INFORM 7

The Program

Appendix B

Build 6F95   Graham Nelson

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

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

Appendix A: the Standard Rules, written in Inform 7

Appendix B: the template layer, written in Inform 6

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

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

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

B The Template Layer

`B/introt`: *Introduction.i6t* A short introduction to the template and its organisation.

`B/maint`: *Main.i6t* The top-level logic of NI: the sequence of operations followed by NI up to the point where the output file is opened, resuming after it is closed again.

`B/lcore`: *Load-Core.i6t* To load the core language definition for Inform, which means creating the basis for its hierarchy of kinds.

`B/ltims`: *Load-Times.i6t* To load the Times of Day language definition element.

`B/lacts`: *Load-Actions.i6t* To load the Actions language definition element.

`B/lscen`: *Load-Scenes.i6t* To load the Scenes language definition element.

`B/lfigs`: *Load-Figures.i6t* To load the Figures language definition element.

`B/lsnd`: *Load-Sounds.i6t* To load the Sounds language definition element.

`B/lfile`: *Load-Files.i6t* To load the Files language definition element.

`B/outt`: *Output.i6t* This is the superstructure of the file of I6 code output by NI: from ICL commands at the top down to the signing-off comments at the bottom.

`B/defnt`: *Definitions.i6t* Miscellaneous constant definitions, usually providing symbolic names for otherwise inscrutable numbers, which are used throughout the template layer.

`B/ordt`: *OrderOfPlay.i6t* The sequence of events in play: the Main routine which runs the startup rulebook, the turn sequence rulebook and the shutdown rulebook; and most of the I6 definitions of primitive rules in those rulebooks.

`B/actt`: *Actions.i6t* To try actions by people in the model world, processing the necessary rulebooks.

`B/acvt`: *Activities.i6t* To run the necessary rulebooks to carry out an activity.

`B/rbt`: *Rulebooks.i6t* To work through the rules in a rulebook until a decision is made.

`B/parst`: *Parser.i6t* The parser for turning the text of the typed command into a proposed action by the player.

`B/lwt`: *ListWriter.i6t* A flexible object-lister taking care of plurals, inventory information, various formats and so on.

`B/oowt`: *OutOfWorld.i6t* To implement some of the out of world actions.

`B/wmt`: *WorldModel.i6t* Testing and changing the fundamental spatial relations.

`B/light`: *Light.i6t* The determination of light, visibility and physical access.

`B/testt`: *Tests.i6t* The command grammar and I6 implementation for testing commands such as TEST, ACTIONS and PURLOIN.

`B/langt`: *Language.i6t* The fundamental definitions needed by the parser and the verb library in order to specify the language of play – that is, the language used for communications between the story file and the player.

`B/stackt`: *MStack.i6t* To allocate space on the memory stack for frames of variables to be used by rulebooks, activities and actions.

`B/chrt`: *Chronology.i6t* To record information now which will be needed later, when a condition phrased in the perfect tense is tested.

`B/print`: *Printing.i6t* To manage the line skips which space paragraphs out, and to handle the printing of names of objects, pieces of text and numbers.

`B/rtpt`: *RTP.i6t* To issue run-time problem messages, and to perform some run-time type checking which may issue such messages.

`B/utilt`: *Utilities.i6t* Miscellaneous utility routines for some fundamental I6 needs.

`B/numt`: *Number.i6t* Support for parsing integers.

`B/timet`: *Time.i6t* Support for parsing and printing times of day.

`B/tabt`: *Tables.i6t* To read, write, search and allocate rows in the Table data structure.

`B/sortt`: *Sort.i6t* To sort arrays.

`B/relt`: *Relations.i6t* To manage run-time storage for relations between objects, and to find routes through relations and the map.

`B/figst`: *Figures.i6t* To display figures and play sound effects.

`B/inxt`: *IndexedText.i6t* Code to support the indexed text kind of value.

`B/regxt`: *RegExp.i6t* Code to match and replace on regular expressions against indexed text strings.

`B/chart`: *Char.i6t* To decide whether letters are upper or lower case, and convert between the two.

`B/unict`: *UnicodeData.i6t* To tabulate casings in the character set.

`B/stact`: *StoredAction.i6t* Code to support the stored action kind of value.

`B/listt`: *Lists.i6t* Code to support the list of... kind of value constructor.

`B/combt`: *Combinations.i6t* Code to support the combination kind of value constructor.

`B/relkt`: *RelationKind.i6t* Code to support the relation kind.

`B/blkvt`: *BlockValues.i6t* Routines for copying, comparing, creating and destroying block values, and for reading and writing them as if they were arrays.

`B/flext`: *Flex.i6t* To allocate flexible-sized blocks of memory as needed to hold arbitrary-length strings of text, stored actions or other block values.

`B/zmt`: *ZMachine.i6t* To provide routines handling low-level Z-machine facilities.

`B/glut`: *Glulx.i6t* To start up the Glk interface for the Glulx virtual machine, and provide Glulx-specific printing functions.

`B/iot`: *FileIO.i6t* Reading and writing external files, in the Glulx virtual machine only.

Introduction Template B/introt

Purpose

A short introduction to the template and its organisation.

§1. The Software Stack. The term "software stack" is sometimes used to refer to the total body of code running in a computer. As the word "stack" implies, there tends to be a fairly clear division between components, and an architecture in which one component is supported by another. The ground on which the stack rests is the hardware. Then come device drivers, for communicating with hardware, and a kernel of very low-level code to regulate who talks to the hardware and when. On top of that are usually several layers of operating-system facilities and utility programs, and on top of all of that are Firefox, iTunes and other consumer programs which the user has made a conscious decision to run. These top-level programs are visible, while the lower layers are concealed from the user and are generally very reliable, so it is easy to forget that they are there at all.

Similarly, when writing and testing a work with I7, it's easy to think that the only code running is the code directly generated by the source text. In fact, though, it runs inside a simulated computer called the virtual machine (or VM), and the VM has a software stack just as other computers do. Here the distinction between "program" and "operating system" is more blurred, because it can afford to be – the VM only ever has a single program running, and is not connected to any valuable hardware or data, so there is no need to protect the system's integrity from a malicious program, or to protect one program from the failings of another running at the same time. But there is still a recognisable software stack. From top to bottom, the VM at run-time contains:

(a) The rules, text and tables written by the author of this specific work of IF;
(b) Material contributed by Extensions which the author chose to borrow from;
(c) Rules, phrases and other constructions made by the Standard Rules extension, whose use is compulsory;
(d1) Infrastructure for rulebooks and code for maintaining a world model common to all works of IF;
(d2) The code needed to manage complex data structures such as relations, tables, indexed text, and so on;
(e) The interface to the VM, which is abstracted so that higher levels do not need to know whether Glulx or the Z-machine is being used.

Here (d1) and (d2) are independent blocks, so the picture is roughly so:

the player typing at a keyboard
(a) MATERIAL WRITTEN BY THE AUTHOR
(b) EXTENSIONS CHOSEN BY THE AUTHOR
(c) THE STANDARD RULES
(d1) MECHANICS OF PLAY (d2) DATA STRUCTURE SUPPORT
(e) VIRTUAL MACHINE INTERFACE
virtual machine "hardware"

NI is not like a compiler for a conventional program, because it has to conjure up the entire software stack whenever it compiles anything. (Until 2008, NI compiled code which ran on top of the traditional I6 library: nowadays it compiles stand-alone code which never uses the `#Include` directive. While it's true that the I6 compiler does add a very thin extra layer of code itself – the "veneer" – in all other respects NI specifies every single line of I6 code which makes up the eventual story file.)

NI compiles layers (a), (b) and (c) from the I7 source text found in the project file, in the Extensions installed by the user, and in the Standard Rules extension which comes pre-installed. But parts (d1), (d2) and (e) are almost exactly the same I6 code whatever may be sitting above them: they are simply copied, with minor variations, from a large body of standing code installed in the I7 application which is called the "template".

§**2. Segments and Paragraphs.** The template is divided into about 40 individual "segments", each stored in its own file and with the `.i6t` file extensions. This stands for "I6 template", because the code generated by NI is Inform 6 (I6) code. What makes the file a template rather than being raw code is that it is divided into named paragraphs which contain both commentary and also I6 code: when NI turns a template into the output code, the commentary (and each paragraph heading) is stripped out.

See the Inform documentation for how to modify the template in use for any particular project: I6 code can be added before, instead of or after any named segment or paragraph, and in addition, it's possible to replace entire segment files with your own versions stored in the Materials folder for a project.

§**3. Architecture.** To recap, then, the template contributes the following parts of the software stack:

(d1) MECHANICS OF PLAY (d2) DATA STRUCTURE SUPPORT

(e) VIRTUAL MACHINE INTERFACE

A more detailed version of this diagram organises its segments follows:

NI Control. `Main.i6t`, `Load-Core.i6t` (and several others like it), `Output.i6t`, `Definitions.i6t`.

Mechanics of Play I: Rules. `OrderOfPlay.i6t`, `Actions.i6t`, `Activities.i6t`, `Rulebooks.i6t`.

II: Infrastructure. `Parser.i6t`, `ListWriter.i6t`, `OutOfWorld.i6t`, `WorldModel.i6t`, `Light.i6t`, `Tests.i6t`.

III: Basic Services. `Language.i6t`, `MStack.i6t`, `Chronology.i6t`, `Printing.i6t`, `RTP.i6t`, `Utilities.i6t`.

Data Structure Support I: Basics. `Number.i6t`, `Time.i6t`, `Tables.i6t`, `Sort.i6t`, `Relations.i6t`, `Figures.i6t`.

II: Block Values. `IndexedText.i6t`, `RegExp.i6t`, `Char.i6t`, `UnicodeData.i6t`, `StoredAction.i6t`, `Lists.i6t`, `RelationKind.i6t`, `BlockValues.i6t`, `Flex.i6t`.

Virtual Machine Interface. `ZMachine.i6t` *or* `Glulx.i6t` and `FileIO.i6t`.

To elaborate:

NI Control. As well as containing commentary and I6 code, template files can also contain commands to tell NI to do something more interesting than simply copying over material verbatim into its output. Four of the segments use this ability a great deal: the remaining segments hardly use it at all. These four segments therefore don't fit anywhere in the diagram of the software stack above. "Main.i6t" controls the top-level logic of NI and gives the sequence of operations; "Load-.i6t" specifies the basic kinds of value known to NI – numbers, times, texts, rulebooks and so on; "Output.i6t" is essentially the arrangement used to join all the many fragments of I6 from the template and the I7 source text into a single end-to-end I6 program which will become the story file; "Definitions.i6t" defines many named constants which are used across the template.

Mechanics of Play I: Rules. "OrderOfPlay.i6t" is the highest-level description of what happens when a story file runs: it contains the I6 `Main` routine, and definitions of the primitive rules in the most important rulebooks.

"Actions.i6t" and "Activities.i6t" contain code which runs actions and activities, respectively: these sit on top of "Rulebooks.i6t", which performs general rulebook-running.

Mechanics of Play II: Infrastructure. "Parser.i6t" breaks down a command typed by the player into a slate of variables which can be formed into an action. "ListWriter.i6t" is a general-purpose service for printing lists of objects satisfying various descriptions, and formatted in different ways. "OutOfWorld.i6t" provides code to handle out-of-world actions not needing direct access to VM internals: PRONOUNS and SUPERBRIEF, for instance. "Tests.i6t" provides code for the testing commands – TEST, ACTIONS, SHOWME and so forth.

"WorldModel.i6t" contains code for performing object movements and the like in such a way that the world model rules are preserved: it handles component parts, decides touchability and so on. "Light.i6t" determines the level of light and decides on visibility.

Mechanics of Play III: Basic Services. "Language.i6t" provides English-language messages issued by template routines and the Standard Rules during play: replacing this segment is the way to translate I7 story files into other languages of play. (It's the equivalent of the old I6 "language definition files", and retains most of the same structure.) "MStack.i6t" provides a small general-purpose memory stack. "Chronology.i6t" performs the continuous monitoring required to ensure that past-tense conditions work: for instance we can only determine whether or not "the Black Door has been open" if we have spent the whole time so far checking on whether it is open or not, and this is where the checking is done. "Printing.i6t" handles paragraph-breaking, the printing of object names with suitable articles attached, and miscellaneous other printing needs. "RTP.i6t" issues run-time problem messages as needed: these should appear only if the story file does something clearly illegal, and point to bugs not yet removed from the author's code. "Utilities.i6t" provides miscellaneous utility functions at a very low level, such as for unsigned comparison of two numbers.

Data Structure Support I: Basics. Many kinds of value need no maintenance, and little or no support. "Number.i6t" and "Time.i6t" contain code to parse numbers and times of day from text typed by the player, together with a few basic operations: for instance, the rounding off of times of day. "Tables.i6t" provides access to table entries, specified in a variety of direct and indirect ways; this makes use of "Sort.i6t", which abstracts a choice of sorting algorithms for use on tables and other I6 data. "Relations.i6t" provides code for testing and asserting relations: the information about what is related to what else is stored in a variety of different ways, depending on the relation, to make best use of memory. "Figures.i6t" displays figures and plays back sound effects, or rather, passes instructions to do so down to the VM.

Data Structure Support II: Block Values. Kinds of value can be divided into the ordinary ones – number, object, time and so on – together with "block values" such as indexed text, stored action and list. Block values have to be stored in dynamically allocated memory from a heap. It would be wasteful to include all of this code, and to spare memory for the heap, if no block values were actually needed, so the following segments are only compiled if the source text makes specific reference to at least one block value. "IndexedText.i6t" manages character-indexed strings of text, which can shrink or stretch to arbitrary lengths: it makes use of "RegExp.i6t" for regular expression matching and search-and-replace; and also of "Char.i6t" for code to deal with lower and upper casing of letters, and "UnicodeData.i6t" to provide character set details, mechanically converted from the Unicode 4.0 standard. "StoredAction.i6t" manages stored actions: it can convert the current action into a stored one, and also try a long-stored action so that it now takes place. "Lists.i6t" manages the flexibly sized lists produced by values whose kind is list of numbers, list of texts, list of lists of lists of stored actions, and so forth: it can merge, insert, delete, resize, rotate, and reverse lists, and makes use once again of "Sort.i6t" (q.v.) to sort them.

"BlockValues.i6t" provides the basic support for kinds of value which are stored as blocks of memory on the heap.

At the lowest level, "Flex.i6t" manages flexible memory allocation as required by "BlockValues.i6t": it organises the heap of unclaimed memory, for instance, and maks allocations and deallocations when needed.

Virtual Machine Interface. Depending on the Settings used for the I7 project being compiled, we either use "ZMachine.i6t" or "Glulx.i6t": if Glulx, we also add "FileIO.i6t", which provides support for the limited file-handling abilities offered by Glulx.

Properly speaking, there is one further template file: "Introduction.i6t". It provides the commentary you are now reading, but has no other function, contains no code, and is finished now anyway.

Main Template B/maint

Purpose

The top-level logic of NI: the sequence of operations followed by NI up to the point where the output file is opened, resuming after it is closed again.

§**1. Startup.** This segment of the template is not like any other. It contains almost the complete logical sequence of operations followed by NI – indeed, NI essentially works by parsing some command-line arguments to set switches and then asking the template interpreter to run through "Main.i6t", the only template file which must always exist. (The other template files are only ever involved when called on by the `{-segment:...}` command from a template file already running.) Whereas most template segments contain I6 code to pass through into NI's output, this one runs both before and after NI's output file is being written, and contains only commands.

The Startup paragraph is not in fact modifiable in any easy way, because of course "Include..." sentences – used to tell the template interpreter to override the template files – will not be understood until long after this paragraph has been fully dealt with. But perhaps that's no bad thing.

```
{-callv:Memory::start}
{-callv:Text::start}
{-callv:Text::Vocabulary::stock}
{-callv:Config::Plugins::start}
{-callv:World::begin}
{-callv:Extensions::Files::handle_census_mode}
{-progress-stage:0}
{-log-phase:Lexical analysis}
{-callv:Text::Reader::read_primary_source_text}
{-callv:Parser::Sentences::create_standard_csps}
```

§**2. Semantic Analysis.** Similarly: Include... sentences are not read until this paragraph is long forgotten.

```
{-progress-stage:1}
{-log-phase:Semantic analysis Ia}
{-callv:Parser::Nodes::plant_parse_tree} ! Initialise the parse tree
{-callv:Parser::Sentences::break_source} ! Build first tranche of sentences
{-callv:Extensions::Inclusion::traverse} ! Expand extension inclusions and build sentences
{-callv:Parser::Sentences::Headings::satisfy_dependencies}
{-log-phase:Initialise language semantics}
{-plugin:load}
{-callv:Semantics::BPs::make_built_in}
{-callv:Semantics::VerbUsage::stock}
{-callv:Semantics::Quantifiers::make_built_in}
{-log-phase:Semantic analysis Ib}
{-callv:Parser::Sentences::VPs::traverse} ! Find verbs in the assertion sentences
{-callv:Text::traverse_for_plural_definitions} ! Build irregular plurals dictionary
{-callv:Parser::Sentences::tidy_up_ofs_and_froms} ! More "of" wrangling
{-callv:Parser::Sentences::register_recently_lexed_phrases} ! To make commands children of their ro
```

```
  ... utines
{-callv:Parser::Sentences::declare_source_loaded}
{-callv:Kinds::Interpreter::include_templates_for_kinds}
{-log-phase:Semantic analysis II}
{-callv:Parser::Nodes::verify} ! Purely to check that NI is running correctly
{-callv:Extensions::Files::check_versions} ! Do the extension version numbers meet needs?
{-callv:Parser::Sentences::Headings::make_tree} ! Stratify headings and subheadings
{-callv:Parser::Sentences::Headings::write_as_xml} ! Dump them to a file for the GUI to use
{-log-phase:Semantic analysis III}
{-callv:Code::Phrases::Adjectives::traverse}
{-callv:Data::Equations::traverse_to_create}
{-callv:Data::Tables::traverse_to_create}
{-callv:Code::Phrases::traverse_for_names}
```

§3. Assertion Reading. Since Include... sentences are read during pass 2, they might just be able to meddle by adding instructions to take place after this paragraph, but would be too late to work before or instead of it.

```
{-progress-stage:2}
{-log-phase:First pass through assertions}
{-read-assertions:1}
{-log-phase:Second pass through assertions}
{-read-assertions:2}
```

§4. Model Construction.

```
{-log-phase:Making the model world}
{-callv:Properties::Implementation::OfObjects::allocate_attributes}
{-callv:Plugins::Actions::name_all}
{-callv:Data::Nametags::name_all}
{-callv:World::complete}
{-callv:Properties::Measurement::validate_definitions}
{-callv:Semantics::BPs::make_built_in_further}
```

§5. Tables and Grammar.

```
{-log-phase:Tables and grammar}
{-callv:Data::Tables::check_tables_for_kind_clashes}
{-callv:Data::Tables::traverse_to_stock}
{-callv:Data::Equations::traverse_to_stock}
{-callv:Plugins::Parsing::traverse}
{-callv:World::complete_additions}
```

§6. Phrases and Rules.

```
{-progress-stage:3}
{-log-phase:Phrases and rules}
{-callv:Semantics::Nouns::LiteralPatterns::define_named_phrases}
{-callv:Code::Phrases::traverse}
{-callv:Code::Phrases::register_meanings}
{-callv:Code::Phrases::parse_rule_parameters}
{-callv:Code::Phrases::add_rules_to_rulebooks}
{-callv:Code::Phrases::parse_rule_placements}
{-callv:Problems::empty_all_headings}
```

§7. Code Generation. This is where we hand over to regular template files – containing code passed through as I6 source, as well as a few further commands – starting with "Output.i6t".

```
{-progress-stage:4}
{-log-phase:Code generation}
{-open-file}{-segment:Output.i6t}{-close-file}
{-log-phase:Compilation now complete}
```

§8. Metadata.

```
{-callv:Plugins::Bibliographic::write_ifiction_and_blurb}
```

§9. Indexing. This paragraph can be skipped without harming any of the rest of NI's work: the only effect is to suppress the generation of the index. (Indeed, if NI is called with the `-noindex` command-line switch, then the template interpreter ignores all commands in between `{-open-index}` and `{-close-index}`, thus skipping exactly this paragraph.)

```
{-open-index}
{-index:Phrasebook Index=A guide to the phrases allowed; [LEXICON]a lexicon of words; a [RELTABLE]t
  ... able of relations, and [VERBTABLE]of verbs.|What are phrases?<PHRASES>; And descriptions?<D
  ... ESCRIPTIONS>; Relations?<RELATIONS>; Verbs?<VERBS>=The Phrasebook Index}
    {-callv:Code::Phrases::index_page_Phrasebook}
{-index:Kinds Index=Table of all the kinds; [ARITHMETIC]how arithmetic affects them; and [KDETAILS]
  ... details of each kind in turn.|What are kinds?<KINDS>; More about kinds of object<NEWKINDS>;
  ...   And kinds of value<KINDSVALUE>=The Kinds Index}
    {-callv:Data::Objects::page_Kinds}
{-index:World Index=A map of the geographical layout; [MDETAILS]contents of each room; and [LEXICON
  ... ]an A to Z of rooms and things.|What's the map?<MAP>; Using regions<REGIONS>; If the map lo
  ... oks wrong...<MAPHINTS>; Exporting to EPS<EPSMAP>=The World Index}
    {-callv:Data::Objects::page_World}
{-index:Actions Index=The actions (click magnifiers for details); [NAP]kinds of action; [COMMANDS]c
  ... ommands A to Z; [LEXICON]actions A to Z; [ARULES]the applicable rules.|What are actions?<AC
  ... TIONS>; Creating new actions<NEWACTIONS>; Out of world actions (in red)<OUTOFWORLD>=The Ac
  ... tions Index}
    {-callv:Plugins::Actions::Index::page}
{-index:Rules Index=Rules not directly tied to actions (see the Actions Index) or scenes (the Scene
  ... s Index).|What are rulebooks?<RULEBOOKS>; What are activities?<ACTIVITIES>; Moving or aboli
  ... shing rules<RLISTING>=The Rules Index}
    {-callv:Code::Rulebooks::index_page}
```

```
{-index:Contents Index=Headings and [EXTLIST]extensions; [NAMES]named values, Tables and so on; the
  ...    [LCARD]Library Card; the [STORYFILE]story file.|How do headings work?<HEADINGS>; What's th
  ...  e Library Card?<LCARDS>=The Contents Index}
    {-callv:Parser::Sentences::Headings::index}
    {-callv:Extensions::Files::index}
    {-callv:Extensions::Files::update_census}
    {-callv:Data::NonlocalVariables::index_all}
    {-callv:Data::Tables::index}
    {-callv:Data::Equations::index}
    {-callv:Plugins::Figures::index_all}
    {-callv:Plugins::Sounds::index_all}
    {-callv:Plugins::Files::index_all}
    {-callv:Plugins::Bibliographic::index_library_card}
{-index:Scenes Index=A map of how the scenes begin and end; [SEVENTS]timed events, if any; [SRULES]
  ...  general rules about scenes; and [SDETAILS]details of each scene in turn.|What are scenes?<S
  ...  CENESINTRO>; How they link together<LINKINGSCENES>; About timed events<TIMEDEVENTS>=The Sc
  ...  enes Index}
    {-callv:Plugins::Scenes::index}
{-callv:Index::complete}
{-close-index}
```

§**10. Shutdown.** Closing the problems report, making any final reports to the debugging log, and freeing all allocated memory: and that's then it.

```
{-callv:Config::Template::report_unacted_upon_interventions}
{-callv:Problems::complete_problems_report}
{! -callv:Specifications::Taxa::report_pairs_observed}
{! -callv:Specifications::Taxa::report_pairs_allowed}
{! -callv:Memory::log_statistics}
{-callv:Index::log_statistics}
{! -callv:Parser::SParser::debug_parser_statistics}
{! -callv:Semantics::VerbUsage::log_all}
{-callv:Memory::free_memory}
```

Load Core Template B/Icore

Purpose

To load the core language definition for Inform, which means creating the basis for its hierarchy of kinds.

§**1. Header.** The following mass of type definitions is not heavily subdivided into paragraphs since it's read so early in Inform's run that "Include..." sentences haven't had time to be read yet – so they aren't easy to customise. (Except by putting a whole customised version of the file into the relevant `Materials/I6T` folder.)

