the wicket
gate.')
The Standard Rules provide locking and unlocking actions at a fairly basic
level: they can be much enhanced using the extension Locksmith by Emily
Short, which is included with all distributions of Inform."
```

§59. Check.

```
Check an actor locking something with (this is the can't lock without a lock rule):
    if the noun provides the property lockable and the noun is lockable,
        continue the action;
    stop the action with library message locking it with action number 1 for the noun.
Check an actor locking something with (this is the can't lock what's already
    locked rule):
    if the noun is locked,
        stop the action with library message locking it with action number 2 for the noun.
Check an actor locking something with (this is the can't lock what's open rule):
    if the noun is open,
        stop the action with library message locking it with action number 3 for the noun.
Check an actor locking something with (this is the can't lock without the correct key rule):
    if the holder of the second noun is not the actor or
        the noun does not provide the property matching key or
        the matching key of the noun is not the second noun,
        stop the action with library message locking it with action number 4 for the second noun.
```

§**60.** Carry out.

```
Carry out an actor locking something with (this is the standard locking rule):
    now the noun is locked.
```

§**61.** Report.

```
Report an actor locking something with (this is the standard report locking rule):
    if the actor is the player:
        issue library message locking it with action number 5 for the noun;
    otherwise:
        if the actor is visible, issue actor-based library message locking it with
            action number 6 for the noun;
```

§62. Unlocking it with.

```
Unlocking it with is an action applying to one thing and one carried thing.
The unlocking it with action translates into I6 as "Unlock".
The specification of the unlocking it with action is "Unlocking undoes the
effect of locking, and renders the noun openable again provided that the
actor is carrying the right key (which must be the second noun).
Unlocking can be performed on any kind of thing which provides the either/or
properties lockable, locked, openable and open. The 'can't unlock without a lock
rule' tests to see if the noun both provides the lockable property, and if
it is in fact lockable: it is then assumed that the other properties can
safely be checked. In the Standard Rules, the container and door kinds both
satisfy these requirements.
We can create a new kind on which opening, closing, locking and unlocking
will work thus: 'A briefcase is a kind of thing. A briefcase can be openable.
A briefcase can be open. A briefcase can be lockable. A briefcase can be
locked. A briefcase is usually openable, lockable, open and unlocked.'
Inform checks whether the key fits using the 'can't unlock without the correct
key rule'. To satisfy this, the actor must be directly holding the second
noun, and it must be the current value of the 'matching key' property for
the noun. (This property is seldom referred to directly because it is
automatically set by assertions like 'The silver key unlocks the wicket
gate.')
The Standard Rules provide locking and unlocking actions at a fairly basic
level: they can be much enhanced using the extension Locksmith by Emily
Short, which is included with all distributions of Inform."
```

§63. Check.

```
Check an actor unlocking something with (this is the can't unlock without a lock rule):
    if the noun provides the property lockable and the noun is lockable,
        continue the action;
    stop the action with library message unlocking it with action number 1 for the noun.
Check an actor unlocking something with (this is the can't unlock what's already unlocked rule):
    if the noun is not locked,
        stop the action with library message unlocking it with action number 2 for the noun.
Check an actor unlocking something with (this is the can't unlock without the correct key rule):
    if the holder of the second noun is not the actor or
        the noun does not provide the property matching key or
        the matching key of the noun is not the second noun,
        stop the action with library message unlocking it with action number 3 for the
            second noun.
```

§64. Carry out.

```
Carry out an actor unlocking something with (this is the standard unlocking rule):
    now the noun is not locked.
```

§65. Report.

```
Report an actor unlocking something with (this is the standard report unlocking rule):
    if the actor is the player:
        issue library message unlocking it with action number 4 for the noun;
    otherwise:
        if the actor is visible, issue actor-based library message unlocking it with
            action number 5 for the noun;
```

§66. Switching on.

```
Switching on is an action applying to one thing.
The switching on action translates into I6 as "SwitchOn".
```

```
The specification of the switching on action is "The switching on and switching
off actions are for the simplest kind of machinery operation: they are for
objects representing machines (or more likely parts of machines), which are
considered to be either off or on at any given time.
```

```
The actions are intended to be used where the noun is a device, but in fact
they could work just as well with any kind which can be 'switched on' or
'switched off'."
```

§67. Check.

```
Check an actor switching on (this is the can't switch on unless switchable rule):
    if the noun provides the property switched on, continue the action;
    stop the action with library message switching on action number 1 for the noun.
```

```
Check an actor switching on (this is the can't switch on what's already on rule):
    if the noun is switched on,
        stop the action with library message switching on action number 2 for the noun.
```

§68. Carry out.

```
Carry out an actor switching on (this is the standard switching on rule):
    now the noun is switched on.
```

§69. Report.

```
Report an actor switching on (this is the standard report switching on rule):
    if the actor is the player, issue library message switching on action number 3
        for the noun;
    otherwise issue actor-based library message switching on action number 4 for the noun;
```

§70. Switching off.

```
Switching off is an action applying to one thing.
The switching off action translates into I6 as "SwitchOff".
```

```
The specification of the switching off action is "The switching off and switching
on actions are for the simplest kind of machinery operation: they are for
objects representing machines (or more likely parts of machines), which are
considered to be either off or on at any given time.
```

```
The actions are intended to be used where the noun is a device, but in fact
they could work just as well with any kind which can be 'switched on' or
'switched off'."
```

§71. Check.

```
Check an actor switching off (this is the can't switch off unless switchable rule):
    if the noun provides the property switched on, continue the action;
    stop the action with library message switching off action number 1 for the noun.
Check an actor switching off (this is the can't switch off what's already off rule):
    if the noun is switched off,
        stop the action with library message switching off action number 2 for the noun.
```

§72. Carry out.

```
Carry out an actor switching off (this is the standard switching off rule):
    now the noun is switched off.
```

§73. Report.

```
Report an actor switching off (this is the standard report switching off rule):
    if the actor is the player, issue library message switching off action number 3
        for the noun;
    otherwise issue actor-based library message switching off action number 4 for the noun;
```

§74. Opening.

```
Opening is an action applying to one thing.
The opening action translates into I6 as "Open".
```

```
The specification of the opening action is "Opening makes something no longer
a physical barrier. The action can be performed on any kind of thing which
provides the either/or properties openable and open. The 'can't open unless
openable rule' tests to see if the noun both can be and actually is openable.
(It is assumed that anything which can be openable can also be open.)
In the Standard Rules, the container and door kinds both satisfy these
requirements.

In the event that the thing to be opened is also lockable, we are forbidden
to open it when it is locked. Both containers and doors can be lockable,
but the opening and closing actions would also work fine with kinds which
cannot be.

We can create a new kind on which opening and closing will work thus:
'A case file is a kind of thing. A case file can be openable.
A case file can be open. A case file is usually openable and closed.'

The meaning of open and closed is different for different kinds of thing.
When a container is closed, that means people outside cannot reach in,
and vice versa; when a door is closed, people cannot use the 'going' action
to pass through it. If we were to create a new kind such as 'case file',
we would also need to write rules to make the open and closed properties
interesting for this kind."
```

§75. Check.

```
Check an actor opening (this is the can't open unless openable rule):
    if the noun provides the property openable and the noun is openable,
        continue the action;
    stop the action with library message opening action number 1 for the noun.

Check an actor opening (this is the can't open what's locked rule):
    if the noun provides the property lockable and the noun is locked,
        stop the action with library message opening action number 2 for the noun.

Check an actor opening (this is the can't open what's already open rule):
    if the noun is open,
        stop the action with library message opening action number 3 for the noun.
```

§76. Carry out.

```
Carry out an actor opening (this is the standard opening rule):
    now the noun is open.
```

§77. Report.

```
Report an actor opening (this is the reveal any newly visible interior rule):
    if the actor is the player and
        the noun is an opaque container and
        the first thing held by the noun is not nothing and
        the noun does not enclose the actor,
        stop the action with library message opening action number 4 for the noun.
Report an actor opening (this is the standard report opening rule):
    if the actor is the player:
        issue library message opening action number 5 for the noun;
    otherwise if the player can see the actor:
        issue actor-based library message opening action number 6 for the noun;
    otherwise:
        issue actor-based library message opening action number 7 for the noun;
```

§78. Closing.

```
Closing is an action applying to one thing.
The closing action translates into I6 as "Close".
The specification of the closing action is "Closing makes something into
a physical barrier. The action can be performed on any kind of thing which
provides the either/or properties openable and open. The 'can't close unless
openable rule' tests to see if the noun both can be and actually is openable.
(It is assumed that anything which can be openable can also be open, and
hence can also be closed.) In the Standard Rules, the container and door
kinds both satisfy these requirements.
We can create a new kind on which opening and closing will work thus:
'A case file is a kind of thing. A case file can be openable.
A case file can be open. A case file is usually openable and closed.'
The meaning of open and closed is different for different kinds of thing.
When a container is closed, that means people outside cannot reach in,
and vice versa; when a door is closed, people cannot use the 'going' action
to pass through it. If we were to create a new kind such as 'case file',
we would also need to write rules to make the open and closed properties
interesting for this kind."
```

§79. Check.

```
Check an actor closing (this is the can't close unless openable rule):
    if the noun provides the property openable and the noun is openable,
        continue the action;
    stop the action with library message closing action number 1 for the noun.
Check an actor closing (this is the can't close what's already closed rule):
    if the noun is closed,
        stop the action with library message closing action number 2 for the noun.
```

§80. Carry out.

```
Carry out an actor closing (this is the standard closing rule):
    now the noun is closed.
```

§81. Report.

```
Report an actor closing (this is the standard report closing rule):
    if the actor is the player:
        issue library message closing action number 3 for the noun;
    otherwise if the player can see the actor:
        issue actor-based library message closing action number 4 for the noun;
    otherwise:
        issue actor-based library message closing action number 5 for the noun;
```

§82. Wearing.

```
Wearing is an action applying to one carried thing.
The wearing action translates into I6 as "Wear".
The specification of the wearing action is "The Standard Rules give Inform
only a simple model of clothing. A thing can be worn only if it has the
either/or property of being 'wearable'. (Typing a sentence like 'Mr Jones
wears the Homburg hat.' automatically implies that the hat is wearable,
which is why we only seldom need to use the word 'wearable' directly.)
There is no checking of how much or how little any actor is wearing, or
how incongruous this may appear: nor any distinction between under or
over-clothes.
To put on an article of clothing, the actor must be directly carrying it,
as enforced by the 'can't wear what's not held rule'."
```

§83. Check.

```
Check an actor wearing (this is the can't wear what's not clothing rule):
    if the noun is not a thing or the noun is not wearable,
        stop the action with library message wearing action number 1 for the noun.
Check an actor wearing (this is the can't wear what's not held rule):
    if the holder of the noun is not the actor,
        stop the action with library message wearing action number 2 for the noun.
Check an actor wearing (this is the can't wear what's already worn rule):
    if the actor is wearing the noun,
        stop the action with library message wearing action number 3 for the noun.
```

§84. Carry out.

```
Carry out an actor wearing (this is the standard wearing rule):
    now the actor wears the noun.
```

§85. Report.

```
Report an actor wearing (this is the standard report wearing rule):
    if the actor is the player, issue library message wearing action number 4
        for the noun;
    otherwise issue actor-based library message wearing action number 5
        for the noun.
```

§86. Taking off.

```
Taking off is an action applying to one carried thing.
The taking off action translates into I6 as "Disrobe".
```

```
The specification of the taking off action is "The Standard Rules give Inform
only a simple model of clothing. A thing can be worn only if it has the
either/or property of being 'wearable'. (Typing a sentence like 'Mr Jones
wears the Homburg hat.' automatically implies that the hat is wearable,
which is why we only seldom need to use the word 'wearable' directly.)
There is no checking of how much or how little any actor is wearing, or
how incongruous this may appear: nor any distinction between under or
over-clothes.
```

```
When an article of clothing is taken off, it becomes a thing directly
carried by its former wearer, rather than being (say) dropped onto the floor."
```

§87. Check.

```
Check an actor taking off (this is the can't take off what's not worn rule):
    if the actor is not wearing the noun,
        stop the action with library message taking off action number 1 for the noun.
```

§88. Carry out.

```
Carry out an actor taking off (this is the standard taking off rule):
    now the actor carries the noun.
```

§89. Report.

```
Report an actor taking off (this is the standard report taking off rule):
    if the actor is the player, issue library message taking off action number 2
        for the noun;
    otherwise issue actor-based library message taking off action number 3 for the noun.
```

§90. Giving it to.

```
Section SR4/6 - Standard actions concerning other people

Giving it to is an action applying to one carried thing and one thing.
The giving it to action translates into I6 as "Give".

The specification of the giving it to action is "This action is indexed by
Inform under 'Actions concerning other people', but it could just as easily
have gone under 'Actions concerning the actor's possessions' because -
like dropping, putting it on or inserting it into - this is an action
which gets rid of something being carried.

The Standard Rules implement this action fully - if it reaches the carry
out and report rulebooks, then the item is indeed transferred to the
recipient, and this is properly reported. But giving something to
somebody is not like putting something on a shelf: the recipient has
to agree. The final check rule, the 'block giving rule', assumes that
the recipient does not consent - so the gift fails to happen. The way
to make the giving action use its abilities fully is to replace the
block giving rule with a rule which makes a more sophisticated decision
about who will accept what from whom, and only blocks some attempts,
letting others run on into the carry out and report rules."
```

§91. Check.

```
Check an actor giving something to (this is the can't give what you haven't got rule):
    if the actor is not the holder of the noun,
        stop the action with library message giving it to action number 1 for the noun.
Check an actor giving something to (this is the can't give to yourself rule):
    if the actor is the second noun,
        stop the action with library message giving it to action number 2 for the noun.
Check an actor giving something to (this is the can't give to a non-person rule):
    if the second noun is not a person,
        stop the action with library message giving it to action number 4 for the
            second noun.
Check an actor giving something to (this is the can't give clothes being worn rule):
    if the actor is wearing the noun:
        issue library message dropping action number 3 for the noun;
        silently try the actor trying taking off the noun;
        if the actor is wearing the noun, stop the action;
Check an actor giving something to (this is the block giving rule):
    stop the action with library message giving it to action number 3 for the
        second noun.
```

§92. Carry out.

```
Carry out an actor giving something to (this is the standard giving rule):
    move the noun to the second noun.
```

§**93.** Report.

```
Report an actor giving something to (this is the standard report giving rule):
    if the actor is the player:
        issue library message giving it to action number 5 for the noun;
    otherwise if the second noun is the player:
        issue actor-based library message giving it to action number 6 for the noun;
    otherwise:
        issue actor-based library message giving it to action number 7 for the noun;
```

§94. Showing it to.

```
Showing it to is an action applying to one carried thing and one visible thing.
The showing it to action translates into I6 as "Show".
```

```
The specification of the showing it to action is "Anyone can show anyone
else something which they are carrying, but not some nearby piece of
scenery, say - so this action is suitable for showing the emerald locket
to Katarina, but not showing the Orange River Rock Room to Mr Douglas.
```

```
The Standard Rules implement this action in only a minimal way, checking
that it makes sense but then blocking all such attempts with a message
such as 'Katarina is not interested.' - this is the task of the 'block
showing rule'. As a result, there are no carry out or report rules. To
make it into a systematic and interesting action, we would need to
unlist the block showing rule and then to write carry out and report
rules: but usually for IF purposes we only need to make a handful of
special cases of showing work properly, and for those we can simply
write Instead rules to handle them."
```

§95. Check.

```
Check an actor showing something to (this is the can't show what you haven't
    got rule):
    if the actor is not the holder of the noun,
        stop the action with library message showing it to action number 1
            for the noun.
```

```
Check an actor showing something to (this is the convert show to yourself to
    examine rule):
    if the actor is the second noun,
        convert to the examining action on the noun.
```

```
Check an actor showing something to (this is the block showing rule):
    stop the action with library message showing it to action number 2
        for the second noun.
```

§96. Waking.

```
Waking is an action applying to one thing.
The waking action translates into I6 as "WakeOther".
The specification of the waking action is "This is the act of jostling
a sleeping person to wake him or her up, and it finds its way into the
Standard Rules only for historical reasons. Inform does not by default
provide any model for people being asleep or awake, so this action does
not do anything in the standard implementation: instead, it is always
stopped by the block waking rule."
```

§97. Check.

```
Check an actor waking (this is the block waking rule):
    stop the action with library message waking action number 1 for the noun.
```

§98. Throwing it at.

```
Throwing it at is an action applying to one carried thing and one visible thing.
The throwing it at action translates into I6 as "ThrowAt".
```

```
The specification of the throwing it at action is "Throwing something at
someone or something is difficult for Inform to model. So many considerations
apply: just because the actor can see the target, does it follow that the
target can accurately hit it? What if the projectile is heavy, like an
anvil, or something not easily aimable, like a feather? What if there
is a barrier in the way, like a cage with bars spaced so that only items
of a certain size get through? And then: what should happen as a result?
Will the projectile break, or do damage, or fall to the floor, or into
a container or onto a supporter? And so on.
```

```
Because it seems hopeless to try to model this in any general way,
Inform instead provides the action for the user to attach specific rules to.
The check rules in the Standard Rules simply require that the projectile
is not an item of clothing still worn (this will be relevant for women
attending a Tom Jones concert) but then, in either the 'futile to throw
things at inanimate objects rule' or the 'block throwing at rule', will
refuse to carry out the action with a bland message.
```

```
To make throwing do something, then, we must either write Instead rules
for special circumstances, or else unlist these check rules and write
suitable carry out and report rules to pick up the thread."
```

§99. Check.

```
Check an actor throwing something at (this is the implicitly remove thrown clothing rule):
    if the actor is wearing the noun:
        issue library message dropping action number 3 for the noun;
        silently try the actor trying taking off the noun;
        if the actor is wearing the noun, stop the action;
```

```
Check an actor throwing something at (this is the futile to throw things at inanimate
    objects rule):
    if the second noun is not a person,
        stop the action with library message throwing it at action number 1
            for the second noun.
```

```
Check an actor throwing something at (this is the block throwing at rule):
    stop the action with library message throwing it at action number 2
        for the noun.
```

§100. Attacking.

