

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.

A/sr5. §2-11 Say phrases; §12 Using the list-writer; §13 Text substitutions using the list-writer; §14 Grouping in the list-writer; §15 Lists written but not by the list-writer; §16 Filtering in the list-writer; §17-24 Values and data structures; §25 Incrementing and decrementing; §26-27 Tables; §28 Indexed text; §29-30 Matching text; §31 Replacing text; §32 Casing of text; §33-40 Lists; §41-51 Loops and conditionals; §52-59 Actions, activities and rules; §60-62 Rules; §63-75 The model world; §76-80 Understanding; §81-82 Using external resources; §83-84 Message support; §85-96 Miscellaneous other phrases

§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. But in fact this is only one of a family of similar phrases. For every kind of value *K* which can be printed out, we have to define “say (something - *K*)”: this first phrase is that definition for the case where *K* is “text”. (Well: strictly speaking, it’s only true that we define this for each atomic *K* and also for each KOV constructor. Thus “list of *K*₁” and “list of *K*₂” are each said by the same phrase, “say (X - list of values)”, which is viable since lists store their types at run-time. This ensures that the number of “say (something - *K*)” doesn’t skyrocket with all of the possibilities for lists, lists of lists and so on. And similarly we only make one “say (X - object)”, not a whole pile of definitions for “say (X - room)”, “say (X - direction)” and so on for each other kind.)

We have this plethora of alternative “say (something - *K*)” definitions in order to give the type-checking machinery options as to which to use on any given value. But the fact that all these alternative definitions exist is not altogether obvious, because they are hidden in no fewer than three ways:

- (1) Very few of them are defined below, because most are defined instead by the data type interpreter, either using definitions tabulated in the `Types.i6t` template file (for built-in KOVS like “table” or “rule”), or when new KOVS are created (when the user creates a unit or a new enumerated KOV).
- (2) The Phrasebook page of the index covers all of them with the generic entry: say “[*a value of some sort*]”.
- (3) Invocation lists built during parsing, which normally hold all possible interpretations of text as a phrase, are optimised a little so that in clear-cut cases (which is almost all cases, in practice) they will only ever contain one “say (something - *K*)” possibility. For instance, if “say 23” is parsed, the invocation list will not bother to store “say (something - text)” as one of the possibilities, because manifestly 23 is not a piece of text.

Anyway, here are three of the four cases of “say (something - K)” to be created in the Standard Rules rather than elsewhere.

Part SR5 - Phrasebook

Section SR5/1/1 - Saying - Values

To say (something - text)
 (documented at ph_say):
 (- print (PrintText) {something}; -).
 To say (something - number):
 (- print (say__n={something}); -).

§3. 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. It would be tidier to abstract this with a function call, but it would cost a function call.

To say (ch - unicode-character) -- running on:
 (- #ifdef TARGET_ZCODE; @print_unicode {ch};
 #ifndef; if (unicode_gestalt_ok) glk_put_char_uni({ch}); else print "?"; #endif; -).

§4. 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 ph_sayn):
 (- print (number) say__n=({something}); -).
 To say (something - time) in words:
 (- print (PrintTimeOfDayEnglish) {something}; -).
 To say s
 (documented at ph_sayn):
 (- STextSubstitution(); -).

§5. 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 (something - object):
    (- print (name) {something}; -).
To say a (something - object)
    (documented at ph_saya):
    (- print (a) {something}; -).
To say an (something - object)
    (documented at ph_saya):
    (- print (a) {something}; -).
To say A (something - object):
    (- CIndefArt({something}); -).
To say An (something - object):
    (- CIndefArt({something}); -).
To say the (something - object):
    (- print (the) {something}; -).
To say The (something - object):
    (- print (The) {something}; -).
```

§6. 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 ph_sayif): (- {-erase}
    if (~~({c})) jump {-next-label:Say};
    -).

To say unless (c - condition): (- {-erase}
    if ({c}) jump {-next-label:Say};
    -).

To say end if: (- {-erase}
    .{-label:Say}; .{-label:SayX};
    -).

To say end unless: (- {-erase}
    .{-label:Say}; .{-label:SayX};
    -).

To say otherwise/else if (c - condition): (- {-erase}
    jump {-next-label:SayX}; .{-label:Say}; if (~~({c})) jump {-next-label:Say};
    -).

To say otherwise/else unless (c - condition): (- {-erase}
    jump {-next-label:SayX}; .{-label:Say}; if ({c}) jump {-next-label:Say};
    -).

To say otherwise: (- {-erase}
    jump {-next-label:SayX}; .{-label:Say};
    -).

To say else: (- {-erase}
    jump {-next-label:SayX}; .{-label:Say};
    -).
```

§7. 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 ph_sayoneof):
    (- {-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:
    (- @nop; {-segment-count}: -).

To say purely at random -- ending say_one_of with marker I7_S00_PAR:
    (- {-close-brace} -).

To say at random -- ending say_one_of with marker I7_S00_RAN:
```

```

    (- {-close-brace} -).
To say sticky random -- ending say_one_of with marker I7_S00_STI:
    (- {-close-brace} -).
To say as decreasingly likely outcomes -- ending say_one_of with marker I7_S00_TAP:
    (- {-close-brace} -).
To say in random order -- ending say_one_of with marker I7_S00_SHU:
    (- {-close-brace} -).
To say cycling -- ending say_one_of with marker I7_S00_CYC:
    (- {-close-brace} -).
To say stopping -- ending say_one_of with marker I7_S00_STOP:
    (- {-close-brace} -).

```

§8. 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 ph_lbreak):
    (- new_line; -).
To say no line break -- running on: do nothing.
To say conditional paragraph break -- running on:
    (- DivideParagraphPoint(); -).
To say command clarification break -- running on:
    (- CommandClarificationBreak(); -).
To say paragraph break -- running on:
    (- DivideParagraphPoint(); new_line; -).
To say run paragraph on -- running on:
    (- RunParagraphOn(); -).
To say run paragraph on with special look spacing -- running on:
    (- SpecialLookSpacingBreak(); -).
To decide if a paragraph break is pending:
    (- (say__p) -).