The `{-lines:type}` command causes all subsequent lines to be sent as parameters to the `{-type:...}` command, until the next `{-endlines}` command.

```
{-callv:Specifications::Taxa::make_type_IDs_table}
{-lines:type}
#DEFAULTS:
defined-in-source-text:no
is-incompletely-defined:no
uses-signed-comparisons:no
can-coincide-with-property:no
named-values-created-with-assertions:no
has-i6-GPR:no
multiple-block:no
constant-compilation-method:none
#KIND-VARIABLE:
group:1
constant-compilation-method:none
i6-printing-routine:DA_Number
i6-printing-routine-actions:DA_Number
index-priority:0
#KIND-OF-KIND:
group:2
constant-compilation-method:none
i6-printing-routine:DA_Number
i6-printing-routine-actions:DA_Number
index-priority:0
#BASE-KIND:
group:3
instance-of:VALUE_TY
index-priority:3
#KIND-CONSTRUCTOR:
group:4
instance-of:VALUE_TY
index-priority:7
```

§**2. Macros.** New kinds of value cause some I7 source to be generated, following various combinations of the macros below. This used to be a mechanism to get around the lack of support for generic phrases in Inform, and since generics are now part of the design, macros like this are used just to create an awkward family of variables which circumvent the typelessness of the I6 parser:

```
*UNDERSTOOD-VARIABLE:
<kind> understood is a <kind> which varies.
*END
```

§**3. Group 1.** These are kinds used in type-checking but not representing specific kinds at run-time. The + notation indicates a type name which NI requires us to construct, and with this specific name.

```
+VALUE_TY:
apply-macro:#KIND-OF-KIND
singular:value
plural:values
! Matches any value at all
+POINTER_VALUE_TY:
apply-macro:#KIND-OF-KIND
singular:pointer value
plural:pointer values
! Matches any value whose runtime representation is a pointer to a block
+WORD_VALUE_TY:
apply-macro:#KIND-OF-KIND
singular:word value
plural:word values
! Matches any value whose runtime representation is a single word
+ARITHMETIC_VALUE_TY:
apply-macro:#KIND-OF-KIND
singular:arithmetic value
plural:arithmetic values
+ENUMERATED_VALUE_TY:
apply-macro:#KIND-OF-KIND
singular:enumerated value
plural:enumerated values
SAYABLE_VALUE_TY:
apply-macro:#KIND-OF-KIND
singular:sayable value
plural:sayable values
COMBINED_VALUE_TY:
apply-macro:#KIND-OF-KIND
singular:combined value of any arity
plural:combined values of any arity
```

§**4. Group 2.** Data types.

```
+OBJECT_TY:
apply-macro:#BASE-KIND
singular:object
plural:objects

instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:nothing

i6-printing-routine:PrintShortName
i6-printing-routine-actions:DA_Name
constant-compilation-method:special
can-coincide-with-property:no

description:an object
index-priority:1
index-default-value:nothing
specification-text:Objects are values intended to simulate physical things: places, people, things,
  ...    and so on. They come in many kinds. The special value 'nothing' is also allowed, and can
  ...  be used to mean 'no object at all'.

+NUMBER_TY:
apply-macro:#BASE-KIND
singular:number
plural:numbers

instance-of:WORD_VALUE_TY
instance-of:ARITHMETIC_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:0
uses-signed-comparisons:yes
can-exchange:yes

has-i6-GPR:yes
explicit-i6-GPR:DECIMAL_TOKEN
i6-printing-routine-actions:DA_Number
constant-compilation-method:literal

description:a number
documentation-reference:kind_number
index-priority:2
index-default-value:0
specification-text:Whole number in the range -32768, -32767, ..., -2, -1, 0, 1, 2, 3, ..., 32767: s
  ...  mall numbers can be written textually as 'one', 'two', 'three', ..., 'ten', 'eleven', 'twe
  ...  lve'. (A much larger number range is allowed if we compile the source to Glulx rather than
  ...    the Z-machine: see the Settings panel.)

+TRUTH_STATE_TY:
apply-macro:#BASE-KIND
singular:truth state
plural:truth states

instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:false
can-exchange:yes

loop-domain-schema:for (*1=0: *1<=1: *1++)
```

```
has-i6-GPR:yes
explicit-i6-GPR:TRUTH_STATE_TOKEN
i6-printing-routine:DA_TruthState
i6-printing-routine-actions:DA_TruthState
constant-compilation-method:literal

description:something which is either true or false
documentation-reference:kind_truthstate
index-default-value:false
specification-text:The state of whether something is 'true' or 'false'. (In other computing languag
  ...  es, this might be called 'boolean', after the 19th-century logician George Boole, who firs
  ...  t realised this was a kind of value.)

+TEXT_TY:
apply-macro:#BASE-KIND
singular:text
plural:texts

instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:EMPTY_TEXT_VALUE

i6-printing-routine:PrintText
constant-compilation-method:special

description:some text
index-default-value:""
specification-text:Some text in double quotation marks, perhaps with substitutions written in squar
  ...  e brackets.

+INDEXED_TEXT_TY:
apply-macro:#BASE-KIND
singular:indexed text
plural:indexed texts

instance-of:POINTER_VALUE_TY
instance-of:SAYABLE_VALUE_TY
has-native-constants:no
multiple-block:yes
heap-size-estimate:256
cast:TEXT_TY
cast:SNIPPET_TY
can-exchange:yes

recognition-only-GPR:INDEXED_TEXT_TY_ROGPR
distinguisher:INDEXED_TEXT_TY_Distinguish
i6-printing-routine:INDEXED_TEXT_TY_Say
constant-compilation-method:none

description:an indexed text
documentation-reference:kind_indexedtext
index-default-value:""
specification-text:A flexible-length form of text which can be internally altered and searched. Inf
  ...  orm automatically changes text to this format when necessary.

+UNICODE_CHARACTER_TY:
apply-macro:#BASE-KIND
singular:unicode character
plural:unicode characters

instance-of:WORD_VALUE_TY
```

```
instance-of:SAYABLE_VALUE_TY
default-value:32

constant-compilation-method:literal

description:a Unicode character
index-default-value:unicode 32 (<i>a space</i>)
specification-text:A single character - a letter or item of punctuation.

+USE_OPTION_TY:
apply-macro:#BASE-KIND
singular:use option
plural:use options

instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:0

loop-domain-schema:for (*1=0: *1<NO_USE_OPTIONS: *1++)
i6-printing-routine:PrintUseOption
constant-compilation-method:special

description:the name of a use option
index-priority:6
index-default-value:the ineffectual option
specification-text:One of the optional ways to configure Inform, such as the 'authorial modesty opt
  ...  ion'.

+SNIPPET_TY:
apply-macro:#BASE-KIND
singular:snippet
plural:snippets

instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:101

i6-printing-routine:PrintSnippet
constant-compilation-method:none

description:a snippet
documentation-reference:kind_snippet
index-default-value:<i>word 1 of command</i>
specification-text:A fragment of the player's most recent typed command, taking in a run of consecu
  ...  tive words.

+TABLE_TY:
apply-macro:#BASE-KIND
singular:table name
plural:table names

instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:TheEmptyTable

loop-domain-schema:for (*2=0, *1=TableOfTables-->*2: *1: *2++, *1=TableOfTables-->*2)
i6-printing-routine:PrintTableName
constant-compilation-method:special

description:the name of a table
documentation-reference:kind_tablename
index-priority:6
index-default-value:<i>a table with no rows or columns</i>
```

```
indexed-grey-if-empty:yes
specification-text:Like tables of information in a book or newspaper, tables in Inform hold values
  ...  which have been organised into rows and columns. A table name is just a single value, iden
  ...  tifying which table is meant - say, 'Table of US Presidents' might be a table name value.

+EQUATION_TY:
apply-macro:#BASE-KIND
singular:equation name
plural:equation names

instance-of:WORD_VALUE_TY
default-value:0

i6-printing-routine:DA_Number
constant-compilation-method:special

description:the name of an equation
index-priority:6
index-default-value:<i>an equation doing nothing</i>
indexed-grey-if-empty:yes
specification-text:Like formulae in a textbook or a scientific paper, equations in Inform are writt
  ...  en out in displayed form and given names.

+RULEBOOK_OUTCOME_TY:
apply-macro:#BASE-KIND
singular:rulebook outcome
plural:rulebook outcomes

instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY

i6-printing-routine:RulebookOutcomePrintingRule
constant-compilation-method:special

description:the name of a rulebook outcome
index-priority:0

+UNDERSTANDING_TY:
apply-macro:#BASE-KIND
singular:topic
plural:topics

instance-of:WORD_VALUE_TY
cast:TEXT_TY
default-value:DefaultTopic

i6-printing-routine-actions:DA_Topic
constant-compilation-method:special

description:a topic
index-priority:0

+INTERMEDIATE_TY:
apply-macro:#BASE-KIND

instance-of:WORD_VALUE_TY
instance-of:ARITHMETIC_VALUE_TY

i6-printing-routine:DA_Number
i6-printing-routine-actions:DA_Number

description:the intermediate result of some complex calculation
index-priority:0
! Represents intermediate results of arithmetic in dimensional formulae
```

```
+NIL_TY:
apply-macro:#BASE-KIND
instance-of:WORD_VALUE_TY
description:nothing
index-priority:0
! Represents a lack of arguments, or of results

+KIND_VARIABLE_TY:
apply-macro:#KIND-VARIABLE
description:a variable denoting a kind
```

§**5. Group 3.** Constructors.

```
+PHRASE_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:phrase k -> l
plural:phrases k -> l
constructor-arity:contravariant list, covariant optional
instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
i6-printing-routine:SayPhraseName
constant-compilation-method:special
index-default-value:<i>always the default value of L</i>

+TUPLE_ENTRY_TY:
apply-macro:#KIND-CONSTRUCTOR
constructor-arity:covariant, covariant
instance-of:WORD_VALUE_TY
index-priority:0
! Represents an entry in a multiple

+RELATION_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:relation | relation of k to l | relation of k
plural:relations | relations of k to l | relations of k
constructor-arity:covariant, covariant
instance-of:POINTER_VALUE_TY
instance-of:SAYABLE_VALUE_TY
has-native-constants:no
multiple-block:yes
heap-size-estimate:8
can-exchange:yes
constant-compilation-method:special
distinguisher:RELATION_TY_Distinguish
i6-printing-routine:RELATION_TY_Say
description:a relation of K to L
index-default-value:<i>a relation never holding</i>

+RULE_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:rule | k based rule | rule producing l | k based rule producing l
plural:rules | k based rules | rules producing l | k based rules producing l
```

```
constructor-arity:contravariant optional, covariant optional
instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
cast:RULEBOOK_TY
default-value:LITTLE_USED_DO_NOTHING_R
i6-printing-routine:RulePrintingRule
constant-compilation-method:special
description:the name of a rule
documentation-reference:kind_rule
index-default-value:the little-used do nothing rule
indexed-grey-if-empty:yes
specification-text:One of many, many rules which determine what happens during play. Rules can be t
  ... riggered by scenes beginning or ending, by certain actions, at certain times, or in the co
  ... urse of carrying out certain activities.
+RULEBOOK_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:rulebook | k based rulebook | rulebook producing l | k based rulebook producing l
plural:rulebooks | k based rulebooks | rulebooks producing l | k based rulebooks producing l
constructor-arity:contravariant optional, covariant optional
instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value: 0
constant-compilation-method:special
i6-printing-routine:RulePrintingRule
description:the name of a rulebook
documentation-reference:kind_rulebook
index-default-value:the action-processing rules
specification-text:A list of rules to follow, in sequence, to get something done. A rulebook is lik
  ... e a ring-binder, with the individual rules as sheets of paper. Inform normally sorts these
  ...   into their 'natural' order, with the most specific rules first, but it's easy to shuffle
  ... the pages if you need to. When some task is carried out during play, Inform is normally wo
  ... rking through a rulebook, turning the pages one by one.
+ACTIVITY_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:activity | activity on k
plural:activities | activities on k
constructor-arity:contravariant optional
instance-of:WORD_VALUE_TY
default-value:PRINTING_THE_NAME_ACT
constant-compilation-method:special
description:an activity
documentation-reference:kind_activity
index-default-value:printing the name
specification-text:An activity is something which Inform does as part of the mechanics of play - fo
  ... r instance, printing the name of an object, which Inform often has to do. An activity can
  ... happen by itself ('printing the banner text', for instance) or can be applied to an object
  ...   ('printing the name of something', say).
+LIST_OF_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:list of k
```

```
plural:lists of k
constructor-arity:covariant

instance-of:POINTER_VALUE_TY
instance-of:SAYABLE_VALUE_TY
has-native-constants:no
multiple-block:yes
heap-size-estimate:256
can-exchange:yes

constant-compilation-method:special
distinguisher:LIST_OF_TY_Distinguish
i6-printing-routine:LIST_OF_TY_Say

description:a list of
documentation-reference:kind_listof
index-default-value:{ }
specification-text:A flexible-length list of values, where all of the items have to have the same k
  ...  ind of value as each other - for instance, a list of rooms, or a list of lists of numbers.
  ...    The empty list, with no items yet, is written { }, and a list with items in is written wi
  ...  th commas dividing them - say {2, 5, 9}.

+DESCRIPTION_OF_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:description of k
plural:descriptions of k
constructor-arity:covariant

instance-of:WORD_VALUE_TY
default-value:Prop_Falsity

constant-compilation-method:special

description:a description of
documentation-reference:kind_description
index-default-value:<i>matching nothing</i>
specification-text:A description of a set of values, where all of the items have to have the same k
  ...  ind of value as each other - for instance, 'even numbers' or 'open doors which are in ligh
  ...  ted rooms'.

+PROPERTY_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:property | k valued property
plural:properties | k valued properties
constructor-arity:covariant

instance-of:WORD_VALUE_TY

constant-compilation-method:special
i6-printing-routine:PROPERTY_TY_Say

description:a property of something

+TABLE_COLUMN_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:table column | k valued table column
plural:table columns | k valued table columns
constructor-arity:covariant

instance-of:WORD_VALUE_TY

constant-compilation-method:special

description:the name of a column
```

```
+COMBINATION_TY:
apply-macro:#KIND-CONSTRUCTOR
singular:combination | combination k | combination k and l
plural:combinations | combinations k | combinations k and l
constructor-arity:covariant list, covariant list
instance-of:POINTER_VALUE_TY
instance-of:COMBINED_VALUE_TY
instance-of:SAYABLE_VALUE_TY
has-native-constants:no
multiple-block:yes
heap-size-estimate:256
can-exchange:yes
constant-compilation-method:special
distinguisher:COMBINATION_TY_Distinguish
i6-printing-routine:COMBINATION_TY_Say
description:a list of
documentation-reference:kind_listof
index-priority:0
index-default-value:{ }
specification-text:A way to combine a fixed small number of values, of possibly different kinds, to
  ... gether.
```

§**6. Groups 4 and 5.** Enumerations and units.

```
! New kinds of value are initially given these settings:
#NEW:
apply-macro:#BASE-KIND
instance-of:WORD_VALUE_TY
is-incompletely-defined:yes
named-values-created-with-assertions:yes
can-coincide-with-property:yes
defined-in-source-text:yes
description:a designed type
! When the source text specifies either a named constant value, or a literal
! pattern, it decides whether the new type is to be an enumeration or a unit,
! at which point one of the following macros is applied to the type:
#ENUMERATION:
instance-of:ENUMERATED_VALUE_TY
instance-of:SAYABLE_VALUE_TY
is-incompletely-defined:no
named-values-created-with-assertions:yes
default-value:1
index-default-value:<first-constant>
index-priority:5
has-i6-GPR:yes
uses-signed-comparisons:yes
can-exchange:yes
description:a designed type
constant-compilation-method:quantitative
apply-template:*UNDERSTOOD-VARIABLE
#UNIT:
```

```
instance-of:ARITHMETIC_VALUE_TY
instance-of:SAYABLE_VALUE_TY
is-incompletely-defined:no
uses-signed-comparisons:yes
can-exchange:yes
defined-in-source-text:yes
named-values-created-with-assertions:no
default-value:0
index-default-value:<0-in-literal-pattern>
index-priority:2
has-i6-GPR:yes
constant-compilation-method:literal
apply-template:*UNDERSTOOD-VARIABLE
```

§7. Tail.

```
{-endlines}
{-callv:Kinds::Interpreter::batch_done}
```

Load Times Template B/Itims

Purpose

To load the Times of Day language definition element.

§1. Data type definitions.

```
{-lines:type}
TIME_TY:
apply-macro:#BASE-KIND
singular:time
plural:times
instance-of:WORD_VALUE_TY
instance-of:ARITHMETIC_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:540
uses-signed-comparisons:yes
can-exchange:yes
loop-domain-schema:for (*1=0: *1<TWENTY_FOUR_HOURS: *1++)
has-i6-GPR:yes
explicit-i6-GPR:TIME_TOKEN
i6-printing-routine:PrintTimeOfDay
i6-printing-routine-actions:PrintTimeOfDay
constant-compilation-method:literal
description:a time
documentation-reference:kind_time
index-default-value:9:00 AM
index-minimum-value:1 minute
index-maximum-value:23 hours 59 minutes
index-priority:2
specification-text:A time of day, written in the form '2:34 AM' or '12:51 PM', or a length of time
  ...  such as '10 minutes' or '3 hours 31 minutes', which must be between 0 minutes and 23 hours
  ...    59 minutes inclusive.
{-endlines}
{-callv:Kinds::Interpreter::batch_done}
```

Load Actions Template B/lacts

Purpose

To load the Actions language definition element.

§1. Data type definitions.

```
{-lines:type}
DESCRIPTION_OF_ACTION_TY:
apply-macro:#BASE-KIND
instance-of:WORD_VALUE_TY
constant-compilation-method:special
description:a stored action
index-priority:0
! To represent an action test implicitly cast as a conditional rvalue

STORED_ACTION_TY:
apply-macro:#BASE-KIND
singular:stored action
plural:stored actions
instance-of:POINTER_VALUE_TY
instance-of:SAYABLE_VALUE_TY
comparison-schema:DESCRIPTION_OF_ACTION_TY>>>*=-((STORED_ACTION_TY_Adopt(*1), SAT_Tmp-->0=(*2), STO
  ...  RED_ACTION_TY_Unadopt()))
has-native-constants:yes
multiple-block:no
heap-size-estimate:16
can-exchange:yes
distinguisher:STORED_ACTION_TY_Distinguish
i6-printing-routine:STORED_ACTION_TY_Say
constant-compilation-method:special
description:a stored action
documentation-reference:kind_storedaction
index-default-value:waiting
specification-text:A stored action, which can later be tried.

ACTION_NAME_TY:
apply-macro:#BASE-KIND
singular:action name
plural:action names
instance-of:WORD_VALUE_TY
instance-of:SAYABLE_VALUE_TY
default-value:##Wait
loop-domain-schema:for (*2=0,*1=ActionNumberIndexed(*2): *2<AD_RECORDS: *2++,*1=ActionNumberIndexed
  ...  (*2))
i6-printing-routine:SayActionName
constant-compilation-method:special
description:an action name
```

```
documentation-reference:kind_actionname
index-priority:6
index-default-value:waiting action
specification-text:An action is what happens when one of the people in the simulated world decides
  ...  to do something. A full action would be something like 'dropping the box', but an action n
  ...  ame is just the choice of which sort of thing is being done: here, it's 'the dropping acti
  ...  on'. (Action names are always written with the word 'action' at the end, to make sure they
  ...    aren't mistaken for full actions.)
{-endlines}
{-callv:Kinds::Interpreter::batch_done}
```

Load Scenes Template B/Iscen

Purpose

To load the Scenes language definition element.

§1. Data type definitions.

```
{-lines:type}
SCENE_TY:
apply-macro:#BASE-KIND
singular:scene
plural:scenes
instance-of:WORD_VALUE_TY
instance-of:ENUMERATED_VALUE_TY
instance-of:SAYABLE_VALUE_TY
named-values-created-with-assertions:yes
default-value: 0
loop-domain-schema:for (*1=1: *1<=NUMBER_SCENES_CREATED: *1++)
has-i6-GPR:yes
i6-printing-routine:PrintSceneName
constant-compilation-method:quantitative
apply-template:*UNDERSTOOD-VARIABLE
description:a scene
documentation-reference:kind_scene
index-priority:4
index-default-value:the Entire Game
indexed-grey-if-empty:yes
specification-text:Like a scene in a play: a period of time which is usually tied to events in the
  ...  plot. Scenes are created by sentences like 'Midnight Arrival is a scene.'
{-endlines}
{-callv:Kinds::Interpreter::batch_done}
```

Load Figures Template B/lfigs

Purpose

To load the Figures language definition element.

§1. Data type definitions.

```
{-lines:type}
FIGURE_NAME_TY:
apply-macro:#BASE-KIND
singular:figure name
plural:figure names
instance-of:WORD_VALUE_TY
instance-of:ENUMERATED_VALUE_TY
instance-of:SAYABLE_VALUE_TY
named-values-created-with-assertions:yes
default-value:1
i6-printing-routine:PrintFigureName
has-i6-GPR:yes
apply-template:*UNDERSTOOD-VARIABLE
constant-compilation-method:quantitative
description:the name of a figure
documentation-reference:kind_figurename
index-priority:6
index-default-value:figure of cover
indexed-grey-if-empty:yes
specification-text:When made with the Glulx setting, an Inform project can include images as well a
  ...  s words, and these are called figures. A figure name is just the name of one of the figure
  ...  s in the current project.
{-endlines}
{-callv:Kinds::Interpreter::batch_done}
```

Load Sounds Template B/lsnd

Purpose

To load the Sounds language definition element.

§1. Data type definitions.

```
{-lines:type}
SOUND_NAME_TY:
apply-macro:#BASE-KIND
singular:sound name
plural:sound names
instance-of:WORD_VALUE_TY
instance-of:ENUMERATED_VALUE_TY
instance-of:SAYABLE_VALUE_TY
named-values-created-with-assertions:yes
default-value:0
i6-printing-routine:PrintSoundName
has-i6-GPR:yes
apply-template:*UNDERSTOOD-VARIABLE
constant-compilation-method:quantitative
description:the name of a sound effect
documentation-reference:kind_soundname
index-priority:6
index-default-value:<i>a silent non-sound</i>
indexed-grey-if-empty:yes
specification-text:When made with the Glulx setting, an Inform project can include sound effects or
  ...    pieces of music. A sound name is just the name of one of these sounds in the current proj
  ...  ect.
{-endlines}
{-callv:Kinds::Interpreter::batch_done}
```

Load Files Template B/lfile

Purpose

To load the Files language definition element.

§1. Data type definitions.

```
{-lines:type}
EXTERNAL_FILE_TY:
apply-macro:#BASE-KIND
singular:external file
plural:external files
instance-of:WORD_VALUE_TY
instance-of:ENUMERATED_VALUE_TY
instance-of:SAYABLE_VALUE_TY
named-values-created-with-assertions:yes
default-value:0
i6-printing-routine:PrintExternalFileName
has-i6-GPR:yes
apply-template:*UNDERSTOOD-VARIABLE
constant-compilation-method:quantitative
description:the name of a file
documentation-reference:kind_externalfile
index-priority:6
index-default-value:<i>a non-file</i>
indexed-grey-if-empty:yes
specification-text:When made with the Glulx setting, an Inform project can make limited use of file
  ...  s stored on the computer which is operating the story at run-time. An external-file is jus
  ...  t the name of one of these files (not the filename in the usual sense, but a name given to
  ...    it in the Inform source text).
{-endlines}
{-callv:Kinds::Interpreter::batch_done}
```

Output Template B/outt

Purpose

This is the superstructure of the file of I6 code output by NI: from ICL commands at the top down to the signing-off comments at the bottom.

§**1. ICL Commands.** The Inform Control Language is a mini-language for controlling the I6 compiler, able to set command-line switches, memory settings and so on. I6 ordinarily discards lines beginning with exclamation marks as comments, but at the very top of the file, lines beginning `!%` are read as ICL commands: as soon as any line (including a blank line) doesn't have this signature, I6 exits ICL mode. So, basically, we'd better do this here:

```
{-call:Config::Inclusions::compile_icl_commands}
```

§**2. Other Configuration.** Setting the `Grammar__Version` constant is a rather creaky old way to tell the I6 compiler to use more detailed grammar tables: GV1 has not in fact been used since about 1994.

```
Constant Grammar__Version 2;
```

§**3. Identification.** Both of the compiler and template layer, and of the story file to be produced.

The "library" identifying texts are now a little anachronistic, since the template layer is not strictly speaking an I6 library in the same way as the original: but it is surely the spiritual successor to 6/11N, so we may as well mark it 6/12N.

```
! This file was compiled by Inform 7: the build number and version of the
! I6 template layer used are as follows.
{-call:Config::Template::compile_build_number}
Constant LibSerial = "080126";
Constant LibRelease = "6/12N";
Constant LIBRARY_VERSION = 612;
{-call:Config::Plugins::define_IFDEF_symbols}
{-array:Plugins::Bibliographic::UUID_ARRAY}
{-call:Plugins::Bibliographic::compile_constants}
Default Story 0;
Default Headline 0;
{-routine:Extensions::Files::ShowExtensionVersions}
```

§**4. Use options.** Use options are translated into I6 constant declarations, and NI puts them here:

```
{-call:Config::Inclusions::UseOptions::compile}
```

§5. Constants.

```
{-segment:Definitions.i6t}
```

§6. Global Variables. These are not the only global variables defined in the template layer: those needed locally only by single sections (and not used in definitions of phrases in the Standard Rules, or referred to by NI directly) are defined within those sections – they can be regarded as unimportant implementation details, subject to change at whim. The variables here, on the other hand, are more important to understand.

(1) The first three variables to be defined are special in that they are significant to very early-style Z-machine interpreters, where they are used to produce the status line display (hence `sline1` and `sline2`). The first variable must always equal a valid object number, which is why we – pretty weirdly – set it equal to the placeholder object `InformLibrary`, which takes no part in play, and is not a valid I7 object. This is not typesafe in I7 terms, but that doesn't matter because initialisation will correct it to a typesafe value before any I7 source text can execute. (`sline1` and `sline2` are entirely unused on when we target Glulx.) Once these variables are defined, the sequence of definition of the rest is not significant.

(2) The `say__*` are used for the finite state machine used in printing text, which keeps track of automatic paragraph breaking and the like. For details see the "Printing.i6t" section. For `subst__v`, see the explanation of "substitution-variable" in the Standard Rules. The variables `ct_0` and `ct_1` hold the current choice of table and table row, respectively, but in some ways these global variables are misleading: in very many routines, `ct_0` and `ct_1` are defined as local variables, and it's the local definitions which take precedence. (The global variables allow table-selecting code to compile in routines where it isn't possible to define these locals because no stack frame exists.)

(3) `standard_interpreter` is used only for the Z-machine VM, and is always 0 for Glulx. For Z, a non-zero value here is the version number of the *Z-Machine Standards Document* which the interpreter claims to support, in the form (upper byte).(lower). `undo_flag`, similarly, behaves slightly differently on the two platforms according to whether they support multiple consecutive UNDOs. UNDO basically works by taking memory snapshots of the whole VM ("saving UNDO") to revert to at a later point ("performing UNDO"), so it is expensive on memory, and traditional VMs can only store a single memory snapshot – making two UNDOs in a row, going back two steps, impossible. Given this, `undo_flag` has three possible states: 0 means UNDO is not available at all, 1 means it is not available now because there is no further saved state to go back to from here, and 2 means it is available.

(4) `deadflag` in normally 0, or false, meaning play continues; 1 means the game ended in death; 2 for ended in victory; higher numbers represent exotic endings. (As from May 2010, the use of 2 for victory is deprecated, and a separate flag, `story_complete`, records whether the story is "complete" in the sense that we don't expect the player to replay.) `deadflag` switching state normally triggers an end to rulebook processing, so is the single most important global variable to the running of a story file.