```
Attacking is an action applying to one thing.
The attacking action translates into I6 as "Attack".
```

```
The specification of the attacking action is "Violence is seldom the answer,
and attempts to attack another person are normally blocked as being unrealistic
or not seriously meant. (I might find a shop assistant annoying, but IF is
not Grand Theft Auto, and responding by killing him is not really one of
my options.) So the Standard Rules simply block attempts to fight people,
but the action exists for rules to make exceptions."
```

§101. Check.

```
Check an actor attacking (this is the block attacking rule):
    stop the action with library message attacking action number 1 for the noun.
```

§102. Kissing.

```
Kissing is an action applying to one thing.
The kissing action translates into I6 as "Kiss".
The specification of the kissing action is "Possibly because Inform was
originally written by an Englishman, attempts at kissing another person are
normally blocked as being unrealistic or not seriously meant. So the
Standard Rules simply block attempts to kiss people, but the action exists
for rules to make exceptions."
```

§103. Check.

```
Check an actor kissing (this is the kissing yourself rule):
    if the noun is the actor,
        stop the action with library message touching action number 3 for the noun.
Check an actor kissing (this is the block kissing rule):
    stop the action with library message kissing action number 1 for the noun.
```

§104. Answering it that.

```
Answering it that is an action applying to one thing and one topic.
The answering it that action translates into I6 as "Answer".
```

```
The specification of the answering it that action is "The Standard Rules do
not include any systematic way to handle conversation: instead, Inform is
set up so that it is as easy as we can make it to write specific rules
handling speech in particular games, and so that if no such rules are
written then all attempts to communicate are gracefully if not very
interestingly rejected.
```

```
The topic here can be any double-quoted text, which can itself contain
tokens in square brackets: see the documentation on Understanding.
```

```
Answering is an action existing so that the player can say something free-form
to somebody else. A convention of IF is that a command such as DAPHNE, TAKE
MASK is a request to Daphne to perform an action: if the persuasion rules in
force mean that she consents, the action 'Daphne taking the mask' does
indeed then result. But if the player types DAPHNE, 12375 or DAPHNE, GREAT
HEAVENS - or anything else not making sense as a command - the action
'answering Daphne that ...' will be generated.
```

```
The name of the action arises because it is also caused by typing, say,
ANSWER 12375 when Daphne (say) has asked a question."
```

§105. Report.

```
Report an actor answering something that (this is the block answering rule):
    stop the action with library message answering it that action number 1
        for the noun.
```

§106. Telling it about.

```
Telling it about is an action applying to one thing and one topic.
The telling it about action translates into I6 as "Tell".
```

```
The specification of the telling it about action is "The Standard Rules do
not include any systematic way to handle conversation: instead, Inform is
set up so that it is as easy as we can make it to write specific rules
handling speech in particular games, and so that if no such rules are
written then all attempts to communicate are gracefully if not very
interestingly rejected.
```

```
The topic here can be any double-quoted text, which can itself contain
tokens in square brackets: see the documentation on Understanding.
```

```
Telling is an action existing only to catch commands like TELL ALEX ABOUT
GUITAR. Customarily in IF, such a command is shorthand which the player
accepts as a conventional form: it means 'tell Alex what I now know about
the guitar' and would make sense if the player had himself recently
discovered something significant about the guitar which might interest
Alex."
```

§107. Check.

```
Check an actor telling something about (this is the telling yourself rule):
    if the actor is the noun,
        stop the action with library message telling it about action number 1
            for the noun.
```

§108. Report.

```
Report an actor telling something about (this is the block telling rule):
    stop the action with library message telling it about action number 2
        for the noun.
```

§109. Asking it about.

```
Asking it about is an action applying to one thing and one topic.
The asking it about action translates into I6 as "Ask".
```

```
The specification of the asking it about action is "The Standard Rules do
not include any systematic way to handle conversation: instead, Inform is
set up so that it is as easy as we can make it to write specific rules
handling speech in particular games, and so that if no such rules are
written then all attempts to communicate are gracefully if not very
interestingly rejected.
```

```
The topic here can be any double-quoted text, which can itself contain
tokens in square brackets: see the documentation on Understanding.
```

```
Asking is an action existing only to catch commands like ASK STEPHEN ABOUT
PENELOPE. Customarily in IF, such a command is shorthand which the player
accepts as a conventional form: it means 'engage Mary in conversation and
try to find out what she might know about'. It's understood as a convention
of the genre that Mary should not be expected to respond in cases where
there is no reason to suppose that she has anything relevant to pass on -
ASK JANE ABOUT RICE PUDDING, for instance, need not conjure up a recipe
even if Jane is a 19th-century servant and therefore almost certainly
knows one."
```

§110. Report.

```
Report an actor asking something about (this is the block asking rule):
    stop the action with library message asking it about action number 1
        for the noun.
```

§111. Asking it for.

```
Asking it for is an action applying to two things.
The asking it for action translates into I6 as "AskFor".
```

```
The specification of the asking it for action is "The Standard Rules do
not include any systematic way to handle conversation, but this is
action is not quite conversation: it doesn't involve any spoken text as
such. It exists to catch commands like ASK SALLY FOR THE EGG WHISK,
where the whisk is something which Sally has and the player can see.

Slightly oddly, but for historical reasons, an actor asking himself for
something is treated to an inventory listing instead. All other cases
are converted to the giving action: that is, ASK SALLY FOR THE EGG WHISK
is treated as if it were SALLY, GIVE ME THE EGG WHISK - an action for
Sally to perform and which then follows rules for giving.

To ask for information or something intangible, see the asking it about
action."
```

§112. Check.

```
Check an actor asking something for (this is the asking yourself for something rule):
    if the actor is the noun and the actor is the player,
        try taking inventory instead.

Check an actor asking something for (this is the translate asking for to giving rule):
    convert to request of the noun to perform giving it to action with the
        second noun and the actor.
```

§113. Waiting.

```
Section SR4/7 - Standard actions which are checked but then do nothing unless rules intervene

Waiting is an action applying to nothing.
The waiting action translates into I6 as "Wait".

The specification of the waiting action is "The inaction action: where would
we be without waiting? Waiting does not cause time to pass by - that happens
anyway - but represents a positive choice by the actor not to fill that time.
It is an action so that rules can be attached to it: for instance, we could
imagine that a player who consciously decides to sit and wait might notice
something which a busy player does not, and we could write a rule accordingly.

Note the absence of check or carry out rules - anyone can wait, at any time,
and it makes nothing happen."
```

§114. Report.

```
Report an actor waiting (this is the standard report waiting rule):
    if the actor is the player, stop the action with library message waiting
        action number 1 for the actor;
    issue actor-based library message waiting action number 2.
```

§115. Touching.

```
Touching is an action applying to one thing.
The touching action translates into I6 as "Touch".
```

```
The specification of the touching action is "Touching is just that, touching
something without applying pressure: a touch-sensitive screen or a living
creature might react, but a standard push-button or lever will probably not.
```

```
In the Standard Rules there are no check touching rules, since touchability
is already a requirement of the noun for the action anyway, and no carry out
rules because nothing in the standard Inform world model reacts to
a mere touch - though report rules do mean that attempts to touch other
people provoke a special reply."
```

§116. Report.

```
Report an actor touching (this is the report touching yourself rule):
    if the noun is the actor:
        if the actor is the player, issue library message touching action number 3
            for the noun;
        otherwise issue actor-based library message touching action number 4;
        stop the action;
    continue the action.
```

```
Report an actor touching (this is the report touching other people rule):
    if the noun is a person:
        if the actor is the player:
            issue library message touching action number 1 for the noun;
        otherwise if the noun is the player:
            issue actor-based library message touching action number 5;
        otherwise:
            issue actor-based library message touching action number 6 for the noun;
        stop the action;
    continue the action.
```

```
Report an actor touching (this is the report touching things rule):
    if the actor is the player, issue library message touching action number 2
        for the noun;
    otherwise issue actor-based library message touching action number 6 for the noun.
```

§117. Waving.

```
Waving is an action applying to one thing.
The waving action translates into I6 as "Wave".
```

```
The specification of the waving action is "Waving in this sense is like
waving a sceptre: the item to be waved must be directly held (or worn)
by the actor.
```

```
In the Standard Rules there are no carry out rules for this action because
nothing in the standard Inform world model which reacts to it. The action
is provided for authors to hang more interesting behaviour onto for special
cases: say, waving a particular rusty iron rod with a star on the end."
```

§118. Check.

```
Check an actor waving (this is the can't wave what's not held rule):
    if the actor is not the holder of the noun,
        stop the action with library message waving action number 1 for the noun.
```

§119. Report.

```
Report an actor waving (this is the report waving things rule):
    if the actor is the player, issue library message waving action number 2
        for the noun;
    otherwise issue actor-based library message waving action number 3 for the noun.
```

§120. Pulling.

```
Pulling is an action applying to one thing.
The Pulling action translates into I6 as "Pull".
```

```
The specification of the pulling action is "Pulling is the act of pulling
something not grossly larger than the actor by an amount which would not
substantially move it.
```

```
In the Standard Rules there are no carry out rules for this action because
nothing in the standard Inform world model which reacts to it. The action
is provided for authors to hang more interesting behaviour onto for special
cases: say, pulling a lever. ('The big red lever is a fixed in place device.
Instead of pulling the big red lever, try switching on the lever. Instead
of pushing the big red lever, try switching off the lever.')"
```

§121. Check.

```
Check an actor pulling (this is the can't pull what's fixed in place rule):
    if the noun is fixed in place,
        stop the action with library message pulling action number 1 for the noun.
Check an actor pulling (this is the can't pull scenery rule):
    if the noun is scenery,
        stop the action with library message pulling action number 2 for the noun.
Check an actor pulling (this is the can't pull people rule):
    if the noun is a person,
        stop the action with library message pulling action number 4 for the noun.
```

§122. Report.

```
Report an actor pulling (this is the report pulling rule):
    if the actor is the player, issue library message pulling action number 3
        for the noun;
    otherwise issue actor-based library message pulling action number 5 for the noun.
```

§123. Pushing.

```
Pushing is an action applying to one thing.
The Pushing action translates into I6 as "Push".
```

```
The specification of the pushing action is "Pushing is the act of pushing
something not grossly larger than the actor by an amount which would not
substantially move it. (See also the pushing it to action, which involves
a longer-distance push between rooms.)
In the Standard Rules there are no carry out rules for this action because
nothing in the standard Inform world model which reacts to it. The action
is provided for authors to hang more interesting behaviour onto for special
cases: say, pulling a lever. ('The big red lever is a fixed in place device.
Instead of pulling the big red lever, try switching on the lever. Instead
of pushing the big red lever, try switching off the lever.')"
```

§124. Check.

```
Check an actor pushing something (this is the can't push what's fixed in place rule):
    if the noun is fixed in place,
        stop the action with library message pushing action number 1 for the noun.
Check an actor pushing something (this is the can't push scenery rule):
    if the noun is scenery,
        stop the action with library message pushing action number 2 for the noun.
Check an actor pushing something (this is the can't push people rule):
    if the noun is a person,
        stop the action with library message pushing action number 4 for the noun.
```

§125. Report.

```
Report an actor pushing something (this is the report pushing rule):
    if the actor is the player, issue library message pushing action number 3
        for the noun;
    otherwise issue actor-based library message pushing action number 6 for the noun.
```

§126. Turning.

```
Turning is an action applying to one thing.
The Turning action translates into I6 as "Turn".
The specification of the turning action is "Turning is the act of rotating
something - say, a dial.
In the Standard Rules there are no carry out rules for this action because
nothing in the standard Inform world model which reacts to it. The action
is provided for authors to hang more interesting behaviour onto for special
cases: say, turning a capstan."
```

§127. Check.

```
Check an actor turning (this is the can't turn what's fixed in place rule):
    if the noun is fixed in place,
        stop the action with library message turning action number 1 for the noun.
Check an actor turning (this is the can't turn scenery rule):
    if the noun is scenery,
        stop the action with library message turning action number 2 for the noun.
Check an actor turning (this is the can't turn people rule):
    if the noun is a person,
        stop the action with library message turning action number 4 for the noun.
```

§128. Report.

```
Report an actor turning (this is the report turning rule):
    if the actor is the player, issue library message turning action number 3
        for the noun;
    otherwise issue actor-based library message turning action number 7 for the noun.
```

§129. Pushing it to.

```
Pushing it to is an action applying to one thing and one visible thing.
The Pushing it to action translates into I6 as "PushDir".
```

```
The specification of the pushing it to action is "This action covers pushing
a large object, not being carried, so that the actor pushes it from one room
to another: for instance, pushing a bale of hay to the east.
```

```
This is rapidly converted into a special form of the going action. If the
noun object has the either/or property 'pushable between rooms', then the
action is converted to going by the 'standard pushing in directions rule'.
If that going action succeeds, then the original pushing it to action
stops; it's only if that fails that we run on into the 'block pushing in
directions rule', which then puts an end to the matter."
```

§130. Check.

```
Check an actor pushing something to (this is the can't push unpushable things rule):
    if the noun is not pushable between rooms,
        stop the action with library message pushing it to action number 1 for
            the noun.
```

```
Check an actor pushing something to (this is the can't push to non-directions rule):
    if the second noun is not a direction,
        stop the action with library message pushing it to action number 2 for
            the noun.
```

```
Check an actor pushing something to (this is the can't push vertically rule):
    if the second noun is up or the second noun is down,
        stop the action with library message pushing it to action number 3 for
            the noun.
```

```
Check an actor pushing something to (this is the standard pushing in directions rule):
    convert to special going-with-push action.
```

```
Check an actor pushing something to (this is the block pushing in directions rule):
    stop the action with library message pushing it to action number 1 for
        the noun.
```

§131. Squeezing.

```
Squeezing is an action applying to one thing.
The Squeezing action translates into I6 as "Squeeze".
```

```
The specification of the squeezing action is "Squeezing is an action which
can conveniently vary from squeezing something hand-held, like a washing-up
liquid bottle, right up to squeezing a pillar in a bear hug.
In the Standard Rules there are no carry out rules for this action because
nothing in the standard Inform world model which reacts to it. The action
is provided for authors to hang more interesting behaviour onto for special
cases. A mildly fruity message is produced to players who attempt to
squeeze people, which is blocked by a check squeezing rule."
```

§132. Check.

```
Check an actor squeezing (this is the innuendo about squeezing people rule):
    if the noun is a person,
        stop the action with library message squeezing action number 1 for
            the noun.
```

§133. Report.

```
Report an actor squeezing (this is the report squeezing rule):
    if the actor is the player, issue library message squeezing action number 2
        for the noun;
    otherwise issue actor-based library message squeezing action number 3 for the noun.
```

§134. Saying yes.

```
Section SR4/8 - Standard actions which always do nothing unless rules intervene

Saying yes is an action applying to nothing.
The Saying yes action translates into I6 as "Yes".

The specification of the saying yes action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§135. Check.

```
Check an actor saying yes (this is the block saying yes rule):
    stop the action with library message saying yes action number 1.
```

§136. Saying no.

```
Saying no is an action applying to nothing.
The Saying no action translates into I6 as "No".

The specification of the saying no action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§137. Check.

```
Check an actor saying no (this is the block saying no rule):
    stop the action with library message saying no action number 1.
```

§138. Burning.

```
Burning is an action applying to one thing.
The Burning action translates into I6 as "Burn".
The specification of the burning action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§139. Check.

```
Check an actor burning (this is the block burning rule):
    stop the action with library message burning action number 1.
```

§140. Waking up.

```
Waking up is an action applying to nothing.
The Waking up action translates into I6 as "Wake".
The specification of the waking up action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§141. Check.

```
Check an actor waking up (this is the block waking up rule):
    stop the action with library message waking up action number 1.
```

§142. Thinking.

```
Thinking is an action applying to nothing.
The Thinking action translates into I6 as "Think".
The specification of the thinking action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§143. Check.

```
Check an actor thinking (this is the block thinking rule):
    stop the action with library message thinking action number 1.
```

§144. Smelling.

```
Smelling is an action applying to one thing.
The Smelling action translates into I6 as "Smell".
The specification of the smelling action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§145. Check.

```
Check an actor smelling (this is the block smelling rule):
    stop the action with library message smelling action number 1 for the noun.
```

§146. Listening to.

```
Listening to is an action applying to one thing.
The Listening to action translates into I6 as "Listen".
The specification of the listening to action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§147. Check.

```
Check an actor listening (this is the block listening rule):
    stop the action with library message listening to action number 1 for the noun.
```

§148. Tasting.

```
Tasting is an action applying to one thing.
The Tasting action translates into I6 as "Taste".
The specification of the tasting action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§149. Check.

```
Check an actor tasting (this is the block tasting rule):
    stop the action with library message tasting action number 1 for the noun.
```

§150. Cutting.

```
Cutting is an action applying to one thing.
The Cutting action translates into I6 as "Cut".
The specification of the cutting action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§151. Check.

```
Check an actor cutting (this is the block cutting rule):
    stop the action with library message cutting action number 1 for the noun.
```

§152. Jumping.

```
Jumping is an action applying to nothing.
The Jumping action translates into I6 as "Jump".
The specification of the jumping action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§153. Check.

```
Check an actor jumping (this is the block jumping rule):
    stop the action with library message jumping action number 1.
```

§154. Tying it to.

```
Tying it to is an action applying to two things.
The Tying it to action translates into I6 as "Tie".
The specification of the tying it to action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§155. Check.

```
Check an actor tying something to (this is the block tying rule):
    stop the action with library message tying it to action number 1 for the noun.
```

§156. Drinking.

```
Drinking is an action applying to one thing.
The Drinking action translates into I6 as "Drink".
The specification of the drinking action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§157. Check.

```
Check an actor drinking (this is the block drinking rule):
    stop the action with library message drinking action number 1 for the noun.
```

§158. Saying sorry.

```
Saying sorry is an action applying to nothing.
The Saying sorry action translates into I6 as "Sorry".
```

```
The specification of the saying sorry action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§159. Check.

```
Check an actor saying sorry (this is the block saying sorry rule):
    stop the action with library message saying sorry action number 1.
```

§160. Swearing obscenely.

```
Swearing obscenely is an action censored, and applying to nothing.
The Swearing obscenely action translates into I6 as "Strong".
```

```
The specification of the swearing obscenely action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§161. Check.

```
Check an actor swearing obscenely (this is the block swearing obscenely rule):
    stop the action with library message swearing obscenely action number 1.
```

§162. Swearing mildly.

