

SR1 - Physical World Model

A/sr1

Purpose

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

A/sr1. §8 Verbal descriptions of numerical comparisons; §9-10 Creating the world model; §11-15 Rooms; §16-22 Things; §23-27 Directions; §28-30 Doors; §31-32 Containers and supporters; §33 Kinds vs patterns; §34 The openability pattern; §35-36 The lockability pattern; §37 Backdrops; §38-40 People; §41 Non-fundamental kinds; §42-43 Men, women and animals; §44 Devices; §45-46 Vehicles; §47-48 Player's holdalls; §49-51 Correspondence between I6 and I7 property and attribute names

§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 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. 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. Thus, to say “The felt hat is worn.” implies “...by the player”: the curious syntax “(adjectival)” below, applied to the past participle, causes this to happen. This is a syntax not documented in the Inform manuals and which may change to be replaced with something more general later on; for the moment, please do not use it.

The verb to carry (he carries, they carry, he carried, it is carried (adjectival), he is carrying) implies the carrying relation.

The verb to hold (he holds, they hold, he held, it is held (adjectival), he is holding) implies the holding relation.

The verb to wear (he wears, they wear, he wore, it is worn (adjectival), he is wearing) implies the wearing relation.

§4. 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.

§5. 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.

§6. 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.

§7. 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" says so (it is indirectly in one of the rooms).

§8. **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.

§9. **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 alternative 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]

§10. 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.

§11. **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."

§12. 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.

A room has a text called printed name.

§13. 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.

§14. 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.

§15. 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."

§16. **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."

§17. 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.

§18. 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, "initially carried" means that the player started with it. ("Carried", which means the player is carrying it now, is not defined as an either/or property but as an adjectival use of the past participle of *to carry*: see above.)

A thing can be handled.
 A thing can be initially carried.

§19. 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 plural-named or singular-named. A thing is usually singular-named.
 A thing can be proper-named or improper-named. A thing is usually improper-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.

§20. 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 an indefinite article.
A thing has a text called a description.
A thing has a text called an initial appearance.
A thing has a text called printed name.
A thing has a text called a printed plural name.
```

§21. The kind “thing”, like all kinds of object, is compiled by NI into an I6 `Class` definition. The three `component_*` properties form up the “part of” hierarchy: if A and B are parts of C then the `component_parent` of both A and B is C, while A is the `component_child` of C and B is the `component_sibling` of A. (This is all directly analogous to I6’s built in object tree for containment and support. Incorporation, being a part of, was a concept which didn’t exist in the I6 world model, and we choose not to implement in a way matching the I6 handling of containment and support because we want the extra flexibility of allowing anything to incorporate anything else – we don’t want the restrictions inherent in the way, in I6, something can be either a container or a supporter but not both. This is why incorporation is not tied to any I7 kind, or any I6 class – there’s no notion of “incorporator” – even though linguistically the incorporation, containment and support relations are so similar to each other. Anyway, the reason something can’t be both `container` and `supporter` in I6 is ultimately because they share the same tree structure: to escape from that, we need incorporation to have a tree all its own at the I6 level, and this is it.)

```
Include (-
  with component_parent nothing, component_sibling nothing, component_child nothing,
  -) when defining a thing.
```

§22. 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.]
```

§23. **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."

§24. 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.

§25. 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.

§26. 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 children 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 `012_north` as it otherwise would.

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

§27. 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 tessellation 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.

§28. **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. ')."

§29. 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.

§30. “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.

§31. **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 teacheat 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.

§32. 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.

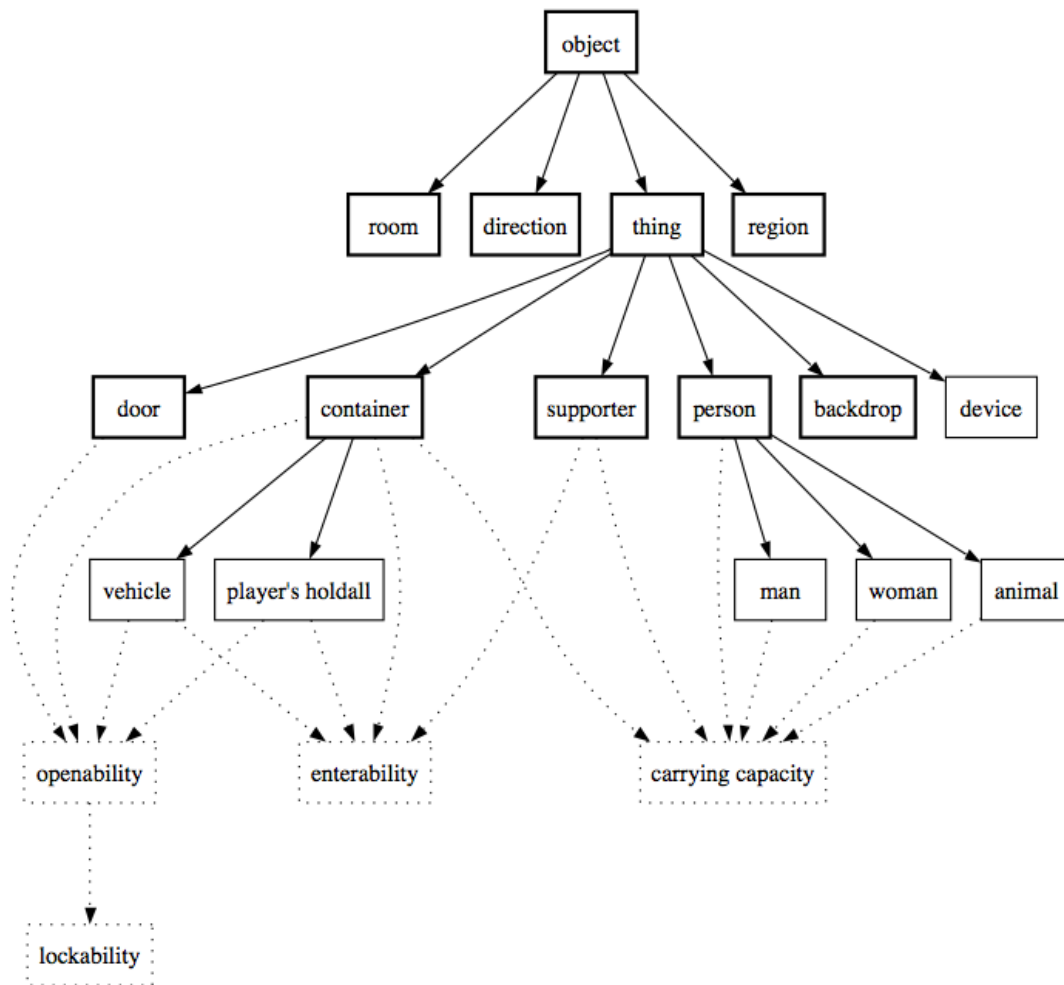
Include (-
 has transparent supporter
 -) when defining a supporter.

§33. **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).



§34. 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.

§35. 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.]

§36. 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.

§37. **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.

§38. **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.

§39. 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, ! number 0,
- ) when defining a person.
```

§40. 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 be 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 (-
    has proper,
    with saved_short_name "yourself",
- ) when defining yourself.
```

§41. **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 are 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.

§42. **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.

§43. 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."

§44. **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."

§45. **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.

§46. 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.

§47. **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.

§48. 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".
```

§49. **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.)

§50. 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 initially carried property translates into I6 as "initially_carried".
 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".

§51. 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".