(5) At present, `time_rate` is not made use of in I7: if positive, it is the number of minutes which pass each turn; if negative, the number of turns which pass each minute. This is quite a neat way to approximate a wide range of time steps with an integer such that fractions are exact and we can approximate any duration to a fair accuracy (the worst case being 3/4 minute, where we have to choose between 1 minute or 1/2 minute).

(6) Note that `notify_mode` is irrelevant if the use option "Use no scoring" is in force: it isn't looked at, and can't be changed, and shouldn't have an effect anyway since `score` will never be altered.

(7) `player` is a variable, not a constant, since the focus of play can change. `SACK_OBJECT` is likewise an unexpected variable: in the I6 library, there could only be one player's holdall, a single rucksack-like possession which had to be the value of the constant `SACK_OBJECT`. Here we define `SACK_OBJECT` as a global variable instead, the value of which is the player's holdall currently in use. `visibility_ceiling` is the highest object in the tree visible from the player's point of view: usually the room, but sometimes nothing (in darkness), and sometimes a closed non-transparent container.

(8) See "OrderOfPlay.i6t" for the meaning of action variables.

(9) This is a slate of global variables used by the parser to give some context to the general parsing routines (GPRs) which it calls; in the I6 design, any object can provide its own GPR, in the form of a `parse_name` property. GPRs are in effect parser plug-ins, and I7 makes extensive use of them.

(10) Similarly, variables for the parser to give context to another sort of plug-in routine: a scope filter. I7 uses these too.

(11) The `move_*` variables are specific to the `##Going` action.

(12) These variables hold current settings for listing objects and, more elaborately, performing room descriptions.

(13) The current colour scheme is stored in variables in order that it can be saved in the save game state, and changed correctly on an UNDO: if it were a transient state inside the VM interpreter's screen model, then a RESTORE or UNDO will upset what the original author may have intended the appearance of text in particular scenes to be. (Cf. Adam Cadre's I6 patch `L61007`.)

(14) These pixel dimensions are used both for the Glulx and v6 Z-machines, but not for the more commonly used versions 5 or 8, whose screen model is based on character cells.

(15) Parameters used by `LibraryMessages`: this may be replaced later.

(16) For debugging. `debug_flag` is traditionally called so, but is actually now a bitmap of flags for tracing actions, calls to object routines, and so on.

```
! [1]
Global location = InformLibrary; ! does not = I7 "location": see below
Global sline1; Global sline2;
! [2]
Global say__p = 1; Global say__pc = 0; Global say__n;
Global ct_0 = 0; Global ct_1 = 0;
Global los_rv = false;
Global subst__v; ! = I7 "substitution-variable"
Global parameter_object; ! = I7 "parameter-object" = I7 "container in question"
Array deferred_calling_list --> 16;
Global property_to_be_totalled; ! used to implement "total P of..."
Global property_loop_sign; ! $+1$ for increasing order, $-1$ for decreasing
Global suppress_scope_loops;
Global temporary_value; ! can be used anywhere side-effects can't occur
Global enable_rte = true; ! reporting of run-time problems is enabled
Constant BLOCKV_STACK_SIZE = 224;
Global blockv_sp = 0;
Array blockv_stack --> BLOCKV_STACK_SIZE;
Global IT_RE_Err = 0;
Array LocalParking --> 16;
! [3]
Global standard_interpreter = 0;
Global undo_flag;
! [4]
Global deadflag = 0;
Global story_complete = 0;
Global resurrect_please = false;
! [5]
Global not_yet_in_play = true; ! set false when first command received
Global turns = 1; ! = I7 "turn count"
Global the_time = NULL; ! = I7 "time of day"
Global time_rate = 1;
```

```
Constant NUMBER_SCENES_CREATED = {-value:NUMBER_CREATED(scene)};
Constant SCENE_ARRAY_SIZE = (NUMBER_SCENES_CREATED+2);
Array scene_started --> SCENE_ARRAY_SIZE;
Array scene_ended --> SCENE_ARRAY_SIZE;
Array scene_status --> SCENE_ARRAY_SIZE;
Array scene_endings --> SCENE_ARRAY_SIZE;
Array scene_latest_ending --> SCENE_ARRAY_SIZE;
{-array:Plugins::Scenes::scene_recurs}

! [6]
Global score; ! = I7 "score"
Global last_score; ! = I7 "last notified score"
Global notify_mode = 1; ! score notification on or off
Global left_hand_status_line = SL_Location; ! = I7 "left hand status line"
Global right_hand_status_line = SL_Score_Moves; ! = I7 "right hand status line"

! [7]
Global player; ! = I7 "player"
Global real_location; ! = I7 "location"
Global visibility_ceiling; ! highest object in tree visible to player
Global visibility_levels; ! distance in tree to that

Global SACK_OBJECT; ! current player's holdall item in use

! [8]
Global act_requester;
Global actor; ! = I7 "person asked" = I7 "person reaching"
Global actors_location; ! like real_location, but for the actor
Global actor_location; ! = I7 "actor-location"
Global action;
Global meta; ! action is out of world
Global inp1;
Global inp2;
Array  multiple_object --> MATCH_LIST_WORDS; ! multiple-object list (I6 table array)
Global toomany_flag; ! multiple-object list overflowed
Global multiflag; ! multiple-object being processed
Global multiple_object_item; ! item currently being processed in multiple-object list
Global noun; ! = I7 "noun"
Global second; ! = I7 "second noun"
Global keep_silent; ! true if current action is being tried silently
Global etype; ! parser error number if command not recognised
Global trace_actions = 0;

Global untouchable_object;
Global untouchable_silence;
Global touch_persona;

Global special_word; ! dictionary address of first word in "[text]" token
Global consult_from; ! word number of start of "[text]" token
Global consult_words; ! number of words in "[text]" token
Global parsed_number; ! value from any token not an object
Global special_number1; ! first value, if token not an object
Global special_number2; ! second value, if token not an object

Array  parser_results --> 16; ! for parser to write its results in
Global parser_trace = 0; ! normally 0, but 1 to 5 traces parser workings
Global pronoun_word; ! records which pronoun ("it", "them", ...) caused an error
Global pronoun_obj; ! and what object it was thought to refer to
Global players_command = 100; ! = I7 "player's command"
Global matched_text; ! = I7 "matched text"
Global reason_the_action_failed; ! = I7 "reason the action failed"
Global understand_as_mistake_number; ! which form of "Understand... as a mistake"
Global particular_possession; ! = I7 "particular possession"
! [9]
Global parser_action; ! written by the parser for the benefit of GPRs
Global parser_one;
Global parser_two;
Global parameters; ! number of I7 tokens parsed on the current line
Global action_to_be; ! (if the current line were accepted)
Global action_reversed; ! (parameters would be reversed in order)
Global wn; ! word number within "parse" buffer (from 1)
Global num_words; ! number of words in buffer
Global verb_word; ! dictionary address of command verb
Global verb_wordnum; ! word number of command verb
! [10]
Global scope_reason = PARSING_REASON; ! current reason for searching scope
Global scope_token; ! for "scope=Routine" grammar tokens
Global scope_error;
Global scope_stage; ! 1, 2 then 3
Global advance_warning; ! what a later-named thing will be
Global reason_code = NULL; ! for the I6 veneer
Global ats_flag = 0; ! for AddToScope routines
Global ats_hls;
! [11]
Global move_pushing;
Global move_from;
Global move_to;
Global move_by;
Global move_through;
! [12]
#Ifdef DEFAULT_BRIEF_DESCRIPTIONS;
Global lookmode = 1; ! 1 = BRIEF, 2 = VERBOSE, 3 = SUPERBRIEF
#Endif;
#Ifdef DEFAULT_VERBOSE_DESCRIPTIONS;
Global lookmode = 2; ! 1 = BRIEF, 2 = VERBOSE, 3 = SUPERBRIEF
#Endif;
#Ifdef DEFAULT_SUPERBRIEF_DESCRIPTIONS;
Global lookmode = 3; ! 1 = BRIEF, 2 = VERBOSE, 3 = SUPERBRIEF
#Endif;
#Ifndef lookmode;
Global lookmode = 2; ! 1 = BRIEF, 2 = VERBOSE, 3 = SUPERBRIEF
#Endif;
Global c_style; ! current list-writer style
Global c_depth; ! current recursion depth
Global c_iterator; ! current iteration function
Global lt_value; ! common value of list_together
Global listing_together; ! object number of one member of a group being listed together
Global listing_size; ! size of such a group
Global c_margin; ! current level of indentation printed by WriteListFrom()
Global inventory_stage = 1; ! 1 or 2 according to the context in which list_together uses
```

```
! [13]
Global clr_fg = 1; ! foreground colour
Global clr_bg = 1; ! background colour
Global clr_fgstatus = 1; ! foreground colour of statusline
Global clr_bgstatus = 1; ! background colour of statusline
Global clr_on; ! has colour been enabled by the player?
Global statuswin_current; ! if writing to top window

! [14]
Global statuswin_cursize = 0;
Global statuswin_size = 1;

! [15]
Global lm_act; Global lm_n; Global lm_o; Global lm_o2;

! [16]
Global debug_flag = 0;
Global debug_rules = 0;
Global debug_scenes = 0;
Global debug_rule_nesting;
```

§**7. VM-Specific Code.** These sections of code contain different definitions of the same routines, and in some cases the same arrays, to handle low-level functions in the virtual machine – saving the game, performing UNDO, parsing typed text into dictionary word addresses and so on.

```
#Ifdef TARGET_GLULX;
{-segment:Glulx.i6t}
#Endif;

#Ifdef TARGET_ZCODE;
{-segment:ZMachine.i6t}
#Endif;
```

§**8. Compass.** I6 identified compass directions as being children of the pseudo-object `Compass`, so we define it. (Note that `Compass` is not a valid I7 object, and is used for no other purpose.) Because of the traditional structure of language definitions, this needs to come first.

```
Object Compass "compass" has concealed;
```

§**9. Language of Play.** The equivalent of I6's language definition file, though here the idea is that a translation should have an inclusion to replace the "Language.i6t" segment, which contains the English definition.

```
{-segment:Language.i6t}

Default LanguageCases 1;
#Ifndef LibraryMessages; Object LibraryMessages; #Endif;
```

§10. The Old Library. The I6 library consisted essentially of the parser, the verb routines, and a pile of utilities and world-modelling code, of which the biggest single component was the list-writer. The parser lives on below; the verb routines are gone, with the equivalent functionality having moved upstairs into I7 source text in the Standard Rules; and the rest of the library largely lives here:

```
{-segment:Light.i6t}
{-segment:ListWriter.i6t}
{-segment:Utilities.i6t}
```

§11. Parser. The largest single block of code in the traditional I6 library part of the template layer is the parser.

The two pseudo-objects `InformParser` and `InformLibrary` are relics of the object-oriented approach in I6, and are used only very slightly in the template layer; they are not used at all in I7, and are not valid for the "object" kind of value.

The parser includes arrays for typed text and some parsing information derived from it, and if these should overrun it would cause enigmatic bugs, as the next arrays in memory would be corrupted: as a tripwire, the `Protect_I7_Arrays` array consists of two magic values in sequence. If it is ever discovered to contain the wrong data, the alarm sounds.

```
Object InformParser "(Inform Parser)" has proper;
{-segment:Parser.i6t}
[ ParserError error_type;
    if (error_type ofclass String or Routine) PrintSingleParagraph(error_type);
    rfalse;
];
Object InformLibrary "(Inform Library)" has proper;
Array Protect_I7_Arrays --> 16339 12345;
```

§12. Order of Play. The `Main` routine, where execution begins, and the primitive rules in the principal rulebooks.

```
{-segment:OrderOfPlay.i6t}
```

§13. Properties. Some either/or properties are compiled to I6 attributes, which must be predeclared, so we do that first. (All other properties can simply be used without declaration.)

What then follows is a table of property metadata: in particular, specifying which properties can be used with which I6 classes or objects. Policing this at run-time costs a little speed, but traps many errors of programming, and keeps everything typesafe. It is the price we pay for the relatively lenient compile-time checking of I7's "object" kind of value. To make it as efficient as possible, we calculate offsets into the metadata: this has to be done (once) at run-time, with the routine compiled.

```
{-call:Properties::alias_translations}
{-call:Properties::Implementation::OfObjects::compile_attributes}
{-array:Properties::Implementation::OfObjects::property_metadata}
Constant attributed_property_offsets_SIZE 48;
Array attributed_property_offsets --> attributed_property_offsets_SIZE;
Constant valued_property_offsets_SIZE (100 + {-value:NUMBER_CREATED(property)} + INDIV_PROP_START-4
  ...  8);
Array valued_property_offsets --> valued_property_offsets_SIZE;
{-routine:Properties::Implementation::OfObjects::CreatePropertyOffsets}
```

§14. Activities. These are numbered upwards from 0 in order of creation. The following arrays taken together provide, for each activity number: (i) the rulebook numbers for the before, for, and after stages of the activity, and (ii) a flag indicating whether the activity is "future action"-capable, that is, is a parsing activity allowed to make use of the action which conjecturally might result from the current grammar line being parsed. (This is called the "action to be", hence "atb".)

```
Constant NUMBER_RULEBOOKS_CREATED = {-value:NUMBER_CREATED(rulebook)};
{-call:Code::Activities::compile_activity_constants}
{-array:Code::Activities::Activity_before_rulebooks}
{-array:Code::Activities::Activity_for_rulebooks}
{-array:Code::Activities::Activity_after_rulebooks}
{-array:Code::Activities::Activity_atb_rulebooks}
```

§15. Relations.

```
{-call:Semantics::Relations::compile_defined_relation_constants}
```

§16. Printing Routines.

```
{-call:Kinds::RunTime::compile_data_type_support_routines}
{-routine:Kinds::RunTime::I7_Kind_Name}
{-routine:Code::Rulebooks::RulebookOutcomePrintingRule}
```

§17. Object Tree. The I6 object tree contains `Class` definitions as well as objects, but we precede both with a pseudo-object called `property_numberspace_forcer`. It does nothing except to ensure that properties are declared in I6 in the same sequence as I7 (which need not otherwise happen); it plays no part in play, and is not a valid I7 "object" value.

```
{-log:Compiling the storage for the model world}
{-call:World::Compile::compile}
```

§18. Tables. The initial state of the I6 arrays corresponding to each I7 table: see "Tables.i6t" for details.

```
{-log:Compiling the tables}
{-call:Data::Tables::compile}
```

§19. Equations. Routines to evaluate from equations.

```
{-log:Compiling the equations}
{-call:Data::Equations::compile}
```

§20. Actions.

```
{-log:Compiling the named action patterns}
{-call:Plugins::Actions::Patterns::Named::compile}
{-log:Compiling the action routines}
{-array:Plugins::Actions::ActionData}
{-array:Plugins::Actions::ActionCoding}
{-array:Plugins::Actions::ActionHappened}
{-call:Plugins::Actions::compile_action_routines}
{-routine:Plugins::Parsing::Lines::MistakeActionSub}
```

§21. Phrases. The following innocent-looking commands tell NI to compile I6 definitions for all of the rules which are not I6-written primitives, and also for adjective definitions, so it results in a fairly enormous cataract of code.

```
{-log:Compiling first block of phrases}
{-call:Code::Phrases::compile_first_block}
```

§22. Timed Events. Some of the phrases are simply called in the course of other phrases, but some are rules in rulebooks or in the table of timed events, so those come next:

```
{-array:Code::Phrases::TimedEventsTable}
{-array:Code::Phrases::TimedEventTimesTable}
```

§23. Rulebooks. The literally hundreds of rulebooks are set up here. (In the end a rulebook is only a (word) array of rule addresses, terminated with a `NULL`.)

```
{-log:Compiling the rulebooks}
{-array:Code::Phrases::rulebooks_array}
{-call:Code::Phrases::compile_rulebooks}
{-call:Code::Phrases::resolve_predeclared_booked_rules}
```

§24. Scenes.

```
{-log:Compiling scene details}
{-routine:Plugins::Scenes::DetectSceneChange}
#IFDEF DEBUG;
{-routine:Plugins::Scenes::ShowSceneStatus}
#ENDIF;
```

§25. The New Library. The gleaming, aluminium and glass extension to the library: almost all of it material new in I7 usage.

```
{-log:CTNL}
{-segment:Actions.i6t}
{-segment:Activities.i6t}
{-segment:Figures.i6t}
{-segment:FileIO.i6t}
{-segment:MStack.i6t}
{-segment:OutOfWorld.i6t}
{-segment:Printing.i6t}
{-segment:Relations.i6t}
{-segment:RTP.i6t}
{-segment:Rulebooks.i6t}
{-segment:Sort.i6t}
{-segment:Tables.i6t}
{-segment:WorldModel.i6t}
{-callv:Specifications::Values::ensure_block_constants}
```

§26. Parsing Tokens. GPRs, scope and noun filters to be used in grammar lines, but no actual grammar lines as yet.

```
{-callv:Plugins::Parsing::Verbs::prepare}
{-call:Plugins::Parsing::Verbs::compile_conditions}
{-log:Compiling GPR tokens for parsing various kinds of value}
{-segment:Number.i6t}
{-segment:Time.i6t}
{-call:Plugins::Parsing::Tokens::Values::compile_type_gprs}
{-log:Compiling noun and scope filter tokens}
{-call:Plugins::Parsing::Tokens::Filters::compile}
```

§27. Testing commands.

```
#IFDEF DEBUG;
{-call:Plugins::Parsing::TestScripts::write_text}
{-segment:Tests.i6t}
{-routine:Plugins::Parsing::TestScripts::InternalTestCases}
#ENDIF; ! DEBUG
```

§28. I6 Inclusions. This paragraph contains no code, by default: it's a hook on which to hang verbatim I6 material.

```
! "Include (- ... -)" inclusions with no specified position appear here.
```

§29. Entries in constant lists. Well: most of them, anyway. In particular, all of those which are lists of texts with substitution will be swept up, which is important for timing reasons. A second round later on will catch any later ones.

```
{-call:Data::Lists::check}
{-call:Data::Lists::compile}
```

§30. To Phrases. We now compile all of the remaining code in the source text: the "To..." phrases and all of their attendant text routines, loop-over-scope routines and so on.

We now have to be quite careful about the sequence of events. Compiling the text routines is an irrevocable step, after which we must not compile any new text with substitutions. On the other hand we mustn't leave it any later, because a text substitution might contain references to the past, or involve propositions which must be deferred into routines.

```
{-log:Compiling second block of phrases}
{-call:Code::Phrases::compile_second_block}
{-call:Data::Lists::check}
{-call:Data::Lists::compile}
{-call:Code::Phrases::compile_second_block}
{-callv:Data::Strings::allow_no_further_text_subs}
```

§31. Chronology. Similarly, this is where we wrap up all references to past tenses: after this point, we cannot safely compile any I7 condition in the past tense.

```
{-log:Compiling chronology}
{-segment:Chronology.i6t}
{-callv:Code::Chronology::allow_no_further_past_tenses}
```

§32. Grammar. This is the trickiest matter of timing. We had to leave the grammar lines until now because the past-tense code above might have needed to investigate whether the player's command matched a given pattern at some time in the past (a case which arose naturally in one of the example games, so which should not be dismissed as an aberration). This is therefore the earliest point at which we can know for sure that no further grammar lines are needed.

```
{-log:Compiling I6 Verb directives}
{-call:Plugins::Parsing::Verbs::compile_all}
#IFTRUE ({-value:no_verb_verb_defined} == 1);
[ UnknownVerb; verb_wordnum = 0; return 'no.verb'; ];
[ PrintVerb v;
    if (v == 'no.verb') { print "do something to"; rtrue; }
    rfalse;
];
#Ifnot;
[ UnknownVerb; rfalse; ]; [ PrintVerb v; rfalse; ];
#ENDIF;
```

§**33. Deferred Propositions.** Most conditions, such as "the score is 10", and descriptions, such as "open doors which are in lighted rooms", are translated by NI into propositions in a form of predicate calculus. Sometimes these can be compiled immediately to I6 code, but other times they involve complicated searches and have to be "deferred" into special routines which will perform them. This is where we compile those routines.

```
{-log:Compiling routines from predicate calculus}
{-call:Semantics::Relations::compile_defined_relations}
{-call:Calculus::Propositions::compile_all_deferred}
{-callv:Calculus::Deferrals::allow_no_further_deferrals}
```

§**34. Miscellaneous Loose Ends.** And we still aren't done, because we still have:

(1) Routines which switch between possible interpretations of phrases by performing run-time type checking. (Note that these cannot involve grammar, or the past tenses, or text substitutions, or deferred propositions.)

(2) Arrays holding constant lists, such as `{2, 3, 4}`, if any.

(3) The string constants, named in the pattern `SC_*`, in alphabetical order. (This ensures that their packed addresses will have unsigned comparison ordering equivalent to alphabetical order.)

(4) "Stub" I6 constants for property names where properties aren't used, to prevent them causing errors if they are referred to in code but not actually present in any object (as can easily happen with extensions presenting optional features which the user chooses not to employ). `cap_short_name` is similarly stubbed: this doesn't correspond to any I7 property, but is used by NI to record capitalised forms of the printed name (which in turn goes into `short_name`).

(5) Counters are used to allocate cells of storage to inline phrases which need a permanent state associated with them: see the Standard Rules. Since all I7 source text has been compiled by now, we know the final values of the counters, and therefore the amount of storage we need to allocate.

(6) Similarly, each "quotation" box needs its own cell of memory.

```
{-call:Code::Invocations::Compiler::resolver_routines}
{-call:Data::Lists::check}
{-call:Data::Lists::compile}
{-array:Data::Lists::ConstantListPointers}
{-call:Data::Strings::compile}
{-call:Properties::Implementation::OfObjects::compile_stub_properties}
#IFNDEF cap_short_name;
Constant cap_short_name = short_name;
#ENDIF;
{-call:Formats::Inform6::Labels::Counters::compile_allocated_storage}
Array Runtime_Quotations_Displayed --> {-value:extent_of_runtime_quotations_array};
```

§35. Block Values. These are values which are pointers to more elaborate data on the memory heap, rather than values in themselves: they point to "blocks". A section of code handles the heap, and there is then one further section to support each of the kinds of value in question.

```
{-call:Kinds::RunTime::compile_heap_allocator}
{-call:Code::Phrases::Constants::compile_closures}
{-call:Kinds::compile_runtime_id_structures}
{-segment:Flex.i6t}
{-segment:BlockValues.i6t}
{-segment:IndexedText.i6t}
{-segment:RegExp.i6t}
{-segment:StoredAction.i6t}
{-segment:Lists.i6t}
{-segment:Combinations.i6t}
{-segment:RelationKind.i6t}
{-array:Plugins::Figures::tableoffigures}
{-array:Plugins::Sounds::tableofsounds}
{-call:Specifications::Values::create_block_constants}
{-callv:Code::Phrases::Timed::check_for_unused}
```

§36. Signing off. And that's all, folks.

```
! End of automatically generated I6 source
! ---------------------------------------------------------------------------
```

Definitions Template B/defnt

Purpose

Miscellaneous constant definitions, usually providing symbolic names for otherwise inscrutable numbers, which are used throughout the template layer.

§**1. VM Target Constants.** Inform can compile story files for four different virtual machines (or VMs): Z-machine versions 5, 6 and 8, and Glulx.

When compiling to Glulx, the I6 compiler predefines the constant `TARGET_GLULX` and also sets `WORDSIZE` to 4; but when compiling to Z, we shouldn't assume it has this modern habit, so we simulate the same effect.

```
#Ifndef WORDSIZE; ! compiling with Z-code only compiler
Constant TARGET_ZCODE;
Constant WORDSIZE 2;
#Endif;
```

§**2. Wordsize-Dependent Definitions.** The old I6 library used to confuse Z-vs-G with 16-vs-32-bit, but we try to separate these distinctions here, even though at present the Z-machine is our only 16-bit target and Glulx our only 32-bit one. The `WORDSIZE` constant is the word size in bytes, so is the multiplier between `->` and `-->` offsets in I6 pointer syntax.

(1) `NULL` is used, as in C, to represent a null value or pointer. In C, this is conventionally 0, but here it is the maximum unsigned value which can be stored, pointing to the topmost byte in the directly addressable memory map; this means it is also -1 when regarded as a signed twos-complement integer, but we write it as an unsigned hexadecimal address for clarity's sake.

(2) `WORD_HIGHBIT` is the most significant bit in the VM's data word.

(3) `IMPROBABLE_VALUE` is one which is *unlikely but still possible* to be a genuine I7 value. The efficiency of some algorithms depends on how well chosen this is: they would ran badly if we chose 1, for instance.

(4) `MAX_POSITIVE_NUMBER` is the largest representable positive (signed) integer, in twos-complement form.

(5) `REPARSE_CODE` is a magic value used in the I6 library's parser to signal that some code which ought to have been parsing a command has in fact rewritten it, so that the whole command must be re-parsed afresh. (Returning this value is like throwing an exception in a language like Java, though we don't implement it that way.) A comment in the 6/11 library reads: "The parser rather gunkily adds addresses to `REPARSE_CODE` for some purposes. And expects the result to be greater than `REPARSE_CODE` (signed comparison). So Glulx Inform is limited to a single gigabyte of storage, for the moment." Guilty as charged, but the gigabyte story file is a remote prospect for now: even megabyte story files are off the horizon. Anyway, it's this comparison issue which means we need a different value for each possible word size.

```
#Iftrue (WORDSIZE == 2);
Constant NULL = $ffff;
Constant WORD_HIGHBIT = $8000;
Constant WORD_NEXTTOHIGHBIT = $4000;
Constant IMPROBABLE_VALUE = $7fe3;
Constant MAX_POSITIVE_NUMBER 32767;
Constant MIN_NEGATIVE_NUMBER -32768;
Constant REPARSE_CODE = 10000;
#Endif;

#Iftrue (WORDSIZE == 4);
Constant NULL = $ffffffff;
Constant WORD_HIGHBIT = $80000000;
Constant WORD_NEXTTOHIGHBIT = $40000000;
Constant IMPROBABLE_VALUE = $deadce11;
Constant MAX_POSITIVE_NUMBER 2147483647;
Constant MIN_NEGATIVE_NUMBER -2147483648;
Constant REPARSE_CODE = $40000000;
#Endif;
```

§**3. Z-Machine Definitions.** The Z-machine contains certain special constants and variables at fixed position in its "header"; the addresses of these are given below. See *The Z-Machine Standards Document*, version 1.0, for details.