```
Swearing mildly is an action censored, and applying to nothing.
The Swearing mildly action translates into I6 as "Mild".
The specification of the swearing mildly action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§163. Check.

```
Check an actor swearing mildly (this is the block swearing mildly rule):
    stop the action with library message swearing mildly action number 1.
```

§164. Swinging.

```
Swinging is an action applying to one thing.
The Swinging action translates into I6 as "Swing".
The specification of the swinging action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§165. Check.

```
Check an actor swinging (this is the block swinging rule):
    stop the action with library message swinging action number 1 for the noun.
```

§166. Rubbing.

```
Rubbing is an action applying to one thing.
The Rubbing action translates into I6 as "Rub".
The specification of the rubbing action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§167. Check.

```
Check an actor rubbing (this is the block rubbing rule):
    stop the action with library message rubbing action number 1 for the noun.
```

§168. Setting it to.

```
Setting it to is an action applying to one thing and one topic.
The Setting it to action translates into I6 as "SetTo".
The specification of the setting it to action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§169. Check.

```
Check an actor setting something to (this is the block setting it to rule):
    stop the action with library message setting it to action number 1 for the noun.
```

§170. Waving hands.

```
Waving hands is an action applying to nothing.
The Waving hands action translates into I6 as "WaveHands".
```

```
The specification of the waving hands action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§171. Check.

```
Check an actor waving hands (this is the block waving hands rule):
    stop the action with library message waving hands action number 1.
```

§172. Buying.

```
Buying is an action applying to one thing.
The Buying action translates into I6 as "Buy".
```

```
The specification of the buying action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§173. Check.

```
Check an actor buying (this is the block buying rule):
    stop the action with library message buying action number 1 for the noun.
```

§174. Singing.

```
Singing is an action applying to nothing.
The Singing action translates into I6 as "Sing".
The specification of the singing action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§175. Check.

```
Check an actor singing (this is the block singing rule):
    stop the action with library message singing action number 1.
```

§176. Climbing.

```
Climbing is an action applying to one thing.
The Climbing action translates into I6 as "Climb".
The specification of the climbing action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§177. Check.

```
Check an actor climbing (this is the block climbing rule):
    stop the action with library message climbing action number 1 for the noun.
```

§178. Sleeping.

```
Sleeping is an action applying to nothing.
The Sleeping action translates into I6 as "Sleep".
```

```
The specification of the sleeping action is
"The Standard Rules define this action in only a minimal way, blocking it
with a check rule which stops it in all cases. It exists so that before
or instead rules can be written to make it do interesting things in special
cases. (Or to reconstruct the action as something more substantial, unlist
the block rule and supply carry out and report rules, together perhaps
with some further check rules.)"
```

§179. Check.

```
Check an actor sleeping (this is the block sleeping rule):
    stop the action with library message sleeping action number 1.
```

§**180. Out of world actions.** We start with a brace of actions which control the (virtual) hardware of the virtual machine: restore, save, quit, restart, verify, and transcript on and off. All of these are implemented at the I6 level where, in fact, they are delegated quickly to assembly language instructions for whichever is the current VM: so these are close to the metal, as they say.

```
Section SR4/9 - Standard actions which happen out of world

Quitting the game is an action out of world and applying to nothing.
The quitting the game action translates into I6 as "Quit".

The quit the game rule is listed in the carry out quitting the game rulebook.
The quit the game rule translates into I6 as "QUIT_THE_GAME_R".

Saving the game is an action out of world and applying to nothing.
The saving the game action translates into I6 as "Save".

The save the game rule is listed in the carry out saving the game rulebook.
The save the game rule translates into I6 as "SAVE_THE_GAME_R".

Restoring the game is an action out of world and applying to nothing.
The restoring the game action translates into I6 as "Restore".

The restore the game rule is listed in the carry out restoring the game rulebook.
The restore the game rule translates into I6 as "RESTORE_THE_GAME_R".

Restarting the game is an action out of world and applying to nothing.
The restarting the game action translates into I6 as "Restart".

The restart the game rule is listed in the carry out restarting the game rulebook.
The restart the game rule translates into I6 as "RESTART_THE_GAME_R".

Verifying the story file is an action out of world applying to nothing.
The verifying the story file action translates into I6 as "Verify".

The verify the story file rule is listed in the carry out verifying the story file rulebook.
The verify the story file rule translates into I6 as "VERIFY_THE_STORY_FILE_R".

Switching the story transcript on is an action out of world and applying to nothing.
The switching the story transcript on action translates into I6 as "ScriptOn".

The switch the story transcript on rule is listed in the carry out switching the story
    transcript on rulebook.
The switch the story transcript on rule translates into I6 as "SWITCH_TRANSCRIPT_ON_R".

Switching the story transcript off is an action out of world and applying to nothing.
The switching the story transcript off action translates into I6 as "ScriptOff".

The switch the story transcript off rule is listed in the carry out switching the story
    transcript off rulebook.
The switch the story transcript off rule translates into I6 as "SWITCH_TRANSCRIPT_OFF_R".
```

§**181.** The VERSION command is not quite so close to the metal – it is implemented in I6, at the end of the day – but it does involve reading the bytes of the story file header, so it needs to take quite different forms for the different formats being compiled to.

```
Requesting the story file version is an action out of world and applying to nothing.
The requesting the story file version action translates into I6 as "Version".

The announce the story file version rule is listed in the carry out requesting the story
    file version rulebook.
The announce the story file version rule translates into I6 as "ANNOUNCE_STORY_FILE_VERSION_R".
```

§**182.** There's really no very good reason why we provide the out-of-world command SCORE but not (say) TIME, or any one of dozens of other traditional what's-my-status commands: DIAGNOSE, say, or PLACES. But we are conservative on this; it's easy for users or extensions to provide these verbs if they want them, and they are not always appropriate for every project. Even SCORE is questionable, but its removal would be a gesture too far.

```
Requesting the score is an action out of world and applying to nothing.
The requesting the score action translates into I6 as "Score".

The announce the score rule is listed in the carry out requesting the score rulebook.
The announce the score rule translates into I6 as "ANNOUNCE_SCORE_R".
```

§**183.** It's perhaps clumsy to have three actions for switching the style of room description, but this accords with I6 custom (and Infocom's, for that matter), and does no harm.

```
Preferring abbreviated room descriptions is an action out of world and applying to nothing.
The preferring abbreviated room descriptions action translates into I6 as "LMode3".

The prefer abbreviated room descriptions rule is listed in the carry out preferring
    abbreviated room descriptions rulebook.
The prefer abbreviated room descriptions rule translates into I6 as "PREFER_ABBREVIATED_R".

The standard report preferring abbreviated room descriptions rule is listed in the
    report preferring abbreviated room descriptions rulebook.
The standard report preferring abbreviated room descriptions rule translates into
    I6 as "REP_PREFER_ABBREVIATED_R".

Preferring unabbreviated room descriptions is an action out of world and applying to nothing.
The preferring unabbreviated room descriptions action translates into I6 as "LMode2".

The prefer unabbreviated room descriptions rule is listed in the carry out preferring
    unabbreviated room descriptions rulebook.
The prefer unabbreviated room descriptions rule translates into I6 as "PREFER_UNABBREVIATED_R".

The standard report preferring unabbreviated room descriptions rule is listed in the
    report preferring unabbreviated room descriptions rulebook.
The standard report preferring unabbreviated room descriptions rule translates into
    I6 as "REP_PREFER_UNABBREVIATED_R".

Preferring sometimes abbreviated room descriptions is an action out of world and
    applying to nothing.
The preferring sometimes abbreviated room descriptions action translates into I6 as "LMode1".

The prefer sometimes abbreviated room descriptions rule is listed in the carry out
    preferring sometimes abbreviated room descriptions rulebook.
The prefer sometimes abbreviated room descriptions rule translates into I6 as
    "PREFER_SOMETIMES_ABBREVIATED_R".

The standard report preferring sometimes abbreviated room descriptions rule is listed
    in the report preferring sometimes abbreviated room descriptions rulebook.
The standard report preferring sometimes abbreviated room descriptions rule translates
    into I6 as "REP_PREFER_SOMETIMES_ABBR_R".
```

§184. Similarly, two different actions handle "notify" and "notify off".

```
Switching score notification on is an action out of world and applying to nothing.
The switching score notification on action translates into I6 as "NotifyOn".
The switch score notification on rule is listed in the carry out switching score
    notification on rulebook.
The switch score notification on rule translates into I6 as "SWITCH_SCORE_NOTIFY_ON_R".
The standard report switching score notification on rule is listed in the report
    switching score notification on rulebook.
The standard report switching score notification on rule translates into
    I6 as "REP_SWITCH_NOTIFY_ON_R".
Switching score notification off is an action out of world and applying to nothing.
The switching score notification off action translates into I6 as "NotifyOff".
The switch score notification off rule is listed in the carry out switching score
    notification off rulebook.
The switch score notification off rule translates into I6 as "SWITCH_SCORE_NOTIFY_OFF_R".
The standard report switching score notification off rule is listed in the report
    switching score notification off rulebook.
The standard report switching score notification off rule translates into
    I6 as "REP_SWITCH_NOTIFY_OFF_R".
```

§185. Lastly, the "pronouns" verb, which is perhaps more often used by people debugging the I6 parser than by actual players.

```
Requesting the pronoun meanings is an action out of world and applying to nothing.
The requesting the pronoun meanings action translates into I6 as "Pronouns".
The announce the pronoun meanings rule is listed in the carry out requesting the
    pronoun meanings rulebook.
The announce the pronoun meanings rule translates into I6 as "ANNOUNCE_PRONOUN_MEANINGS_R".
```

§186. Miscellaneous Grammar Tokens. There's only one of these, at present.

```
The understand token a time period translates into I6 as "RELATIVE_TIME_TOKEN".
```

§187. Grammar.

```
Section SR4/10 - Grammar

Understand "take [things]" as taking.
Understand "take off [something]" as taking off.
Understand "take [something] off" as taking off.
Understand "take [things inside] from [something]" as removing it from.
Understand "take [things inside] off [something]" as removing it from.
Understand "take inventory" as taking inventory.
Understand the commands "carry" and "hold" as "take".

Understand "get in/on" as entering.
Understand "get out/off/down/up" as exiting.
Understand "get [things]" as taking.
Understand "get in/into/on/onto [something]" as entering.
Understand "get off/down [something]" as getting off.
Understand "get [things inside] from [something]" as removing it from.

Understand "pick up [things]" or "pick [things] up" as taking.

Understand "stand" or "stand up" as exiting.
Understand "stand on [something]" as entering.

Understand "remove [something preferably held]" as taking off.
Understand "remove [things inside] from [something]" as removing it from.

Understand "shed [something preferably held]" as taking off.
Understand the commands "doff" and "disrobe" as "shed".

Understand "wear [something preferably held]" as wearing.
Understand the command "don" as "wear".

Understand "put [other things] in/inside/into [something]" as inserting it into.
Understand "put [other things] on/onto [something]" as putting it on.
Understand "put on [something preferably held]" as wearing.
Understand "put [something preferably held] on" as wearing.
Understand "put down [things preferably held]" or "put [things preferably held] down" as dropping.

Understand "insert [other things] in/into [something]" as inserting it into.

Understand "drop [things preferably held]" as dropping.
Understand "drop [other things] in/into/down [something]" as inserting it into.
Understand "drop [other things] on/onto [something]" as putting it on.
Understand "drop [something preferably held] at/against [something]" as throwing it at.
Understand the commands "throw" and "discard" as "drop".

Understand "give [something preferably held] to [someone]" as giving it to.
Understand "give [someone] [something preferably held]" as giving it to (with nouns reversed).
Understand the commands "pay" and "offer" and "feed" as "give".

Understand "show [someone] [something preferably held]" as showing it to (with nouns reversed).
Understand "show [something preferably held] to [someone]" as showing it to.
Understand the commands "present" and "display" as "show".

Understand "go" as going.
Understand "go [direction]" as going.
Understand "go [something]" as entering.
Understand "go into/in/inside/through [something]" as entering.
Understand the commands "walk" and "run" as "go".

Understand "inventory" as taking inventory.
Understand the commands "i" and "inv" as "inventory".

Understand "look" as looking.
Understand "look at [something]" as examining.
Understand "look [something]" as examining.
Understand "look inside/in/into/through [something]" as searching.
Understand "look under [something]" as looking under.
Understand "look up [text] in [something]" as consulting it about (with nouns reversed).
Understand the command "l" as "look".

Understand "consult [something] on/about [text]" as consulting it about.

Understand "open [something]" as opening.
Understand "open [something] with [something preferably held]" as unlocking it with.
Understand the commands "unwrap", "uncover" as "open".

Understand "close [something]" as closing.
Understand "close up [something]" as closing.
Understand "close off [something]" as switching off.
Understand the commands "shut" and "cover" as "close".

Understand "enter" as entering.
Understand "enter [something]" as entering.
Understand the command "cross" as "enter".

Understand "sit on top of [something]" as entering.
Understand "sit on/in/inside [something]" as entering.

Understand "exit" as exiting.
Understand the commands "leave" and "out" as "exit".

Understand "examine [something]" as examining.
Understand the commands "x", "watch", "describe" and "check" as "examine".

Understand "read [something]" as examining.
Understand "read about [text] in [something]" as consulting it about (with nouns reversed).
Understand "read [text] in [something]" as consulting it about (with nouns reversed).

Understand "yes" as saying yes.
Understand the command "y" as "yes".

Understand "no" as saying no.

Understand "sorry" as saying sorry.

Understand "bother" as swearing mildly.
```

...and so on...

```
Understand "search [something]" as searching.

Understand "wave" as waving hands.

Understand "wave [something]" as waving.

Understand "set [something] to [text]" as setting it to.
Understand the command "adjust" as "set".

Understand "pull [something]" as pulling.
Understand the command "drag" as "pull".

Understand "push [something]" as pushing.
Understand "push [something] [direction]" or "push [something] to [direction]" as pushing it to.
Understand the commands "move", "shift", "clear" and "press" as "push".

Understand "turn [something]" as turning.
Understand "turn [something] on" or "turn on [something]" as switching on.
Understand "turn [something] off" or "turn off [something]" as switching off.
Understand the commands "rotate", "twist", "unscrew" and "screw" as "turn".

Understand "switch [something switched on]" as switching off.
Understand "switch [something]" or "switch on [something]" or "switch [something] on" as
```

```
    switching on.
Understand "switch [something] off" or "switch off [something]" as switching off.

Understand "lock [something] with [something preferably held]" as locking it with.

Understand "unlock [something] with [something preferably held]" as unlocking it with.

Understand "attack [something]" as attacking.
Understand the commands "break", "smash", "hit", "fight", "torture", "wreck", "crack", "destroy",
    "murder", "kill", "punch" and "thump" as "attack".

Understand "wait" as waiting.
Understand the command "z" as "wait".

Understand "answer [text] to [someone]" as answering it that (with nouns reversed).
Understand the commands "say", "shout" and "speak" as "answer".

Understand "tell [someone] about [text]" as telling it about.

Understand "ask [someone] about [text]" as asking it about.
Understand "ask [someone] for [something]" as asking it for.

Understand "eat [something preferably held]" as eating.

Understand "sleep" as sleeping.
Understand the command "nap" as "sleep".

Understand "sing" as singing.

Understand "climb [something]" or "climb up/over [something]" as climbing.
Understand the command "scale" as "climb".

Understand "buy [something]" as buying.
Understand the command "purchase" as "buy".

Understand "squeeze [something]" as squeezing.
Understand the command "squash" as "squeeze".

Understand "swing [something]" or "swing on [something]" as swinging.

Understand "wake" or "wake up" as waking up.
Understand "wake [someone]" or "wake [someone] up" or "wake up [someone]" as waking.
Understand the commands "awake" and "awaken" as "wake".

Understand "kiss [someone]" as kissing.
Understand the commands "embrace" and "hug" as "kiss".

Understand "think" as thinking.

Understand "smell" as smelling.
Understand "smell [something]" as smelling.
Understand the command "sniff" as "smell".

Understand "listen" as listening.
Understand "hear [something]" as listening.
Understand "listen to [something]" as listening.

Understand "taste [something]" as tasting.

Understand "touch [something]" as touching.
Understand the command "feel" as "touch".

Understand "rub [something]" as rubbing.
Understand the commands "shine", "polish", "sweep", "clean", "dust", "wipe" and "scrub" as "rub".

Understand "tie [something] to [something]" as tying it to.
Understand the commands "attach" and "fasten" as "tie".

Understand "burn [something]" as burning.
Understand the command "light" as "burn".

Understand "drink [something]" as drinking.
Understand the commands "swallow" and "sip" as "drink".

Understand "cut [something]" as cutting.
Understand the commands "slice", "prune" and "chop" as "cut".

Understand "jump" as jumping.
Understand the commands "skip" and "hop" as "jump".

Understand "score" as requesting the score.
Understand "quit" or "q" as quitting the game.
Understand "save" as saving the game.
Understand "restart" as restarting the game.
Understand "restore" as restoring the game.
Understand "verify" as verifying the story file.
Understand "version" as requesting the story file version.
Understand "script" or "script on" or "transcript" or "transcript on" as switching the story
    transcript on.
Understand "script off" or "transcript off" as switching the story transcript off.
Understand "superbrief" or "short" as preferring abbreviated room descriptions.
Understand "verbose" or "long" as preferring unabbreviated room descriptions.
Understand "brief" or "normal" as preferring sometimes abbreviated room descriptions.
Understand "nouns" or "pronouns" as requesting the pronoun meanings.
Understand "notify" or "notify on" as switching score notification on.
Understand "notify off" as switching score notification off.
```

SR5 - Phrase Definitions A/sr5

Purpose

The phrases making up the Inform language, and in terms of which all other phrases and rules are defined; and the final sign-off of the Standard Rules extension, including its minimal documentation.

§**1.** Our last task is to create the phrases: more or less all of them, but that does need a little qualification. NI has no phrase definitions built in, but it does contain assumptions about how "say ...", "repeat ...", "let ...", "otherwise ..." and "end ..." will behave when defined: we would not be allowed to call these something else, or redefine them in fundamentally different ways. Apart from that, we are more or less free.

Most of these phrases are defined in terms of I6 code, using the `(-` and `-)` notation – it would be too cumbersome to use the "... translates into I6 as ..." verb for this, too. The fact that phrases are not so much translated as transliterated was one source of early criticism of Inform 7. Phrases appeared to have very simplistic definitions, with the natural language simply being a verbose description of obviously equivalent I6 code. However, the simplicity is misleading, because the definitions below tend to conceal where the complexity of the translation process suddenly increases. If the preamble includes "(c - condition)", and the definition includes the expansion `{c}`, then the text forming c is translated in a way much more profound than any simple substitution process could describe. Type-checking also complicates the code produced below, since NI automatically generates the code needed to perform run-time type checking at any point where doubt remains as to the phrase definition which must be used.

§**2. Say phrases.** We begin with saying phrases: the very first phrase to exist is the one printing a single piece of static text, which seems only appropriate for an IF system. We then have a general way to say any value...