```

§9. 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:
    (- print "["; -).
To say close bracket -- running on:
    (- print "]" ; -).
To say apostrophe/' -- running on:
    (- print "'"; -).
To say quotation mark -- running on:
    (- print "~"; -).

```

§10. 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 ph_types):
 (- style bold; -).

To say italic type -- running on:
 (- style underline; -).

To say roman type -- running on:
 (- style roman; -).

To say fixed letter spacing -- running on:
 (- font off; -).

To say variable letter spacing -- running on:
 (- font on; -).

To display the boxed quotation (Q - boxed-quotation)
 (documented at ph_boxed):
 (- DisplayBoxedQuotation({Q}); -).

§11. 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 act_banner):
 (- Banner(); -).

To say the/-- list of extension credits
 (documented at ph_extcr):
 (- ShowExtensionVersions(); -).

To say the/-- complete list of extension credits:
 (- ShowFullExtensionVersions(); -).

To say the/-- player's surroundings
 (documented at ph_surrounds):
 (- SL_Location(); -).

§12. **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_list):
(- WriteListFrom(child({0}), {phrase options}); -).

```

To say contents of (0 - an object):

```
list the contents of 0, as a sentence.
```

To say the contents of (0 - an object):

```
list the contents of 0, as a sentence, using the definite article.
```

§13. 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):

```

(- @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):

```

(- @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):

```

(- @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):

```

(- @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):

```

(- @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):

```

(- @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):

```
(- @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):

```
(- @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) including contents:

```
(- @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; -).
```

§14. 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) 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) together giving articles:

```
(- @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) together as (T - text):

```
(- @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); -).
```

§15. 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 ph_saylv):

```
(- LIST_OF_TY_Say({-pointer-to:L}, 1); -).
```

To say (L - a list of objects) with definite articles:

```
(- LIST_OF_TY_Say({-pointer-to:L}, 2); -).
```

To say (L - a list of objects) with indefinite articles:

```
(- LIST_OF_TY_Say({-pointer-to:L}, 3); -).
```

§16. 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):

```
(- list_filter_routine = {D}; -).
```

To unfilter list recursion:

```
(- list_filter_routine = 0; -).
```

§17. **Values and data structures.** To begin with, counting the number of items matching a description.

Section SR5/2/1 - Values and data structures - Counting

To decide which number is number of (S - description)

(documented at ph_numof):
(- {-number-of:S} -).

To decide which number is number of (S - domain-description)

(documented at ph_numof):
(- {-number-of:S} -).

§18. 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 strictly to do with arithmetic in the same way, but it's needed to flag the phrase for the Inform typechecker's special attention.

Section SR5/2/2 - Values and data structures - Arithmetic

To decide which number is (X - number) + (Y - number)

(arithmetic operation 0)
(documented at ph_plus): (- ({X}+{Y}) -).

To decide which number is (X - number) - (Y - number)

(arithmetic operation 1):
(- ({X}-{Y}) -).

To decide which number is (X - number) * (Y - number)

(arithmetic operation 2):
(- ({X}*{Y}{-rescale-times}) -).

To decide which number is (X - number) / (Y - number)

(arithmetic operation 3):
(- (IntegerDivide({X}{-rescale-divide},{Y})) -).

To decide which number is (X - number) plus (Y - number)

(arithmetic operation 0):
(- ({X}+{Y}) -).

To decide which number is (X - number) minus (Y - number)

(arithmetic operation 1):
(- ({X}-{Y}) -).

To decide which number is (X - number) times (Y - number)

(arithmetic operation 2):
(- ({X}*{Y}{-rescale-times}) -).

To decide which number is (X - number) multiplied by (Y - number)

(arithmetic operation 2):
(- ({X}*{Y}{-rescale-times}) -).

To decide which number is (X - number) divided by (Y - number)

(arithmetic operation 3):
(- (IntegerDivide({X}{-rescale-divide},{Y})) -).

To decide which number is remainder after dividing (X - number)
by (Y - number)

(arithmetic operation 4):
(- (IntegerRemainder({X},{Y})) -).

To decide which number is (X - number) to the nearest (Y - number)

(arithmetic operation 5):
(- (RoundOffTime({X},{Y})) -).

To decide which number is the square root of (X - number)
 (arithmetic operation 6):
 (- (SquareRoot({X}{-rescale-root})) -).

To decide which number is the cube root of (X - number)
 (arithmetic operation 7):
 (- (CubeRoot({X}{-rescale-cuberoot})) -).

To decide which number is total (p - property) of (S - description)
 (arithmetic operation 11)
 (documented at ph_total):
 (- {-total-of:S} -).

§19. 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 SR5/2/3 - Values and data structures - Truth states

To decide what truth state is whether or not (C - condition)
 (documented at ph_wornot):
 (- {C} -).

§20. Random numbers and random items chosen from sets of objects matching a given description (“a random closed door”).

Section SR5/2/4 - Values and data structures - Randomness

To decide which object is a/-- random (S - description)
 (documented at ph_randobj):
 (- {-random-of:S} -).

To decide which number is a random number from (N - number) to (M - number)
 (documented at ph_random):
 (- (GenerateRandomNumber({N}, {M})) -).

To decide which number is a random number between (N - number) and (M - number):
 (- (GenerateRandomNumber({N}, {M})) -).

To decide whether a random chance of (N - number) in (M - number) succeeds:
 (- (GenerateRandomNumber(1, {M}) <= {N}) -).

To seed the random-number generator with (N - number):
 (- VM_Seed_RNG({N}); -).

To decide which value is a/-- random (S - domain-description)
 (amended kind of value 1):
 (- {-random-of:S} -).

§21. The “now” phrase is somewhat unclassifiable, and though it doesn’t quite belong here, it doesn’t quite belong anywhere else either:

Section SR5/2/5 - Values and data structures - Changing stored values

To now (cn - now-condition)
 (documented at ph_now):
 (- {cn} -).

§22. 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). Moreover, the assignment itself works differently depending on whether the values are pointers to blocks of data on the heap, needing a deep destruction and a deep copy, or are simple values, so that one can simply overwrite the other. All of this makes the “To let” section here only slightly shorter than John Galsworthy’s Forsyte novel of the same name:

```
To let (t - nonexisting variable) be (u - word value)
  (assignment operation)
  (documented at ph_let):
  (- {t} = {u}; -).