`INDIV_PROP_START` is the lowest number of any "individual property", an I6 internal quantity defined by the compiler when the target is Glulx but not for Z.

```
#Ifdef TARGET_ZCODE;

Global max_z_object;

Constant INDIV_PROP_START 64;

! Offsets into Z-machine header:

Constant HDR_ZCODEVERSION     = $00;     ! byte
Constant HDR_TERPFLAGS        = $01;     ! byte
Constant HDR_GAMERELEASE      = $02;     ! word
Constant HDR_HIGHMEMORY       = $04;     ! word
Constant HDR_INITIALPC        = $06;     ! word
Constant HDR_DICTIONARY       = $08;     ! word
Constant HDR_OBJECTS          = $0A;     ! word
Constant HDR_GLOBALS          = $0C;     ! word
Constant HDR_STATICMEMORY     = $0E;     ! word
Constant HDR_GAMEFLAGS        = $10;     ! word
Constant HDR_GAMESERIAL       = $12;     ! six ASCII characters
Constant HDR_ABBREVIATIONS    = $18;     ! word
Constant HDR_FILELENGTH       = $1A;     ! word
Constant HDR_CHECKSUM         = $1C;     ! word
Constant HDR_TERPNUMBER       = $1E;     ! byte
Constant HDR_TERPVERSION      = $1F;     ! byte
Constant HDR_SCREENHLINES     = $20;     ! byte
Constant HDR_SCREENWCHARS     = $21;     ! byte
Constant HDR_SCREENWUNITS     = $22;     ! word
Constant HDR_SCREENHUNITS     = $24;     ! word
Constant HDR_FONTWUNITS       = $26;     ! byte
Constant HDR_FONTHUNITS       = $27;     ! byte
Constant HDR_ROUTINEOFFSET    = $28;     ! word
Constant HDR_STRINGOFFSET     = $2A;     ! word
Constant HDR_BGCOLOUR         = $2C;     ! byte
Constant HDR_FGCOLOUR         = $2D;     ! byte
Constant HDR_TERMCHARS        = $2E;     ! word
Constant HDR_PIXELSTO3        = $30;     ! word
Constant HDR_TERPSTANDARD     = $32;     ! two bytes
Constant HDR_ALPHABET         = $34;     ! word
Constant HDR_EXTENSION        = $36;     ! word
Constant HDR_UNUSED           = $38;     ! two words
Constant HDR_INFORMVERSION    = $3C;     ! four ASCII characters
#Endif;
```

§**4. Glulx Definitions.** We make similar header definitions for Glulx. Extensive further definitions, of constants needed to handle the Glk I/O layer, can be found in the "Infglk" section of "Glulx.i6t"; they are not used in the rest of the template layer, and would only get in the way here.

```
#IFDEF TARGET_GLULX;
Global unicode_gestalt_ok; ! Set if interpreter supports Unicode
! Offsets into Glulx header and start of ROM:
Constant HDR_MAGICNUMBER      = $00;     ! long word
Constant HDR_GLULXVERSION     = $04;     ! long word
Constant HDR_RAMSTART         = $08;     ! long word
Constant HDR_EXTSTART         = $0C;     ! long word
Constant HDR_ENDMEM           = $10;     ! long word
Constant HDR_STACKSIZE        = $14;     ! long word
Constant HDR_STARTFUNC        = $18;     ! long word
Constant HDR_DECODINGTBL      = $1C;     ! long word
Constant HDR_CHECKSUM         = $20;     ! long word
Constant ROM_INFO             = $24;     ! four ASCII characters
Constant ROM_MEMORYLAYOUT     = $28;     ! long word
Constant ROM_INFORMVERSION    = $2C;     ! four ASCII characters
Constant ROM_COMPVERSION      = $30;     ! four ASCII characters
Constant ROM_GAMERELEASE      = $34;     ! short word
Constant ROM_GAMESERIAL       = $36;     ! six ASCII characters
#Endif;
```

§**5. Powers of Two.** I6 lacks support for logical shifts, and the Z-machine opcodes which bear on this are not always well supported, so the I6 library has traditionally used a lookup table for the values of 2^{15-n} where $0 \leq n \leq 11$.

```
Array PowersOfTwo_TB
--> $$100000000000
    $$010000000000
    $$001000000000
    $$000100000000
    $$000010000000
    $$000001000000
    $$000000100000
    $$000000010000
    $$000000001000
    $$000000000100
    $$000000000010
    $$000000000001;
Array IncreasingPowersOfTwo_TB
```

```
--> $$0000000000000001
    $$0000000000000010
    $$0000000000000100
    $$0000000000001000
    $$0000000000010000
    $$0000000000100000
    $$0000000001000000
    $$0000000010000000
    $$0000000100000000
    $$0000001000000000
    $$0000010000000000
    $$0000100000000000
    $$0001000000000000
    $$0010000000000000
    $$0100000000000000
    $$1000000000000000;
```

§**6. Text Styles.** These are the styles of text distinguished by the template layer, though they are not required to look different from each other on any given VM. The codes are independent of the VM targetted, though in fact they are equal to Glulx style numbers as conventionally used. (The Z-machine renders some as roman, some as bold, but for instance makes `HEADER_VMSTY` and `SUBHEADER_VMSTY` indistinguishable to the eye.) Glulx's system of styles is one of its weakest points from an IF author's perspective, since it is all but impossible to achieve the text effects one might want – boldface, for instance, or red text – and text rendering is almost the only area in which it is clearly inferior to the Z-machine, which it was designed to replace. Still, using these styles when we can will get the most out of it, and for unornamental works Glulx is fine in practice.

```
Constant NORMAL_VMSTY     = 0;
Constant HEADER_VMSTY     = 3;
Constant SUBHEADER_VMSTY  = 4;
Constant ALERT_VMSTY      = 5;
Constant NOTE_VMSTY       = 6;
Constant BLOCKQUOTE_VMSTY = 7;
Constant INPUT_VMSTY      = 8;
```

§**7. Colour Numbers.** These are traditional colour names: quite who it was who thought that azure was the same colour as cyan is now unclear. Colour is, again, not easy to arrange on Glulx, but there is some workaround code.

```
Constant CLR_DEFAULT = 1;
Constant CLR_BLACK   = 2;
Constant CLR_RED     = 3;
Constant CLR_GREEN   = 4;
Constant CLR_YELLOW  = 5;
Constant CLR_BLUE    = 6;
Constant CLR_MAGENTA = 7; Constant CLR_PURPLE  = 7;
Constant CLR_CYAN    = 8; Constant CLR_AZURE   = 8;
Constant CLR_WHITE   = 9;
```

§8. Window Numbers. Although Glulx can support elaborate tessalations of windows on screen (if the complexity of handling this can be mastered), the Z-machine has much more limited resources in general, so the template layer assumes a simple screen model: there are just two screen areas, the scrolling main window in which commands are typed and responses printed, and the fixed status line bar at the top of the screen.

```
Constant WIN_ALL     = 0; ! Both windows at once
Constant WIN_STATUS  = 1;
Constant WIN_MAIN    = 2;
```

§9. Paragraphing Flags. I am not sure many dictionaries would countenance "to paragraph" as a verb, but never mind: the reference here is to the algorithm used to place paragraph breaks within text, which uses bitmaps composed of the following.

```
Constant PARA_COMPLETED          = 1;
Constant PARA_PROMPTSKIP         = 2;
Constant PARA_SUPPRESSPROMPTSKIP = 4;
Constant PARA_NORULEBOOKBREAKS   = 8;
Constant PARA_CONTENTEXPECTED    = 16;
```

§10. Descriptors in the Language of Play. The following constants, which must be different in value from the number of any I6 attribute, are used in the `LanguageDescriptors` table found in the definition of the language of play.

```
Constant POSSESS_PK  = $100;
Constant DEFART_PK   = $101;
Constant INDEFART_PK = $102;
```

§11. Run-Time Problem Numbers. The enumeration sequence here must correspond to that in the file of RTP texts, which is used to generate the HTML pages displayed by the Inform user interface when a run-time problem has occurred. (For instance, the file of RTP texts includes the heading "P17 - Can't divide by zero", equivalent to `RTP_DIVZERO` being 17 below.) There is no significance to the sequence, which is simply the historical order in which they were added to I7.

```
Constant RTP_BACKDROP                = 1;
Constant RTP_EXITDOOR                = 2;
Constant RTP_NOEXIT                  = 3;
Constant RTP_CANTCHANGE              = 4;
Constant RTP_IMPREL                  = 5;
Constant RTP_RULESTACK               = 6;
Constant RTP_TOOMANYRULEBOOKS        = 7;
Constant RTP_TOOMANYEVENTS           = 8;
Constant RTP_BADPROPERTY             = 9;
Constant RTP_UNPROVIDED              = 10;
Constant RTP_UNSET                   = 11;
Constant RTP_TOOMANYACTS             = 12;
Constant RTP_CANTABANDON             = 13;
Constant RTP_CANTEND                 = 14;
Constant RTP_CANTMOVENOTHING         = 15;
Constant RTP_CANTREMOVENOTHING       = 16;
Constant RTP_DIVZERO                 = 17;
Constant RTP_BADVALUEPROPERTY        = 18;
Constant RTP_NOTBACKDROP             = 19;
Constant RTP_TABLE_NOCOL            = 20;
Constant RTP_TABLE_NOCORR           = 21;
Constant RTP_TABLE_NOROW            = 22;
Constant RTP_TABLE_NOENTRY          = 23;
Constant RTP_TABLE_NOTABLE          = 24;
Constant RTP_TABLE_NOMOREBLANKS     = 25;
Constant RTP_TABLE_NOROWS           = 26;
Constant RTP_TABLE_CANTSORT         = 27;
Constant RTP_NOTINAROOM             = 28;
Constant RTP_BADTOPIC               = 29;
Constant RTP_ROUTELESS              = 30;
Constant RTP_PROPOFNOTHING          = 31;
Constant RTP_DECIDEONWRONGKIND      = 32;
Constant RTP_DECIDEONNOTHING        = 33;
Constant RTP_TABLE_CANTSAVE         = 34;
Constant RTP_TABLE_WONTFIT          = 35;
Constant RTP_TABLE_BADFILE          = 36;
Constant RTP_LOWLEVELERROR          = 37;
Constant RTP_DONTIGNORETURNSEQUENCE  = 38;
Constant RTP_SAYINVALIDSNIPPET      = 39;
Constant RTP_SPLICEINVALIDSNIPPET   = 40;
Constant RTP_INCLUDEINVALIDSNIPPET  = 41;
Constant RTP_LISTWRITERMEMORY       = 42;
Constant RTP_CANTREMOVEPLAYER       = 43;
Constant RTP_CANTREMOVEDOORS        = 44;
Constant RTP_CANTCHANGEOFFSTAGE     = 45;
Constant RTP_MSTACKMEMORY           = 46;
Constant RTP_TYPECHECK              = 47;
Constant RTP_FILEIOERROR            = 48;
Constant RTP_HEAPERROR              = 49;
Constant RTP_LISTRANGEERROR         = 50;
Constant RTP_REGEXPSYNTAXERROR      = 51;
Constant RTP_NOGLULXUNICODE         = 52;
Constant RTP_BACKDROPONLY           = 53;
Constant RTP_NOTTHING               = 54;
Constant RTP_SCENEHASNTSTARTED      = 55;
Constant RTP_SCENEHASNTENDED        = 56;
Constant RTP_NEGATIVEROOT           = 57;
Constant RTP_TABLE_CANTRUNTHROUGH   = 58;
Constant RTP_CANTITERATE            = 59;
Constant RTP_WRONGASSIGNEDKIND      = 60;
```

§**12. Template Activities.** These are the activities defined in the Standard Rules. Most, though not all, are carried out by explicit function calls in the template layer, which is why we need their ID numbers: note that NI assigns each activity a unique ID number on creation, counting upwards from 0, and that it processes the Standard Rules before any other source text. (These numbers must correspond *both* to those in the source of NI, *and* to the creation sequence in the Standard Rules.)

```
Constant PRINTING_THE_NAME_ACT          = 0;
Constant PRINTING_THE_PLURAL_NAME_ACT   = 1;
Constant PRINTING_A_NUMBER_OF_ACT       = 2;
Constant PRINTING_ROOM_DESC_DETAILS_ACT = 3;
Constant LISTING_CONTENTS_ACT           = 4;
Constant GROUPING_TOGETHER_ACT          = 5;
Constant WRITING_A_PARAGRAPH_ABOUT_ACT  = 6;
Constant LISTING_NONDESCRIPT_ITEMS_ACT  = 7;

Constant PRINTING_NAME_OF_DARK_ROOM_ACT = 8;
Constant PRINTING_DESC_OF_DARK_ROOM_ACT = 9;
Constant PRINTING_NEWS_OF_DARKNESS_ACT  = 10;
Constant PRINTING_NEWS_OF_LIGHT_ACT     = 11;
Constant REFUSAL_TO_ACT_IN_DARK_ACT     = 12;

Constant CONSTRUCTING_STATUS_LINE_ACT   = 13;
Constant PRINTING_BANNER_TEXT_ACT       = 14;

Constant READING_A_COMMAND_ACT          = 15;
Constant DECIDING_SCOPE_ACT             = 16;
Constant DECIDING_CONCEALED_POSSESS_ACT = 17;
Constant DECIDING_WHETHER_ALL_INC_ACT   = 18;
Constant CLARIFYING_PARSERS_CHOICE_ACT  = 19;
Constant ASKING_WHICH_DO_YOU_MEAN_ACT   = 20;
Constant PRINTING_A_PARSER_ERROR_ACT    = 21;
Constant SUPPLYING_A_MISSING_NOUN_ACT   = 22;
Constant SUPPLYING_A_MISSING_SECOND_ACT = 23;
Constant IMPLICITLY_TAKING_ACT          = 24;
Constant STARTING_VIRTUAL_MACHINE_ACT   = 25;

Constant AMUSING_A_VICTORIOUS_PLAYER_ACT = 26;
Constant PRINTING_PLAYERS_OBITUARY_ACT  = 27;
Constant DEALING_WITH_FINAL_QUESTION_ACT = 28;

Constant PRINTING_LOCALE_DESCRIPTION_ACT = 29;
Constant CHOOSING_NOTABLE_LOCALE_OBJ_ACT = 30;
Constant PRINTING_LOCALE_PARAGRAPH_ACT  = 31;
```

§**13. Template Rulebooks.** Rulebooks are created in a similar way, and again are numbered upwards from 0 in order of creation. These are the ones used in the template layer. (These numbers must correspond *both* to those in the source of NI, *and* to the creation sequence in the Standard Rules.)

```
Constant PROCEDURAL_RB                    = 0;
Constant STARTUP_RB                       = 1;
Constant TURN_SEQUENCE_RB                 = 2;
Constant SHUTDOWN_RB                      = 3;
Constant WHEN_PLAY_BEGINS_RB              = 5;
Constant WHEN_PLAY_ENDS_RB                = 6;
Constant WHEN_SCENE_BEGINS_RB             = 7;
Constant WHEN_SCENE_ENDS_RB               = 8;
Constant ACTION_PROCESSING_RB             = 10;
Constant SETTING_ACTION_VARIABLES_RB      = 11;
Constant SPECIFIC_ACTION_PROCESSING_RB    = 12;
Constant ACCESSIBILITY_RB                 = 14;
Constant REACHING_INSIDE_RB               = 15;
Constant REACHING_OUTSIDE_RB              = 16;
Constant VISIBLE_RB                       = 17;
Constant PERSUADE_RB                      = 18;
Constant UNSUCCESSFUL_ATTEMPT_RB          = 19;
Constant AFTER_RB                         = 24;
Constant REPORT_RB                        = 25;
```

§**14. Kind IDs.** These are filled in automatically by NI, and have the same names as are used in the NI source (and in the Load-.i6t sections): for instance `NUMBER_TY`.

```
{-call:Kinds::Constructors::compile_I6_constants}
```

§**15. Parser Error Numbers.** The traditional ways in which the I6 library's parser, which we adopt here more or less intact, can give up on a player's command. See the *Inform Designer's Manual*, 4th edition, for details.

```
Constant STUCK_PE     = 1;
Constant UPTO_PE      = 2;
Constant NUMBER_PE    = 3;
Constant ANIMA_PE     = 4;
Constant CANTSEE_PE   = 5;
Constant TOOLIT_PE    = 6;
Constant NOTHELD_PE   = 7;
Constant MULTI_PE     = 8;
Constant MMULTI_PE    = 9;
Constant VAGUE_PE     = 10;
Constant EXCEPT_PE    = 11;
Constant VERB_PE      = 12;
Constant SCENERY_PE   = 13;
Constant ITGONE_PE    = 14;
Constant JUNKAFTER_PE = 15;
Constant TOOFEW_PE    = 16;
Constant NOTHING_PE   = 17;
Constant ASKSCOPE_PE  = 18;
Constant NOTINCONTEXT_PE = 19;
Constant BLANKLINE_PE = 20; ! Not formally a parser error, but used by I7 as if
```

§16. Scope Searching Reasons. The parser makes use of a mechanism for searching through the objects currently "in scope", which basically means visible to the actor who would perform the action envisaged by the command being parsed. It is sometimes useful to behave differently depending on why this scope searching is being done, so the following constants enumerate the possibilities.

I6's `EACH_TURN_REASON`, `REACT_BEFORE_REASON` and `REACT_AFTER_REASON` have been removed from this list as no longer meaningful; hence the lacuna in numbering.

```
Constant PARSING_REASON       = 0;
Constant TALKING_REASON       = 1;
Constant EACH_TURN_REASON     = 2;
Constant LOOPOVERSCOPE_REASON = 5;
Constant TESTSCOPE_REASON     = 6;
```

§17. Token Types. Tokens are the indecomposable pieces of a grammar line making up a possible reading of a command; some are literal words, others stand for "any named object in scope", and so on. The following codes identify the possibilities. The `*_TOKEN` constants must not be altered without modifying the I6 compiler to match (so, basically, they must not be altered at all).

```
Constant ILLEGAL_TT         = 0;    ! Types of grammar token: illegal
Constant ELEMENTARY_TT      = 1;    !     (one of those below)
Constant PREPOSITION_TT     = 2;    !     e.g. 'into'
Constant ROUTINE_FILTER_TT  = 3;    !     e.g. noun=CagedCreature
Constant ATTR_FILTER_TT     = 4;    !     e.g. edible
Constant SCOPE_TT           = 5;    !     e.g. scope=Spells
Constant GPR_TT             = 6;    !     a general parsing routine

Constant NOUN_TOKEN         = 0;    ! The elementary grammar tokens, and
Constant HELD_TOKEN         = 1;    ! the numbers compiled by I6 to
Constant MULTI_TOKEN        = 2;    ! encode them
Constant MULTIHELD_TOKEN    = 3;
Constant MULTIEXCEPT_TOKEN  = 4;
Constant MULTIINSIDE_TOKEN  = 5;
Constant CREATURE_TOKEN     = 6;
Constant SPECIAL_TOKEN      = 7;
Constant NUMBER_TOKEN       = 8;
Constant TOPIC_TOKEN        = 9;
Constant ENDIT_TOKEN        = 15;   ! Value used to mean "end of grammar line"
```

§18. GPR Return Values. GRP stands for "General Parsing Routine", an I6 routine which acts as a grammar token: again, see the *Inform Designer's Manual*, 4th edition, for details.

In Library 6/11, `GPR_NOUN` is defined as `$ff00`, but this would fail on Glulx: it needs to be `$ffffff00` on 32-bit virtual machines. It appears that `GPR_NOUN` to `GPR_CREATURE`, though documented in the old *Inform Translator's Manual*, were omitted when this was consolidated into the DM4, so that they effectively disappeared from view. But they might still be useful for implementing inflected forms of nouns, so we have retained them here regardless.

```
Constant GPR_FAIL           = -1;   ! Return values from General Parsing
Constant GPR_PREPOSITION    = 0;    ! Routines
Constant GPR_NUMBER         = 1;
Constant GPR_MULTIPLE       = 2;
Constant GPR_REPARSE        = REPARSE_CODE;
Constant GPR_NOUN           = -256; ! Reparse, but as |NOUN_TOKEN| this time
Constant GPR_HELD           = GPR_NOUN + 1; ! And so on
Constant GPR_MULTI          = GPR_NOUN + 2;
Constant GPR_MULTIHELD      = GPR_NOUN + 3;
Constant GPR_MULTIEXCEPT    = GPR_NOUN + 4;
Constant GPR_MULTIINSIDE    = GPR_NOUN + 5;
Constant GPR_CREATURE       = GPR_NOUN + 6;
```

§19. List Styles. These constants make up bitmaps of the options in use when listing objects.

```
Constant NEWLINE_BIT        = $$0000000000000001; ! New-line after each entry
Constant INDENT_BIT         = $$0000000000000010; ! Indent each entry by depth
Constant FULLINV_BIT        = $$0000000000000100; ! Full inventory information after entry
Constant ENGLISH_BIT        = $$0000000000001000; ! English sentence style, with commas and and
Constant RECURSE_BIT        = $$0000000000010000; ! Recurse downwards with usual rules
Constant ALWAYS_BIT         = $$0000000000100000; ! Always recurse downwards
Constant TERSE_BIT          = $$0000000001000000; ! More terse English style
Constant PARTINV_BIT        = $$0000000010000000; ! Only brief inventory information after entry
Constant DEFART_BIT         = $$0000000100000000; ! Use the definite article in list
Constant WORKFLAG_BIT       = $$0000001000000000; ! At top level (only), only list objects
                                                   ! which have the "workflag" attribute
Constant ISARE_BIT          = $$0000010000000000; ! Print " is" or " are" before list
Constant CONCEAL_BIT        = $$0000100000000000; ! Omit objects with "concealed" or "scenery":
                                                   ! if WORKFLAG_BIT also set, then does not
                                                   ! apply at top level, but does lower down
Constant NOARTICLE_BIT      = $$0001000000000000; ! Print no articles, definite or not
Constant EXTRAINDENT_BIT    = $$0010000000000000; ! New in I7: extra indentation of 1 level
Constant CFIRSTART_BIT      = $$0100000000000000; ! Capitalise first article in list
```

§20. Lengths Of Time. Inform measures time in minutes.

```
Constant QUARTER_HOUR = 15;
Constant HALF_HOUR = 30;
Constant ONE_HOUR = 60;
Constant TWELVE_HOURS = 720;
Constant TWENTY_FOUR_HOURS = 1440;
```

§**21. Empty Text.** The I6 compiler does not optimise string compilation: if it needs to compile the (packed, read-only) string `"exemplum"` twice, it will compile two copies. This is slightly wasteful on memory, though in practice the loss is not enough for us to care. But we do want to avoid this in I7 because, to make string-sorting algorithms more efficient, we want direct numerical comparison of packed addresses to be equivalent to string comparison: and that means the text "exemplum" has to be compiled once and once only. There's a general mechanism for this in NI, but the single case most often needed is the empty text, since this is the default value for text variables and properties: we give it a name as follows.

(This works because I6 constant definition is not like C preprocessor macro expansion: `EMPTY_TEXT_VALUE` is equated with the address resulting from compiling `""`, rather than being replaced with the blank text to be recompiled each time.)

```
Constant EMPTY_TEXT_VALUE "";
```

§**22. Empty Table.** Similarly: the default value for the "table" kind of value, a Table containing no rows and no columns.

```
Array TheEmptyTable --> 0 0;
```

§**23. Empty Rulebook.** Similarly. An empty rulebook is one whose array has first word equal to `NULL`: we define a sequence of four `$ff` bytes so that the first word will be `NULL` on either 16-bit or 32-bit VMs. As with the empty text, this array is the value of many empty rulebooks: rulebook arrays are read-only, so there is no risk of a clash.

```
Array EMPTY_RULEBOOK -> $ff $ff $ff $ff;
```

§**24. Empty Set.** The falsity proposition describes the empty set of objects, and is the zero value for the "description" kind of value.

```
[ Prop_Falsity reason obj; return 0; ];
```

§**25. Score and Rankings Table.** The following command tells NI to compile constant definitions for `MAX_SCORE` and/or `RANKING_TABLE`, in cases where there are scores and rankings. If there's no score, we define `MAX_SCORE` as 0 anyway; if there's no ranking table, `RANKING_TABLE` is left undefined, so that we can `#ifdef` this possibility later.

```
{-call:Data::Tables::compile_max_score}
#Ifndef MAX_SCORE;
Global MAX_SCORE = 0;
#Endif;
```

§26. Template Attributes. An I6 attribute is equivalent to an I7 “either/or property”, though the latter are not always translated into I6 attributes because the Z-machine has only a limited number of attributes to use. Here, we define attributes used by the template.