```
Part SR5 - Phrasebook

Section SR5/1/1 - Saying - Values

To say (something - text)
    (documented at ph_say):
    (- print (PrintText) {something}; -).
To say (value - sayable value of kind K)
    (documented at phs_value):
    (- print ({-printing-routine:K}) {-pointer-to:value}; -).
```

§**3.** ...but despite that we override this to give numbers their own definition:

```
To say (something - number)
    (documented at phs_value):
    (- print (say__n={something}); -).
```

§4. And similarly for Unicode characters. It would be tidier to abstract this with a function call, but it would cost a function call.

Note that emitting a Unicode character requires different code on the Z-machine to Glulx; we have to handle this within I6 conditional compilation blocks because neither syntax will compile when I6 is compiling for the other VM.

```
To say (ch - unicode character) -- running on
    (documented at phs_unicode):
    (- #ifdef TARGET_ZCODE; @push self; self = {ch}; @print_unicode self; @pull self;
    #ifnot; if (unicode_gestalt_ok) glk_put_char_uni({ch}); else print "?"; #endif; -).
```

§5. Three little grace-notes for printing values: we can have numbers or times of day "in words", rather than given as digits, and we can produce an optional "s" where a number not equal to 1 has recently been printed. This is how "You see [X] balloon[s]." is handled: the printing of the value *X* sets the I6 variable `say__n` as a side-effect (see definition above) and the routine handling "[s]" looks at this variable to see whether to print an "s" or not.

```
To say (something - number) in words
    (documented at phs_numwords):
    (- print (number) say__n=({something}); -).
To say (something - time) in words
    (documented at phs_timewords):
    (- print (PrintTimeOfDayEnglish) {something}; -).
To say s
    (documented at phs_s):
    (- STextSubstitution(); -).
```

§6. Now we come to the fourth and last of the "say (something - *K*)" definitions in the SR, followed by six close variations. Note that say phrases are case sensitive on the first word, so that "to say a something" and "to say A something" are different.

A curiosity of the original I6 design, arising I think mostly from the need to save property memory in *Curses* (1993), the work of IF for which Inform 1 had been created, is that it lacks the `print (A) ...` syntax to match the other forms. The omission is made good by using a routine in the I6 library instead.

```
Section SR5/1/2 - Saying - Names with articles

To say a (something - object)
    (documented at phs_a):
    (- print (a) {something}; -).
To say an (something - object)
    (documented at phs_a):
    (- print (a) {something}; -).
To say A (something - object)
    (documented at phs_A):
    (- CIndefArt({something}); -).
To say An (something - object)
    (documented at phs_A):
    (- CIndefArt({something}); -).
To say the (something - object)
    (documented at phs_the):
    (- print (the) {something}; -).
To say The (something - object)
    (documented at phs_The):
    (- print (The) {something}; -).
```

§**7.** Now for "[if ...]", which expands into a rather assembly-language-like usage of `jump` statements, I6's form of goto. For instance, the text "[if the score is 10]It's ten![otherwise]It's not ten, alas." compiles thus:

```
    if (~~(score == 10)) jump L_Say3;
        ...
    jump L_SayX2; .L_Say3;
        ...
    .L_Say4; .L_SayX2;
```

Though labels actually have local namespaces in I6 routines, we use globally unique labels throughout the whole program: compiling the same phrase again would involve say labels 5 and 6 and "say exit" label 3. This example text demonstrates the reason we `jump` about, rather than making use of `if... else...` and bracing groups of statements: it is legal in I7 either to conclude with or to omit the "[end if]". (If statements in I6 compile to jump instructions in any event, and on our virtual machines there is no speed penalty for branches.) We also need the same definitions to accommodate what amounts to a switch statement. The trickier text "[if the score is 10]It's ten![otherwise if the score is 8]It's eight?[otherwise]It's not ten, alas." comes out as:

```
    if (~~(score == 10)) jump L_Say5;
        ...
    jump L_SayX3; .L_Say5; if (~~(score == 8)) jump L_Say6;
        ...
    jump L_SayX3; .L_Say6;
        ...
    .L_Say7; .L_SayX3;
```

In either form of the construct, control passes into at most one of the pieces of text. The terminal labels (the two on the final line) are automatically generated; often – when there is a simple "otherwise" or "end if" to conclude the construct – they are not needed, but labels are quick to process in I6, are soon discarded from I6's memory when not needed any more, and compile no code.

We assume in each case that the next say label number to be free is always the start of the next block, and that the next say exit label number is always the one at the end of the current construct. This is true because NI does not allow "say if" to be nested.

(The use of `{-erase}` below only tidies up the white space to set out the compiled I6 code neatly, and has no effect on the eventual story file.)

```
Section SR5/1/3 - Saying - Say if and otherwise

To say if (c - condition)
    (documented at phs_if): (- {-erase}
    if (~~({c})) jump {-next-label:Say};
        -).
To say unless (c - condition)
    (documented at phs_unless): (- {-erase}
    if ({c}) jump {-next-label:Say};
        -).
To say otherwise/else if (c - condition)
    (documented at phs_elseif): (- {-erase}
    jump {-next-label:SayX}; .{-label:Say}; if (~~({c})) jump {-next-label:Say};
        -).
To say otherwise/else unless (c - condition)
    (documented at phs_elseunless): (- {-erase}
    jump {-next-label:SayX}; .{-label:Say}; if ({c}) jump {-next-label:Say};
        -).
To say otherwise
    (documented at phs_otherwise): (- {-erase}
    jump {-next-label:SayX}; .{-label:Say};
```

```
        -).
To say else
    (documented at phs_otherwise): (- {-erase}
    jump {-next-label:SayX}; .{-label:Say};
        -).
To say end if
    (documented at phs_endif): (- {-erase}
    .{-label:Say}; .{-label:SayX};
        -).
To say end unless
    (documented at phs_endunless): (- {-erase}
    .{-label:Say}; .{-label:SayX};
        -).
```

§**8.** The other control structure: the random variations form of saying. This part of the Standard Rules was in effect contributed by the community: it reimplements a form of Jon Ingold's former extension Text Variations, which itself built on code going back to the days of I6.

The head phrase here has one of the most complicated definitions in the SR, but is actually documented fairly explicitly in the *Extensions* chapter of *Writing with Inform*, so we won't repeat all that here. Essentially it uses its own allocated cell of storage in an array to remember a state between uses, and compiles as a switch statement based on the current state.

```
Section SR5/1/4 - Saying - Say one of

To say one of -- beginning say_one_of (documented at phs_oneof):
    (- {-allocate-storage:say_one_of}I7_ST_say_one_of-->{-counter:say_one_of} =
    {-final-segment-marker}(I7_ST_say_one_of-->{-counter:say_one_of}, {-segment-count});
    switch((I7_ST_say_one_of-->{-advance-counter:say_one_of})%({-segment-count}+1)-1) {-open-brace}
        0: -).
To say or -- continuing say_one_of (documented at phs_or):
    (- @nop; {-segment-count}: -).
To say at random -- ending say_one_of with marker I7_SOO_RAN (documented at phs_random):
    (- {-close-brace} -).
To say purely at random -- ending say_one_of with marker I7_SOO_PAR (documented at phs_purelyrandom
  ...  ):
    (- {-close-brace} -).
To say then at random -- ending say_one_of with marker I7_SOO_TRAN (documented at phs_thenrandom):
    (- {-close-brace} -).
To say then purely at random -- ending say_one_of with marker I7_SOO_TPAR (documented at phs_thenpu
  ...  relyrandom):
    (- {-close-brace} -).
To say sticky random -- ending say_one_of with marker I7_SOO_STI (documented at phs_sticky):
    (- {-close-brace} -).
To say as decreasingly likely outcomes -- ending say_one_of with marker I7_SOO_TAP (documented at p
  ...  hs_decreasing):
    (- {-close-brace} -).
To say in random order -- ending say_one_of with marker I7_SOO_SHU (documented at phs_order):
    (- {-close-brace} -).
To say cycling -- ending say_one_of with marker I7_SOO_CYC (documented at phs_cycling):
    (- {-close-brace} -).
To say stopping -- ending say_one_of with marker I7_SOO_STOP (documented at phs_stopping):
    (- {-close-brace} -).

To say first time -- beginning say_first_time (documented at phs_firsttime):
    (- {-allocate-storage:say_first_time}
    if ((I7_ST_say_first_time-->{-advance-counter:say_first_time})++ == 0) {-open-brace}
        -).
To say only -- ending say_first_time (documented at phs_firsttime):
    (- {-close-brace} -).
```

§**9.** For an explanation of the paragraph breaking algorithm, see the template file "Printing.i6t".

```
Section SR5/1/5 - Saying - Paragraph control

To say line break -- running on
    (documented at phs_linebreak):
    (- new_line; -).
To say no line break -- running on
    (documented at phs_nolinebreak): do nothing.
To say conditional paragraph break -- running on
    (documented at phs_condparabreak):
    (- DivideParagraphPoint(); -).
To say command clarification break -- running on
    (documented at phs_clarifbreak):
    (- CommandClarificationBreak(); -).
To say paragraph break -- running on
    (documented at phs_parabreak):
    (- DivideParagraphPoint(); new_line; -).
To say run paragraph on -- running on
    (documented at phs_runparaon):
    (- RunParagraphOn(); -).
To say run paragraph on with special look spacing -- running on
    (documented at phs_runparaonsls):
    (- SpecialLookSpacingBreak(); -).
To decide if a paragraph break is pending
    (documented at ph_breakpending):
    (- (say__p) -).
```

§**10.** Now some text substitutions which are the equivalent of escape characters. (In double-quoted I6 text, the notation for a literal quotation mark is a tilde ~.)

```
Section SR5/1/6 - Saying - Special characters

To say bracket -- running on
    (documented at phs_bracket):
    (- print "["; -).
To say close bracket -- running on
    (documented at phs_closebracket):
    (- print "]"; -).
To say apostrophe/' -- running on
    (documented at phs_apostrophe):
    (- print "'"; -).
To say quotation mark -- running on
    (documented at phs_quotemark):
    (- print "~"; -).
```

§11. Now some visual effects, which may or may not be rendered the way the user hopes: that's partly up to the virtual machine, unfortunately.

```
Section SR5/1/7 - Saying - Fonts and visual effects

To say bold type -- running on
    (documented at phs_bold):
    (- style bold; -).
To say italic type -- running on
    (documented at phs_italic):
    (- style underline; -).
To say roman type -- running on
    (documented at phs_roman):
    (- style roman; -).
To say fixed letter spacing -- running on
    (documented at phs_fixedspacing):
    (- font off; -).
To say variable letter spacing -- running on
    (documented at phs_varspacing):
    (- font on; -).
To display the boxed quotation (Q - text)
    (documented at ph_boxed):
    (- DisplayBoxedQuotation({-box-quotation-text:Q}); -).
```

§12. And now some oddball special texts which must sometimes be said.

```
Section SR5/1/8 - Saying - Some built-in texts

To say the/-- banner text
    (documented at phs_banner):
    (- Banner(); -).
To say the/-- list of extension credits
    (documented at phs_extcredits):
    (- ShowExtensionVersions(); -).
To say the/-- complete list of extension credits
    (documented at phs_compextcredits):
    (- ShowFullExtensionVersions(); -).
To say the/-- player's surroundings
    (documented at phs_surroundings):
    (- SL_Location(); -).
```

§13. Using the list-writer. The I7 list-writer resembles the old I6 library one, but has been reimplemented in a more general way: see the template file "ListWriter.i6t". The following is the main routine for listing:

```
Section SR5/1/9 - Saying - Saying lists of things

To list the contents of (O - an object),
    with newlines,
    indented,
    giving inventory information,
    as a sentence,
    including contents,
    including all contents,
    tersely,
    giving brief inventory information,
    using the definite article,
    listing marked items only,
    prefacing with is/are,
    not listing concealed items,
    suppressing all articles
    and/or with extra indentation
    (documented at ph_listcontents):
    (- WriteListFrom(child({O}), {phrase options}); -).

To say contents of (O - an object)
    (deprecated)
    (documented at phs_contents_dep):
    list the contents of O, as a sentence.

To say the contents of (O - an object)
    (deprecated)
    (documented at phs_contents_dep):
    list the contents of O, as a sentence, using the definite article.
```

§14. Text substitutions using the list-writer. These all look (and are) repetitive. We want to avoid passing a description value to some routine, because that's tricky if the description needs to refer to a value local to the current stack frame. (There are ways round that, but it minimises nuisance to avoid the need.) So we mark out the set of objects matching by giving them, and only them, the `workflag2` attribute.

```
To say a list of (OS - description of objects)
    (documented at phs_alistof):
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT);
        @pull subst__v; -).
To say A list of (OS - description of objects)
    (documented at phs_Alistof):
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT+CFIRSTART_BIT);
        @pull subst__v; -).
To say list of (OS - description of objects)
    (documented at phs_listof):
```

```
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT+NOARTICLE_BIT);
        @pull subst__v; -).
To say the list of (OS - description of objects)
    (documented at phs_thelistof):
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT+DEFART_BIT);
        @pull subst__v; -).
To say The list of (OS - description of objects)
    (documented at phs_Thelistof):
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT+DEFART_BIT+CFIRSTART_BIT);
        @pull subst__v; -).
To say is-are a list of (OS - description of objects)
    (documented at phs_isalistof):
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT+ISARE_BIT);
        @pull subst__v; -).
To say is-are list of (OS - description of objects)
    (documented at phs_islistof):
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT+ISARE_BIT+NOARTICLE_BIT);
        @pull subst__v; -).
To say is-are the list of (OS - description of objects)
    (documented at phs_isthelistof):
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT+DEFART_BIT+ISARE_BIT);
        @pull subst__v; -).
To say a list of (OS - description of objects) including contents
    (documented at phs_alistofconts):
    (- @push subst__v;
        objectloop (subst__v ofclass Object) if ({-bind-variable:OS})
        give subst__v workflag2; else give subst__v ~workflag2;
        WriteListOfMarkedObjects(ENGLISH_BIT+RECURSE_BIT+PARTINV_BIT+
            TERSE_BIT+CONCEAL_BIT);
        @pull subst__v; -).
```

§15. Grouping in the list-writer. See the specifications of `list_together` and `c_style` in the DM4, which are still broadly accurate.

```
Section SR5/1/10 - Saying - Group in and omit from lists

To group (OS - description of objects) together
    (documented at ph_group):
    (- @push subst__v;
        objectloop (subst__v provides list_together) if ({-bind-variable:OS})
        subst__v.list_together = {-list-together};
        @pull subst__v; -).
To group (OS - description of objects) together giving articles
    (documented at ph_grouppart):
    (- @push subst__v;
        objectloop (subst__v provides list_together) if ({-bind-variable:OS})
        subst__v.list_together = {-articled-list-together};
        @pull subst__v; -).
To group (OS - description of objects) together as (T - text)
    (documented at ph_grouptext):
    (- @push subst__v;
        objectloop (subst__v provides list_together) if ({-bind-variable:OS})
        subst__v.list_together = {T};
        @pull subst__v; -).
To omit contents in listing
    (documented at ph_omit):
    (- c_style = c_style &~ (RECURSE_BIT+FULLINV_BIT+PARTINV_BIT); -).
```

§16. Lists written but not by the list-writer. These are lists in the sense of the "list of" kind of value constructor, and they might contain any values, not just objects. They're printed by code in "Lists.i6t".

```
Section SR5/1/11 - Saying - Lists of values

To say (L - a list of values) in brace notation
    (documented at phs_listbraced):
    (- LIST_OF_TY_Say({-pointer-to:L}, 1); -).
To say (L - a list of objects) with definite articles
    (documented at phs_listdef):
    (- LIST_OF_TY_Say({-pointer-to:L}, 2); -).
To say (L - a list of objects) with indefinite articles
    (documented at phs_listindef):
    (- LIST_OF_TY_Say({-pointer-to:L}, 3); -).
```

§17. Filtering in the list-writer. Something of a last resort, which is intentionally not documented. It's needed by the Standard Rules to tidy up an implementation and avoid I6, but is not an ideal trick and may be dropped in later builds. Recursion occurs when the list-writer descends to the contents of, or items supported by, something it lists. Here we can restrict to just those contents, or supportees, matching a description `D`.

```
Section SR5/1/12 - Saying - Filtering contents - Unindexed

To filter list recursion to (D - description of objects):
    (- list_filter_routine = {D}; -).
To unfilter list recursion:
    (- list_filter_routine = 0; -).
```

§18. Values. To begin with, the magic “now”.

```
Section SR5/2/1 - Values - Making conditions true

To now (cn - now-condition)
    (documented at ph_now):
    (- {cn} -).
```

§19. Assignment is probably the most difficult thing the type-checker has to cope with, since “let” has to work when applied to both unknown names (it creates a new variable) and existing ones (kind of value permitting). There are also four different ways to create with “let”, and two to use existing variables. Note that the “given by” forms are not strictly speaking assignments at all; the value placed in `t` is found by solving the equation `Q`. This does require special typechecking, but of a different kind to that requested by “(assignment operation)”. All of which makes the “To let” section here only slightly shorter than John Galsworthy’s Forsyte novel of the same name:

```
Section SR5/2/2 - Values - Giving values temporary names

To let (t - nonexisting variable) be (u - value)
    (assignment operation)
    (documented at ph_let):
    (- {-creationassignment}; -).
To let (t - nonexisting variable) be (u - name of kind of value)
    (assignment operation)
    (documented at ph_letdefault):
    (- {-creationdefault}; -).
To let (t - nonexisting variable) be (u - description of relations)
    (assignment operation)
    (documented at ph_letrelation):
    (- {-pointer-to:t} = {-allocate-storage-for:u}; {-adapt-relation}; -).
To let (t - nonexisting variable) be given by (Q - equation name)
    (documented at ph_letequation):
    (- {-solve-equation}; -).