To let (t - nonexisting variable) be (u - pointer value)
  (assignment operation):
  (- {-pointer-to:t}={-allocate-storage-for:u};BlkValueCopy({-pointer-to:t},{-pointer-to:u}); -).

To let (t - nonexisting variable) be (u - kind of word value)
  (assignment operation):
  (- {t} = {-default-value-for:u}; -).

To let (t - nonexisting variable) be (u - kind of pointer value)
  (assignment operation):
  (- {-pointer-to:t} = {-allocate-storage-for:u}; -).

To let (t - existing variable) be (u - assignable-value)
  (assignment operation):
  (- {-assignment}; -).
```

§23. These are not strictly speaking assignments; 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)”.

```
To let (t - existing variable) be given by (Q - equation-name):
  (- {-solve-equation}; -).

To let (t - nonexisting variable) be given by (Q - equation-name):
  (- {-solve-equation}; -).
```

§24. 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”.

```
To change (p - property-value) to (w - assignable-value)
  (assignment operation)
  (documented at ph_changev):
  (- {-assignment}; -).

To change (gv - global variable) to (w - assignable-value)
  (assignment operation)
  (documented at ph_change):
  (- {-assignment}; -).

To change (tr - table-reference) to (w - assignable-value)
  (assignment operation):
  (- {-assignment}; -).

To change (le - list-entry) to (w - assignable-value)
```

```

    (assignment operation):
    (- {-assignment}); -).
To change (o - object) to (p - property)
    (assignment operation)
    (documented at ph_change):
    (- SetEitherOrProperty({o}, {p}, false, {-adjective-definition:p}); -).
To change (o - object) to (w - value)
    (assignment operation):
    (- WriteValueProperty({o},{-convert-adjectival-constants:w},{w}); -).
To change (lv - existing variable) to (w - assignable-value)
    (assignment operation):
    (- {-assignment}); -).

```

§25. Incrementing and decrementing. 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. Special cases in the type-checker are gradually moving into instructions in phrase preambles, but these haven't done so yet.

Section SR5/2/6 - Values and data structures - Increase and decrease