Many concepts in I6 correspond directly to their successors in I7, even if details may vary. (Value properties are a case in point.) Attributes are the opposite of this: indeed, no I6 concept is more fragmented in its I7 equivalents. All but one of the old I6 library attributes are still used (the `general` attribute, for miscellaneous use, has been removed: it more often invited abuse than use); and a few new attributes have been added. But they are used for a variety of purposes. Some do correspond exactly to either/or properties in I7, but others are a sort of signature for I7 kinds. (So that for I7 use they are read-only.) Others still are used by the template layer as part of the implementation of services for I7, but are not visible to I7 source text as storage.

```
Attribute absent; ! Used to mark objects removed from play
Attribute animate; ! I6-level marker for I7 kind "person"
Attribute clothing; ! = I7 "wearable"
Attribute concealed; ! = I7 "undescribed"
Attribute container; ! I6-level marker for I7 kind "container"
Attribute door; ! I6-level marker for I7 kind "door"
Attribute edible; ! = I7 "edible" vs "inedible"
Attribute enterable; ! = I7 "enterable"
Attribute light; ! = I7 "lighted" vs "dark"
Attribute lockable; ! = I7 "lockable"
Attribute locked; ! = I7 "locked"
Attribute moved; ! = I7 "handled"
Attribute on; ! = I7 "switched on" vs "switched off"
Attribute open; ! = I7 "open" vs "closed"
Attribute openable; ! = I7 "openable"
Attribute scenery; ! = I7 "scenery"
Attribute static; ! = I7 "fixed in place" vs "portable"
Attribute supporter; ! I6-level marker for I7 kind "supporter"
Attribute switchable; ! I6-level marker for I7 kind "device"
Attribute talkable; ! Not currently used by I7, but retained for possible future use
Attribute transparent; ! = I7 "transparent" vs "opaque"
Attribute visited; ! = I7 "visited"
Attribute worn; ! marks that an object tree edge represents wearing

Attribute male; ! not directly used by I7, but available for languages with genders
Attribute female; ! = I7 "female" vs "male"
Attribute neuter; ! = I7 "neuter"
Attribute pluralname; ! = I7 "plural-named"
Attribute proper; ! = I7 "proper-named"
Attribute remove_proper; ! remember to remove proper again when using ChangePlayer next

Attribute privately_named; ! New in I7
Attribute mentioned; ! New in I7
Attribute pushable; ! New in I7

Attribute mark_as_room; ! Used in I7 to speed up testing "ofclass K1_room"
Attribute mark_as_thing; ! Used in I7 to speed up testing "ofclass K2_thing"

Attribute workflag; ! = I7 "marked for listing", but basically temporary workspace
Attribute workflag2; ! new in I7 and also temporary workspace
Constant list_filter_permits = privately_named; ! another I7 listwriter convenience
```

§27. Template Properties. As remarked above, these more often correspond to value properties in I7. To an experienced I6 user, though, the death toll of abolished I6 properties in I7 is breathtaking: in alphabetical order, `after`, `cant_go`, `daemon`, `each_turn`, `invent`, `life`, `number`, `orders`, `react_after`, `react_before`, `time_left`, `time_out`, `when_closed`, `when_off`, `when_on`, `when_open`. In May 2008, the direction properties `n_to`, `s_to`, `e_to`, ..., `out_to` joined the list of the missing.

The losses are numerous because of the shift from I6's object orientation to I7's rule orientation: information about the behaviour of objects is no longer thought of as data attached to them. At that, it could have been worse: a few unused I6 library properties have been retained for possible future use.

```
Property add_to_scope; ! used as in I6 to place component parts in scope
Property article "a"; ! used as in I6 to implement articles
Property capacity 100; ! = I7 "carrying capacity"
Property component_child; ! new in I7: forest structure holding "part of" relation
Property component_parent; ! new in I7
Property component_sibling; ! new in I7
Property description; ! = I7 "description"
Property door_dir; ! used to implement two-sided doors, but holds direction object, not a property
Property door_to; ! used as in I6 to implement two-sided doors
Property found_in; ! used as in I6 to implement two-sided doors and backdrops
Property initial; ! = I7 "initial description"
Property list_together; ! used as in I6 to implement "grouping together" activity
Property map_region; ! new in I7
Property parse_name 0; ! used as in I6 to implement "Understand... as..." grammars
Property plural; ! used as in I6 to implement plural names for duplicate objects
Property regional_found_in; ! new in I7
Property room_index; ! new in I7: storage for route-finding
Property short_name 0; ! = I7 "printed name"
Property vector; ! new in I7: storage for route-finding
Property with_key; ! = I7 "matching key"

Property KD_Count; ! Instance count of the kind of the current object
Property IK1_Count; ! These are instance counts within kinds K1, K2, ...
Property IK2_Count; ! and it is efficient to declare the common ones with Property
Property IK4_Count; ! since this results in a slightly smaller story file
Property IK5_Count;
Property IK6_Count;
Property IK8_Count;

Property IK1_link; ! These are for linked lists used to make searches faster
Property IK2_link; ! and again it's memory-efficient to declare the common ones
Property IK5_link; !
Property IK6_link; !
Property IK8_link; !

Property articles; ! not used by I7, but an interesting hook in the parser
Property grammar; ! not used by I7, but an interesting hook in the parser
Property inside_description; ! not used by I7, but an interesting hook in the locale code
Property short_name_indef 0; ! not used by I7, but an interesting hook in the listmaker
```

§**28. Loss of Life.** The loss of `life` is so appalling that I6 will not even compile a story file which doesn't define the property number `life` (well, strictly speaking, it checks the presence of constants suggesting the I6 library first, but the template layer does define constants like that). We define it as a null constant to be sure of avoiding any valid property number; I6 being typeless, that enables the veneer to compile again. (The relevant code is in `CA__Pr`, defined in the `veneer.c` section of I6.)

```
Constant life = NULL;
```

§**29. Action Count.** The number of valid I7 actions in existence.

```
Constant ActionCount = {-value:NUMBER_CREATED(action_name)};
```

§**30. Fake Actions.** Though sometimes useful for I6 coding tricks, fake actions – action numbers not corresponding to any action, but different from those of valid actions, and useable with a number of action syntaxes – are not conceptually present in I7 source text. They can only really exist at the I6 level because I6 is typeless; in I7 terms, there is no convenient kind of value which could represent both actions and fake actions while protecting each from confusion with the other.

See the *Inform Designer's Manual*, 4th edition, for what these are used for.

The following fake actions from the I6 library have been dropped here: `##LetGo`, `##Receive`, `##ThrownAt`, `##Prompt`, `##Places`, `##Objects`, `##Order`, `##NotUnderstood`.

```
Fake_Action ListMiscellany;
Fake_Action Miscellany;
Fake_Action PluralFound;
Fake_Action TheSame;
```

OrderOfPlay Template B/ordt

Purpose

The sequence of events in play: the Main routine which runs the startup rulebook, the turn sequence rulebook and the shutdown rulebook; and most of the I6 definitions of primitive rules in those rulebooks.

§**1. Main.** This is where every I6 story file begins execution: it can end either by returning, or by a `quit` statement or equivalent opcode. (In I7 this does indeed happen when the quitting the game action is carried out, or when QUIT is typed as a reply to the final question; it's only if the user has altered the shutdown rulebook that we might ever actually return from `Main`.) The return value from `Main` is not meaningful.

The `EarlyInTurnSequence` flag is used to enforce the requirement that the "parse command rule" and "generate action rule" do nothing unless the turn sequence rulebook is being followed directly from `Main`, an anomaly explained in the Standard Rules.

```
Global EarlyInTurnSequence;

[ Main;
    #ifdef TARGET_ZCODE; max_z_object = #largest_object - 255; #endif;
    ProcessRulebook(STARTUP_RB);
    #ifdef DEBUG; InternalTestCases(); #endif;
    while (true) {
        while (deadflag == false) {
            EarlyInTurnSequence = true;
            action = ##Wait; meta = false; noun = nothing; second = nothing;
            actor = player;
            FollowRulebook(TURN_SEQUENCE_RB);
        }
        if (FollowRulebook(SHUTDOWN_RB) == false) return;
    }
];
```

§**2. Virtual Machine Startup Rule.** Note that we consider the three rulebooks for the "starting the virtual machine" activity, but do not formally carry it out, because that might invoke procedural rules: this early in the run, before the screen can accept text, for instance, procedural rules would be risky. We then delegate to the appropriate VM-specific section of code for the real work. The printing of three blank lines at the start of play is traditional: on early Z-machine interpreters such as InfoTaskForce and Zip it was a necessity because of the way they buffered output. On modern windowed ones it still helps to space the opening text better.

```
[ VIRTUAL_MACHINE_STARTUP_R;
    ProcessRulebook(Activity_before_rulebooks-->STARTING_VIRTUAL_MACHINE_ACT);
    ProcessRulebook(Activity_for_rulebooks-->STARTING_VIRTUAL_MACHINE_ACT);
    ProcessRulebook(Activity_after_rulebooks-->STARTING_VIRTUAL_MACHINE_ACT);

    VM_Initialise();

    print "^^^";
    rfalse;
];
```

§**3. Initial Situation.** The array `InitialSituation` is compiled by NI and contains:

(0) The object number for the player, which is usually `selfobj`.
(1) The object in or on which the player begins, if he does. (This will always be an enterable container or supporter, or `nothing`.)
(2) The room in which the player begins, which is usually the first room created in the source text.
(3) The initial time of day, which is usually 9 AM.

The start object and start room are meaningful only if the player's object is compiled outside of the object tree (as can happen if the source text reads, say, "Mrs Bridges is a woman. The player is Mrs Bridges."): in other circumstances they are often correct, but this must not be relied on.

```
Constant PLAYER_OBJECT_INIS = 0;
Constant START_OBJECT_INIS = 1;
Constant START_ROOM_INIS = 2;
Constant START_TIME_INIS = 3;
Constant DONE_INIS = 4;

{-array:Plugins::Player::InitialSituation}
```

§**4. Initialise Memory Rule.** This rule amalgamates some minimal initialisations which all need to happen before we can risk using some of the more exotic I7 kinds:

(a) The language definition might call for initialisation, although the default language of play (English) does not.
(b) A handful of variables are filled in. `I7_LOOKMODE` is a constant created by the use options "use full-length room descriptions" or "use abbreviated room descriptions", but otherwise not existing. It is particularly important that `player` have the correct value, as the process of initialising the memory heap uses the player as the presumed actor when creating memory representations of literal stored actions where no actor was specified; this is why `player` is initialised here and not in the "position player in model world rule" below. The other interesting point here is that we explicitly set `location` and `real_location` to `nothing`, which is certainly incorrect, even though we know better. We do this so that the "update chronological records rule" cannot see where the player is: see the Standard Rules for an explanation of why this is, albeit perhaps dubiously, a good thing.
(c) We start the machinery needed to check that property accesses are valid during play.
(d) And we initialise the memory allocation heap, and expand the literal constants, as hinted above: these are called "block constants" since they occupy blocks of memory.

The `not_yet_in_play` flag, which is cleared when the first command is about to be read from the keyboard, suppresses the standard status line text: thus, if there is some long text to read before the player finds out where he is, the surprise will not be spoiled.

```
[ INITIALISE_MEMORY_R;
    #ifdef TARGET_GLULX; VM_PreInitialise(); #Endif;
    #Ifdef LanguageInitialise; LanguageInitialise(); #Endif;

    not_yet_in_play = true;
    #ifdef I7_LOOKMODE; lookmode = I7_LOOKMODE; #endif;
    player = InitialSituation-->PLAYER_OBJECT_INIS;
    the_time = InitialSituation-->START_TIME_INIS;
    real_location = nothing;
    location = nothing;

    CreatePropertyOffsets();
    HeapInitialise(); ! Create a completely unused memory allocation heap
    InitialHeapAllocation(); ! Allocate empty blocks for variables, properties, and such
    CreateBlockConstants(); ! Allocate and fill in blocks for constant values
    DistributeBlockConstants(); ! Ensure these exist in multiple independent copies when needed
    CreateDynamicRelations(); ! Create relation structures on the heap

    rfalse;
];
```

§**5. Seed Random Number Generator Rule.** Unless a seed is provided by NI, and it won't be for released story files, the VM's interpreter is supposed to start up with a good seed in its random number generator: something usually derived from, say, the milliseconds part of the current time of day, which is unlikely to repeat or show any pattern in real-world use. However, early Z-machine interpreters often did this quite badly, starting with poor seed values which meant that the first few random numbers always had something in common (being fairly small in their range, for instance). To obviate this we extract and throw away 100 random numbers to get the generator going, shaking out more obvious early patterns, but that cannot really help much if the VM interpreter's RNG is badly written. "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin" (von Neumann).

```
[ SEED_RANDOM_NUMBER_GENERATOR_R i;
    if ({-value:rng_seed_at_start_of_play}) VM_Seed_RNG({-value:rng_seed_at_start_of_play});
    for (i=1: i<=100: i++) random(i);
    rfalse;
];
```

§6. Position Player In Model World Rule. This seems as good a place as any to write down the invariant we attempt to maintain for the player's position variables:

(1) The `player` variable is the object through which the player plays, which is always a person: its value is always set by starting from `selfobj` and then making a sequence of 0 or more `ChangePlayer(new_value)` calls. (This enables us to make sure it has the correct property values for printed name and so forth.)

(2) The `location` is always either the current room, a valid I7 room, or `thedark`, which is not a valid I7 object but is distinguishable both from all I7 objects and from `nothing`. The `real_location` is always the current room, which is always a valid I7 room. `location` equals `thedark` if and only if the player does not have light to see by; the routine `SilentlyConsiderLight` updates this without printing any messages to announce the fall or lifting of darkness (hence "silently").

(3) The `player` object is always in the subtree of `real_location`, and is always in a chain $O_1 < O_2 < ... < O_n$ where O_1 is the player, O_n is `real_location`, $n \geq 2$ and $O_2, ..., O_{n-1}$ are all either enterable containers, enterable supporters, or component parts of such. The routine `LocationOf`, applied to the player object, always agrees with `real_location`.

(4) "Floating" objects, such as backdrops and two-sided doors, are in theory present in more than one room at once. In practice they can only be in a single position in the I6 object tree at any one time. The rule is that if they are theoretically present in the `real_location`, then they are actually present in the subtree of `real_location`. The routine `MoveFloatingObjects` updates this, and must be called whenever the player moves from one room to another.

(5) Any objects carried by the player have the I6 `moved` attribute set. The `SACK_OBJECT` variable is always set to the object of kind "player's holdall" which the player has most recently been carrying, or had as a component part of himself. The "note object acquisitions" rule updates this.

These invariants are usually all false before the following rule is executed; they are all true once it has completed. In addition, because the global action variables usually hold details of the action most recently carried out, we initialise these as if the most recent action had been the player waiting. (Nobody ought to use these variables at this point, but in case they do use them by accident in a "when play begins" rule, we want Inform to behave predictably and without type-unsafe values entering code.)

```
[ POSITION_PLAYER_IN_MODEL_R player_to_be;
    player = selfobj;
    player_to_be = InitialSituation-->PLAYER_OBJECT_INIS;
    location = LocationOf(player_to_be);
    if (location == 0) {
        location = InitialSituation-->START_ROOM_INIS;
        if (InitialSituation-->START_OBJECT_INIS)
            move player_to_be to InitialSituation-->START_OBJECT_INIS;
        else move player_to_be to location;
    }
    if (player_to_be ~= player) { remove selfobj; ChangePlayer(player_to_be); }
    real_location = location; SilentlyConsiderLight();
    NOTE_OBJECT_ACQUISITIONS_R(); MoveFloatingObjects();
    actor = player; act_requester = nothing; actors_location = real_location; action = ##Wait;
    InitialSituation-->DONE_INIS = true;
    rfalse;
];
```

§**7. Parse Command Rule.** This section contains only two primitive rules from the turn sequence rulebook, the matched pair of the "parse command rule" and the "generate action rule"; the others are found in the sections on Light and Time.

We use almost identically the same parser as that in the I6 library, since it is a well-proven and understood algorithm. The I6 parser returns some of its results in a supplied array (here `parser_results`, though the I6 library used to call this `inputobjs`), but others are in global variables:

(1) The `parser_results` array holds four words, used as indexed by the constants below.
   (a) The action can be a valid I6 action number, or an I6 "fake action", a concept not used overtly in I7. Most valid I6 actions correspond exactly to I7 actions, but in principle it is possible to define (say) extra debugging commands entirely at the I6 level.
   (b) The count `NO_INPS_PRES` is always 0, 1 or 2, and then that many of the next two words are meaningful.
   (c) Each of the "inp" values is either 0, meaning "put the multiple object list here"; or 1, meaning "not an object but a value"; or a valid I6 object. (We use the scoping rules to ensure that any I6 object visible to the parser is also a valid I7 object, so – unlike with actions – we need not distinguish between the two.)

(2) The global variable `actor` is set to the person asked to carry out the command, or is the same as `player` if nobody was mentioned. Thus it will be the object for Floyd in the command FLOYD, GET PERMIT, but will be just `player` in the command EAST.

(3) The global variables `special_number1` and, if necessary, `special_number2` hold values corresponding to the first and second of the "inps" to be returned as 1. Thus, if one of the "inps" is a value and the other is an object, then `special_number1` is that value; only if both are values rather than objects will `special_number2` be used. There is no indication of the kind of these values: I6 is typeless.

(4) At most one of the "inps" is permitted to be 1, referring to a multiple object list. (And a multiple value list is forbidden.) If this happens, the list of objects is stored in an I6 table array (i.e., with the 0th word being the number of subsequent words) called `multiple_object`, and the parser will have set the `toomany_flag` if an overflow occurred – that is, if the list was truncated because it originally called for more than 63 objects.

(5) The global variable `meta` is set if the action is one marked as such in the I6 grammar. A confusion in the design of I6 is that being out of world, as we would say in I7 terms, is associated not with an action as such but with the command verb triggering it. (This in practice caused no trouble since we never used, say, the word SAVE for both saving the game and saving, I don't know, box top coupons.) The state of `meta` returned by the I6 parser does not quite correspond to I7's "out of world" concept, so we will alter it in a few cases.

Some of these conventions are a little odd-looking now: why not simply have a larger results array, rather than this pile of occasionally used variables? The reasons are purely historical: the I6 parser developed gradually over about a decade.

```
Constant ACTION_PRES = 0;
Constant NO_INPS_PRES = 1;
Constant INP1_PRES = 2;
Constant INP2_PRES = 3; ! Parser.i6t code assumes this is INP1_PRES + 1

[ PARSE_COMMAND_R;
    if (EarlyInTurnSequence == false) rfalse; ! Prevent use outside top level
    not_yet_in_play = false;

    Parser__parse();
    TreatParserResults();
    rfalse;
];
```

§8. Treat Parser Results. We don't quite use the results exactly as they are returned by the parser: we make modifications in a few special cases.

(1) `##MistakeAction` is a valid I6 action, but not an I7 one. It is used to implement "Understand ... as a mistake", which provides a short-cut way for I7 source text to specify responses to mistaken guesses at the syntax expected for commands. It can therefore result from a whole variety of different commands, some of which might be flagged `meta`, others not. We forcibly set the `meta` flag: a mistake in guessing the command always happens out of world.

(2) A command in the form PERSON, TELL ME ABOUT SOMETHING is altered to the action resulting from ASK PERSON ABOUT SOMETHING, so that `##Tell` is converted to `##Ask` in these cases.

```
[ TreatParserResults;
    if (parser_results-->ACTION_PRES == ##MistakeAction) meta = true;
    if (parser_results-->ACTION_PRES == ##Tell &&
        parser_results-->INP1_PRES == player && actor ~= player) {
        parser_results-->ACTION_PRES = ##Ask;
        parser_results-->INP1_PRES = actor; actor = player;
    }
];
```

§9. Generate Action Rule. For what are, again, historical reasons to do with the development of I6, the current action is recorded in a slate of global variables:

(1) `actor` is as above; `action` is the I6 action number or fake action number, though in I7 usage no fake actions should ever reach this point.

(2) `act_requester` is the person requesting that the actor should perform the action, or `nothing` if the action is the actor's own choice. In the command FLOYD, MOP FLOOR, the `act_requester` is the player and the actor is Floyd; but when an action arises from a try phrase in I7, such as "try Floyd mopping the floor", `act_requester` is `nothing` because it is Floyd's own decision to do this. (The computer, of course, represents the will-power of all characters other than the player.)

(3) `inp1` and `inp2` are global variables whose contents mean the same as those of `parser_results-->INP1_PRES` and `parser_results-->INP2_PRES`. (This is not duplication, because actions also arise from "try" rather than the parser.)

(4) The variable `multiflag` is set during the processing of a multiple object list, and clear otherwise. (It is used for instance by the Standard Rules to give more concise reports of some successful actions.) Note that it remains set during any knock-on actions caused by actions in the multiple object list: the following rule is the only place where `multiflag` is set or cleared.

(5) `noun` and `second` are global variables which are equal to `inp1` and `inp2` when the latter hold valid object numbers, and are equal to `nothing` otherwise. (This is not duplication either, because it provides us with type-safe access to objects: there is no KOV which can safely represent `inp1` and `inp2`, but `noun` and `second` are valid for the I7 kind of value "object".)

In the following rule, we create this set of variables for the action or multiple action(s) suggested by the parser: each action is sent on to `BeginAction` for processing. Once done, we reset the above variables in what might seem an odd way: we allow straightforward actions by the player to remain in the variables, but convert requests to other people to the neutral "waiting" action carried out by the player (which is the zero value for actions). Now, in a better world, we would always erase the action like this, because an action once completed ought to be forgotten. The value of `noun` ought to be visible only during the action's processing.

But in practice many I7 users write "every turn" rules which are predicated on what the turn's main action was: say, "Every turn when going: ..." The every turn stage is not until later in the turn sequence, so such rules can only work if we keep the main parser-generated action of the turn in the action variables when we finish up here: so that's what we do. (Note that `BeginAction` preserves the values of the action variables, storing copies on the stack, so whatever may have happened during processing, we finish this routine with the same action variable values that we set at the beginning.)

Finally, note that an out of world action stops the turn sequence early, at the end of action generation: this is what prevents the time of day advancing, every turn rules from firing, and so forth – see the Standard Rules.

```
[ GENERATE_ACTION_R i j k l;
    if (EarlyInTurnSequence == false) rfalse; ! Prevent use outside top level
    EarlyInTurnSequence = false;

    action = parser_results-->ACTION_PRES;
    act_requester = nothing; if (actor ~= player) act_requester = player;

    inp1 = 0; inp2 = 0; multiflag = false;
    if (parser_results-->NO_INPS_PRES >= 1) {
        inp1 = parser_results-->INP1_PRES; if (inp1 == 0) multiflag = true;
    }
    if (parser_results-->NO_INPS_PRES >= 2) {
        inp2 = parser_results-->INP2_PRES; if (inp2 == 0) multiflag = true;
    }

    if (inp1 == 1) {
        noun = nothing; ! noun = special_number1;
    } else noun = inp1;
    if (inp2 == 1) {
        second = nothing;
        ! if (inp1 == 1) second = special_number2; else second = special_number1;
    } else second = inp2;

    if (multiflag) {
        if (multiple_object-->0 == 0) { L__M(##Miscellany, 2); return; }
        if (toomany_flag) { toomany_flag = false; L__M(##Miscellany, 1); }
        GenerateMultipleActions();
        multiflag = false;
    } else BeginAction(action, noun, second);

    if ((actor ~= player) || (act_requester)) action = ##Wait;
    actor = player; act_requester = 0;

    if (meta) { RulebookSucceeds(); rtrue; }
    rfalse;
];
```

§10. Generate Multiple Actions. So this routine is used to issue the individual actions necessary when a multiple object list has been supplied as either the noun or second noun part of an action generated by the parser. Note that we stop processing the list in the event of the game ending, or of the `location` variable changing its value, which can happen either through movement of the player, or through passage from darkness to light or vice versa.

We use `RunParagraphOn` to omit skipped lines as paragraph breaks between the results from any item in the list: this is both more condensed on screen in ordinary lists, and might allow the user to play tricks such as gathering up reports from a list and delivering them later in some processed way.

```
[ GenerateMultipleActions initial_location k item;
    initial_location = location;
    for (k=1: k<=multiple_object-->0: k++) {
        item = multiple_object-->k;
        RunParagraphOn();
        if (inp1 == 0) { inp1 = item; BeginAction(action, item, second, item); inp1 = 0; }
        else { inp2 = item; BeginAction(action, noun, item, item); inp2 = 0; }
        if (deadflag) return;
        if (location ~= initial_location) { L__M(##Miscellany, 51); return; }
    }
];
```

§11. Timed Events Rule. A timed event is a rule stored in the `TimedEventsTable`, an I6 table array: zero entries in this table are ignored, and the sequence is significant only if more than one event goes off at the same moment, in which case earlier entries go off first. Each rule in the table has a corresponding timer value in `TimedEventTimesTable`. If this is negative, it represents a number of turns to go before the event happens – or properly speaking, the number of times the timed events rule is invoked. Otherwise the timer value must be a valid time of day at which the event happens (note that valid times are all non-negative integers). We allow a bracket of 30 minutes after the event time proper; this is designed to cope with situations in which the user sets some timed events, then advances the clock by hand (or uses a long step time, say in which each turn equates to 20 minutes).

Because an event is struck out of the table just before it is fired, it will not continue to go off the rest of the half-hour. Moreover, because the striking out happens *before* rather than after the rule fires, a rule can re-time itself to go off again later, somewhat like the snooze feature on an alarm clock, without the risk of it going off again immediately in the same use of the timed events rule: there is guaranteed to be a blank slot in the timer array at or before the current position because we have just blanked one.

```
[ TIMED_EVENTS_R i event_timer fire rule;
    for (i=1: i<=(TimedEventsTable-->0): i++)
        if ((rule=TimedEventsTable-->i) ~= 0) {
            event_timer = TimedEventTimesTable-->i; fire = false;
            if (event_timer<0) {
                (TimedEventTimesTable-->i)++;
                if (TimedEventTimesTable-->i == 0) fire = true;
            } else {
                if ((the_time >= event_timer) && (the_time < event_timer+30)) fire = true;
            }
            if (fire) {
                TimedEventsTable-->i = 0;
                ProcessRulebook(rule);
            }
        }
    rfalse;
];
```

§12. Setting Timed Events. This is the corresponding routine which adds events to the timer tables, and is used to define phrases like "the cuckoo clock explodes in 7 turns from now" or "the cuckoo clock explodes at 4 PM". Here the `rule` would be "cuckoo clock explodes", and the `event_time` would either be 4 PM with `absolute_time` set, or simply 7 with `absolute_time` clear.