To let (t - existing variable) be (u - value)
    (assignment operation)
    (documented at ph_let):
    (- {-assignment}; -).
To let (t - existing variable) be given by (Q - equation name)
    (documented at ph_letequation):
    (- {-solve-equation}; -).
```

§20. There are five sorts of storage in I7 at present: local variables (i.e., “let” and “called” variables, together with parameters in “To...” preambles), global variables, table entries, property values and list entries. Objects are not a form of storage in this sense. The reason we handle “change O to X” specially where O is an object is that it’s legal to “change O to open” if O is a container or door, or to “change O to 20kg” if O has a weight, for instance. The meaning then is to do with the transference of a property, not an assignment. This requires careful disambiguation, since O may be, e.g., a global variable whose kind of value is “object”.

```
Section SR5/2/3 - Values - Changing stored values

To change (S - storage) to (w - value)
    (deprecated)
    (assignment operation)
    (documented at ph_change_dep):
    (- {-assignment}; -).
To change (o - object) to (p - property)
    (deprecated)
    (assignment operation)
    (documented at ph_change_dep):
    (- SetEitherOrProperty({o}, {p}, false, {-adjective-definition:p}); -).
To change (o - object) to (w - enumerated value)
    (deprecated)
    (assignment operation)
    (documented at ph_change_dep):
    (- WriteGProperty(OBJECT_TY,{o},{-convert-adjectival-constants:w},{w}); -).
```

§21. It is not explicit in the following definitions that Inform should typecheck that the values held by these storage objects can be incremented or decremented (as a room, say, cannot, but a number or a height can): Inform nevertheless contains code which does this.

```
To increase (S - storage) by (w - value)
    (assignment operation)
    (documented at ph_increase):
    (- {-increase}; -).
To decrease (S - storage) by (w - value)
    (assignment operation)
    (documented at ph_decrease):
    (- {-decrease}; -).
To increment (S - storage)
    (documented at ph_increment):
    (- {-increment}; -).
To decrement (S - storage)
    (documented at ph_decrement):
    (- {-decrement}; -).
```

§**22.** There are eight arithmetic operations, internally numbered 0 to 7, and given verbal forms below. These are handled unusually in the type-checker because they need to be more polymorphic than most phrases: Inform uses dimension-checking to determine the kind of value resulting. (Thus a height times a number is another height, and so on.)

The totalling code (10) is not structly to do with arithmetic in the same way, but it's needed to flag the phrase for the Inform typechecker's special attention.

```
Section SR5/2/4 - Values - Arithmetic

To decide which arithmetic value is (X - arithmetic value) + (Y - arithmetic value)
    (arithmetic operation 0)
    (documented at ph_plus): (- ({X}+{Y}) -).
To decide which arithmetic value is (X - arithmetic value) plus (Y - arithmetic value)
    (arithmetic operation 0)
    (documented at ph_plus):
    (- ({X}+{Y}) -).
To decide which arithmetic value is (X - arithmetic value) - (Y - arithmetic value)
    (arithmetic operation 1)
    (documented at ph_minus):
    (- ({X}-{Y}) -).
To decide which arithmetic value is (X - arithmetic value) minus (Y - arithmetic value)
    (arithmetic operation 1)
    (documented at ph_minus):
    (- ({X}-{Y}) -).
To decide which arithmetic value is (X - arithmetic value) * (Y - arithmetic value)
    (arithmetic operation 2)
    (documented at ph_times):
    (- ({X}*{Y}{-rescale-times}) -).
To decide which arithmetic value is (X - arithmetic value) times (Y - arithmetic value)
    (arithmetic operation 2)
    (documented at ph_times):
    (- ({X}*{Y}{-rescale-times}) -).
To decide which arithmetic value is (X - arithmetic value) multiplied by (Y - arithmetic value)
    (arithmetic operation 2)
    (documented at ph_times):
    (- ({X}*{Y}{-rescale-times}) -).
To decide which arithmetic value is (X - arithmetic value) / (Y - arithmetic value)
    (arithmetic operation 3)
    (documented at ph_divide):
    (- (IntegerDivide({X}{-rescale-divide},{Y})) -).
To decide which arithmetic value is (X - arithmetic value) divided by (Y - arithmetic value)
    (arithmetic operation 3)
    (documented at ph_divide):
    (- (IntegerDivide({X}{-rescale-divide},{Y})) -).
To decide which arithmetic value is remainder after dividing (X - arithmetic value)
    by (Y - arithmetic value)
    (arithmetic operation 4)
    (documented at ph_remainder):
    (- (IntegerRemainder({X},{Y})) -).
To decide which arithmetic value is (X - arithmetic value) to the nearest (Y - arithmetic value)
    (arithmetic operation 5)
    (documented at ph_nearest):
    (- (RoundOffTime({X},{Y})) -).
To decide which arithmetic value is the square root of (X - arithmetic value)
```

```
    (arithmetic operation 6)
    (documented at ph_squareroot):
    (- (SquareRoot({X}{-rescale-root})) -).
To decide which arithmetic value is the cube root of (X - arithmetic value)
    (arithmetic operation 7)
    (documented at ph_cuberoot):
    (- (CubeRoot({X}{-rescale-cuberoot})) -).
To decide which arithmetic value is total (p - arithmetic value valued property)
    of (S - description of values)
    (arithmetic operation 11)
    (documented at ph_total):
    (- {-total-of:S} -).
```

§**23.** Some of the things we can do with enumerations, others being listed under randomness below.

```
Section SR5/2/5 - Values - Enumerations

To decide which number is number of (S - description of values)
    (documented at ph_numberof):
    (- {-number-of:S} -).
To decide which K is (name of kind of enumerated value K) after (X - K)
    (documented at ph_enumafter):
    (- A_{-printing-routine:K}({X}) -).
To decide which K is (name of kind of enumerated value K) before (X - K)
    (documented at ph_enumbefore):
    (- B_{-printing-routine:K}({X}) -).
To decide which K is the first value of (name of kind of enumerated value K)
    (documented at ph_enumfirst):
    decide on the default value of K.
To decide which K is the last value of (name of kind of enumerated value K)
    (documented at ph_enumlast):
    decide on K before the default value of K.
```

§**24.** The following exists only to convert a condition to a value, and is needed because I7 does not silently cast from one to the other in the way that C would.

```
Section SR5/2/6 - Values - Truth states

To decide what truth state is whether or not (C - condition)
    (documented at ph_whether):
    (- ({C}) -).
```

§**25.** Random numbers and random items chosen from sets of objects matching a given description ("a random closed door").

```
Section SR5/2/7 - Values - Randomness

To decide which K is a/-- random (S - description of values of kind K)
    (documented at ph_randomdesc):
    (- {-random-of:S} -).
To decide which K is a random (name of kind of arithmetic value K) between (first value - K) and (s
  ...  econd value - K)
    (documented at ph_randombetween):
    (- R_{-printing-routine:K}({first value}, {second value}) -).
To decide which K is a random (name of kind of arithmetic value K) from (first value - K) to (secon
  ...  d value - K)
    (documented at ph_randombetween):
    (- R_{-printing-routine:K}({first value}, {second value}) -).
To decide which K is a random (name of kind of enumerated value K) between (first value - K) and (s
  ...  econd value - K)
    (documented at ph_randombetween):
    (- R_{-printing-routine:K}({first value}, {second value}) -).
To decide which K is a random (name of kind of enumerated value K) from (first value - K) to (secon
  ...  d value - K)
    (documented at ph_randombetween):
    (- R_{-printing-routine:K}({first value}, {second value}) -).
To decide whether a random chance of (N - number) in (M - number) succeeds
    (documented at ph_randomchance):
    (- (GenerateRandomNumber(1, {M}) <= {N}) -).
To seed the random-number generator with (N - number)
    (documented at ph_seed):
    (- VM_Seed_RNG({N}); -).
```

§**26. Tables.** And off we go into the world of tables, the code for which is all in "Tables.i6t". Note that changing a table entry is not here: it's above, with the phrases for changing other storage objects.

```
Section SR5/2/8 - Values - Tables

To choose a/the/-- row (N - number) in/from (T - table name)
    (documented at ph_chooserow):
    (- {-require-ctvs}ct_0 = {T}; ct_1 = {N}; -).
To choose a/the/-- row with (TC - K valued table column) of (w - value of kind K)
    in/from (T - table name)
    (documented at ph_chooserowwith):
    (- {-require-ctvs}ct_0 = {T}; ct_1 = TableRowCorr(ct_0, {TC}, {w}); -).
To choose a/the/-- blank row in/from (T - table name)
    (documented at ph_chooseblankrow):
    (- {-require-ctvs}ct_0 = {T}; ct_1 = TableBlankRow(ct_0); -).
To choose a/the/-- random row in/from (T - table name)
    (documented at ph_chooserandomrow):
    (- {-require-ctvs}ct_0 = {T}; ct_1 = TableRandomRow(ct_0); -).
To decide which number is number of rows in/from (T - table name)
    (documented at ph_numrows):
    (- TableRows({T}) -).
To decide which number is number of blank rows in/from (T - table name)
    (documented at ph_numblank):
```

```
    (- TableBlankRows({T}) -).
To decide which number is number of filled rows in/from (T - table name)
    (documented at ph_numfilled):
    (- TableFilledRows({T}) -).
To decide if there is (TR - table-reference)
    (documented at ph_thereis):
    (- (Exists{-do-not-dereference:TR}) -).
To decide if there is no (TR - table-reference)
    (documented at ph_thereisno):
    (- (Exists{-do-not-dereference:TR} == false) -).
To blank out (tr - table-reference)
    (documented at ph_blankout):
    (- {tr}{-backspace},4); -).
To blank out the whole row
    (documented at ph_blankoutrow):
    (- {-require-ctvs}TableBlankOutRow(ct_0, ct_1); -).
To blank out the whole (TC - table column) in/from (T - table name)
    (documented at ph_blankoutcol):
    (- TableBlankOutColumn({T}, {TC}); -).
To blank out the whole of (T - table name)
    (documented at ph_blankouttable):
    (- TableBlankOutAll({T}); -).
To delete (tr - table-reference)
    (deprecated)
    (documented at ph_deleteentry_dep):
    (- {tr}{-backspace},4); -).
```

§27. Sorting.

```
Section SR5/2/9 - Values - Sorting tables

To sort (T - table name) in random order
    (documented at ph_sortrandom):
    (- TableShuffle({T}); -).
To sort (T - table name) in (TC - table column) order
    (documented at ph_sortcolumn):
    (- TableSort({T}, {TC}, 1); -).
To sort (T - table name) in reverse (TC - table column) order
    (documented at ph_sortcolumnreverse):
    (- TableSort({T}, {TC}, -1); -).
```

§28. Indexed text. As repetitive as the following is, it's much simpler and less prone to possible namespace trouble if we don't define kinds of value for the different structural levels of text (character, word, punctuated word, etc.).

```
Section SR5/2/10 - Values - Indexed text

To decide what number is the number of characters in (txb - indexed text)
    (documented at ph_numchars):
    (- IT_BlobAccess({-pointer-to:txb}, CHR_BLOB) -).
To decide what number is the number of words in (txb - indexed text)
    (documented at ph_numwords):
    (- IT_BlobAccess({-pointer-to:txb}, WORD_BLOB) -).
To decide what number is the number of punctuated words in (txb - indexed text)
    (documented at ph_numpwords):
    (- IT_BlobAccess({-pointer-to:txb}, PWORD_BLOB) -).
To decide what number is the number of unpunctuated words in (txb - indexed text)
    (documented at ph_numupwords):
    (- IT_BlobAccess({-pointer-to:txb}, UWORD_BLOB) -).
To decide what number is the number of lines in (txb - indexed text)
    (documented at ph_numlines):
    (- IT_BlobAccess({-pointer-to:txb}, LINE_BLOB) -).
To decide what number is the number of paragraphs in (txb - indexed text)
    (documented at ph_numparas):
    (- IT_BlobAccess({-pointer-to:txb}, PARA_BLOB) -).

To decide what indexed text is character number (N - a number) in (txb - indexed text)
    (documented at ph_charnum):
    (- IT_GetBlob({-pointer-to-new:indexed text}, {-pointer-to:txb}, {N}, CHR_BLOB) -).
To decide what indexed text is word number (N - a number) in (txb - indexed text)
    (documented at ph_wordnum):
    (- IT_GetBlob({-pointer-to-new:indexed text}, {-pointer-to:txb}, {N}, WORD_BLOB) -).
To decide what indexed text is punctuated word number (N - a number) in (txb - indexed text)
    (documented at ph_pwordnum):
    (- IT_GetBlob({-pointer-to-new:indexed text}, {-pointer-to:txb}, {N}, PWORD_BLOB) -).
To decide what indexed text is unpunctuated word number (N - a number) in (txb - indexed text)
    (documented at ph_upwordnum):
    (- IT_GetBlob({-pointer-to-new:indexed text}, {-pointer-to:txb}, {N}, UWORD_BLOB) -).
To decide what indexed text is line number (N - a number) in (txb - indexed text)
    (documented at ph_linenum):
    (- IT_GetBlob({-pointer-to-new:indexed text}, {-pointer-to:txb}, {N}, LINE_BLOB) -).
To decide what indexed text is paragraph number (N - a number) in (txb - indexed text)
    (documented at ph_paranum):
    (- IT_GetBlob({-pointer-to-new:indexed text}, {-pointer-to:txb}, {N}, PARA_BLOB) -).
```

§29. Matching text. A common matching engine is used for matching plain text...

```
Section SR5/2/11 - Values - Matching text

To decide if (txb - indexed text) exactly matches the text (ftxb - indexed text),
    case insensitively
    (documented at ph_exactlymatches):
    (- IT_Replace_RE(CHR_BLOB,{-pointer-to:txb},{-pointer-to:ftxb},0,{phrase options},1) -).
To decide if (txb - indexed text) matches the text (ftxb - indexed text),
    case insensitively
    (documented at ph_matches):
    (- IT_Replace_RE(CHR_BLOB,{-pointer-to:txb},{-pointer-to:ftxb},0,{phrase options}) -).
To decide what number is number of times (txb - indexed text) matches the text
    (ftxb - indexed text), case insensitively
    (documented at ph_nummatches):
    (- IT_Replace_RE(CHR_BLOB,{-pointer-to:txb},{-pointer-to:ftxb},1,{phrase options}) -).
```

§30. ...and for regular expressions, though here we also have access to the exact text which matched (not interesting in the plain text case since it's the same as the search text, up to case at least), and the values of matched subexpressions (which the plain text case doesn't have).

```
To decide if (txb - indexed text) exactly matches the regular expression (ftxb - indexed text),
    case insensitively
    (documented at ph_exactlymatchesre):
    (- IT_Replace_RE(REGEXP_BLOB,{-pointer-to:txb},{-pointer-to:ftxb},0,{phrase options},1) -).
To decide if (txb - indexed text) matches the regular expression (ftxb - indexed text),
    case insensitively
    (documented at ph_matchesre):
    (- IT_Replace_RE(REGEXP_BLOB,{-pointer-to:txb},{-pointer-to:ftxb},0,{phrase options}) -).
To decide what indexed text is text matching regular expression
    (documented at ph_matchtext):
    (- IT_RE_GetMatchVar({-pointer-to-new:indexed text}, 0) -).
To decide what indexed text is text matching subexpression (N - a number)
    (documented at ph_subexpressiontext):
    (- IT_RE_GetMatchVar({-pointer-to-new:indexed text}, {N}) -).
To decide what number is number of times (txb - indexed text) matches the regular expression
    (ftxb - indexed text),case insensitively
    (documented at ph_nummatchesre):
    (- IT_Replace_RE(REGEXP_BLOB,{-pointer-to:txb},{-pointer-to:ftxb},1,{phrase options}) -).
```

§31. Replacing text. The same engine, in "RegExp.i6t", handles replacement.

```
Section SR5/2/12 - Values - Replacing text
To replace the text (ftxb - indexed text) in (txb - indexed text) with (rtxb - indexed text),
    case insensitively
    (documented at ph_replace):
    (- IT_Replace_RE(CHR_BLOB, {-pointer-to:txb}, {-pointer-to:ftxb},
        {-pointer-to:rtxb}, {phrase options}); -).
To replace the regular expression (ftxb - indexed text) in (txb - indexed text) with
    (rtxb - indexed text), case insensitively
    (documented at ph_replacere):
    (- IT_Replace_RE(REGEXP_BLOB, {-pointer-to:txb}, {-pointer-to:ftxb},
        {-pointer-to:rtxb}, {phrase options}); -).
To replace the word (ftxb - indexed text) in (txb - indexed text) with
    (rtxb - indexed text)
    (documented at ph_replacewordin):
    (- IT_ReplaceText(WORD_BLOB, {-pointer-to:txb}, {-pointer-to:ftxb}, {-pointer-to:rtxb}); -).
To replace the punctuated word (ftxb - indexed text) in (txb - indexed text)
    with (rtxb - indexed text)
    (documented at ph_replacepwordin):
    (- IT_ReplaceText(PWORD_BLOB, {-pointer-to:txb}, {-pointer-to:ftxb}, {-pointer-to:rtxb}); -).
To replace character number (N - a number) in (txb - indexed text)
    with (rtxb - indexed text)
    (documented at ph_replacechar):
    (- IT_ReplaceBlob(CHR_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).
To replace word number (N - a number) in (txb - indexed text)
    with (rtxb - indexed text)
    (documented at ph_replaceword):
    (- IT_ReplaceBlob(WORD_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).
To replace punctuated word number (N - a number) in (txb - indexed text)
    with (rtxb - indexed text)
    (documented at ph_replacepword):
    (- IT_ReplaceBlob(PWORD_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).
To replace unpunctuated word number (N - a number) in (txb - indexed text)
    with (rtxb - indexed text)
    (documented at ph_replaceupword):
    (- IT_ReplaceBlob(UWORD_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).
To replace line number (N - a number) in (txb - indexed text) with (rtxb - indexed text)
    (documented at ph_replaceline):
    (- IT_ReplaceBlob(LINE_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).
To replace paragraph number (N - a number) in (txb - indexed text) with (rtxb - indexed text)
    (documented at ph_replacepara):
    (- IT_ReplaceBlob(PARA_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).
```

§32. Casing of text.