```

To increase/increment (p - property-value) by (w - value)
    (documented at ph_increase):
    (- Write{p}{-delete},{p}+{w}); -).
To increase/increment (gv - global variable) by (w - value):
    (- {gv} = {gv} + {w}; -).
To increase/increment (lv - existing variable) by (w - value):
    (- {lv} = {lv} + {w}; -).
To increase/increment (tr - table-reference) by (w - value):
    (- {tr}{-delete},2,{w}); -).
To decrease/decrement (p - property-value) by (w - value):
    (- Write{p}{-delete},{p}-{w}); -).
To decrease/decrement (gv - global variable) by (w - value):
    (- {gv} = {gv} - {w}; -).
To decrease/decrement (lv - existing variable) by (w - value):
    (- {lv} = {lv} - {w}; -).
To decrease/decrement (tr - table-reference) by (w - value):
    (- {tr}{-delete},3,{w}); -).

```

§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 and data structures - Tables

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):

(- 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):

(- (Exists{-do-not-dereference:TR} == false) -).

To delete (tr - table-reference)

(documented at ph_blankout):

(- {tr}{-delete},4); -).

To blank out the whole row

(documented at ph_blankout):

(- {-require-ctvs}TableBlankOutRow(ct_0, ct_1); -).

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 - table-column) of (w - value) in/from (T - table-name):

(- {-require-ctvs}ct_0 = {T}; ct_1 = TableRowCorr(ct_0, {TC}, {w}); -).

To choose a/the/-- blank row in/from (T - table-name):

(- {-require-ctvs}ct_0 = {T}; ct_1 = TableBlankRow(ct_0); -).

To choose a/the/-- random row in/from (T - table-name):

(- {-require-ctvs}ct_0 = {T}; ct_1 = TableRandomRow(ct_0); -).

§27. Sorting.

Section SR5/2/9 - Values and data structures - Sorting tables

To sort (T - table-name) in random order

(documented at ph_sort):

(- TableShuffle({T}); -).

To sort (T - table-name) in (TC - table-column) order:

(- TableSort({T}, {TC}, 1); -).

To sort (T - table-name) in reverse (TC - table-column) order:

(- 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 and data structures - Indexed text

To decide what number is the number of characters in (txb - indexed text)

(documented at ph_numofc):

(- IT_BlobAccess({-pointer-to:txb}, CHR_BLOB) -).

To decide what indexed text is character number (N - a number) in (txb - indexed text):

(- IT_GetBlob({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, {N}, CHR_BLOB) -).

To replace character number (N - a number) in (txb - indexed text)

with (rtxb - indexed text):

(- IT_ReplaceBlob(CHR_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).

To decide what number is the number of words in (txb - indexed text):

(- IT_BlobAccess({-pointer-to:txb}, WORD_BLOB) -).

To decide what indexed text is word number (N - a number) in (txb - indexed text):

(- IT_GetBlob({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, {N}, WORD_BLOB) -).

To replace word number (N - a number) in (txb - indexed text)

with (rtxb - indexed text):

(- IT_ReplaceBlob(WORD_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).

To decide what number is the number of punctuated words in (txb - indexed text):

(- IT_BlobAccess({-pointer-to:txb}, PWORD_BLOB) -).

To decide what indexed text is punctuated word number (N - a number) in (txb - indexed text):

(- IT_GetBlob({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, {N}, PWORD_BLOB) -).

To replace punctuated word number (N - a number) in (txb - indexed text)

with (rtxb - indexed text):

(- IT_ReplaceBlob(PWORD_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).

To decide what number is the number of unpunctuated words in (txb - indexed text):

(- IT_BlobAccess({-pointer-to:txb}, UWORD_BLOB) -).

To decide what indexed text is unpunctuated word number (N - a number) in (txb - indexed text):

(- IT_GetBlob({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, {N}, UWORD_BLOB) -).

To replace unpunctuated word number (N - a number) in (txb - indexed text)

with (rtxb - indexed text):

(- IT_ReplaceBlob(UWORD_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).

To decide what number is the number of lines in (txb - indexed text):

(- IT_BlobAccess({-pointer-to:txb}, LINE_BLOB) -).

To decide what indexed text is line number (N - a number) in (txb - indexed text):

(- IT_GetBlob({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, {N}, LINE_BLOB) -).

To replace line number (N - a number) in (txb - indexed text) with (rtxb - indexed text):

(- IT_ReplaceBlob(LINE_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).

To decide what number is the number of paragraphs in (txb - indexed text):

(- IT_BlobAccess({-pointer-to:txb}, PARA_BLOB) -).

To decide what indexed text is paragraph number (N - a number) in (txb - indexed text):

(- IT_GetBlob({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, {N}, PARA_BLOB) -).

To replace paragraph number (N - a number) in (txb - indexed text) with (rtxb - indexed text):

(- IT_ReplaceBlob(PARA_BLOB, {-pointer-to:txb}, {N}, {-pointer-to:rtxb}); -).

§29. **Matching text.** A common matching engine is used for matching plain text...

Section SR5/2/11 - Values and data structures - Matching text

To decide if (txb - indexed text) exactly matches the text (ftxb - indexed text),
 case insensitively
 (documented at ph_exactm):
 (- 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:
 (- 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:
 (- 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_regexp):
 (- 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:
 (- IT_Replace_RE(REGEXP_BLOB,{-pointer-to:txb},{-pointer-to:ftxb},0,{phrase options}) -).
 To decide what number is number of times (txb - indexed text) matches the regular expression
 (ftxb - indexed text), case insensitively:
 (- IT_Replace_RE(REGEXP_BLOB,{-pointer-to:txb},{-pointer-to:ftxb},1,{phrase options}) -).
 To decide what indexed text is text matching regular expression:
 (- IT_RE_GetMatchVar({-pointer-to-new:INDEXED_TEXT_TY}, 0) -).
 To decide what indexed text is text matching subexpression (N - a number):
 (- IT_RE_GetMatchVar({-pointer-to-new:INDEXED_TEXT_TY}, {N}) -).

§31. **Replacing text.** The same engine, in "RegExp.i6t", handles replacement.

Section SR5/2/12 - Values and data structures - 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(REGEXP_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:
 (- 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):
 (- 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):
 (- IT_ReplaceText(PWORD_BLOB, {-pointer-to:txb}, {-pointer-to:ftxb}, {-pointer-to:rtxb}); -).

§32. Casing of text.

Section SR5/2/13 - Values and data structures - Casing of text

To decide what indexed text is (txb - indexed text) in lower case

(documented at ph_casing):

(- IT_CharactersToCase({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, 0) -).

To decide what indexed text is (txb - indexed text) in upper case:

(- IT_CharactersToCase({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, 1) -).

To decide what indexed text is (txb - indexed text) in title case:

(- IT_CharactersToCase({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, 2) -).

To decide what indexed text is (txb - indexed text) in sentence case:

(- IT_CharactersToCase({-pointer-to-new:INDEXED_TEXT_TY}, {-pointer-to:txb}, 3) -).

To decide if (txb - indexed text) is in lower case:

(- IT_CharactersOfCase({-pointer-to:txb}, 0) -).

To decide if (txb - indexed text) is in upper case:

(- 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 and data structures - Lists

To add (N - value = kov reference 1) to (L - list of values = list of kov marker 1),

if absent

(documented at ph_addlist):

(- LIST_OF_TY_InsertItem({-pointer-to:L}, {N}, 0, 0, {phrase options}); -).

To add (N - value = kov reference 1) at entry (E - number) in

(L - list of values = list of kov marker 1), if absent:

(- LIST_OF_TY_InsertItem({-pointer-to:L}, {N}, 1, {E}, {phrase options}); -).

To add (LX - list of values = list of kov reference 1) to

(L - list of values = list of kov marker 1), if absent:

(- LIST_OF_TY_AppendList({-pointer-to:L}, {-pointer-to:LX}, 0, 0, {phrase options}); -).

To add (LX - list of values = list of kov reference 1) at entry (E - number)

in (L - list of values = list of kov marker 1):

(- LIST_OF_TY_AppendList({-pointer-to:L}, {-pointer-to:LX}, 1, {E}, 0); -).

To remove (N - value = kov reference 1) from (L - list of values = list of kov marker 1),

if present

(documented at ph_remlist):

(- LIST_OF_TY_RemoveValue({-pointer-to:L}, {N}, {phrase options}); -).

To remove (N - list of values = list of kov reference 1) from

(L - list of values = list of kov marker 1), if present:

(- 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:

(- 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:

(- 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 - value = kov reference 1) is listed in
  (L - list of values = list of kov marker 1)
  (documented at ph_islistedin):
  (- (LIST_OF_TY_FindItem({-pointer-to:L}, {N})) -).
To decide if (N - value = kov reference 1) is not listed in
  (L - list of values = list of kov marker 1):
  (- (LIST_OF_TY_FindItem({-pointer-to:L}, {N}) == false) -).
```

§35. The following are casts to and from other data types 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 objects is the list of (D - description)
  (documented at ph_listd):
  (- LIST_OF_TY_Desc({-pointer-to-new:LIST_OF_TY}, {D}) -).
To decide what list of values is the list of (D - domain-description)
  (amended kind of value 0):
  (- LIST_OF_TY_Desc({-pointer-to-new:LIST_OF_TY}, {D}, {-domain-kov:D}) -).
To decide what list of objects is the multiple object list
  (documented at ph_mobj1):
  (- LIST_OF_TY_Mol({-pointer-to-new:LIST_OF_TY}) -).
To alter the multiple object list to (L - list of objects):
  (- LIST_OF_TY_Set_Mol({-pointer-to:L}); -).
```

§36. Determining and setting the length:

Section SR5/2/15 - Values and data structures - Length of lists