Note that the same event can occur only once in the timer tables: a new setting for its firing overwrites an old one. (This ensures that the table does not slowly balloon in size if the user has not been careful to ensure that events always fire.)

```
[ SetTimedEvent rule event_time absolute_time i b;
    for (i=1: i<=(TimedEventsTable-->0): i++) {
        if (rule == TimedEventsTable-->i) { b=i; break; }
        if ((b==0) && (TimedEventsTable-->i == 0)) b=i;
    }
    if (b==0) return RunTimeProblem(RTP_TOOMANYEVENTS);
    TimedEventsTable-->b = rule;
    if (absolute_time) TimedEventTimesTable-->b = event_time;
    else TimedEventTimesTable-->b = -event_time;
];
```

§13. Setting Time Of Day. This is the old I6 library routine `SetTime`, which is no longer used in I7 at present; but might be, some day.

```
Global time_step;

[ SetTime t s;
    the_time = t; time_rate = s; time_step = 0;
    if (s < 0) time_step = 0-s;
];
```

§14. Advance Time Rule. This rule advances the two measures of the passing of time: the number of `turns` of play, and `the_time` of day.

```
[ ADVANCE_TIME_R;
    turns++;
    if (the_time ~= NULL) {
        if (time_rate >= 0) the_time = the_time+time_rate;
        else {
            time_step--;
            if (time_step == 0) {
                the_time++;
                time_step = -time_rate;
            }
        }
        the_time = the_time % TWENTY_FOUR_HOURS;
    }
    rfalse;
];
```

§15. Note Object Acquisitions Rule. See the Standard Rules for comment on this.

```
[ NOTE_OBJECT_ACQUISITIONS_R obj;
    objectloop (obj in player) give obj moved;
    objectloop (obj has concealed)
        if (IndirectlyContains(player, obj)) give obj ~concealed;
    #Ifdef RUCKSACK_CLASS;
    objectloop (obj in player)
        if (obj ofclass RUCKSACK_CLASS)
            SACK_OBJECT = obj;
    objectloop (obj ofclass RUCKSACK_CLASS && obj provides component_parent
        && obj.component_parent == player)
        SACK_OBJECT = obj;
    #Endif;
    rfalse;
];
```

§16. Resurrect Player If Asked Rule. If a rule in the "when play ends" rulebook set `resurrect_please`, by executing the "resume the game" phrase, then this is where we notice that: making the shutdown rulebook succeed then tells `Main` to fall back into the turn sequence.

```
[ RESURRECT_PLAYER_IF_ASKED_R;
    if (resurrect_please) {
        RulebookSucceeds(); resurrect_please = false;
        deadflag = 0; story_complete = false; rtrue;
    }
    rfalse;
];
```

§17. Ask The Final Question Rule. And so we come to the bittersweet end: we ask the final question endlessly, until the player gives a reply which takes drastic enough action to destroy the current execution context in the VM, for instance by typing QUIT, RESTART, UNDO or RESTORE. The question and answer are all managed by the activity, which is defined in I7 source text in the Standard Rules.

```
[ ASK_FINAL_QUESTION_R;
    print "^";
    while (true) {
        CarryOutActivity(DEALING_WITH_FINAL_QUESTION_ACT);
        DivideParagraphPoint();
    }
];
```

§18. Read The Final Answer Rule. This erases the current command, so is a technique we couldn't use during actual play, but here commands are but a distant memory. So we can use the same buffers for the final question as for game commands.

```
[ READ_FINAL_ANSWER_R;
    DrawStatusLine();
    KeyboardPrimitive(buffer, parse);
    players_command = 100 + WordCount();
    num_words = WordCount();
    wn = 1;
    rfalse;
];
```

§19. Immediately Restart VM Rule. Now for four rules acting on typical responses to the final question.

```
[ IMMEDIATELY_RESTART_VM_R; @restart; ];
```

§20. Immediately Restore Saved Game Rule. It is almost certainly unnecessary to set `actor` to `player` here, but we do so just in case, because `RESTORE_THE_GAME_R` is protected against doing anything when it thinks it might have been called erroneously through a command like "DAPHNE, RESTORE". (Out of world actions should never be carried out that way, but again, it's a precaution.)

```
[ IMMEDIATELY_RESTORE_SAVED_R; actor = player; RESTORE_THE_GAME_R(); ];
```

§21. Immediately Quit Rule.

```
[ IMMEDIATELY_QUIT_R; @quit; ];
```

§22. Immediately Undo Rule. An UNDO is disallowed when `turns` is 1, because there is nothing to revert to: but suppose the player died or won as a result of the very first command? Then the game will be over with `turns` equal to 1, but UNDO disallowed, even though there is a saved state to revert to, captured just before that command was typed. To prevent this, we increment `turns` to include the one only partially completed, but only if a command was actually typed. (If the player died as a result of a monstrously unfair rule applied before the very first command had been typed, UNDO is indeed impossible, and `turns` is left at 1.)

```
[ IMMEDIATELY_UNDO_R;
    if (not_yet_in_play == false) turns++;
    Perform_Undo();
    if (not_yet_in_play == false) turns--;
];
```

§23. Print Obituary Headline Rule. Finally, definitions of three primitive rules for the "printing the player's obituary" activity.

```
[ PRINT_OBITUARY_HEADLINE_R;
    print "^^    ";
    VM_Style(ALERT_VMSTY);
    print "***";
    if (deadflag == 1) L__M(##Miscellany, 3);
    if (deadflag == 2) L__M(##Miscellany, 4);
    if (deadflag == 3) L__M(##Miscellany, 75);
    if (deadflag ~= 0 or 1 or 2 or 3)  {
        print " ";
        if (deadflag ofclass Routine) (deadflag)();
        if (deadflag ofclass String) print (string) deadflag;
        print " ";
    }
    print "***";
    VM_Style(NORMAL_VMSTY);
    print "^^"; #Ifndef NO_SCORE; print "^"; #Endif;
    rfalse;
];
```

§24. Print Final Score Rule.

```
[ PRINT_FINAL_SCORE_R;
    #Ifndef NO_SCORING; ANNOUNCE_SCORE_R(); #Endif;
    rfalse;
];
```

§25. Display Final Status Line Rule.

```
[ DISPLAY_FINAL_STATUS_LINE_R;
    sline1 = score; sline2 = turns;
    rfalse;
];
```

Actions Template B/actt

Purpose

To try actions by people in the model world, processing the necessary rulebooks.

§**1. Summary.** To review: an action is an impulse to do something by a person in the model world. Commands such as DROP POTATO are converted into actions ("dropping the Idaho potato"); sometimes they succeed, sometimes they fail. While they run, the fairly complicated details are stored in a suite of I6 global variables such as `actor`, `noun`, `inp1`, and so on (see "OrderOfPlay.i6t" for details); the running of an action is mainly a matter of processing many rulebooks, chief among them they "action processing rules".

In general, actions can come from five different sources:

(i) As a result of parsing the player's command: there are actually two ways this can happen, one if the command calls for a single action, and another if it calls for a whole run of them (like TAKE ALL). See the rules in "OrderOfPlay.i6t".

(ii) From an I7 "try" phrase, in which case `TryAction` is called.

(iii) From an I6 angle-bracket-notation such as `<Wait>`, though this is a syntax which is deprecated now, and is never normally used in I7. The I6 compiler converts such a syntax into a call to the `R_Process` below.

(iv) Through conversion of an existing action. For instance, "removing the cup from the table" is converted in the Standard Rules to "taking the cup". This is done via a routine called `GVS_Convert`.

(v) When a request is successful, the "carry out requested actions" rule turns the original action – a request by the player, such as is produced by CHOPIN, PLAY POLONAISE – into an action by the person asked, such as "Chopin playing the Polonaise".

Certain exceptional cases can arise for other reasons:

(vi) Implicit taking actions are generated by the "carrying requirements rule" when the actor tries something which requires him to be holding an item which he can see, but is not currently holding.

(vii) A partial but intentionally incomplete form of the "looking" action is generated when describing the new location at the end of a "going" action.

In every case except (vii), the action is carried out by `BeginAction`, the single routine which unifies all of these approaches. Except the last one.

This segment of the template is divided into two: first, the I6 code needed for (i) to (vii), the alternative ways for actions to begin; and secondly the common machinery into which all actions eventually pass.

§**2. Action Data.** This is perhaps a good place to document the `ActionData` array, a convenient table of metadata about the actions. Since this is compiled by NI, the following structure can't be modified here without making matching changes in NI. `ActionData` is an I6 `table` containing a series of fixed-length records, one on each action.

The routine `FindAction` locates the record in this table for a given action number, returning its word offset within the table: the argument −1 means "the current action".

```
Constant AD_ACTION = 0; ! The I6 action number (0 to 4095)
Constant AD_REQUIREMENTS = 1; ! Such as requiring light; a bitmap, see below
Constant AD_NOUN_KOV = 2; ! Kind of value of the first noun
Constant AD_SECOND_KOV = 3; ! Kind of value of the second noun
Constant AD_VARIABLES_CREATOR = 4; ! Routine to initialise variables owned
Constant AD_VARIABLES_ID = 5; ! Frame ID for variables owned by action
Constant AD_RECORD_SIZE = 6;
[ FindAction fa t;
    if (fa == -1) fa = action;
    t = 1;
    while (t <= ActionData-->0) {
        if (fa == ActionData-->t) return t;
        t = t + AD_RECORD_SIZE;
    }
    rfalse;
];
[ ActionNumberIndexed i;
    if ((i>=0) && (i < AD_RECORDS)) return ActionData-->(i*AD_RECORD_SIZE + AD_ACTION + 1);
    return 0;
];
```

§**3. Requirements Bitmap.** As noted above, the `AD_REQUIREMENTS` field is a bitmap of flags for various possible action requirements:

```
Constant TOUCH_NOUN_ABIT   = $$00000001;
Constant TOUCH_SECOND_ABIT = $$00000010;
Constant LIGHT_ABIT        = $$00000100;
Constant NEED_NOUN_ABIT    = $$00001000;
Constant NEED_SECOND_ABIT  = $$00010000;
Constant OUT_OF_WORLD_ABIT = $$00100000;
Constant CARRY_NOUN_ABIT   = $$01000000;
Constant CARRY_SECOND_ABIT = $$10000000;
[ NeedToCarryNoun;       return TestActionMask(CARRY_NOUN_ABIT); ];
[ NeedToCarrySecondNoun; return TestActionMask(CARRY_SECOND_ABIT); ];
[ NeedToTouchNoun;       return TestActionMask(TOUCH_NOUN_ABIT); ];
[ NeedToTouchSecondNoun; return TestActionMask(TOUCH_SECOND_ABIT); ];
[ NeedLightForAction;    return TestActionMask(LIGHT_ABIT); ];
[ TestActionMask match mask at;
    at = FindAction(-1);
    if (at == 0) rfalse;
    mask = ActionData-->(at+AD_REQUIREMENTS);
    if (mask & match) rtrue;
    rfalse;
];
```

§**4. Try Action.** This is method (ii) in the summary above.

```
[ TryAction req by ac n s stora smeta tbits saved_command text_of_command;
    if (stora) return STORED_ACTION_TY_New(ac, n, s, by, req, stora);
    tbits = req & (16+32);
    req = req & 1;
    @push actor; @push act_requester; @push inp1; @push inp2;
    @push parsed_number; smeta = meta;
    actor = by; if (req) act_requester = player; else act_requester = 0;
    by = FindAction(ac);
    if (by) {
        if (ActionData-->(by+AD_NOUN_KOV) == OBJECT_TY) inp1 = n;
        else { inp1 = 1; parsed_number = n; }
        if (ActionData-->(by+AD_SECOND_KOV) == OBJECT_TY) inp2 = s;
        else { inp2 = 1; parsed_number = s; }
        if (((ActionData-->(by+AD_NOUN_KOV) == UNDERSTANDING_TY) ||
            (ActionData-->(by+AD_SECOND_KOV) == UNDERSTANDING_TY)) && (tbits)) {
            saved_command = INDEXED_TEXT_TY_Create();
            INDEXED_TEXT_TY_Cast(players_command, SNIPPET_TY, saved_command);
            text_of_command = INDEXED_TEXT_TY_Create();
            INDEXED_TEXT_TY_Cast(parsed_number, TEXT_TY, text_of_command);
            SetPlayersCommand(text_of_command);
            if (tbits == 16) {
                n = players_command; inp1 = 1; parsed_number = players_command;
            } else {
                s = players_command; inp2 = 1; parsed_number = players_command;
            }
            BlkFree(text_of_command);
            @push consult_from; @push consult_words;
            consult_from = 1; consult_words = parsed_number - 100;
        }
    }
    BeginAction(ac, n, s, 0, true);
    if (saved_command) {
        @pull consult_words; @pull consult_from;
        SetPlayersCommand(saved_command);
        BlkFree(saved_command);
    }
    meta = smeta; @pull parsed_number;
    @pull inp2; @pull inp1; @pull act_requester; @pull actor;
    TrackActions(true, smeta);
];
```

§**5. I6 Angle Brackets.** This is method (iii) in the summary above. The routine here has slightly odd conventions and a curious name which would take too long to explain: neither can be changed without amending the veneer code within the I6 compiler.

```
[ R_Process a i j;
    @push inp1; @push inp2;
    inp1 = i; inp2 = j; BeginAction(a, i, j);
    @pull inp2; @pull inp1;
];
```

§**6. Conversion.** This is method (iv) in the summary above.

```
Global converted_action_outcome = -1;
[ GVS_Convert ac n s;
    converted_action_outcome = BeginAction(ac, n, s);
    rtrue;
];
[ ConvertToGoingWithPush i oldrm newrm infl;
    i=noun;
    if (IndirectlyContains(noun, actor) == false) { move i to actor; infl = true; }
    move_pushing = i;
    oldrm = LocationOf(noun);
    BeginAction(##Go, second);
    newrm = LocationOf(actor);
    move_pushing = nothing; move i to newrm;
    if (newrm ~= oldrm) {
        if (IndirectlyContains(i, player)) TryAction(0, player, ##Look, 0, 0);
        RulebookSucceeds();
    } else RulebookFails();
];
```

§**7. Implicit Take.** This is method (vi) in the summary above.

```
[ ImplicitTake obj ks;
    if (actor == player) L__M(##Miscellany, 69, obj);
    else L__M(##Miscellany, 68, obj);
    ClearParagraphing();
    @push keep_silent; keep_silent = true;
    if (act_requester) TryAction(true, actor, ##Take, obj, nothing);
    else TryAction(false, actor, ##Take, obj, nothing);
    @pull keep_silent;
    if (obj in actor) rtrue;
    rfalse;
];
```

§**8. Look After Going.** This is method (vii) in the summary above.

Fundamentally, room descriptions arise through looking actions, but they are also printed after successful going actions, with a special form of paragraph break (see "Printing.i6t" for an explanation of this). Room descriptions through looking are always given in full, unless we have SUPERBRIEF mode set.

```
[ LookAfterGoing;
    GoingLookBreak();
    AbbreviatedRoomDescription();
];
```

§**9. Abbreviated Room Description.** This is used when we want a room description with the same abbreviation conventions as after a going action, and we don't quite want a looking action fully to take place. We nevertheless want to be sure that the action variables for looking exist, and in particular, we want to set the "room-describing action" variable to the action which was prevailing when the room description was called for. We also set "abbreviated form allowed" to "true": when the ordinary looking action is running, this is "false".

The actual description occurs during `LookSub`, which is the specific action processing stage for the "looking" action: thus, we use the check, carry out, after and report rules as if we were "looking", but are unaffected by before or instead rules.

Uniquely, this pseudo-action does not use `BeginAction`: it works only through the specific action processing rules, not the main action-processing ones, though that is not easy to see from the code below because it is hidden in the call to `LookSub`. The `-Sub` suffix is an I6 usage identifying this as the routine to go along with the action `##Look`, and so it is, but it looks nothing like the `LookSub` of the old I6 library. NI compiles `-Sub` routines like so:

```
    [ LookSub; return GenericVerbSub(153,154,155); ];
```

(with whatever rulebook numbers are appropriate). `GenericVerbSub` then runs through the specific action processing stage.

```
[ AbbreviatedRoomDescription  prior_action pos frame_id;
    prior_action = action;
    action = ##Look;
    pos = FindAction(##Look);
    if ((pos) && (ActionData-->(pos+AD_VARIABLES_CREATOR))) {
        frame_id = ActionData-->(pos+AD_VARIABLES_ID);
        Mstack_Create_Frame(ActionData-->(pos+AD_VARIABLES_CREATOR), frame_id);
        ProcessRulebook(SETTING_ACTION_VARIABLES_RB);
        (MStack-->MstVO(frame_id, 0)) = prior_action; ! "room-describing action"
        (MStack-->MstVO(frame_id, 1)) = true; ! "abbreviated form allowed"
    }
    LookSub(); ! The I6 verb routine for "looking"
    if (frame_id) Mstack_Destroy_Frame(ActionData-->(pos+AD_VARIABLES_CREATOR), frame_id);
    action = prior_action;
];
```

§10. Begin Action. We now begin the second half of the segment: the machinery which handles all actions.

The significance of 4096 here is that this is how I6 distinguishes genuine actions – numbered upwards in order of creation – from what I6 calls "fake actions" – numbered upwards from 4096. Fake actions are hardly used at all in I7, and certainly shouldn't get here, but it's possible nonetheless using I6 angled-brackets, so... In other respects all we do is to save details of whatever current action is happening onto the stack, and then call `ActionPrimitive`.

```
[ BeginAction a n s moi notrack  rv;
    ChronologyPoint();
    @push action; @push noun; @push second; @push self; @push multiple_object_item;
    action = a; noun = n; second = s; self = noun; multiple_object_item = moi;
    if (action < 4096) rv = ActionPrimitive();
    @pull multiple_object_item; @pull self; @pull second; @pull noun; @pull action;
    if (notrack == false) TrackActions(true, meta);
    return rv;
];
```

§11. Action Primitive. This is somewhat different from the I6 library counterpart which gives it its name, but the idea is the same. It has no arguments at all: everything it needs to know is now stored in global variables. The routine looks long, but really contains little: it's all just book-keeping, printing debugging information if ACTIONS is in force, etc., with all of the actual work delegated to the action processing rulebook.

We use a rather sneaky device to handle out-of-world actions, those for which the `meta` flag is set: we make it look to the system as if the "action processing rulebook" is being followed, so that all its variables are created and placed in scope, but at the crucial moment we descend to the specific action processing rules directly instead of processing the main rulebook. This is what short-circuits out of world actions and protects them from before and instead rules: see the Standard Rules for more discussion of this.

```
[ ActionPrimitive  rv p1 p2 p3 p4 p5 frame_id;
    MStack_CreateRBVars(ACTION_PROCESSING_RB);
    if ((keep_silent == false) && (multiflag == false)) DivideParagraphPoint();
    reason_the_action_failed = 0;
    frame_id = -1;
    p1 = FindAction(action);
    if ((p1) && (ActionData-->(p1+AD_VARIABLES_CREATOR))) {
        frame_id = ActionData-->(p1+AD_VARIABLES_ID);
        Mstack_Create_Frame(ActionData-->(p1+AD_VARIABLES_CREATOR), frame_id);
    }
    if (ActionVariablesNotTypeSafe()) {
        if (frame_id ~= -1)
            Mstack_Destroy_Frame(ActionData-->(p1+AD_VARIABLES_CREATOR), frame_id);
        MStack_DestroyRBVars(ACTION_PROCESSING_RB);
        return;
    }
    ProcessRulebook(SETTING_ACTION_VARIABLES_RB);
    #IFDEF DEBUG;
    if ((trace_actions) && (FindAction(-1))) {
        print "["; p1=actor; p2=act_requester; p3=action; p4=noun; p5=second;
        DB_Action(p1,p2,p3,p4,p5);
        print "]^"; ClearParagraphing();
    }
    ++debug_rule_nesting;
    #ENDIF;
    TrackActions(false, meta);
    BeginFollowRulebook();
    if ((meta) && (actor ~= player)) { L__M(##Miscellany, 74, actor); rv = RS_FAILS; }
    else if (meta) { DESCEND_TO_SPECIFIC_ACTION_R(); rv = RulebookOutcome(); }
    else { ProcessRulebook(ACTION_PROCESSING_RB); rv = RulebookOutcome(); }
    #IFDEF DEBUG;
    --debug_rule_nesting;
    if ((trace_actions) && (FindAction(-1))) {
        print "["; DB_Action(p1,p2,p3,p4,p5); print " - ";
        switch (rv) {
            RS_SUCCEEDS: print "succeeded";
            RS_FAILS: print "failed";
                #IFNDEF MEMORY_ECONOMY;
                if (reason_the_action_failed)
                    print " the ",
                        (RulePrintingRule) reason_the_action_failed;
                #ENDIF;
            default: print "ended without result";
        }
        print "]^"; say__p = 1;
        SetRulebookOutcome(rv); ! In case disturbed by printing activities
    }
    #ENDIF;
    if (rv == RS_SUCCEEDS) UpdateActionBitmap();
    EndFollowRulebook();
    if (frame_id ~= -1) {
        p1 = FindAction(action);
        Mstack_Destroy_Frame(ActionData-->(p1+AD_VARIABLES_CREATOR), frame_id);
    }
    MStack_DestroyRBVars(ACTION_PROCESSING_RB);
    if ((keep_silent == false) && (multiflag == false)) DivideParagraphPoint();
    if (rv == RS_SUCCEEDS) rtrue;
    rfalse;
];
```

§**12. Type Safety.** Some basic action requirements have to be met before we can go any further: if they aren't, then it isn't type-safe even to run the action processing rulebook.

(i) For an out of world action, we set the `meta` flag. Otherwise:
(ii) If either the noun or second noun is a topic, then this is an action arising from parsing (such actions do not arise through the "try" phrase, unless by stored actions in which case this has all happened before and doesn't need to be done again) – the parser places details of which words make up the topic in the I6 global variables `consult_words` and `consult_from`. We convert them to a valid I7 snippet value.
(iii) If either the first or second noun is supposed to be an object but seems here to be a value, or vice versa, we stop with a parser error. (This should be fairly difficult to provoke: NI's type-checking will make it difficult to arrange without I6 subterfuges.)
(iv) If either the first or second noun is supposed to be an object and required to exist, yet is missing, we use the "supplying a missing noun" or "supplying a missing second noun" activities to fill the void.

We return `true` if type safety is violated, `false` if all is well.

```
[ ActionVariablesNotTypeSafe mask noun_kova second_kova at;
    at = FindAction(-1); if (at == 0) rfalse; ! For any I6-defined actions

    noun_kova = ActionData-->(at+AD_NOUN_KOV);
    second_kova = ActionData-->(at+AD_SECOND_KOV);

    !print "at = ", at, " nst = ", noun_kova, "^";
    !print "consult_from = ", consult_from, " consult_words = ", consult_from, "^";
    !print "inp1 = ", inp1, " noun = ", noun, "^";
    !print "inp2 = ", inp2, " second = ", second, "^";
    !print "sst = ", second_kova, "^";

    if (noun_kova == SNIPPET_TY or UNDERSTANDING_TY) {
        if (inp1 ~= 1) { inp2 = inp1; second = noun; }
        parsed_number = 100*consult_from + consult_words;
        inp1 = 1; noun = nothing; ! noun = parsed_number;
    }
    if (second_kova == SNIPPET_TY or UNDERSTANDING_TY) {
        parsed_number = 100*consult_from + consult_words;
        inp2 = 1; second = nothing; ! second = parsed_number;
    }

    mask = ActionData-->(at+AD_REQUIREMENTS);
    if (mask & OUT_OF_WORLD_ABIT) { meta = 1; rfalse; }

    if (inp1 == 1) {
        if (noun_kova == OBJECT_TY) {
            return L__M(##Miscellany, 61); }
    } else {
        if (noun_kova ~= OBJECT_TY) {
            return L__M(##Miscellany, 62); }
        if ((mask & NEED_NOUN_ABIT) && (noun == nothing)) {
            @push act_requester; act_requester = nothing;
            CarryOutActivity(SUPPLYING_A_MISSING_NOUN_ACT);
            @pull act_requester;
            if (noun == nothing) {
                if (say__p) rtrue;
                return L__M(##Miscellany, 59);
            }
        }
        if (((mask & NEED_NOUN_ABIT) == 0) && (noun ~= nothing)) {
            return L__M(##Miscellany, 60); }
    }
    if (inp2 == 1) {
        if (second_kova == OBJECT_TY) {
            return L__M(##Miscellany, 63); }
    } else {
        if (second_kova ~= OBJECT_TY) {
            return L__M(##Miscellany, 64); }
        if ((mask & NEED_SECOND_ABIT) && (second == nothing)) {
            @push act_requester; act_requester = nothing;
            CarryOutActivity(SUPPLYING_A_MISSING_SECOND_ACT);
            @pull act_requester;
            if (second == nothing) {
                if (say__p) rtrue;
                return L__M(##Miscellany, 65);
            }
        }
        if (((mask & NEED_SECOND_ABIT) == 0) && (second ~= nothing)) {
            return L__M(##Miscellany, 66); }
    }
    rfalse;
];
```

§**13. Basic Visibility Rule.** This is one of the I6 primitive rules in the action processing rulebook: see the account in the Standard Rules for details.