```
Section SR5/2/13 - Values - Casing of text

To decide what indexed text is (txb - indexed text) in lower case
    (documented at ph_lowercase):
    (- IT_CharactersToCase({-pointer-to-new:indexed text}, {-pointer-to:txb}, 0) -).
To decide what indexed text is (txb - indexed text) in upper case
    (documented at ph_uppercase):
    (- IT_CharactersToCase({-pointer-to-new:indexed text}, {-pointer-to:txb}, 1) -).
To decide what indexed text is (txb - indexed text) in title case
    (documented at ph_titlecase):
    (- IT_CharactersToCase({-pointer-to-new:indexed text}, {-pointer-to:txb}, 2) -).
To decide what indexed text is (txb - indexed text) in sentence case
    (documented at ph_sentencecase):
    (- IT_CharactersToCase({-pointer-to-new:indexed text}, {-pointer-to:txb}, 3) -).
To decide if (txb - indexed text) is in lower case
    (documented at ph_inlower):
    (- IT_CharactersOfCase({-pointer-to:txb}, 0) -).
To decide if (txb - indexed text) is in upper case
    (documented at ph_inupper):
    (- IT_CharactersOfCase({-pointer-to:txb}, 1) -).
```

§33. Lists. The following are all for adding and removing values to dynamic lists:

```
Section SR5/2/14 - Values - Lists

To add (new entry - K) to (L - list of values of kind K), if absent
    (documented at ph_addtolist):
    (- LIST_OF_TY_InsertItem({-pointer-to:L}, {new entry}, 0, 0, {phrase options}); -).

To add (new entry - K) at entry (E - number) in (L - list of values of kind K), if absent
    (documented at ph_addatentry):
    (- LIST_OF_TY_InsertItem({-pointer-to:L}, {new entry}, 1, {E}, {phrase options}); -).

To add (LX - list of Ks) to (L - list of values of kind K), if absent
    (documented at ph_addlisttolist):
    (- LIST_OF_TY_AppendList({-pointer-to:L}, {-pointer-to:LX}, 0, 0, {phrase options}); -).

To add (LX - list of Ks) at entry (E - number) in (L - list of values of kind K)
    (documented at ph_addlistatentry):
    (- LIST_OF_TY_AppendList({-pointer-to:L}, {-pointer-to:LX}, 1, {E}, 0); -).

To remove (existing entry - K) from (L - list of values of kind K), if present
    (documented at ph_remfromlist):
    (- LIST_OF_TY_RemoveValue({-pointer-to:L}, {existing entry}, {phrase options}); -).

To remove (N - list of Ks) from (L - list of values of kind K), if present
    (documented at ph_remlistfromlist):
    (- LIST_OF_TY_Remove_List({-pointer-to:L}, {-pointer-to:N}, {phrase options}); -).

To remove entry (N - number) from (L - list of values), if present
    (documented at ph_rementry):
    (- LIST_OF_TY_RemoveItemRange({-pointer-to:L}, {N}, {N}, {phrase options}); -).

To remove entries (N - number) to (N2 - number) from (L - list of values), if present
    (documented at ph_rementries):
    (- LIST_OF_TY_RemoveItemRange({-pointer-to:L}, {N}, {N2}, {phrase options}); -).
```

§**34.** Searching a list is implemented in a somewhat crude way at present, and the following syntax may later be replaced with a suitable verb "to be listed in", so that there's no need to imitate.

```
To decide if (N - K) is listed in (L - list of values of kind K)
    (documented at ph_islistedin):
    (- (LIST_OF_TY_FindItem({-pointer-to:L}, {N})) -).
To decide if (N - K) is not listed in (L - list of values of kind K)
    (documented at ph_isnotlistedin):
    (- (LIST_OF_TY_FindItem({-pointer-to:L}, {N}) == false) -).
```

§**35.** The following are casts to and from other kinds or objects. Lists are not the only I7 way to hold list-like data, because sometimes the memory requirements for dynamic lists are beyond what the virtual machine can sustain.

(a) A description is a representation of a set of objects by means of a predicate (e.g., "open unlocked doors"), and it converts into a list of current members (in creation order), but there's no reverse process.

(b) The multiple object list is a data structure used in the parser when processing commands like TAKE ALL.

```
To decide what list of Ks is the list of (D - description of values of kind K)
    (documented at ph_listofdesc):
    (- LIST_OF_TY_Desc({-pointer-to-new:list of K}, {D}, {-strong-kind:K}) -).
To decide what list of objects is the multiple object list
    (documented at ph_multipleobjectlist):
    (- LIST_OF_TY_Mol({-pointer-to-new:list of objects}) -).
To alter the multiple object list to (L - list of objects)
    (documented at ph_altermultipleobjectlist):
    (- LIST_OF_TY_Set_Mol({-pointer-to:L}); -).
```

§**36.** Determining and setting the length:

```
Section SR5/2/15 - Values - Length of lists
To decide what number is the number of entries in/of (L - a list of values)
    (documented at ph_numberentries):
    (- LIST_OF_TY_GetLength({-pointer-to:L}) -).
To truncate (L - a list of values) to (N - a number) entries/entry
    (documented at ph_truncate):
    (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, -1, 1); -).
To truncate (L - a list of values) to the first (N - a number) entries/entry
    (documented at ph_truncatefirst):
    (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, -1, 1); -).
To truncate (L - a list of values) to the last (N - a number) entries/entry
    (documented at ph_truncatelast):
    (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, -1, -1); -).
To extend (L - a list of values) to (N - a number) entries/entry
    (documented at ph_extend):
    (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, 1); -).
To change (L - a list of values) to have (N - a number) entries/entry
    (documented at ph_changelength):
    (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, 0); -).
```

§**37.** Easy but useful list operations:

```
Section SR5/2/16 - Values - Reversing and rotating lists

To reverse (L - a list of values)
    (documented at ph_reverselist):
    (- LIST_OF_TY_Reverse({-pointer-to:L}); -).
To rotate (L - a list of values)
    (documented at ph_rotatelist):
    (- LIST_OF_TY_Rotate({-pointer-to:L}, 0); -).
To rotate (L - a list of values) backwards
    (documented at ph_rotatelistback):
    (- LIST_OF_TY_Rotate({-pointer-to:L}, 1); -).
```

§**38.** Sorting ultimately uses a common sorting mechanism, in “Sort.i6t”, which handles both lists and tables.

```
Section SR5/2/17 - Values - Sorting lists

To sort (L - a list of values)
    (documented at ph_sortlist):
    (- LIST_OF_TY_Sort({-pointer-to:L}, 1); -).
To sort (L - a list of values) in reverse order
    (documented at ph_sortlistreverse):
    (- LIST_OF_TY_Sort({-pointer-to:L}, -1); -).
To sort (L - a list of values) in random order
    (documented at ph_sortlistrandom):
    (- LIST_OF_TY_Sort({-pointer-to:L}, 2); -).
To sort (L - a list of objects) in (P - property) order
    (documented at ph_sortlistproperty):
    (- LIST_OF_TY_Sort({-pointer-to:L}, 1, {P}, {-block-value:P}); -).
To sort (L - a list of objects) in reverse (P - property) order
    (documented at ph_sortlistpropertyreverse):
    (- LIST_OF_TY_Sort({-pointer-to:L}, -1, {P}, {-block-value:P}); -).
```

§**39.** To call use options a “data structure” is a stretch. Moreover, the following phrase is now deprecated: write “if O is active” in preference to “if using O”.

```
Section SR5/2/18 - Values - Use options

To decide whether using the/-- (UO - use option)
    (deprecated)
    (documented at ph_testuseoption_dep):
    (- (TestUseOption({UO})) -).
```

§**40.** Relations are the final data structure given here. In some ways they are the most fundamental of all, but they're not either set or tested by procedural phrases – they lie in the linguistic structure of conditions. So all we have here are the route-finding phrases:

```
Section SR5/2/19 - Values - Relations

To show relation (R - relation)
    (documented at ph_showrelation):
    (- {-show-me}; RelationTest({-pointer-to:R}, RELS_SHOW); -).

To decide which object is next step via (R - relation of values to values)
    from (O1 - object) to (O2 - object)
    (documented at ph_nextstep):
    (- RelationRouteTo({-pointer-to:R},{O1},{O2},false) -).
To decide which number is number of steps via (R - relation of values to values)
    from (O1 - object) to (O2 - object)
    (documented at ph_numbersteps):
    (- RelationRouteTo({-pointer-to:R},{O1},{O2},true) -).

To decide which list of Ks is list of (name of kind of value K)
    that/which/whom (R - relation of Ks to values of kind L) relates
    (documented at ph_leftdomain):
    (- RelationTest({-pointer-to:R}, RELS_LIST, {-pointer-to-new:list of K}, RLIST_ALL_X) -).

To decide which list of Ls is list of (name of kind of value L)
    to which/whom (R - relation of values of kind K to Ls) relates
    (documented at ph_rightdomain):
    (- RelationTest({-pointer-to:R}, RELS_LIST, {-pointer-to-new:list of L}, RLIST_ALL_Y) -). [1]

To decide which list of Ls is list of (name of kind of value L)
    that/which/whom (R - relation of values of kind K to Ls) relates to
    (documented at ph_rightdomain):
    (- RelationTest({-pointer-to:R}, RELS_LIST, {-pointer-to-new:list of L}, RLIST_ALL_Y) -). [2]

To decide which list of Ks is list of (name of kind of value K) that/which/who
    relate to (Y - L) by (R - relation of Ks to values of kind L)
    (documented at ph_leftlookuplist):
    (- RelationTest({-pointer-to:R}, RELS_LOOKUP_ALL_X, {Y}, {-pointer-to-new:list of K}) -).

To decide which list of Ls is list of (name of kind of value L) to which/whom (X - K)
    relates by (R - relation of values of kind K to Ls)
    (documented at ph_rightlookuplist):
    (- RelationTest({-pointer-to:R}, RELS_LOOKUP_ALL_Y, {X}, {-pointer-to-new:list of L}) -). [1]

To decide which list of Ls is list of (name of kind of value L)
    that/which/whom (X - K) relates to by (R - relation of values of kind K to Ls)
    (documented at ph_rightlookuplist):
    (- RelationTest({-pointer-to:R}, RELS_LOOKUP_ALL_Y, {X}, {-pointer-to-new:list of L}) -). [2]

To decide whether (name of kind of value K) relates to (Y - L) by
    (R - relation of Ks to values of kind L)
    (documented at ph_ifright):
    (- RelationTest({-pointer-to:R}, RELS_LOOKUP_ANY, {Y}, RLANY_CAN_GET_X) -).

To decide whether (X - K) relates to (name of kind of value L) by
    (R - relation of values of kind K to Ls)
    (documented at ph_ifleft):
    (- RelationTest({-pointer-to:R}, RELS_LOOKUP_ANY, {X}, RLANY_CAN_GET_Y) -).

To decide which K is (name of kind of value K) that/which/who relates to
    (Y - L) by (R - relation of Ks to values of kind L)
    (documented at ph_leftlookup):
```

```
    (- RelationTest({-pointer-to:R}, RELS_LOOKUP_ANY, {Y}, RLANY_GET_X) -).
To decide which L is (name of kind of value L) to which/whom (X - K)
    relates by (R - relation of values of kind K to Ls)
    (documented at ph_rightlookup):
    (- RelationTest({-pointer-to:R}, RELS_LOOKUP_ANY, {X}, RLANY_GET_Y) -). [1]
To decide which L is (name of kind of value L) that/which/whom (X - K)
    relates to by (R - relation of values of kind K to Ls)
    (documented at ph_rightlookup):
    (- RelationTest({-pointer-to:R}, RELS_LOOKUP_ANY, {X}, RLANY_GET_Y) -). [2]
```

§**41.** The standard map and reduce operations found in most functional programming languages:

```
Section SR5/2/20 - Values - Functional programming
To decide whether (val - K) matches (desc - description of values of kind K)
    (documented at ph_valuematch):
    (- {-description-application} -).
To decide what K is (function - phrase nothing -> value of kind K) applied
    (documented at ph_applied0):
    (- {-function-application} -).
To decide what L is (function - phrase value of kind K -> value of kind L)
    applied to (input - K)
    (documented at ph_applied1):
    (- {-function-application} -).
To decide what M is (function - phrase (value of kind K, value of kind L) -> value of kind M)
    applied to (input - K) and (second input - L)
    (documented at ph_applied2):
    (- {-function-application} -).
To decide what N is (function - phrase (value of kind K, value of kind L, value of kind M) -> value
  ...    of kind N)
    applied to (input - K) and (second input - L) and (third input - M)
    (documented at ph_applied3):
    (- {-function-application} -).
To apply (function - phrase nothing -> nothing)
    (documented at ph_apply0):
    (- {-function-application}; -).
To apply (function - phrase value of kind K -> nothing)
    to (input - K)
    (documented at ph_apply1):
    (- {-function-application}; -).
To apply (function - phrase (value of kind K, value of kind L) -> nothing)
    to (input - K) and (second input - L)
    (documented at ph_apply2):
    (- {-function-application}; -).
To apply (function - phrase (value of kind K, value of kind L, value of kind M) -> nothing)
    to (input - K) and (second input - L) and (third input - M)
    (documented at ph_apply3):
    (- {-function-application}; -).
To decide what list of L is (function - phrase K -> value of kind L) applied to (original list - li
  ...  st of values of kind K)
    (documented at ph_appliedlist):
```

```
    let the result be a list of Ls;
    repeat with item running through the original list:
        let the mapped item be the function applied to the item;
        add the mapped item to the result;
    decide on the result.
To decide what K is the (function - phrase (K, K) -> K) reduction of (original list - list of value
  ... s of kind K)
    (documented at ph_reduction):
    let the total be a K;
    let the count be 0;
    repeat with item running through the original list:
        increase the count by 1;
        if the count is 1, now the total is the item;
        otherwise now the total is the function applied to the total and the item;
    decide on the total.
To decide what list of K is the filter to (criterion - description of Ks) of
    (full list - list of values of kind K)
    (documented at ph_filter):
    let the filtered list be a list of K;
    repeat with item running through the full list:
        if the item matches the criterion:
            add the item to the filtered list;
    decide on the filtered list.
To showme (V - value)
    (documented at ph_showme):
    (- {-show-me} -).
To decide what K is the default value of (V - name of kind of value of kind K)
    (documented at ph_defaultvalue):
    (- {-default-value-for:V} -).
```

§42. Using external resources. The following all refer to “FileIO.i6t” and work only on Glulx.

```
Section SR5/2/21 - Values - Files (for Glulx external files language element only)

To read (filename - external file) into (T - table name)
    (documented at ph_readtable):
    (- FileIO_GetTable({filename}, {T}); -).
To write (filename - external file) from (T - table name)
    (documented at ph_writetable):
    (- FileIO_PutTable({filename}, {T}); -).
To decide if (filename - external file) exists
    (documented at ph_fileexists):
    (- (FileIO_Exists({filename}, false)) -).
To decide if ready to read (filename - external file)
    (documented at ph_fileready):
    (- (FileIO_Ready({filename}, false)) -).
To mark (filename - external file) as ready to read
    (documented at ph_markfileready):
    (- FileIO_MarkReady({filename}, true); -).
To mark (filename - external file) as not ready to read
    (documented at ph_markfilenotready):
    (- FileIO_MarkReady({filename}, false); -).
To write (T - text) to (FN - external file)
    (documented at ph_writetext):
    (- FileIO_PutContents({FN}, {-allow-stack-frame-access:T}, false); -).
To append (T - text) to (FN - external file)
    (documented at ph_appendtext):
    (- FileIO_PutContents({FN}, {-allow-stack-frame-access:T}, true); -).
To say text of (FN - external file)
    (documented at ph_saytext):
    (- FileIO_PrintContents({FN}); say__p = 1; -).
```

§43. Figures and sound effects. Ditto, but for “Figures.i6t”.

```
Section SR5/2/22 - Values - Figures (for figures language element only)

To display (F - figure name), one time only
    (documented at ph_displayfigure):
    (- DisplayFigure(ResourceIDsOfFigures-->{F}, {phrase options}); -).
To decide which number is the Glulx resource ID of (F - figure name)
    (documented at ph_figureid):
    (- ResourceIDsOfFigures-->{F} -).

Section SR5/2/23 - Values - Sound effects (for sounds language element only)

To play (SFX - sound name), one time only
    (documented at ph_playsf):
    (- PlaySound(ResourceIDsOfSounds-->{SFX}, {phrase options}); -).
To decide which number is the Glulx resource ID of (SFX - sound name)
    (documented at ph_soundid):
    (- ResourceIDsOfSounds-->{SFX} -).
```

§**44. Control phrases.** While “unless” is supposed to be exactly like “if” but with the reversed sense of the condition, that isn’t quite true. For example, there is no “unless ... then ...”: logical it might be, English it is not.

```
Section SR5/3/1 - Control phrases - If and unless

To if (c - condition) , (ph - phrase)
    (documented at ph_if):
    (- if {c} {ph} -).
To if (c - condition) begin -- end
    (documented at ph_if):
    (- if {c}  -).
To if (c - condition) then (ph - phrase)
    (deprecated)
    (documented at ph_if_dep):
    (- if {c} {ph} -).
To unless (c - condition) , (ph - phrase)
    (documented at ph_unless):
    (- if (~~{c}) {ph} -).
To unless (c - condition) begin -- end
    (documented at ph_unless):
    (- if (~~{c})  -).
```

§**45.** It looks as if the definitions below could be abbreviated a little by making more use of the slash (“To else/otherwise if...”, say), but in fact the slash isn’t compatible with these built-in control structure definitions – or at any rate using it causes a handful of problem messages from the type-checker to be worded badly in some cases.