```
To decide what number is the number of entries in/of (L - a list of values)
  (documented at ph_numel):
  (- LIST_OF_TY_GetLength({-pointer-to:L}) -).
To truncate (L - a list of values) to (N - a number) entries
  (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:
  (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, -1, 1); -).
To truncate (L - a list of values) to the last (N - a number) entries:
  (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, -1, -1); -).
To extend (L - a list of values) to (N - a number) entries:
  (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, 1); -).
To change (L - a list of values) to have (N - a number) entries:
  (- LIST_OF_TY_SetLength({-pointer-to:L}, {N}, 0); -).
```

§37. Easy but useful list operations:

Section SR5/2/16 - Values and data structures - Reversing and rotating lists

To reverse (L - a list of values)

(documented at ph_rev1):

(- LIST_OF_TY_Reverse({-pointer-to:L}); -).

To rotate (L - a list of values)

(documented at ph_rot1):

(- LIST_OF_TY_Rotate({-pointer-to:L}, 0); -).

To rotate (L - a list of values) backwards:

(- 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 and data structures - Sorting lists

To sort (L - a list of values)

(documented at ph_sort1):

(- LIST_OF_TY_Sort({-pointer-to:L}, 1); -).

To sort (L - a list of values) in reverse order:

(- LIST_OF_TY_Sort({-pointer-to:L}, -1); -).

To sort (L - a list of values) in random order:

(- LIST_OF_TY_Sort({-pointer-to:L}, 2); -).

To sort (L - a list of objects) in (P - property) order:

(- LIST_OF_TY_Sort({-pointer-to:L}, 1, {P}); -).

To sort (L - a list of objects) in reverse (P - property) order:

(- LIST_OF_TY_Sort({-pointer-to:L}, -1, {P}); -).

§39. To call use options a “data structure” is a stretch. Still:

Section SR5/2/18 - Values and data structures - Use options

To decide whether using the/-- (U0 - use-option)

(documented at ph_testuo):

(- (TestUseOption({U0})) -).

§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 and data structures - Relations

To decide which object is next step via (R - abstract-relation)

from (O1 - object) to (O2 - object)

(documented at ph_nextstep):

(- RelationRouteTo({R},{O1},{O2},false) -).

To decide which number is number of steps via (R - abstract-relation)

from (O1 - object) to (O2 - object):

(- RelationRouteTo({R},{O1},{O2},true) -).

§41. **Loops and conditionals.** 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 - Loops and conditionals - If and unless

To if (c - condition) then (ph - phrase)

(documented at ph_if):

(- if {c} {ph} -).

To if (c - condition) , (ph - phrase):

(- if {c} {ph} -).

To unless (c - condition) , (ph - phrase):

(- if (~{c}) {ph} -).

§42. 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):

(- } else if {c} { -).

To else if (c - condition):

(- } else if {c} { -).

To otherwise unless (c - condition):

(- } else if (~{c}) { -).

To else unless (c - condition):

(- } else if (~{c}) { -).

To otherwise (ph - phrase)

(documented at ph_otherwise):

(- else {ph} -).

To else (ph - phrase)

(documented at ph_otherwise):

(- else {ph} -).

To if (c - condition) begin -- end:

(- if {c} -).

To unless (c - condition) begin -- end:

(- if (~{c}) -).

§43. 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:

(- switch({V}) -).

§44. 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 - Loops and conditionals - While

```
To while (c - condition) repeatedly (ph - phrase)
    (documented at ph_while):
    (- while {c} {ph} -).
To while (c - condition) , (ph - phrase):
    (- while {c} {ph} -).
To while (c - condition) begin -- end:
    (- while {c} -).
```

§45. 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 - Loops and conditionals - Repeat

```
To repeat with (loopvar - nonexisting number variable)
    running from (v - number) to (w - number) begin -- end
    (documented at ph_repeat):
    (- for ({loopvar}={v}: {loopvar}<={w}: {loopvar}++) -).
To repeat with (loopvar - nonexisting object variable)
    running through (OS - description) begin -- end
    (documented at ph_runthrough):
    (- {-loop-over:OS} -).
To repeat with (loopvar - nonexisting object variable)
    running through (OS - domain-description) begin -- end
    (documented at ph_runthrough):
    (- {-loop-over-domain:OS} -).
```

§46. 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.

Section SR5/3/4 - Loops and conditionals - Repeat through tables

```
To repeat through (T - table-name) begin -- end
    (documented at ph_tabrepeat):
    (- {-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:
    (- {-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:
    (- {-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:
  (- {-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}) -).
```

§47. Finally, we can repeat through the other dynamic data structure: the list.

Section SR5/3/5 - Loops and conditionals - Repeat through lists

```
To repeat with (loopvar - nonexisting object variable)
  running through (L - list of values) begin -- end:
  (- {-loop-over-list:L} -).
```

§48. 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 - Loops and conditionals - Changing the flow of loops

```
To break -- in loop (documented at ph_break):
  (- break; -).
To next -- in loop (documented at ph_next):
  (- continue; -).
```

§49. 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 - Loops and conditionals - Deciding outcomes

```
To yes
  (documented at ph_yes):
  (- rtrue; -) - in to decide if only.
To decide yes:
  (- rtrue; -) - in to decide if only.
To no:
  (- rfalse; -) - in to decide if only.
To decide no:
  (- rfalse; -) - in to decide if only.
```

§50. 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_result):
  (- return {-check-return-type:something}; -).
```

§51. “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 - Loops and conditionals - Stop or go

To do nothing (documented at `ph_nothing`):

```
(- ; -).
```

To stop (documented at `ph_stop`):

```
(- return; -) - in to only.
```

§52. **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):

(- @push keep_silent; keep_silent=1; {doing something}; @pull keep_silent; -).

To try silently (doing something - action):

(- @push keep_silent; keep_silent=1; {doing something}; @pull keep_silent; -).

§53. 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_requires`):

(- (NeedToTouchNoun()) -).

To decide whether the action requires a touchable second noun:

(- (NeedToTouchSecondNoun()) -).

To decide whether the action requires a carried noun:

(- (NeedToCarryNoun()) -).

To decide whether the action requires a carried second noun:

(- (NeedToCarrySecondNoun()) -).