Note that this rule only blocks the player from acting in darkness: this is because light is only reckoned from the player's perspective in any case, so that it would be unfair to apply the rule to any other person.

```
[ BASIC_VISIBILITY_R;
    if (act_requester) rfalse;
    if ((NeedLightForAction()) &&
        (actor == player) &&
        (ProcessRulebook(VISIBLE_RB)) &&
        (RulebookSucceeded())) {
        BeginActivity(REFUSAL_TO_ACT_IN_DARK_ACT);
        if (ForActivity(REFUSAL_TO_ACT_IN_DARK_ACT)==false) L__M(##Miscellany, 17);
        EndActivity(REFUSAL_TO_ACT_IN_DARK_ACT);
        reason_the_action_failed = BASIC_VISIBILITY_R;
        RulebookFails();
        rtrue;
    }
    rfalse;
];
```

§14. Basic Accessibility Rule. This is one of the I6 primitive rules in the action processing rulebook: see the account in the Standard Rules for details.

```
[ BASIC_ACCESSIBILITY_R mask at;
    if (act_requester) rfalse;
    at = FindAction(-1);
    if (at == 0) rfalse;
    mask = ActionData-->(at+AD_REQUIREMENTS);
    if ((mask & TOUCH_NOUN_ABIT) && noun && (inp1 ~= 1)) {
        if (noun ofclass K3_direction) {
            RulebookFails();
            reason_the_action_failed = BASIC_ACCESSIBILITY_R;
            if (actor~=player) rtrue;
            return L__M(##Miscellany, 67);
        }
        if (ObjectIsUntouchable(noun, (actor~=player), FALSE, actor)) {
            RulebookFails();
            reason_the_action_failed = BASIC_ACCESSIBILITY_R;
            rtrue;
        }
    }
    if ((mask & TOUCH_SECOND_ABIT) && second && (inp2 ~= 1)) {
        if (second ofclass K3_direction) {
            RulebookFails();
            reason_the_action_failed = BASIC_ACCESSIBILITY_R;
            if (actor~=player) rtrue;
            return L__M(##Miscellany, 67);
        }
        if (ObjectIsUntouchable(second, (actor~=player), FALSE, actor)) {
            RulebookFails();
            reason_the_action_failed = BASIC_ACCESSIBILITY_R;
            rtrue;
        }
    }
    rfalse;
];
```

§15. Carrying Requirements Rule. This is one of the I6 primitive rules in the action processing rulebook: see the account in the Standard Rules for details.

```
[ CARRYING_REQUIREMENTS_R mask at;
    at = FindAction(-1);
    if (at == 0) rfalse;
    mask = ActionData-->(at+AD_REQUIREMENTS);
    if ((mask & TOUCH_NOUN_ABIT) && noun && (inp1 ~= 1)) {
        if ((mask & CARRY_NOUN_ABIT) && (noun notin actor)) {
            BeginActivity(IMPLICITLY_TAKING_ACT, noun);
            if (ForActivity(IMPLICITLY_TAKING_ACT, noun)==false)
                ImplicitTake(noun);
            EndActivity(IMPLICITLY_TAKING_ACT, noun);
            !if (act_requester) rfalse;
            if (noun notin actor) {
                RulebookFails();
                reason_the_action_failed = CARRYING_REQUIREMENTS_R;
                rtrue;
            }
        }
    }
    if ((mask & TOUCH_SECOND_ABIT) && second && (inp2 ~= 1)) {
        if ((mask & CARRY_SECOND_ABIT) && (second notin actor)) {
            BeginActivity(IMPLICITLY_TAKING_ACT, second);
            if (ForActivity(IMPLICITLY_TAKING_ACT, second)==false)
                ImplicitTake(second);
            EndActivity(IMPLICITLY_TAKING_ACT, second);
            !if (act_requester) rfalse;
            if (second notin actor) {
                RulebookFails();
                reason_the_action_failed = CARRYING_REQUIREMENTS_R;
                rtrue;
            }
        }
    }
    rfalse;
];
```

§16. Requested Actions Require Persuasion Rule. This is one of the I6 primitive rules in the action processing rulebook: see the account in the Standard Rules for details.

```
[ REQUESTED_ACTIONS_REQUIRE_R rv;
    if ((actor ~= player) && (act_requester)) {
        @push say__p;
        say__p = 0;
        rv = ProcessRulebook(PERSUADE_RB);
        if (RulebookSucceeded() == false) {
            if ((deadflag == false) && (say__p == FALSE)) L__M(##Miscellany, 72, actor);
            ActRulebookFails(rv); rtrue;
        }
        @pull say__p;
    }
    rfalse;
];
```

§17. Carry Out Requested Actions Rule. This is one of the I6 primitive rules in the action processing rulebook: see the account in the Standard Rules for details.

```
[ CARRY_OUT_REQUESTED_ACTIONS_R rv;
    if ((actor ~= player) && (act_requester)) {
        @push act_requester; act_requester = nothing;
        rv = BeginAction(action, noun, second);
        if (((meta) || (rv == false)) && (deadflag == false)) {
            if (ProcessRulebook(UNSUCCESSFUL_ATTEMPT_RB) == false) L__M(##Miscellany, 58);
        }
        @pull act_requester;
        ActRulebookSucceeds();
        rtrue;
    }
    rfalse;
];
```

§18. Generic Verb Subroutine. In I6, actions are carried out by routines with names like `TakeSub`, consisting of `-Sub` tacked on to the action name `Take`. `Sub` stands for “subroutine”: this is all a convention going back to Inform 1, which was in 1993 practically an assembler. In the I6 code generated by I7, every `-Sub` routine corresponding to an I7 action consists only of a call to `GenericVerbSub` which specifies the three rulebooks it owns: its check, carry out and report rulebooks.

```
Array Details_of_Specific_Action-->5;

[ GenericVerbSub ch co re vis rv;
    @push converted_action_outcome;
    converted_action_outcome = -1;

    Details_of_Specific_Action-->0 = true;
    if (meta) Details_of_Specific_Action-->0 = false;
    Details_of_Specific_Action-->1 = keep_silent;
    Details_of_Specific_Action-->2 = ch; ! Check rules for the action
    Details_of_Specific_Action-->3 = co; ! Carry out rules for the action
    Details_of_Specific_Action-->4 = re; ! Report rules for the action

    ProcessRulebook(SPECIFIC_ACTION_PROCESSING_RB, 0, true);
    if ((RulebookFailed()) && (converted_action_outcome == 1)) ActRulebookSucceeds();
    @pull converted_action_outcome;
    rtrue;
];
```

§**19. Work Out Details Of Specific Action Rule.** This is one of the I6 primitive rules in the specific action processing rulebook, and it's basically a trick to allow information known to the `GenericVerbSub` routine to be passed down as rulebook variables for the specific action-processing rules – in effect allowing us to pass not one but five parameters to the rulebook: the out-of-world and silence flags, plus the three specific rulebooks needed to process the action.

```
[ WORK_OUT_DETAILS_OF_SPECIFIC_R;
    MStack-->MstVO(SPECIFIC_ACTION_PROCESSING_RB, 0) = Details_of_Specific_Action-->0;
    MStack-->MstVO(SPECIFIC_ACTION_PROCESSING_RB, 1) = Details_of_Specific_Action-->1;
    MStack-->MstVO(SPECIFIC_ACTION_PROCESSING_RB, 2) = Details_of_Specific_Action-->2;
    MStack-->MstVO(SPECIFIC_ACTION_PROCESSING_RB, 3) = Details_of_Specific_Action-->3;
    MStack-->MstVO(SPECIFIC_ACTION_PROCESSING_RB, 4) = Details_of_Specific_Action-->4;
    rfalse;
];
```

§**20. Actions Bitmap.** This is a fairly large bitmap recording which actions have succeeded thus far on which nouns. It was to some extent an early attempt at implementing a past-tense system; I'm not at all sure it was successful, since it is hindered by certain restrictions – it only records action/noun combinations, for instance, and the notion of "success" is a vexed one for actions anyway. There is a clearly defined meaning, but it doesn't always correspond to what the user might expect, which is unfortunate.

```
[ TestActionBitmap obj act i j k bitmap;
    if (obj == nothing) bitmap = ActionHappened;
    else {
        if (~~(obj provides action_bitmap)) rfalse;
        bitmap = obj.&action_bitmap;
    }
    if (act == -1) return (((bitmap->0) & 1) ~= 0);
    for (i=0, k=2; i<ActionCount; i++) {
        if (act == ActionCoding-->i) {
            return (((bitmap->j) & k) ~= 0);
        }
        k = k*2; if (k == 256) { k = 1; j++; }
    }
    rfalse;
];
[ UpdateActionBitmap;
    SetActionBitmap(noun, action);
    if (action == ##Go) SetActionBitmap(location, ##Enter);
];
[ SetActionBitmap obj act i j k bitmap;
    for (i=0, k=2; i<ActionCount; i++) {
        if (act == ActionCoding-->i) {
            if (obj provides action_bitmap) {
                bitmap = obj.&action_bitmap;
                bitmap->0 = (bitmap->0) | 1;
                bitmap->j = (bitmap->j) | k;
            }
            ActionHappened->0 = (ActionHappened->0) | 1;
            ActionHappened->j = (ActionHappened->j) | k;
        }
        k = k*2; if (k == 256) { k = 1; j++; }
    }
];
```

§**21. Printing Actions.** This is really for debugging purposes, but also provides us with a way to print a stored action, for instance, or to print an action name value. (For instance, printing an action name might result in “taking”; printing a whole action might produce “Henry taking the grapefruit”.)

```
[ SayActionName act; DB_Action(0, 0, act, 0, 0, 2); ];
[ DA_Name n; if (n ofclass K3_direction) print (name) n; else print (the) n; ];
[ DA_Topic x a b c d i cf cw;
    cw = x%100; cf = x/100;
    print "~";
    for (a=cf:d<cw:d++,a++) {
        wn = a; b = WordAddress(a); c = WordLength(a);
        for (i=b:i<b+c:i++) {
            print (char) 0->i;
        }
        if (d<cw-1) print " ";
    }
    print "~";
];
[ DA_Number n; print n; ];
[ DA_TruthState n; if (n==0) print "false"; else print "true"; ];
[ DB_Action ac acr act n s for_say t at l j v c clc;
    if ((for_say == 0) && (debug_rule_nesting > 0))
        print "(", debug_rule_nesting, ") ";
    if ((ac ~= player) && (for_say ~= 2)) {
        if (acr) print "asking ", (the) ac, " to try ";
        else print (the) ac, " ";
    }
    DB_Action_Details(act, n, s, for_say);
    if ((keep_silent) && (for_say == 0)) print " - silently";
];
```

Activities Template B/acvt

Purpose

To run the necessary rulebooks to carry out an activity.

§**1. The Activities Stack.** Activities are more like nested function calls than independent processes; they finish in reverse order of starting, and are placed on a stack. This needs only very limited size in practice: 20 might seem a bit low, but making it much higher simply means that oddball bugs in the user's code – where activities recursively cause themselves ad infinitum – will be caught less efficiently.

```
Constant MAX_NESTED_ACTIVITIES = 20;
Global activities_sp = 0;
Array activities_stack --> MAX_NESTED_ACTIVITIES;
Array activity_parameters_stack --> MAX_NESTED_ACTIVITIES;
```

§**2. Rule Debugging Inhibition.** The output from RULES or RULES ALL becomes totally illegible if it is applied even to the activities printing names of objects, so this is inhibited when any such activity is running. `FixInhibitFlag` is called each time the stack changes and ensures that `inhibit_flag` has exactly this meaning.

```
Global inhibit_flag = 0;
Global saved_debug_rules = 0;
[ FixInhibitFlag n act inhibit_rule_debugging;
    for (n=0:n<activities_sp:n++) {
        act = activities_stack-->n;
        if (act == PRINTING_THE_NAME_ACT or PRINTING_THE_PLURAL_NAME_ACT or
            PRINTING_ROOM_DESC_DETAILS_ACT or LISTING_CONTENTS_ACT or
            GROUPING_TOGETHER_ACT) inhibit_rule_debugging = true;
    }
    if ((inhibit_flag == false) && (inhibit_rule_debugging)) {
        saved_debug_rules = debug_rules;
        debug_rules = 0;
    }
    if ((inhibit_flag) && (inhibit_rule_debugging == false)) {
        debug_rules = saved_debug_rules;
    }
    inhibit_flag = inhibit_rule_debugging;
];
```

§3. Testing Activities. The following tests whether a given activity `A` is currently running whose parameter-object matches description `desc`, where as usual the description is represented by a routine testing membership, and where zero `desc` means that any parameter is valid. Alternatively, we can require a specific parameter value of `val`.

```
[ TestActivity A desc val i;
    for (i=0:i<activities_sp:i++)
        if (activities_stack-->i == A) {
            if (desc) {
                if ((desc)(activity_parameters_stack-->i)) rtrue;
            } else if (val) {
                if (val == activity_parameters_stack-->i) rtrue;
            } else rtrue;
        }
    rfalse;
];
```

§4. Emptiness. An activity is defined by its three rulebooks: it is empty if they are all empty.

```
[ ActivityEmpty A x;
    x = Activity_before_rulebooks-->A;
    if (((rulebooks_array-->x)-->0) ~= NULL) rfalse;
    x = Activity_for_rulebooks-->A;
    if (((rulebooks_array-->x)-->0) ~= NULL) rfalse;
    x = Activity_after_rulebooks-->A;
    if (((rulebooks_array-->x)-->0) ~= NULL) rfalse;
    rtrue;
];
[ RulebookEmpty rb;
    if (((rulebooks_array-->rb)-->0) ~= NULL) rfalse;
    rtrue;
];
```

§5. Process Activity Rulebook. This is really much like processing any rulebook, except that `self` is temporarily set to the parameter, and is preserved by the process.

```
[ ProcessActivityRulebook rulebook parameter  rv;
    @push self;
    if (parameter) self = parameter;
    rv = ProcessRulebook(rulebook, parameter, true);
    @pull self;
    if (rv) rtrue;
    rfalse;
];
```

§6. Carrying Out Activities. This is a three-stage process; most activities are run by calling the following simple routine, but some are run by calling the three subroutines independently.

```
[ CarryOutActivity A o rv;
    BeginActivity(A, o);
    rv = ForActivity(A, o);
    EndActivity(A, o);
    return rv;
];
```

§7. Begin. Note that when an activity based on the conjectural "future action" is being run – in a few parser-related cases, that is – the identity of this action is put temporarily into `action`, and the current value saved while this takes place. That allows rules in the activity rulebooks to have preambles based on the current action, and yet be tested against what is not yet the current action.

```
[ BeginActivity A o x;
    if (activities_sp == MAX_NESTED_ACTIVITIES) return RunTimeProblem(RTP_TOOMANYACTS);
    activity_parameters_stack-->activities_sp = o;
    activities_stack-->(activities_sp++) = A;
    FixInhibitFlag();
    MStack_CreateAVVars(A);
    if (Activity_atb_rulebooks->A) { x = action; action = action_to_be; }
    o = ProcessActivityRulebook(Activity_before_rulebooks-->A, o);
    if (Activity_atb_rulebooks->A) action = x;
    return o;
];
```

§8. For.

```
[ ForActivity A o x;
    if (Activity_atb_rulebooks->A) { x = action; action = action_to_be; }
    o = ProcessActivityRulebook(Activity_for_rulebooks-->A, o);
    if (Activity_atb_rulebooks->A) action = x;
    return o;
];
```

§9. End.

```
[ EndActivity A o rv x;
    if ((activities_sp > 0) && (activities_stack-->(activities_sp-1) == A)) {
        if (Activity_atb_rulebooks->A) { x = action; action = action_to_be; }
        rv = ProcessActivityRulebook(Activity_after_rulebooks-->A, o);
        if (Activity_atb_rulebooks->A) action = x;
        activities_sp--; FixInhibitFlag();
        MStack_DestroyAVVars(A);
        return rv;
    }
    return RunTimeProblem(RTP_CANTABANDON);
];
```

§**10. Abandon.** For (very) rare cases where an activity must be abandoned midway; such an activity must be being run by calling the three stages individually, and `EndActivity` must not have been called yet.

```
[ AbandonActivity A o;
    if ((activities_sp > 0) && (activities_stack-->(activities_sp-1) == A)) {
        activities_sp--; FixInhibitFlag();
        MStack_DestroyAVVars(A);
        return;
    }
    return RunTimeProblem(RTP_CANTEND);
];
```

Rulebooks Template B/rbt

Purpose

To work through the rules in a rulebook until a decision is made.

§**1. Rule Change Stack.** The rulebook is a fundamental data structure in Inform 7: it's the basic way in which code is organised. The metaphor is that a rulebook is a loose-leaf ring-binder rather than a bound volume: so we can not only tell NI how to bind the rulebooks at compile time, we can also rearrange the pages during use at run-time.

We keep track of such run-time changes not by actually changing the rulebook arrays but by using the "rule change stack". This is a tally of any temporary abolition or modification of rules: thus, before using any rule from a rulebook we need to check that there is no note of its abolition. As the rule change stack gets larger, rule processing gets slower: this is why it's always more efficient to change rulebooks at compile time than at run-time, if there's a choice between the two.

The rule change stack contains 3-word (6 byte) records, a usage code and two operands (normally both rules).

```
Constant RULECHANGE_STACK_SIZE = 501;
Global rulechange_sp = 0;
Array rulechange_stack --> RULECHANGE_STACK_SIZE;
[ PushRuleChange usage rule1 rule2;
    if (rulechange_sp >= RULECHANGE_STACK_SIZE) return RunTimeProblem(RTP_RULESTACK);
    if ((rulechange_stack-->rulechange_sp == RS_SUCCEEDS or RS_FAILS) &&
        (KOVIsBlockValue(rulechange_stack-->(rulechange_sp+1))))
        BlkValueDestroy(rulechange_stack-->(rulechange_sp+2));
    if ((usage == RS_SUCCEEDS or RS_FAILS) && (KOVIsBlockValue(rule1)))
        rule2 = BlkValueCopy(BlkValueCreate(rule1), rule2);
    rulechange_stack-->rulechange_sp++ = usage;
    rulechange_stack-->rulechange_sp++ = rule1;
    rulechange_stack-->rulechange_sp++ = rule2;
];
```

§**2. Usage Codes.** The first word of each record indicates a usage type, of which one value is special and indicates the start of a frame. A new frame is opened on the stack each time a new "follow" occurs; recall that this processes a rulebook after consulting procedural rules, which may in turn modify the behaviour of other rules. These frame divisions, therefore, mark local scopes for modifications: anything from the top of the stack down to the topmost frame marker is currently in force.

There are then seven kinds of frame marking different ways in which rules have been modified; and three further values which indicate possible outcomes. In the following, we refer to the record as containing word 1, one of the usage codes below, word 2 and word 3.

(0) `RS_FRAME` marks the lowest point on the stack of a frame. Words 2 and 3 are not used.

(1) `RS_DONOTRUN` marks the rule or rulebook in word 1 as to be ignored.
(2) `RS_RUN` reinstates the rule or rulebook in word 1. This is useful only to reverse the effect of an `RS_DONOTRUN` frame.
(3) `RS_MOVEBEFORE` moves the rule/rulebook in word 1 to immediately before the rule/rulebook in word 2. If the latter is never invoked, nor is the former.
(4) `RS_MOVEAFTER` moves the rule/rulebook in word 1 to immediately after the rule/rulebook in word 2. If the latter is never invoked, nor is the former.
(5) `RS_DONOTUSE` marks the rule/rulebook in word 1 as to be invoked in the normal way, but then have its result (if any) ignored.
(6) `RS_USE` reverses the effect of an earlier `RS_DONOTUSE` frame.
(7) `RS_SUBSTITUTE` moves the rule/rulebook in word 1 so that it is invoked instead of the rule/rulebook in word 2. If the latter is never invoked, nor is the former.

(8) `RS_SUCCEEDS` occurs only exactly above the top of the stack: it is an ephemeral frame wiped out as soon as the stack is used again. It indicates that the most recent rule or rulebook processed ended in success. Word 2 is a flag: `true` means that a value was returned, `false` that it wasn't. If this is `true` then word 3 contains the value.
(9) `RS_FAILS` is similar, but for a failure. Note that failures can also return values.
(10) `RS_NEITHER` is similar except that it cannot return any value, so that words 2 and 3 are meaningless.

```
Constant RS_FRAME       = -1;

Constant RS_DONOTRUN = 1;
Constant RS_RUN = 2;
Constant RS_MOVEBEFORE = 3;
Constant RS_MOVEAFTER = 4;
Constant RS_DONOTUSE = 5;
Constant RS_USE = 6;
Constant RS_SUBSTITUTE = 7;

Constant RS_SUCCEEDS = 8;
Constant RS_FAILS = 9;
Constant RS_NEITHER = 10;
```

§**3. Following.** At the I6 level, there are two ways to invoke a rulebook: we can "follow" it, or simply "process" it. The former is a grander and slightly slower method which contains the latter. The I7 language talks about "considering" and "abiding by" rules and rulebooks, but these aren't handled at the I6 level: they are both achieved by "process" and the difference between the two is a matter of what is done with the result. (See the Standard Rules for the definitions.)

To "follow" a rulebook, we start a new frame, process the procedural rules, then process the rulebook, then clear the frame back off the stack. (To avoid circularity, the procedural rulebook is the only one which is exempted from the procedural rules.)

```
Global rule_frames = 0; ! Number of frames currently in force
Constant MAX_SIMULTANEOUS_FRAMES = 20;

[ FollowRulebook rulebook parameter no_paragraph_skips  rv;
    @push self;
    if ((Protect_I7_Arrays-->0 ~= 16339) || (Protect_I7_Arrays-->1 ~= 12345)) {
        print "^^*** Fatal programming error: I7 arrays corrupted ***^^";
        @quit;
    }
    if (parameter) self = parameter;
    if (rulebook ~= PROCEDURAL_RB) BeginFollowRulebook();
    rv = ProcessRulebook(rulebook, parameter, no_paragraph_skips);
    if (rulebook ~= PROCEDURAL_RB) EndFollowRulebook();
    @pull self;
    if (rv) rtrue;
    rfalse;
];

[ BeginFollowRulebook;
    PushRuleChange(RS_FRAME, RS_FRAME, RS_FRAME);
    rule_frames++;
    if (rule_frames == MAX_SIMULTANEOUS_FRAMES) {
        RunTimeProblem(RTP_TOOMANYRULEBOOKS);
        rule_frames = -1; ! For recovery: this terminates rulebook processing
        return;
    }
    ProcessRulebook(PROCEDURAL_RB, 0, true);
];

[ EndFollowRulebook r x y;
    if (rulechange_stack-->rulechange_sp == RS_SUCCEEDS) r = 1;
    else if (rulechange_stack-->rulechange_sp == RS_FAILS) r = 0;
    else r = -1;
    if (r ~= -1) {
        x = rulechange_stack-->(rulechange_sp+1);
        y = rulechange_stack-->(rulechange_sp+2);
    }
    rule_frames--;
    while (rulechange_sp > 0) {
        rulechange_sp = rulechange_sp - 3;
        if (rulechange_stack-->rulechange_sp == RS_FRAME) break;
    }
    if (rulechange_sp == 0) rule_frames = 0;
    if (r == 1) rulechange_stack-->rulechange_sp = RS_SUCCEEDS;
    else if (r == 0) rulechange_stack-->rulechange_sp = RS_FAILS;
    if (r ~= -1) {
        rulechange_stack-->(rulechange_sp+1) = x;
        rulechange_stack-->(rulechange_sp+2) = y;
    }
];
```

§4. Processing. The routine `ProcessRulebook` is arguably the most important in the whole of I7. It does something essentially simple but has deceptively complicated implications. To complicate matters, it reuses its variables to keep the virtual machine stack usage to an absolute minimum – here we use 10 locals per call to `ProcessRulebook`, which is the fewest I can comfortably manage. In the early days of I7, stack usage became a serious issue since some forms of the Frotz Z-machine interpreter provided only 4K of stack by default. ("Only" 4K. In the mid-1980s, one of the obstacles facing IF authors at Infocom was the need to get the stack usage down to fewer than 600 bytes in order that the story file could be run on the smaller home computers of the day.)

`ProcessRulebook` takes three arguments, of which only the first is compulsory:

(a) The `rulebook` is an I7 value of kind "rule", which means it can be either the ID number of a rulebook – from 0 up to $N - 1$, where N is the number of rulebooks compiled by NI, typically about 600 – or else the address of a routine representing an individual rule.
(b) The `parameter` supplied to the rulebook. Much as arguments can be supplied to a function in a conventional language's function call, so a parameter can be supplied whenever a rulebook is invoked.
(c) The value `bits` is initially a flag: if not supplied, this is `false`; if explicitly set `true`, then the rulebook is run with paragraph breaking suppressed. This is the process by which paragraph division points are placed between rules, so that if two rules both print text then a paragraph break appears between. While that is appropriate for rulebooks attached to actions or for "every turn" rules, it is disastrous for rulebooks attached to activities such as "printing the name of something". Once the routine is running, however, `bits` becomes a bitmap containing five flags, made up from the `RS_*_BIT` values defined below.