```
To otherwise if (c - condition)
    (documented at ph_otherwise):
    (- } else if {c} { -).
To otherwise unless (c - condition)
    (documented at ph_otherwise):
    (- } else if (~~{c}) { -).
To otherwise (ph - phrase)
    (documented at ph_otherwise):
    (- else {ph} -).
To else if (c - condition)
    (documented at ph_otherwise):
    (- } else if {c} { -).
To else unless (c - condition)
    (documented at ph_otherwise):
    (- } else if (~~{c}) { -).
To else (ph - phrase)
    (documented at ph_otherwise):
    (- else {ph} -).
```

§**46.** The switch form of "if" is subtly different, and here again "unless" is not allowed in its place.

The begin and end markers here are in a sense bogus, in that the end user isn't supposed to type them: they are inserted automatically in the sentence subtree maker, which converts indentation into block structure.

```
To if (V - word value) is begin -- end
    (documented at ph_switch):
    (- switch({V})  -).
```

§**47.** After all that, the while loop is simplicity itself. Perhaps the presence of "unless" for "if" argues for a similarly negated form, "until" for "while", but users haven't yet petitioned for this.

```
Section SR5/3/2 - Control phrases - While

To while (c - condition) repeatedly (ph - phrase)
    (deprecated)
    (documented at ph_while_dep):
    (- while {c} {ph} -).
To while (c - condition) , (ph - phrase)
    (deprecated)
    (documented at ph_while_dep):
    (- while {c} {ph} -).
To while (c - condition) begin -- end
    (documented at ph_while):
    (- while {c}  -).
```

§**48.** The repeat loop looks like a single construction, but isn't, because the range can be given in four fundamentally different ways (and the loop variable then has a different kind of value accordingly). First, the equivalents of BASIC's `for` loop and of Inform 6's `objectloop`, respectively:

```
Section SR5/3/3 - Control phrases - Repeat

To repeat with (loopvar - nonexisting K variable)
    running from (v - arithmetic value of kind K) to (w - K) begin -- end
    (documented at ph_repeat):
        (- for ({loopvar}={v}: {loopvar}<={w}: {loopvar}++)  -).
To repeat with (loopvar - nonexisting K variable)
    running from (v - enumerated value of kind K) to (w - K) begin -- end
    (documented at ph_repeat):
        (- for ({loopvar}={v}: {loopvar}<={w}: {loopvar}++)  -).
To repeat with (loopvar - nonexisting K variable)
    running through (OS - description of values of kind K) begin -- end
    (documented at ph_runthrough):
        (- {-loop-over:OS}  -).
To repeat with (loopvar - nonexisting object variable)
    running through (L - list of values) begin -- end
    (documented at ph_repeatlist):
    (- {-loop-over-list:L}  -).
```

§**49.** The following are all repeats where the range is the set of rows of a table, taken in some order, and the repeat variable – though it does exist – is never specified since the relevant row is instead the one selected during each iteration of the loop.

```
To repeat through (T - table name) begin -- end
    (documented at ph_repeattable):
    (- {-push-ctvs}
        for ({-ct-v0}={T},{-ct-v1}=1,ct_0={-ct-v0},ct_1={-ct-v1}:
            {-ct-v1}<=TableRows({-ct-v0}):{-ct-v1}++,ct_0={-ct-v0},ct_1={-ct-v1})
            if (TableRowIsBlank(ct_0,ct_1)==false)  -).
To repeat through (T - table name) in reverse order begin -- end
    (documented at ph_repeattablereverse):
    (- {-push-ctvs}
        for ({-ct-v0}={T},{-ct-v1}=TableRows({-ct-v0}),ct_0={-ct-v0},ct_1={-ct-v1}:
            {-ct-v1}>=1:{-ct-v1}--,ct_0={-ct-v0},ct_1={-ct-v1})
            if (TableRowIsBlank(ct_0,ct_1)==false)  -).
To repeat through (T - table name) in (TC - table column) order begin -- end
    (documented at ph_repeattablecol):
    (- {-push-ctvs}
        for ({-ct-v0}={T},{-ct-v1}=TableNextRow({-ct-v0},{TC},0,1),ct_0={-ct-v0},ct_1={-ct-v1}:
            {-ct-v1}~=0:
            {-ct-v1}=TableNextRow({-ct-v0},{TC},{-ct-v1},1),ct_0={-ct-v0},ct_1={-ct-v1})  -).
To repeat through (T - table name) in reverse (TC - table column) order begin -- end
    (documented at ph_repeattablecolreverse):
    (- {-push-ctvs}
        for ({-ct-v0}={T},{-ct-v1}=TableNextRow({-ct-v0},{TC},0,-1),ct_0={-ct-v0},ct_1={-ct-v1}:
            {-ct-v1}~=0:
            {-ct-v1}=TableNextRow({-ct-v0},{TC},{-ct-v1},-1),ct_0={-ct-v0},ct_1={-ct-v1})  -).
```

§**50.** The equivalent of `break` or `continue` in C or I6, or of `last` or `next` in Perl. Here “in loop” means “in any of the forms of while or repeat”.

```
Section SR5/3/6 - Control phrases - Changing the flow of loops

To break -- in loop
    (documented at ph_break):
    (- break; -).
To next -- in loop
    (documented at ph_next):
    (- continue; -).
```

§**51.** The following certainly aren't loops and aren't quite conditionals either, but they reflect the use of rules within rulebooks as being a form of control structure, and they have to be indexed somewhere.

The antique forms "yes" and "no" are now somewhat to be regretted, with "decide yes" and "decide no" being clearer ways to write the same thing. But we seem to be stuck with them.

```
Section SR5/3/7 - Control phrases - Deciding outcomes

To yes
    (documented at ph_yes):
    (- rtrue; -) - in to decide if only.
To decide yes
    (documented at ph_yes):
    (- rtrue; -) - in to decide if only.
To no
    (documented at ph_no):
    (- rfalse; -) - in to decide if only.
To decide no
    (documented at ph_no):
    (- rfalse; -) - in to decide if only.
```

§**52.** Note that returning a value has to invoke the type-checker to ensure that the return value matches the kind of value expected. This certainly rejects the phrase if it's used in a definition which isn't meant to be deciding a value at all, so an "in... only" clause is not needed.

```
To decide on (something - value)
    (documented at ph_decideon):
    (- return {-check-return-type:something}; -).
```

§**53.** "Do nothing" is useful mainly when other syntax has backed us into something clumsy, but it can't be dispensed with. (In the examples, it used to be used when conditions were awkward to negate – if condition, do nothing, otherwise blah blah blah – but the creation of "unless" made it possible to remove most of the "do nothing"s.)

```
Section SR5/3/8 - Control phrases - Stop or go

To do nothing (documented at ph_nothing):
    (- ; -).
To stop (documented at ph_stop):
    (- return; -) - in to only.
```

§**54. Actions, activities and rules.** We begin with the firing off of new actions. The current action runs silently if the I6 global variable `keep_silent` is set, so the result of the definitions below is that one can go into silence mode, using "try silently", but not climb out of it again. This is done because many actions try other actions as part of their normal workings: if we want action X to be tried silently, then any action X itself tries should also be tried silently.

```
Section SR5/4/1 - Actions, activities and rules - Trying actions

To try (doing something - action)
    (documented at ph_try):
    (- {doing something}; -).
To silently try (doing something - action)
    (documented at ph_trysilently):
    (- @push keep_silent; keep_silent=1; {doing something}; @pull keep_silent; -).
To try silently (doing something - action)
    (documented at ph_trysilently):
    (- @push keep_silent; keep_silent=1; {doing something}; @pull keep_silent; -).
```

§**55.** The requirements of the current action can be tested. The following may be reimplemented using a verb *to require* at some future point.

```
Section SR5/4/2 - Actions, activities and rules - Action requirements

To decide whether the action requires a touchable noun
    (documented at ph_requirestouch):
    (- (NeedToTouchNoun()) -).
To decide whether the action requires a touchable second noun
    (documented at ph_requirestouch2):
    (- (NeedToTouchSecondNoun()) -).
To decide whether the action requires a carried noun
    (documented at ph_requirescarried):
    (- (NeedToCarryNoun()) -).
To decide whether the action requires a carried second noun
    (documented at ph_requirescarried2):
    (- (NeedToCarrySecondNoun()) -).
To decide whether the action requires light
    (documented at ph_requireslight):
    (- (NeedLightForAction()) -).
```

§**56.** Within the rulebooks to do with an action, returning `true` from a rule is sufficient to stop the rulebook early: there is no need to specify success or failure because that is determined by the rulebook itself. (For instance, if the check taking rules stop for any reason, the action failed; if the after rules stop, it succeeded.) In some rulebooks, notably "instead" and "after", the default is to stop, so that execution reaching the end of the I6 routine for a rule will run into an `rtrue`. "Continue the action" prevents this.

```
Section SR5/4/3 - Actions, activities and rules - Stop or continue

To stop the action
    (documented at ph_stopaction):
    (- rtrue; -) - in to only.
To continue the action
    (documented at ph_continueaction):
    (- rfalse; -) - in to only.
```

§**57.** Note that we define “try”, “silently try” and “try silently” all over again here, but for stored actions rather than actions: NI’s type-checking means that it will automatically use whichever definition is appropriate.

```
Section SR5/4/4 - Actions, activities and rules - Stored actions

To decide what stored action is the current action
    (documented at ph_currentaction):
    (- STORED_ACTION_TY_Current({-pointer-to-new:stored action}) -).
To decide what stored action is the action of (A - action)
    (documented at ph_actionof):
    (- {A}{-backspace}{-backspace}, STORED_ACTION_TY_Current({-pointer-to-new:stored action})) -).
To try (S - stored action)
    (documented at ph_trystored):
    (- STORED_ACTION_TY_Try({S}); -).
To silently try (S - stored action)
    (documented at ph_trystoredsilently):
    (- STORED_ACTION_TY_Try({S}, true); -).
To try silently (S - stored action)
    (documented at ph_trystoredsilently):
    (- STORED_ACTION_TY_Try({S}, true); -).
To decide if (act - a stored action) involves (X - an object)
    (documented at ph_involves):
    (- (STORED_ACTION_TY_Involves({-pointer-to:act}, {X})) -).
To decide what action name is the action name part of (act - a stored action)
    (documented at ph_actionpart):
    (- (STORED_ACTION_TY_Part({-pointer-to:act}, 0)) -).
To decide what object is the noun part of (act - a stored action)
    (documented at ph_nounpart):
    (- (STORED_ACTION_TY_Part({-pointer-to:act}, 1)) -).
To decide what object is the second noun part of (act - a stored action)
    (documented at ph_secondpart):
    (- (STORED_ACTION_TY_Part({-pointer-to:act}, 2)) -).
To decide what object is the actor part of (act - a stored action)
    (documented at ph_actorpart):
    (- (STORED_ACTION_TY_Part({-pointer-to:act}, 3)) -).
```

§**58.** Firing off activities:

```
Section SR5/4/5 - Actions, activities and rules - Carrying out activities

To carry out the (A - activity on nothing) activity
    (documented at ph_carryout):
    (- CarryOutActivity({A}); -).
To carry out the (A - activity on value of kind K) activity with (val - K)
    (documented at ph_carryoutwith):
    (- CarryOutActivity({A}, {val}); -).
```

§**59.** This is analogous to “continue the action”:

```
To continue the activity
    (documented at ph_continueactivity):
    (- rfalse; -) - in to only.
```

§60. Advanced activity phrases: for setting up one's own activities structured around I7 source text. People tend not to use this much, and perhaps that's a good thing, but it does open up possibilities, and it's good for retro-fitting onto extensions to make the more customisable.

```
Section SR5/4/6 - Actions, activities and rules - Advanced activities
To begin the (A - activity on nothing) activity
    (documented at ph_beginactivity):
    (- BeginActivity({A}); -).
To begin the (A - activity on value of kind K) activity with (val - K)
    (documented at ph_beginactivitywith):
    (- BeginActivity({A}, {val}); -).
To decide whether handling (A - activity) activity
    (documented at ph_handlingactivity):
    (- (~~(ForActivity({A}))) -).
To decide whether handling (A - activity on value of kind K) activity with (val - K)
    (documented at ph_handlingactivitywith):
    (- (~~(ForActivity({A}, {val}))) -).
To end the (A - activity on nothing) activity
    (documented at ph_endactivity):
    (- EndActivity({A}); -).
To end the (A - activity on value of kind K) activity with (val - K)
    (documented at ph_endactivitywith):
    (- EndActivity({A}, {val}); -).
To abandon the (A - activity on nothing) activity
    (documented at ph_abandonactivity):
    (- AbandonActivity({A}); -).
To abandon the (A - activity on value of kind K) activity with (val - K)
    (documented at ph_abandonactivitywith):
    (- AbandonActivity({A}, {val}); -).
```

§61. Rules. Four different ways to invoke a rule or rulebook:

```
Section SR5/4/7 - Actions, activities and rules - Following rules

To follow (RL - a rule)
    (documented at ph_follow):
    (- FollowRulebook({RL}); -).
To follow (RL - value of kind K based rule producing a value) for (V - K)
    (documented at ph_followfor):
    (- FollowRulebook({RL}, {V}, true); -).
To consider (RL - a rule)
    (documented at ph_consider):
    (- ProcessRulebook({RL}); -).
To consider (RL - value of kind K based rule producing a value) for (V - K)
    (documented at ph_considerfor):
    (- ProcessRulebook({RL}, {V}, true); -).
To decide what K is the (name of kind K) produced by (RL - rule producing a value of kind K)
    (documented at ph_producedby):
    (- ResultOfRule({RL}, 0, false, {-strong-kind:K}) -).
To decide what L is the (name of kind L) produced by (RL - value of kind K based rule
    producing a value of kind L) for (V - K)
    (documented at ph_producedbyfor):
    (- ResultOfRule({RL}, {V}, true, {-strong-kind:L}) -).
To abide by (RL - a rule)
    (documented at ph_abide):
    (- if (ProcessRulebook({RL})) rtrue; -) - in to only.
To abide by (RL - value of kind K based rule producing a value) for (V - K)
    (documented at ph_abidefor):
    (- if (ProcessRulebook({RL}, {V}, true)) rtrue; -) - in to only.
To anonymously abide by (RL - a rule)
    (documented at ph_abideanon):
    (- if (temporary_value = ProcessRulebook({RL})) {
        if (RulebookSucceeded()) ActRulebookSucceeds(temporary_value);
        else ActRulebookFails(temporary_value);
        return 2;
    } -) - in to only.
To anonymously abide by (RL - value of kind K based rule producing a value) for (V - K)
    (documented at ph_abideanon):
    (- if (temporary_value = ProcessRulebook({RL}, {V}, true)) {
        if (RulebookSucceeded()) ActRulebookSucceeds(temporary_value);
        else ActRulebookFails(temporary_value);
        return 2;
    } -) - in to only.
```

§**62.** Rules return `true` to indicate a decision, which could be either a success or a failure, and optionally may also return a value. If they return `false`, there's no decision.

```
Section SR5/4/8 - Actions, activities and rules - Success and failure

To make no decision
    (documented at ph_nodecision): (- rfalse; -) - in to only.
To rule succeeds
    (documented at ph_succeeds):
    (- RulebookSucceeds(); rtrue; -) - in to only.
To rule fails
    (documented at ph_fails):
    (- RulebookFails(); rtrue; -) - in to only.
To rule succeeds with result (val - a value)
    (documented at ph_succeedswith):
    (- RulebookSucceeds({-weak-kind-to-be-produced:val},{-check-success-type:val}); rtrue; -) - in to
o
  ... nly.
To decide if rule succeeded
    (documented at ph_succeeded):
    (- (RulebookSucceeded()) -).
To decide if rule failed
    (documented at ph_failed):
    (- (RulebookFailed()) -).
To decide which rulebook outcome is the outcome of the rulebook
    (documented at ph_rulebookoutcome):
    (- (ResultOfRule()) -).
```

§**63.** And lastly the suite of procedural rule tricks, though happily these have been less and less needed as Inform has grown in flexibility. (They incur an appreciable speed penalty at run-time.)

```
Section SR5/4/9 - Actions, activities and rules - Procedural manipulation

To ignore (RL - a rule)
    (deprecated)
    (documented at ph_ignore_dep):
    (- SuppressRule({RL}); -).
To reinstate (RL - a rule)
    (deprecated)
    (documented at ph_reinstate_dep):
    (- ReinstateRule({RL}); -).
To reject the result of (RL - a rule)
    (deprecated)
    (documented at ph_reject_dep):
    (- DonotuseRule({RL}); -).
To accept the result of (RL - a rule)
    (deprecated)
    (documented at ph_accept_dep):
    (- DonotuseRule({RL}); -).
To substitute (RL1 - a rule) for (RL2 - a rule)
    (deprecated)
    (documented at ph_substitute_dep):
    (- SubstituteRule({RL1},{RL2}); -).
To restore the original (RL1 - a rule)
```

```
    (deprecated)
    (documented at ph_restore_dep):
    (- SubstituteRule({RL1},{RL1}); -).
To move (RL1 - a rule) to before (RL2 - a rule)
    (deprecated)
    (documented at ph_movebefore_dep):
    (- MoveRuleBefore({RL1},{RL2}); -).
To move (RL1 - a rule) to after (RL2 - a rule)
    (deprecated)
    (documented at ph_moveafter_dep):
    (- MoveRuleAfter({RL1},{RL2}); -).
```

§**64. The model world.** Phrase definitions with wordings like “the story has ended” are a necessary evil. The “has” here is parsed literally, not as the present tense of *to have*, so inflected forms like “the story had ended” are not available: nor is there any value “the story” for the subject noun phrase to hold... and so on. Ideally, we would word all conditional phrases so as to avoid the verbs, but natural language just doesn't work that way.

```
Section SR5/5/1 - Model world - Ending the story

To end the story
    (documented at ph_end):
    (- deadflag=3; story_complete=false; -).
To end the story finally
    (documented at ph_endfinally):
    (- deadflag=3; story_complete=true; -).
To end the story saying (finale - text)
    (documented at ph_endsaying):
    (- deadflag={finale}; story_complete=false; -).
To end the story finally saying (finale - text)
    (documented at ph_endfinallysaying):
    (- deadflag={finale}; story_complete=true; -).
To decide whether the story has ended
    (documented at ph_ended):
    (- (deadflag~=0) -).
To decide whether the story has ended finally
    (documented at ph_finallyended):
    (- (story_complete) -).
To decide whether the story has not ended
    (documented at ph_notended):
    (- (deadflag==0) -).
To decide whether the story has not ended finally
    (documented at ph_notfinallyended):
    (- (story_complete==false) -).
To resume the story
    (documented at ph_resume):
    (- resurrect_please = true; -).