To decide whether the action requires light:

(- (NeedLightForAction()) -).

§54. 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_stopac`):

(- `rtrue`; -) - in to only.

To continue the action:

(- `rfalse`; -) - in to only.

§55. 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_curract):

```
(- STORED_ACTION_TY_Current({-pointer-to-new:STORED_ACTION_TY}) -).
```

To decide what stored action is the action of (A - action):

```
(- {A}{-delete}{-delete}, STORED_ACTION_TY_Current({-pointer-to-new:STORED_ACTION_TY})) -).
```

To try (S - stored action):

```
(- STORED_ACTION_TY_Try({S}); -).
```

To silently try (S - stored action):

```
(- STORED_ACTION_TY_Try({S}, true); -).
```

To try silently (S - stored action):

```
(- 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):

```
(- (STORED_ACTION_TY_Part({-pointer-to:act}, 0)) -).
```

To decide what object is the noun part of (act - a stored action):

```
(- (STORED_ACTION_TY_Part({-pointer-to:act}, 1)) -).
```

To decide what object is the second noun part of (act - a stored action):

```
(- (STORED_ACTION_TY_Part({-pointer-to:act}, 2)) -).
```

To decide what object is the actor part of (act - a stored action):

```
(- (STORED_ACTION_TY_Part({-pointer-to:act}, 3)) -).
```

§56. Firing off activities:

Section SR5/4/5 - Actions, activities and rules - Carrying out activities

To carry out the (A - activity) activity

(documented at ph_carryout):

```
(- CarryOutActivity({A}); -).
```

To carry out the (A - activity) activity with (O - object):

```
(- CarryOutActivity({A}, {O}); -).
```

§57. The following circumlocutions are needed because, at present, we can’t define adjectives for kinds of value other than “object”.

[To decide whether (A - activity) activity is going on

(documented at ph_goingon):

```
(- (TestActivity({A})) -).
```

To decide whether (A - activity) activity is not going on:

```
(- (~ (TestActivity({A}))) -).
```

To decide whether (A - activity) activity is empty:

```
(- (ActivityEmpty({A})) -).
```

To decide whether (A - activity) activity is not empty:

```
(- (~ (ActivityEmpty({A}))) -).]
```

§58. This is analogous to “continue the action”:

To continue the activity:
 (- rfalse; -) - in to only.

§59. 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) activity
 (documented at ph_beginact):
 (- BeginActivity({A}); -).

To begin the (A - activity) activity with (O - object):
 (- BeginActivity({A}, {O}); -).

To decide whether handling (A - activity) activity:
 (- (~~(ForActivity({A}))) -).

To decide whether handling (A - activity) activity with (O - object):
 (- (~~(ForActivity({A}, {O}))) -).

To end the (A - activity) activity:
 (- EndActivity({A}); -).

To end the (A - activity) activity with (O - object):
 (- EndActivity({A}, {O}); -).

To abandon the (A - activity) activity:
 (- AbandonActivity({A}); -).

To abandon the (A - activity) activity with (O - object):
 (- AbandonActivity({A}, {O}); -).

§60. **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 - a rule) for (O - object):
    (- FollowRulebook({RL}, {O}, true); -).
To consider (RL - a rule)
    (documented at ph_consider):
    (- ProcessRulebook({RL}); -).
To consider (RL - a rule) for (O - object):
    (- ProcessRulebook({RL}, {O}, true); -).
To abide by (RL - a rule)
    (documented at ph_abide):
    (- if (ProcessRulebook({RL})) rtrue; -) - in to only.
To abide by (RL - a rule) for (O - object):
    (- if (ProcessRulebook({RL}, {O}, true)) rtrue; -) - in to only.
To anonymously abide by (RL - a rule):
    (- if (temporary_value = ProcessRulebook({RL})) {
        if (RulebookSucceeded()) ActRulebookSucceeds(temporary_value);
        else ActRulebookFails(temporary_value);
        rtrue;
    } -) - in to only.
To anonymously abide by (RL - a rule) for (O - object):
    (- if (temporary_value = ProcessRulebook({RL}, {O}, true)) {
        if (RulebookSucceeded()) ActRulebookSucceeds(temporary_value);
        else ActRulebookFails(temporary_value);
        rtrue;
    } -) - in to only.

```

§61. 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: (- rfalse; -) - in to only.
To rule succeeds
    (documented at ph_succeeds):
    (- RulebookSucceeds(); rtrue; -) - in to only.
To rule fails:
    (- RulebookFails(); rtrue; -) - in to only.
To rule succeeds with result (O - a miscellaneous-value):
    (- RulebookSucceeds(true,{O}); rtrue; -) - in to only.
To rule fails with result (O - a miscellaneous-value):
    (- RulebookFails(true,{O}); rtrue; -) - in to only.
To decide if rule succeeded:
    (- (RulebookSucceeded()) -).
To decide if rule succeeded with result (O - miscellaneous-value):
    (- ((RulebookSucceeded()) && (ResultOfRule() == {O})) -).
To decide if rule failed:
    (- (RulebookFailed()) -).
To decide if rule failed with result (O - miscellaneous-value):

```

```

    (- ((RulebookFailed()) && (ResultOfRule() == {0})) -).
To decide which miscellaneous-value is the result of the rule
    (documented at ph_resulttr):
    (- (ResultOfRule()) -).
To decide which rulebook-outcome is the outcome of the rulebook
    (documented at ph_outcomer):
    (- (ResultOfRule()) -).

```

§62. 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)
    (documented at ph_ignore):
    (- SuppressRule({RL}); -).
To reinstate (RL - a rule):
    (- ReinstateRule({RL}); -).
To reject the result of (RL - a rule):
    (- DonotuseRule({RL}); -).
To accept the result of (RL - a rule):
    (- DonotuseRule({RL}); -).
To substitute (RL1 - a rule) for (RL2 - a rule):
    (- SubstituteRule({RL1},{RL2}); -).
To restore the original (RL1 - a rule):
    (- SubstituteRule({RL1},{RL1}); -).
To move (RL1 - a rule) to before (RL2 - a rule):
    (- MoveRuleBefore({RL1},{RL2}); -).
To move (RL1 - a rule) to after (RL2 - a rule):
    (- MoveRuleAfter({RL1},{RL2}); -).