To end the game in death (deprecated)
    (documented at ph_enddeath_dep):
    (- deadflag=1; story_complete=false; -).
To end the game in victory (deprecated)
    (documented at ph_endvictory_dep):
    (- deadflag=2; story_complete=true; -).
To end the game saying (finale - text) (deprecated)
    (documented at ph_endgamesaying_dep):
    (- deadflag={finale}; story_complete=false; -).
To resume the game (deprecated)
    (documented at ph_resumegame_dep):
    (- resurrect_please = true; -).
To decide whether the game is in progress (deprecated)
    (documented at ph_inprogress_dep):
    (- (deadflag==0) -).
To decide whether the game is over (deprecated)
    (documented at ph_over_dep):
    (- (deadflag~=0) -).
To decide whether the game ended in death (deprecated)
```

```
    (documented at ph_endeddeath_dep):
    (- (deadflag==1) -).
To decide whether the game ended in victory (deprecated)
    (documented at ph_endedvictory_dep):
    (- (deadflag==2) -).
```

§65. Times of day.

```
Section SR5/5/2 - Model world - Times of day

To decide which number is the minutes part of (t - time)
    (documented at ph_minspart):
    (- ({t}%ONE_HOUR) -).
To decide which number is the hours part of (t - time)
    (documented at ph_hourspart):
    (- ({t}/ONE_HOUR) -).
```

§66. Comparing times of day is inherently odd, because the day is circular. Every 2 PM comes after a 1 PM, but it also comes before another 1 PM. Where do we draw the meridian on this circle? The legal day divides at midnight but for other purposes (daylight savings time, for instance) society often chooses 2 AM as the boundary. Inform uses 4 AM instead as the least probable time through which play continues. (Modulo a 24-hour clock, adding 20 hours is equivalent to subtracting 4 AM from the current time: hence the use of `20*ONE_HOUR` below.) Thus 3:59 AM is after 4:00 AM, the former being at the very end of a day, the latter at the very beginning.

```
To decide if (t - time) is before (t2 - time)
    (documented at ph_timebefore):
    (- ((({t}+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))<(({t2}+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))) -).
To decide if (t - time) is after (t2 - time)
    (documented at ph_timeafter):
    (- ((({t}+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))>(({t2}+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))) -).
To decide if it is before (t2 - time) (deprecated)
    (documented at ph_itisbefore_dep):
    (- (((the_time+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))<(({t2}+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))) -).
To decide if it is after (t2 - time) (deprecated)
    (documented at ph_itisafter_dep):
    (- (((the_time+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))>(({t2}+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))) -).
To decide which time is (t - time) before (t2 - time)
    (documented at ph_shiftbefore):
    (- (({t2}-{t}+TWENTY_FOUR_HOURS)%(TWENTY_FOUR_HOURS)) -).
To decide which time is (t - time) after (t2 - time)
    (documented at ph_shiftafter):
    (- (({t2}+{t}+TWENTY_FOUR_HOURS)%(TWENTY_FOUR_HOURS)) -).
```

§67. Durations are in effect casts from "number" to "time".

```
Section SR5/5/3 - Model world - Durations

To decide which time is (n - number) minutes
    (documented at ph_durationmins):
    (- (({n})%(TWENTY_FOUR_HOURS)) -).
To decide which time is (n - number) hours
    (documented at ph_durationhours):
    (- (({n}*ONE_HOUR)%(TWENTY_FOUR_HOURS)) -).
```

§68. Timed events.

```
Section SR5/5/4 - Model world - Timed events

To (R - rule) in (t - number) turn/turns from now
    (documented at ph_turnsfromnow):
    (- SetTimedEvent({-mark-event-used:R}, {t}+1, 0); -).
To (R - rule) at (t - time)
    (documented at ph_attime):
    (- SetTimedEvent({-mark-event-used:R}, {t}, 1); -).
To (R - rule) in (t - time) from now
    (documented at ph_timefromnow):
    (- SetTimedEvent({-mark-event-used:R}, (the_time+{t})%(TWENTY_FOUR_HOURS), 1); -).
```

§69. Scenes.

```
Section SR5/5/5 - Model world - Scenes

To decide if (sc - scene) has happened
    (documented at ph_hashappened):
    (- (scene_endings-->({sc}-1)) -).
To decide if (sc - scene) has not happened
    (documented at ph_hasnothappened):
    (- (scene_endings-->({sc}-1) == 0) -).
To decide if (sc - scene) has ended
    (documented at ph_hasended):
    (- (scene_endings-->({sc}-1) > 1) -).
To decide if (sc - scene) has not ended
    (documented at ph_hasnotended):
    (- (scene_endings-->({sc}-1) <= 1) -).
```

§70. Timing of scenes.

```
Section SR5/5/6 - Model world - Timing of scenes

To decide which time is the time since (sc - scene) began
    (documented at ph_scenetimesincebegan):
    (- (SceneUtility({sc}, 1)) -).
To decide which time is the time when (sc - scene) began
    (documented at ph_scenetimewhenbegan):
    (- (SceneUtility({sc}, 2)) -).
To decide which time is the time since (sc - scene) ended
    (documented at ph_scenetimesinceended):
    (- (SceneUtility({sc}, 3)) -).
To decide which time is the time when (sc - scene) ended
    (documented at ph_scenetimewhenended):
    (- (SceneUtility({sc}, 4)) -).
```

§71. Player's identity and location.

```
Section SR5/5/7 - Model world - Player's identity and location

To change the/-- player to (O - an object)
    (deprecated)
    (documented at ph_changeplayer_dep):
    (- ChangePlayer({O}); -).
To decide whether in (somewhere - an object)
    (deprecated)
    (documented at ph_in_dep):
    (- (WhetherIn({somewhere})) -).
To decide whether in darkness
    (documented at ph_indarkness):
    (- (location==thedark) -).
```

§72. Moving and removing things.

```
Section SR5/5/8 - Model world - Moving and removing things

To move (something - object) to (something else - object),
    without printing a room description
    or printing an abbreviated room description
    (documented at ph_move):
    (- MoveObject({something}, {something else}, {phrase options}, false); -).
To remove (something - object) from play
    (documented at ph_remove):
    (- RemoveFromPlay({something}); -).
To move (O - object) backdrop to all (D - description of objects)
    (documented at ph_movebackdrop):
    (- MoveBackdrop({O}, {D}); -).
To update backdrop positions
    (documented at ph_updatebackdrop):
    (- MoveFloatingObjects(); -).
```

§73. The map.

```
Section SR5/5/9 - Model world - The map

To decide which room is location of (O - object)
    (documented at ph_locationof):
    (- LocationOf({O}) -).
To decide which room is room (D - direction) from/of (R1 - room)
    (documented at ph_roomdirof):
    (- MapConnection({R1},{D}) -).
To decide which door is door (D - direction) from/of (R1 - room)
    (documented at ph_doordirof):
    (- DoorFrom({R1},{D}) -).
To decide which object is the other side of (D - door) from (R1 - room)
    (documented at ph_othersideof):
    (- OtherSideOfDoor({D},{R1}) -).
To decide which object is the direction of (D - door) from (R1 - room)
    (documented at ph_directionofdoor):
    (- DirectionDoorLeadsIn({D},{R1}) -).
To decide which object is room-or-door (D - direction) from/of (R1 - room)
    (documented at ph_roomordoor):
    (- RoomOrDoorFrom({R1},{D}) -).
To change (D - direction) exit of (R1 - room) to (R2 - room)
    (documented at ph_changeexit):
    (- AssertMapConnection({R1},{D},{R2}); -).
To change (D - direction) exit of (R1 - room) to nothing/nowhere
    (documented at ph_changenoexit):
    (- AssertMapConnection({R1},{D},nothing); -).
To decide which room is the front side of (D - object)
    (documented at ph_frontside):
    (- FrontSideOfDoor({D}) -).
To decide which room is the back side of (D - object)
    (documented at ph_backside):
    (- BackSideOfDoor({D}) -).
```

§74. Route-finding.

```
Section SR5/5/10 - Model world - Route-finding

To decide which object is best route from (R1 - object) to (R2 - object),
    using doors or using even locked doors
    (documented at ph_bestroute):
    (- MapRouteTo({R1},{R2},0,{phrase options}) -).
To decide which number is number of moves from (R1 - object) to (R2 - object),
    using doors or using even locked doors
    (documented at ph_bestroutelength):
    (- MapRouteTo({R1},{R2},0,{phrase options},true) -).
To decide which object is best route from (R1 - object) to (R2 - object) through
    (RS - description of objects),
    using doors or using even locked doors
    (documented at ph_bestroutethrough):
    (- MapRouteTo({R1},{R2},{RS},{phrase options}) -).
To decide which number is number of moves from (R1 - object) to (R2 - object) through
    (RS - description of objects),
    using doors or using even locked doors
    (documented at ph_bestroutethroughlength):
    (- MapRouteTo({R1},{R2},{RS},{phrase options},true) -).
```

§**75.** The object tree.

```
Section SR5/5/11 - Model world - The object tree

To decide which object is holder of (something - object)
    (documented at ph_holder):
    (- (HolderOf({something})) -).
To decide which object is next thing held after (something - object)
    (documented at ph_nextheld):
    (- (sibling({something})) -).
To decide which object is first thing held by (something - object)
    (documented at ph_firstheld):
    (- (child({something})) -).
```

§**76.** Last, and since it's deprecated, least:

```
Section SR5/5/12 - Model world - Score

To award (some - number) point/points
    (deprecated)
    (documented at ph_awardpoints_dep):
    (- score=score+{some}; -).
```

§**77. Understanding.** First, asking yes/no questions.

```
Section SR5/6/1 - Understanding - Asking yes/no questions

To decide whether player consents
    (documented at ph_consents):
        (- YesOrNo() -).
```

§**78.** Support for snippets, which are substrings of the player's command.

```
Section SR5/6/2 - Understanding - The player's command

To decide if (S - a snippet) matches (T - a topic)
    (documented at ph_snippetmatches):
    (- (SnippetMatches({S}, {T})) -).
To decide if (S - a snippet) does not match (T - a topic)
    (documented at ph_snippetdoesnotmatch):
    (- (SnippetMatches({S}, {T}) == false) -).
To decide if (S - a snippet) includes (T - a topic)
    (documented at ph_snippetincludes):
    (- (matched_text=SnippetIncludes({T},{S})) -).
To decide if (S - a snippet) does not include (T - a topic)
    (documented at ph_snippetdoesnotinclude):
    (- (SnippetIncludes({T},{S})==0) -).
```

§**79.** Changing the player's command.

```
Section SR5/6/3 - Understanding - Changing the player's command

To change the text of the player's command to (txb - indexed text)
    (documented at ph_changecommand):
    (- SetPlayersCommand({-pointer-to:txb}); -).
To replace (S - a snippet) with (T - text)
    (documented at ph_replacesnippet):
    (- SpliceSnippet({S}, {T}); -).
To cut (S - a snippet)
    (documented at ph_cutsnippet):
    (- SpliceSnippet({S}, 0); -).
To reject the player's command
    (documented at ph_rejectcommand):
    (- RulebookFails(); rtrue; -) - in to only.
```

§**80.** Scope and pronouns.

```
Section SR5/6/4 - Understanding - Scope and pronouns
To place (O - an object) in scope, but not its contents
    (documented at ph_placeinscope):
    (- PlaceInScope({O}, {phrase options}); -).
To place the/-- contents of (O - an object) in scope
    (documented at ph_placecontentsinscope):
    (- ScopeWithin({O}); -).
To set pronouns from (O - an object)
    (documented at ph_setpronouns):
    (- PronounNotice({O}); -).
To set pronouns from possessions of the player
    (deprecated)
    (documented at ph_setpronouns_dep):
    (- PronounNoticeHeldObjects(); -).
```

§**81. Message support.** "Unindexed" here is a euphemism for "undocumented". This is where experimental or intermediate phrases go: things we don't want people to use because we will probably revise them heavily in later builds of Inform. For now, the Standard Rules do make use of these phrases, but nobody else should. They will change without comment in the change log.

```
Section SR5/8/1 - Message support - Issuance - Unindexed

To stop the action with library message (AN - an action name) number
    (N - a number) for (H - an object):
    (- return GL__M({AN},{N},{H}); -) - in to only.
To stop the action with library message (AN - an action name) number
    (N - a number):
    (- return GL__M({AN},{N},noun); -) - in to only.
To issue miscellaneous library message number (N - a number):
    (- GL__M(##Miscellany,{N}); -).
To issue miscellaneous library message number (N - a number) for (H - an object):
    (- GL__M(##Miscellany,{N}, {H}); -).
To issue library message (AN - an action name) number
    (N - a number) for (H - an object):
    (- GL__M({AN},{N},{H}); -).
To issue library message (AN - an action name) number
    (N - a number) for (H - an object) and (H2 - an object):
    (- GL__M({AN},{N},{H},{H2}); -).
To issue library message (AN - an action name) number (N - a number):
    (- GL__M({AN},{N},noun); -).
To issue actor-based library message (AN - an action name) number
    (N - a number) for (H - an object) and (H2 - an object):
    (- AGL__M({AN},{N},{H},{H2}); -).
To issue actor-based library message (AN - an action name) number
    (N - a number) for (H - an object):
    (- AGL__M({AN},{N},{H}); -).
To issue actor-based library message (AN - an action name) number (N - a number):
    (- AGL__M({AN},{N},noun); -).

To issue score notification message:
    (- NotifyTheScore(); -).
To say pronoun dictionary word:
    (- print (address) pronoun_word; -).
To say recap of command:
    (- PrintCommand(); -).
The pronoun reference object is an object that varies.
The pronoun reference object variable translates into I6 as "pronoun_obj".
The library message action is an action name that varies.
The library message action variable translates into I6 as "lm_act".
The library message number is a number that varies.
The library message number variable translates into I6 as "lm_n".
The library message amount is a number that varies.
The library message amount variable translates into I6 as "lm_o".
The library message object is an object that varies.
The library message object variable translates into I6 as "lm_o".
The library message actor is an object that varies.
The library message actor variable translates into I6 as "actor".
The second library message object is an object that varies.
The second library message object variable translates into I6 as "lm_o2".
```

§**82.** Intervention. These are hooks for the new library messages system coming in a later build; they won't last.

```
Section SR5/8/2 - Message support - Intervention - Unindexed

To decide if intervened in miscellaneous message:
    decide on false;

To decide if intervened in miscellaneous list message:
    decide on false;

To decide if intervened in action message:
    decide on false;
```

§**83. Miscellaneous other phrases.** Again, *these are not part of Inform's public specification.*

```
Section SR5/9/1 - Miscellaneous other phrases - Unindexed
```

§**84.** These are actually sensible concepts in the world model, and could even be opened to public use, but they're quite complicated to explain.

```
To decide which object is the component parts core of (X - an object):
    (- CoreOf({X}) -).
To decide which object is the common ancestor of (O - an object) with
    (P - an object):
    (- (CommonAncestor({O}, {P})) -).
To decide which object is the not-counting-parts holder of (O - an object):
    (- (CoreOfParentOfCoreOf({O})) -).
To decide which object is the visibility-holder of (O - object):
    (- VisibilityParent({O}) -).
To calculate visibility ceiling at low level:
    (- FindVisibilityLevels(); -).
```

§**85.** These are in effect global variables, but aren't defined as such, to prevent people using them. Their contents are only very briefly meaningful, and they would be dangerous friends to know.

```
To decide which number is the visibility ceiling count calculated:
    (- visibility_levels -).
To decide which object is the visibility ceiling calculated:
    (- visibility_ceiling -).
```

§**86.** This is a unique quasi-action, using the secondary action processing stage only. A convenience, but also an anomaly, and let's not encourage its further use.

```
To produce a room description with going spacing conventions:
    (- LookAfterGoing(); -).
```

§**87.** An ugly little trick needed because of the mismatch between I6 and I7 property implementation, and because of legacy code from the old I6 library. Please don't touch.

```
To print the location's description:
    (- PrintOrRun(location, description); -).
```

§**88.** This is a bit trickier than it looks, because it isn't always set when one thinks it is.

```
To decide whether the I6 parser is running multiple actions:
    (- (multiflag==1) -).
```

§89. Again, the following cries out for an enumerated kind of value.

```
To decide if set to sometimes abbreviated room descriptions:
    (- (lookmode == 1) -).
To decide if set to unabbreviated room descriptions:
    (- (lookmode == 2) -).
To decide if set to abbreviated room descriptions:
    (- (lookmode == 3) -).
```

§90. Action conversion is a trick used in the Standard Rules to simplify the implementation of actions: it allows one action to become another one mid-way, without causing spurious action failures. (There are better ways to make user-defined actions convert, and some of the examples show this.)

```
To convert to (AN - an action name) on (O - an object):
    (- return GVS_Convert({AN},{O},0); -) - in to only.
To convert to request of (X - object) to perform (AN - action name) with
    (Y - object) and (Z - object):
    (- TryAction(true, {X}, {AN}, {Y}, {Z}); rtrue; -).
To convert to special going-with-push action:
    (- ConvertToGoingWithPush(); rtrue; -).
```

§91. The "surreptitiously" phrases shouldn't be used except in the Standard Rules because they temporarily violate invariants for the object tree and the light variables; the SR uses them carefully in situations where it's known to work out all right.

```
To surreptitiously move (something - object) to (something else - object):
    (- move {something} to {something else}; -).
To surreptitiously move (something - object) to (something else - object) during going:
    (- MoveDuringGoing({something}, {something else}); -).
To surreptitiously reckon darkness:
    (- SilentlyConsiderLight(); -).
```

§92. This is convenient for debugging Inform, but for no other purpose. It toggles verbose logging of the type-checker.

```
To ***:
    (- {-verbose-checking} -).
```

§93. And so, at last...

```
The Standard Rules end here.
```

§94. ...except that this is not quite true, because like most extensions they then quote some documentation for Inform to weave into index pages: though here it's more of a polite refusal than a manual, since the entire system documentation is really the description of what was defined in this extension.

```
---- DOCUMENTATION ----
Unlike other extensions, the Standard Rules are compulsorily included
with every project. They define the phrases, kinds and relations which
are basic to Inform, and which are described throughout the documentation.
```