```

§63. **The model world.** The score and outcome of play phrases only interface with two I6 library variables, `score` and `deadflag`. The latter takes values 0 (in progress), 1 (lost), 2 (won), or the packed address of a string or routine to print a string which explains in what way the world ended (“You have sailed off into the sunset”).

Section SR5/5/1 - Model world - Score

To award (some - number) point/points
 (documented at `ph_awardp`):
 (- `score=score+{some}`; -).

§64. Phrase definitions with wordings like “the game is in progress” are a necessary evil. The “is” here is parsed literally, not as the present tense of *to be*, so inflected forms like “the game had been in progress” are not available: nor is there any value “the game” for the subject noun phrase to hold, nor can the relation “in”... 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 SR5/5/2 - Model world - Outcome of play

To end the game in death (documented at `ph_end`): (- `deadflag=1`; -).
 To end the game in victory: (- `deadflag=2`; -).
 To end the game saying (finale - text): (- `deadflag={finale}`; -).
 To resume the game: (- `resurrect_please = true`; -).
 To decide whether the game is in progress: (- (`deadflag==0`) -).
 To decide whether the game is over: (- (`deadflag~=0`) -).
 To decide whether the game ended in death: (- (`deadflag==1`) -).
 To decide whether the game ended in victory: (- (`deadflag==2`) -).

§65. Times of day.

Section SR5/5/3 - 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):
 (- (`{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_timeshift):
  (- ((({t}+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))<((t2)+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))) -).
To decide if (t - time) is after (t2 - time):
  (- ((({t}+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))>((t2)+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))) -).
To decide if it is before (t2 - time):
  (- (((the_time+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))<((t2)+20*ONE_HOUR)%(TWENTY_FOUR_HOURS))) -).
To decide if it is after (t2 - time):
  (- (((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_timeshift):
  (- (((t2)-{t}+TWENTY_FOUR_HOURS)%(TWENTY_FOUR_HOURS)) -).
To decide which time is (t - time) after (t2 - time):
  (- (((t2)+{t}+TWENTY_FOUR_HOURS)%(TWENTY_FOUR_HOURS)) -).
```

§67. Durations are in effect casts from “number” to “time”.

Section SR5/5/4 - Model world - Durations

```
To decide which time is (n - number) minutes
  (documented at ph_durations):
  (- (({n})%(TWENTY_FOUR_HOURS)) -).
To decide which time is (n - number) hours:
  (- (({n}*ONE_HOUR)%(TWENTY_FOUR_HOURS)) -).
```

§68. Timed events.

Section SR5/5/5 - Model world - Timed events

```
To (R - rule) in (t - number) turn from now
  (documented at ph_future):
  (- SetTimedEvent({R}, {t}+1, 0); -).
To (R - rule) in (t - number) turns from now:
  (- SetTimedEvent({R}, {t}+1, 0); -).
To (R - rule) at (t - time):
  (- SetTimedEvent({R}, {t}, 1); -).
To (R - rule) in (t - time) from now:
  (- SetTimedEvent({R}, (the_time+{t})%(TWENTY_FOUR_HOURS), 1); -).
```

§69. Scenes.

Section SR5/5/6 - Model world - Scenes

To decide if (sc - scene) has happened:

```
(- (scene_endings-->({sc}-1)) -).
```

To decide if (sc - scene) has not happened:

```
(- (scene_endings-->({sc}-1) == 0) -).
```

To decide if (sc - scene) has ended:

```
(- (scene_endings-->({sc}-1) > 1) -).
```

To decide if (sc - scene) has not ended:

```
(- (scene_endings-->({sc}-1) <= 1) -).
```

§70. Timing of scenes.

Section SR5/5/7 - Model world - Timing of scenes

To decide which time is the time since (sc - scene) began

```
(documented at ph_tsince):
```

```
(- (SceneUtility({sc}, 1)) -).
```

To decide which time is the time when (sc - scene) began:

```
(- (SceneUtility({sc}, 2)) -).
```

To decide which time is the time since (sc - scene) ended:

```
(- (SceneUtility({sc}, 3)) -).
```

To decide which time is the time when (sc - scene) ended:

```
(- (SceneUtility({sc}, 4)) -).
```

§71. Player's identity and location.

Section SR5/5/8 - Model world - Player's identity and location

To change the/-- player to (0 - an object)

```
(documented at ph_changep):
```

```
(- ChangePlayer({0}); -).
```

To decide whether in (somewhere - an object):

```
(- (WhetherIn({somewhere})) -).
```

To decide whether in darkness

```
(documented at ph_indarkness):
```

```
(- (location==thedark) -).
```

§72. Moving and removing things.

Section SR5/5/9 - 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)

(documented at ph_movebd):
(- MoveBackdrop({O}, {D}); -).

To update backdrop positions

(documented at ph_updatebp):
(- MoveFloatingObjects(); -).

§73. The map.

Section SR5/5/10 - Model world - The map

To decide which room is location of (O - object)

(documented at ph_locof):
(- 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_roomdirof):
(- DoorFrom({R1},{D}) -).

To decide which object is the other side of (D - door) from (R1 - room):

(- OtherSideOfDoor({D},{R1}) -).

To decide which object is the direction of (D - door) from (R1 - room):

(- DirectionDoorLeadsIn({D},{R1}) -).

To decide which object is room-or-door (D - direction) from/of (R1 - room):

(- RoomOrDoorFrom({R1},{D}) -).

To change (D - direction) exit of (R1 - room) to (R2 - room)

(documented at ph_changex):
(- AssertMapConnection({R1},{D},{R2}); -).

To change (D - direction) exit of (R1 - room) to nothing/nowhere:

(- 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):

(- BackSideOfDoor({D}) -).

§74. Route-finding.

Section SR5/5/11 - 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:

(- MapRouteTo({R1},{R2},0,{phrase options},true) -).

To decide which object is best route from (R1 - object) to (R2 - object) through
(RS - description),

using doors or using even locked doors:

(- MapRouteTo({R1},{R2},{RS},{phrase options}) -).

To decide which number is number of moves from (R1 - object) to (R2 - object) through
(RS - description),

using doors or using even locked doors:

(- MapRouteTo({R1},{R2},{RS},{phrase options},true) -).

§75. The object tree.

Section SR5/5/12 - 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):

(- (sibling({something})) -).

To decide which object is first thing held by (something - object):

(- (child({something})) -).

§76. **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() -).

§77. 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 act_reading):
 (- (SnippetMatches({S}, {T})) -).

To decide if (S - a snippet) does not match (T - a topic)
 (documented at act_reading):
 (- (SnippetMatches({S}, {T}) == false) -).

To decide if (S - a snippet) includes (T - a topic):
 (- (matched_text=SnippetIncludes({T},{S})) -).

To decide if (S - a snippet) does not include (T - a topic):
 (- (SnippetIncludes({T},{S})==0) -).

§78. 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 act_reading):
 (- SetPlayersCommand({-pointer-to:txb}); -).

To replace (S - a snippet) with (T - text):
 (- SpliceSnippet({S}, {T}); -).

To cut (S - a snippet):
 (- SpliceSnippet({S}, 0); -).

To reject the player's command:
 (- RulebookFails(); rtrue; -) - in to only.

§79. 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_scope):
 (- PlaceInScope({O}, {phrase options}); -).

To place the/-- contents of (O - an object) in scope
 (documented at ph_scope):
 (- ScopeWithin({O}); -).

To set pronouns from possessions of the player:
 (- PronounNoticeHeldObjects(); -).

To set pronouns from (O - an object):
 (- PronounNotice({O}); -).

§80. The I6 parser errors cry out to be an enumerated kind of value, and probably will be in a later build. The following will then be more concise.

Section SR5/6/5 - Understanding - I6 parser errors

To decide whether parser error is didn't understand
(documented at act_parsererror):

```
(- (etype == STUCK_PE) -).
```

To decide whether parser error is only understood as far as:

```
(- (etype == UPTO_PE) -).
```

To decide whether parser error is didn't understand that number:

```
(- (etype == NUMBER_PE) -).
```

To decide whether parser error is can't see any such thing:

```
(- (etype == CANTSEE_PE) -).
```

To decide whether parser error is said too little:

```
(- (etype == TOOLIT_PE) -).
```

To decide whether parser error is aren't holding that:

```
(- (etype == NOTHELD_PE) -).
```

To decide whether parser error is can't use multiple objects:

```
(- (etype == MULTI_PE) -).
```

To decide whether parser error is can only use multiple objects:

```
(- (etype == MMULTI_PE) -).
```

To decide whether parser error is not sure what it refers to:

```
(- (etype == VAGUE_PE) -).
```

To decide whether parser error is excepted something not included:

```
(- (etype == EXCEPT_PE) -).
```

To decide whether parser error is can only do that to something animate:

```
(- (etype == ANIMA_PE) -).
```

To decide whether parser error is not a verb I recognise:

```
(- (etype == VERB_PE) -).
```

To decide whether parser error is not something you need to refer to:

```
(- (etype == SCENERY_PE) -).
```

To decide whether parser error is can't see it at the moment:

```
(- (etype == ITGONE_PE) -).
```

To decide whether parser error is didn't understand the way that finished:

```
(- (etype == JUNKAFTER_PE) -).
```

To decide whether parser error is not enough of those available:

```
(- (etype == TOOFEW_PE) -).
```

To decide whether parser error is nothing to do:

```
(- (etype == NOTHING_PE) -).
```

To decide whether parser error is I beg your pardon:

```
(- (etype == BLANKLINE_PE) -).
```

To decide whether parser error is noun did not make sense in that context:

```
(- (etype == NOTINCONTEXT_PE) -).
```

§81. Using external resources. The following all refer to “FileIO.i6t” and work only on Glux.

Section SR5/7/1 - Using external resources - Files

To read (filename - external-file) into (T - table-name)
 (documented at ph_filetables):
 (- FileIO_GetTable({filename}, {T}); -).

To write (filename - external-file) from (T - table-name):
 (- FileIO_PutTable({filename}, {T}); -).

To decide if (filename - external-file) exists:
 (- (FileIO_Exists({filename}, false)) -).

To decide if ready to read (filename - external-file)
 (documented at ph_readytr):
 (- (FileIO_Ready({filename}, false)) -).

To mark (filename - external-file) as ready to read:
 (- FileIO_MarkReady({filename}, true); -).

To mark (filename - external-file) as not ready to read:
 (- FileIO_MarkReady({filename}, false); -).

To write (T - text) to (FN - external-file)
 (documented at ph_writef):
 (- FileIO_PutContents({FN}, {-allow-stack-frame-access:T}, false); -).

To append (T - text) to (FN - external-file):
 (- FileIO_PutContents({FN}, {-allow-stack-frame-access:T}, true); -).

To say text of (FN - external-file):
 (- FileIO_PrintContents({FN}); say__p = 1; -).

§82. Figures and sound effects. Ditto, but for “Figures.i6t”.

Section SR5/7/2 - Using external resources - Figures and sound effects

To display (F - figure-name), one time only
 (documented at ph_displayf):
 (- DisplayFigure({F}, {phrase options}); -).

To play (SFX - sound-name), one time only
 (documented at ph_playsf):
 (- PlaySound({SFX}, {phrase options}); -).

§83. **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 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):

(- 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".

§84. 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;

§85. **Miscellaneous other phrases.** Again, *these are not part of Inform's public specification.*

Section SR5/9/1 - Miscellaneous other phrases - Unindexed

§86. 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(); -).
```

§87. 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 -).
```

§88. 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(); -).
```

§89. Two ugly little tricks 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); -).
```

To decide if (O - object) goes undescribed by source text:

```
(- ({O}.description == 0) -).
```

§90. 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) -).
```

§91. 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) -).
```

§92. 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},O); -) - 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; -).
```

§93. 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(); -).
```

§94. And so, at last...

The Standard Rules end here.

§95. ...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.