`ProcessRulebook` returns `R` if rule `R` in the rulebook (or rule) chose to "succeed" or "fail", and `false` if it made no choice. (To repeat: if the rule explicitly fails, then `ProcessRulebook` returns `true`. It's easy to write plausible-looking code which goes wrong because it assumes that the return value is success vs. failure.) The outcome of `ProcessRulebook` is lodged just above the top of this stack: thus the most recent rule or rulebook succeeded or failed if –

```
    (rulechange_stack-->rulechange_sp == RS_SUCCEEDS)
    (rulechange_stack-->rulechange_sp == RS_FAILS)
```

and otherwise there was no decision. If there was indeed a decision, then the second word of this record is a flag: `true` means that a value was returned, in which case the third word is that value, and `false` means that no value was returned, so that the third word is meaningless.

```
Constant RS_ACTIVE_BIT = 1;
Constant RS_MOVED_BIT = 2;
Constant RS_USERESULT_BIT = 4;
Constant RS_ACTIVITY = 8;
Constant RS_NOSKIPS = 16;
Constant RS_AFFECTED_BIT = 32;

Global process_rulebook_count; ! Depth of processing recursion
Global debugging_rules = false; ! Are we tracing rule invocations?

[ ProcessRulebook rulebook parameter bits rv
    x frame_base substituted_rule usage original_deadflag rbaddress ra acf gc ga;
    if (bits) bits = RS_ACTIVITY + RS_NOSKIPS;
    if (say__pc & PARA_NORULEBOOKBREAKS) bits = bits | RS_NOSKIPS;
    if (rule_frames<0) rfalse;
    if (parameter) parameter_object = parameter;
    for (x = rulechange_sp-3: x>=0: x = x - 3) {
        usage = rulechange_stack-->x;
        if (usage == RS_FRAME) { x=x+3; break; }
        if (rulechange_stack-->(x+1) == rulebook) {
            bits = bits | (RS_AFFECTED_BIT);
            if (usage == RS_MOVEBEFORE or RS_MOVEAFTER)
                bits = bits | (RS_MOVED_BIT);
        }
        if (rulechange_stack-->(x+2) == rulebook) {
            bits = bits | (RS_AFFECTED_BIT);
        }
    } if (x<0) x=0; frame_base = x;
    if ((bits & RS_MOVED_BIT) && (rv == false)) { rfalse; }
    ! rv was a call parameter: it's no longer needed and is now reused
    bits = bits | (RS_ACTIVE_BIT + RS_USERESULT_BIT);
    substituted_rule = rulebook; rv = 0;
    if (bits & RS_AFFECTED_BIT)
        for (: x<rulechange_sp: x = x + 3) {
            usage = rulechange_stack-->x;
            if (rulechange_stack-->(x+1) == rulebook) {
                if (usage == RS_DONOTRUN) bits = bits & (~RS_ACTIVE_BIT);
                if (usage == RS_RUN) bits = bits | (RS_ACTIVE_BIT);
                if (usage == RS_DONOTUSE) bits = bits & (~RS_USERESULT_BIT);
                if (usage == RS_USE) bits = bits | (RS_USERESULT_BIT);
                if (usage == RS_SUBSTITUTE)
                    substituted_rule = rulechange_stack-->(x+2);
            }
            if ((usage == RS_MOVEBEFORE) && (rulechange_stack-->(x+2) == rulebook)) {
                rv = ProcessRulebook(rulechange_stack-->(x+1),
                    parameter, (bits & RS_ACTIVITY ~= 0), true);
                if (rv) return rv;
            }
        }
    if ((bits & RS_ACTIVE_BIT) == 0) rfalse;
    ! We now reuse usage to keep the stack frame slimmer
    usage = debugging_rules;
    #ifndef MEMORY_ECONOMY;
    if (debugging_rules) DebugRulebooks(substituted_rule, parameter);
    #endif;
    ! (A routine defined in the I7 code generator)
    process_rulebook_count = process_rulebook_count + debugging_rules;
    if ((substituted_rule >= 0) && (substituted_rule < NUMBER_RULEBOOKS_CREATED)) {
        rbaddress = rulebooks_array-->substituted_rule;
        ra = rbaddress-->0; x = 0; original_deadflag = deadflag;
        if (ra ~= NULL) {
            acf = (bits & RS_ACTIVITY ~= 0);
            if (substituted_rule ~= ACTION_PROCESSING_RB) MStack_CreateRBVars(substituted_rule);
            if (ra == (-2)) {
                for (x=1: original_deadflag == deadflag: x++) {
                    ra = rbaddress-->x;
                    if (ra == NULL) break;
                    if (gc == 0) {
                        ga = ra; x++; gc = rbaddress-->x;
                        if ((gc<1) || (gc>31)) { gc = 1; x--; }
                        x++; ra = rbaddress-->x;
                    }
                    gc--;
                    if (ga ~= (-2) or action) continue;
                    if ((rv = (ProcessRulebook(ra, parameter, acf)))
                        && (bits & RS_USERESULT_BIT)) jump NonNullResult;
                }
            } else {
                for (: original_deadflag == deadflag: x++) {
                    ra = rbaddress-->x;
                    if (ra == NULL) break;
                    if ((rv = (ProcessRulebook(ra, parameter, acf)))
                        && (bits & RS_USERESULT_BIT)) jump NonNullResult;
                }
            }
            rv = 0;
            .NonNullResult;
            if (substituted_rule ~= ACTION_PROCESSING_RB) MStack_DestroyRBVars(substituted_rule);
        }
    } else {
        if ((say__p) && (bits & RS_NOSKIPS == 0)) DivideParagraphPoint();
        rv = indirect(substituted_rule);
        if (rv == 2) rv = reason_the_action_failed;
        else if (rv) rv = substituted_rule;
    }
    if (rv && (bits & RS_USERESULT_BIT)) {
        process_rulebook_count = process_rulebook_count - debugging_rules;
        if (process_rulebook_count < 0) process_rulebook_count = 0;
        #ifndef MEMORY_ECONOMY;
        if (debugging_rules) {
            spaces(2*process_rulebook_count);
            if (rulechange_stack-->rulechange_sp == RS_SUCCEEDS)
                print "[stopped: success]^";
            if (rulechange_stack-->rulechange_sp == RS_FAILS)
                print "[stopped: fail]^";
        }
        #endif;
        debugging_rules = usage;
        return rv;
    }
    if (bits & RS_AFFECTED_BIT)
        for (x=rulechange_sp-3: x>=frame_base: x = x-3) {
            if ((rulechange_stack-->x == RS_MOVEAFTER) &&
                (rulechange_stack-->(x+2) == rulebook)) {
                rv = ProcessRulebook(rulechange_stack-->(x+1),
                    parameter, (bits & RS_ACTIVITY ~= 0), true);
                if (rv) {
                    process_rulebook_count--;
                    debugging_rules = usage;
                    return rv;
                }
            }
        }
    process_rulebook_count = process_rulebook_count - debugging_rules;
    rulechange_stack-->rulechange_sp = 0;
    debugging_rules = usage;
    rfalse;
];
```

§5. Specifying Outcomes. The following provide ways for rules to succeed, fail or decline to do either. `SetRulebookOutcome` is a little different: it changes the outcome state of the most recent rule completed, not the current one. (It's used only when saving and restoring this in the actions machinery: rules should not call it.)

```
[ ActRulebookSucceeds rule_id;
    if (rule_id) reason_the_action_failed = rule_id;
    RulebookSucceeds();
];
[ ActRulebookFails rule_id;
    if (rule_id) reason_the_action_failed = rule_id;
    RulebookFails();
];
[ RulebookSucceeds weak_kind value;
    PushRuleChange(RS_SUCCEEDS, weak_kind, value);
    rulechange_sp = rulechange_sp - 3;
];
[ RulebookFails weak_kind value;
    PushRuleChange(RS_FAILS, weak_kind, value);
    rulechange_sp = rulechange_sp - 3;
];
[ RuleHasNoOutcome;
    PushRuleChange(RS_NEITHER, 0, 0);
    rulechange_sp = rulechange_sp - 3;
];
[ SetRulebookOutcome a;
    rulechange_stack-->rulechange_sp = a;
];
```

§6. Discovering Outcomes. And here is how to tell what the results were.

```
[ RulebookOutcome a;
    a = rulechange_stack-->rulechange_sp;
    if ((a == RS_FAILS) || (a == RS_SUCCEEDS)) return a;
    return RS_NEITHER;
];
[ RulebookFailed;
    if (rulechange_stack-->rulechange_sp == RS_FAILS) rtrue; rfalse;
];
[ RulebookSucceeded;
    if (rulechange_stack-->rulechange_sp == RS_SUCCEEDS) rtrue; rfalse;
];
[ ResultOfRule RB V F K a;
    if (RB) ProcessRulebook(RB, V, F);
    a = rulechange_stack-->rulechange_sp;
    if ((a == RS_FAILS) || (a == RS_SUCCEEDS)) {
        a = rulechange_stack-->(rulechange_sp + 1);
        if (a) return rulechange_stack-->(rulechange_sp + 2);
    }
    if (K) return DefaultValueOfKOV(K);
    return 0;
];
```

§7. Procedural Rule Changes. The following routines provide a sort of rule-changing API, and correspond closely to the I7 phrases documented in *Writing with Inform*, so they won't be discussed in any detail here.

```
Global DITS_said = false;
[ SuppressRule rule;
    if (rule == TURN_SEQUENCE_RB) {
        if (DITS_said == false) RunTimeProblem(RTP_DONTIGNORETURNSEQUENCE);
        DITS_said = true;
    } else PushRuleChange(RS_DONOTRUN, rule, 0);
];
[ ReinstateRule rule; PushRuleChange(RS_RUN, rule, 0); ];
[ DonotuseRule rule; PushRuleChange(RS_DONOTUSE, rule, 0); ];
[ UseRule rule; PushRuleChange(RS_USE, rule, 0); ];
[ SubstituteRule rule1 rule2; PushRuleChange(RS_SUBSTITUTE, rule2, rule1); ];
[ MoveRuleBefore rule1 rule2; PushRuleChange(RS_MOVEBEFORE, rule1, rule2); ];
[ MoveRuleAfter rule1 rule2; PushRuleChange(RS_MOVEAFTER, rule1, rule2); ];
```

§**8. Printing Rule Names.** This is the I6 printing rule used for a value of kind "rule", which as noted above can either be rulebook ID numbers in the range 0 to $N-1$ or are addresses of individual rules.

Names of rules and rulebooks take up a fair amount of space, and one of the main memory economies enforced by the "Use memory economy" option is to omit the necessary arrays. (It's not the text which is the problem so much as the table of addresses pointing to that text, which has to live in precious readable memory on the Z-machine.)

```
#IFNDEF MEMORY_ECONOMY;
{-array:Code::Phrases::RulebookNames}
#ENDIF; ! MEMORY_ECONOMY
[ RulePrintingRule R p1;
#ifndef MEMORY_ECONOMY;
    if ((R>=0) && (R<NUMBER_RULEBOOKS_CREATED)) {
        print (string) (RulebookNames-->R);
    } else {
{-call:Code::Phrases::compile_rule_printing_switch}
        print "(nameless rule at address ", R, ")";
    }
#ifnot;
    if ((R>=0) && (R<NUMBER_RULEBOOKS_CREATED)) {
        print "(rulebook ", R, ")";
    } else {
        print "(rule at address ", R, ")";
    }
#endif;
];
```

§**9. Debugging.** Two modest routines to print out the names of rules and rulebooks when they occur, in so far as memory economy allows this.

```
[ DebugRulebooks subs parameter i;
    spaces(2*process_rulebook_count);
    print "[", (RulePrintingRule) subs;
    if (parameter) print " / on O", parameter;
    print "]^";
];
[ DB_Rule R N blocked;
    if (R==0) return;
    print "[Rule ~", (RulePrintingRule) R, "~ ";
    #ifdef NUMBERED_RULES; print "(", N, ") "; #endif;
    if (blocked == false) "applies.]";
    "does not apply.]";
];
```

Parser Template B/parst

Purpose

The parser for turning the text of the typed command into a proposed action by the player.

§**1. Grammar Line Variables.** This is the I6 library parser in mostly untouched form: reformatted for template file use, and with paragraph divisions, but otherwise hardly changed at all. It is a complex algorithm but one which is known to produce good results for the most part, and it is well understood from (at time of writing) fifteen years of use. A few I7 additions have been made, but none disrupting the basic method. For instance, I7's system for resolving ambiguities is implemented by providing a `ChooseObjects` routine, just as a user of the I6 library would do.

The I6 parser uses a huge number of global variables, which is not to modern programming tastes: in the early days of Inform, the parser was essentially written in assembly-language only lightly structured by C-like syntaxes, and the Z-machine's 240 globals were more or less registers. The I6 library made no distinction between which were "private" to the parser and which allowed to be accessed by the user's code at large. The I7 template does impose that boundary, though not very strongly: the variables defined in "Output.i6t" are for general access, while the ones below should only be read or written by the parser.

```
Global best_etype;                  ! Preferred error number so far
Global nextbest_etype;              ! Preferred one, if ASKSCOPE_PE disallowed

Global parser_inflection;           ! A property (usually "name") to find object names in

Array pattern --> 32;               ! For the current pattern match
Global pcount;                      ! and a marker within it
Array pattern2 --> 32;              ! And another, which stores the best match
Global pcount2;                     ! so far

Array  line_ttype-->32;             ! For storing an analysed grammar line
Array  line_tdata-->32;
Array  line_token-->32;

Global nsns;                        ! Number of special_numbers entered so far

Global params_wanted;               ! Number of parameters needed (which may change in parsing)

Global inferfrom;                   ! The point from which the rest of the command must be inferred
Global inferword;                   ! And the preposition inferred
Global dont_infer;                  ! Another dull flag

Global cobj_flag = 0;

Global oops_from;                   ! The "first mistake" word number
Global saved_oops;                  ! Used in working this out
Array  oops_workspace -> 64;        ! Used temporarily by "oops" routine
```

```
Global held_back_mode;           ! Flag: is there some input from last time
Global hb_wn;                    ! left over?  (And a save value for wn.)
                                   ! (Used for full stops and "then".)
Global usual_grammar_after;      ! Point from which usual grammar is parsed (it may vary from
                                   ! the above if user's routines match multi-word verbs)
```

§2. Grammar Token Variables. More globals, but dealing at the level of individual tokens now.

```
Constant PATTERN_NULL = $ffff;   ! Entry for a token producing no text
Global found_ttype;              ! Used to break up tokens into type
Global found_tdata;              ! and data (by AnalyseToken)
Global token_filter;             ! For noun filtering by user routines
Global length_of_noun;           ! Set by NounDomain to no of words in noun
Global lookahead;                ! The token after the one now being matched
Global multi_mode;               ! Multiple mode
Global multi_wanted;             ! Number of things needed in multitude
Global multi_had;                ! Number of things actually found
Global multi_context;            ! What token the multi-obj was accepted for
Global indef_mode;               ! "Indefinite" mode - ie, "take a brick"
                                   ! is in this mode
Global indef_type;               ! Bit-map holding types of specification
Global indef_wanted;             ! Number of items wanted (INDEF_ALL_WANTED for all)
Constant INDEF_ALL_WANTED = 32767;
Global indef_guess_p;            ! Plural-guessing flag
Global indef_owner;              ! Object which must hold these items
Global indef_cases;              ! Possible gender and numbers of them
Global indef_possambig;          ! Has a possibly dangerous assumption
                                   ! been made about meaning of a descriptor?
Global indef_nspec_at;           ! Word at which a number like "two" was parsed
                                   ! (for backtracking)
Global allow_plurals;            ! Whether plurals presently allowed or not
Global take_all_rule;            ! Slightly different rules apply to "take all" than other uses
                                   ! of multiple objects, to make adjudication produce more
                                   ! pragmatically useful results
                                   ! (Not a flag: possible values 0, 1, 2)
Global dict_flags_of_noun;       ! Of the noun currently being parsed
                                   ! (a bitmap in #dict_par1 format)
Global pronoun__word;            ! Saved value
Global pronoun__obj;             ! Saved value
Constant comma_word = 'comma,';  ! An "untypeable word" used to substitute
                                   ! for commas in parse buffers
```

§3. Match List Variables. The most difficult tokens to match are those which refer to objects, since there is such a variety of names which can be given to any individual object, and we don't of course know which object or objects are meant. We store the possibilities (up to `MATCH_LIST_WORDS`, anyway) in a data structure called the match list.

```
Array  match_list --> MATCH_LIST_WORDS;    ! An array of matched objects so far
Array  match_classes --> MATCH_LIST_WORDS; ! An array of equivalence classes for them
Array  match_scores --> MATCH_LIST_WORDS;  ! An array of match scores for them
Global number_matched;              ! How many items in it?  (0 means none)
Global number_of_classes;           ! How many equivalence classes?
Global match_length;                ! How many words long are these matches?
Global match_from;                  ! At what word of the input do they begin?
```

§4. Words. The player's command is broken down into a numbered sequence of words, which break at spaces or certain punctuation (see the DM4). The numbering runs upwards from 1 to `WordCount()`. The following utility routines provide access to words in the current command; because buffers have different definitions in Z and Glulx, so these routines must vary also.

The actual text of each word is stored as a sequence of ZSCII values in a `->` (byte) array, with address `WordAddress(x)` and length `WordLength(x)`.

We picture the command as a stream of words to be read one at a time, with the global variable `wn` being the "current word" marker. `NextWord`, which takes no arguments, returns:
(a) 0 if the word at `wn` is unrecognised by the dictionary or `wn` is out of range,
(b) `comma_word` if the word was a comma,
(c) `THEN1__WD` if it was a full stop (because of the Infocom tradition that a full stop abbreviates for the word "then": e.g., TAKE BOX. EAST was read as two commands in succession),
(d) or the dictionary address if the word was recognised.

The current word marker `wn` is always advanced.

`NextWordStopped` does the same, but returns −1 when `wn` is out of range (e.g., by having advanced past the last word in the command).

`MoveWord(at1, b2, at2)` copies word `at2` from parse buffer `b2` – which doesn't need to be `buffer` – to word `at1` in `parse`.

```
#Ifdef TARGET_ZCODE;
[ WordCount; return parse->1; ];
[ WordAddress wordnum; return buffer + parse->(wordnum*4+1); ];
[ WordLength wordnum; return parse->(wordnum*4); ];
[ MoveWord at1 b2 at2 x y;
    x = at1*2-1; y = at2*2-1;
    parse-->x++ = b2-->y++;
    parse-->x = b2-->y;
];
#Ifnot;
[ WordCount; return parse-->0; ];
[ WordAddress wordnum; return buffer + parse-->(wordnum*3); ];
[ WordLength wordnum; return parse-->(wordnum*3-1); ];
[ MoveWord at1 b2 at2 x y;
    x = at1*3-2; y = at2*3-2;
    parse-->x++ = b2-->y++;
    parse-->x++ = b2-->y++;
    parse-->x = b2-->y;
];
#Endif;
```

```
[ WordFrom w p i j wc;
    #Ifdef TARGET_ZCODE; wc = p->1; i = w*2-1;
    #Ifnot; wc = p-->0; i = w*3-2; #Endif;
    if ((w < 1) || (w > wc)) return 0;
    j = p-->i;
    if (j == ',//') j = comma_word;
    if (j == './/') j = THEN1__WD;
    return j;
];
[ NextWord i j wc;
    #Ifdef TARGET_ZCODE; wc = parse->1; i = wn*2-1;
    #Ifnot; wc = parse-->0; i = wn*3-2; #Endif;
    wn++;
    if ((wn < 2) || (wn > wc+1)) return 0;
    j = parse-->i;
    if (j == ',//') j = comma_word;
    if (j == './/') j = THEN1__WD;
    return j;
];
[ NextWordStopped wc;
    #Ifdef TARGET_ZCODE; wc = parse->1; #Ifnot; wc = parse-->0; #Endif;
    if ((wn < 1) || (wn > wc)) { wn++; return -1; }
    return NextWord();
];
```

§**5. Snippets.** Although the idea is arguably implicit in I6, the formal concept of "snippet" is new in I7. A snippet is a value which represents a word range in the command most recently typed by the player. These words number consecutively upwards from 1, as noted above. The correspondence between (w_1, w_2), the word range, and V, the number used to represent it as an I6 value, is:

$$V = 100w_1 + (w_2 - w_1 + 1)$$

so that the remainder mod 100 is the number of words in the range. We require that $1 \leq w_1 \leq w_2 \leq N$, where N is the number of words in the current player's command. The entire command is therefore represented by:

$$C = 100 + N$$

```
[ PrintSnippet snip from to i w1 w2;
    w1 = snip/100; w2 = w1 + (snip%100) - 1;
    if ((w2<w1) || (w1<1) || (w2>WordCount())) {
        if ((w1 == 1) && (w2 == 0)) rfalse;
        return RunTimeProblem(RTP_SAYINVALIDSNIPPET, w1, w2);
    }
    from = WordAddress(w1); to = WordAddress(w2) + WordLength(w2) - 1;
    for (i=from: i<=to: i++) print (char) i->0;
];
[ SpliceSnippet snip t i w1 w2 nextw at endsnippet newlen;
    w1 = snip/100; w2 = w1 + (snip%100) - 1;
    if ((w2<w1) || (w1<1)) {
        if ((w1 == 1) && (w2 == 0)) return;
        return RunTimeProblem(RTP_SPLICEINVALIDSNIPPET, w1, w2);
    }
    @push say__p; @push say__pc;
    nextw = w2 + 1;
    at = WordAddress(w1) - buffer;
    if (nextw <= WordCount()) endsnippet = 100*nextw + (WordCount() - nextw + 1);
    buffer2-->0 = 120;
    newlen = VM_PrintToBuffer(buffer2, 120, SpliceSnippet__TextPrinter, t, endsnippet);
    for (i=0: (i<newlen) && (at+i<120): i++) buffer->(at+i) = buffer2->(WORDSIZE+i);
    #Ifdef TARGET_ZCODE; buffer->1 = at+i; #ifnot; buffer-->0 = at+i; #endif;
    for (:at+i<120:i++) buffer->(at+i) = ' ';
    VM_Tokenise(buffer, parse);
    players_command = 100 + WordCount();
    @pull say__pc; @pull say__p;
];
[ SpliceSnippet__TextPrinter t endsnippet;
    PrintText(t);
    if (endsnippet) { print " "; PrintSnippet(endsnippet); }
];
[ SnippetIncludes test snippet w1 w2 wlen i j;
    w1 = snippet/100; w2 = w1 + (snippet%100) - 1;
    if ((w2<w1) || (w1<1)) {
        if ((w1 == 1) && (w2 == 0)) rfalse;
        return RunTimeProblem(RTP_INCLUDEINVALIDSNIPPET, w1, w2);
    }
    if (metaclass(test) == Routine) {
        wlen = snippet%100;
        for (i=w1, j=wlen: j>0: i++, j--) {
            if (((test)(i, 0)) ~= GPR_FAIL) return i*100+wn-i;
        }
    }
    rfalse;
];
[ SnippetMatches snippet topic_gpr rv;
    wn=1;
    if (topic_gpr == 0) rfalse;
    if (metaclass(topic_gpr) == Routine) {
        rv = (topic_gpr)(snippet/100, snippet%100);
        if (rv ~= GPR_FAIL) rtrue;
        rfalse;
    }
    RunTimeProblem(RTP_BADTOPIC);
    rfalse;
];
```

§6. Unpacking Grammar Lines. Grammar lines are sequences of tokens in an array built into the story file, but in a format which differs depending on the virtual machine in use, so the following code unpacks the data into more convenient if larger arrays which are VM-independent.

```
[ UnpackGrammarLine line_address i size;
    for (i=0 : i<32 : i++) {
        line_token-->i = ENDIT_TOKEN;
        line_ttype-->i = ELEMENTARY_TT;
        line_tdata-->i = ENDIT_TOKEN;
    }
#Ifdef TARGET_ZCODE;
    action_to_be = 256*(line_address->0) + line_address->1;
    action_reversed = ((action_to_be & $400) ~= 0);
    action_to_be = action_to_be & $3ff;
    line_address--;
    size = 3;
#Ifnot; ! GLULX
    @aloads line_address 0 action_to_be;
    action_reversed = (((line_address->2) & 1) ~= 0);
    line_address = line_address - 2;
    size = 5;
#Endif;
    params_wanted = 0;
    for (i=0 : : i++) {
        line_address = line_address + size;
        if (line_address->0 == ENDIT_TOKEN) break;
        line_token-->i = line_address;
        AnalyseToken(line_address);
        if (found_ttype ~= PREPOSITION_TT) params_wanted++;
        line_ttype-->i = found_ttype;
        line_tdata-->i = found_tdata;
    }
    return line_address + 1;
];

[ AnalyseToken token;
    if (token == ENDIT_TOKEN) {
        found_ttype = ELEMENTARY_TT;
        found_tdata = ENDIT_TOKEN;
        return;
    }
    found_ttype = (token->0) & $$1111;
    found_tdata = (token+1)-->0;
];
```

§**7. Extracting Verb Numbers.** A long tale of woe lies behind the following. Infocom games stored verb numbers in a single byte in dictionary entries, but they did so counting downwards, so that verb number 0 was stored as 255, 1 as 254, and so on. Inform followed suit so that debugging of Inform 1 could be aided by using the then-available tools for dumping dictionaries from Infocom story files; by using the Infocom format for dictionary tables, Inform's life was easier.

But there was an implicit restriction there of 255 distinct verbs (not 256 since not all words were verbs). When Glulx raised almost all of the Z-machine limits, it made space for 65535 verbs instead of 255, but it appears that nobody remembered to implement this in I6-for-Glulx and the Glulx form of the I6 library. This was only put right in March 2009, and the following routine was added to concentrate lookups of this field in one place.

```
[ DictionaryWordToVerbNum dword verbnum;
#Ifdef TARGET_ZCODE;
    verbnum = $ff-(dword->#dict_par2);
#Ifnot; ! GLULX
    dword = dword + #dict_par2 - 1;
    @aloads dword 0 verbnum;
    verbnum = $ffff-verbnum;
#Endif;
    return verbnum;
];
```

§**8. Keyboard Primitive.** This is the primitive routine to read from the keyboard: it usually delegates this to a routine specific to the virtual machine being used, but sometimes uses a hacked version to allow TEST commands to work. (When a TEST is running, the text in the walk-through provided is fed into the buffer as if it had been typed at the keyboard.)

```
[ KeyboardPrimitive a_buffer a_table;
#Ifdef DEBUG; #Iftrue ({-value:NUMBER_CREATED(test_scenario)} > 0);
    return TestKeyboardPrimitive(a_buffer, a_table);
#Endif; #Endif;
    return VM_ReadKeyboard(a_buffer, a_table);
];
```

§**9. Reading the Command.** The `Keyboard` routine actually receives the player's words, putting the words in `a_buffer` and their dictionary addresses in `a_table`. It is assumed that the table is the same one on each (standard) call. Much of the code handles the OOPS and UNDO commands, which are not actions and do not pass through the rest of the parser. The undo state is saved – it is essentially an internal saved game, in the VM interpreter's memory rather than in an external file – and note that this is therefore also where execution picks up if an UNDO has been typed. Since UNDO recreates the former machine state perfectly, it might seem impossible to tell that an UNDO had occurred, but in fact the VM passes information back in the form of a return code from the relevant instruction, and this allows us to detect an undo. (We deal with it by printing the current location and asking another command.)

`Keyboard` can also be used by miscellaneous routines in the game to ask yes/no questions and the like, without invoking the rest of the parser.

The return value is the number of words typed.

```
[ Keyboard  a_buffer a_table  nw i w w2 x1 x2;
    sline1 = score; sline2 = turns;

    while (true) {
        ! Save the start of the buffer, in case "oops" needs to restore it
```

```
for (i=0 : i<64 : i++) oops_workspace->i = a_buffer->i;

! In case of an array entry corruption that shouldn't happen, but would be
! disastrous if it did:
#Ifdef TARGET_ZCODE;
a_buffer->0 = INPUT_BUFFER_LEN;
a_table->0 = 15;  ! Allow to split input into this many words
#Endif; ! TARGET_

! Print the prompt, and read in the words and dictionary addresses
PrintPrompt();
DrawStatusLine();
KeyboardPrimitive(a_buffer, a_table);

! Set nw to the number of words
#Ifdef TARGET_ZCODE; nw = a_table->1; #Ifnot; nw = a_table-->0; #Endif;

! If the line was blank, get a fresh line
if (nw == 0) {
    @push etype; etype = BLANKLINE_PE;
    players_command = 100;
    BeginActivity(PRINTING_A_PARSER_ERROR_ACT);
    if (ForActivity(PRINTING_A_PARSER_ERROR_ACT) == false) L__M(##Miscellany,10);
    EndActivity(PRINTING_A_PARSER_ERROR_ACT);
    @pull etype;
    continue;
}

! Unless the opening word was OOPS, return
! Conveniently, a_table-->1 is the first word on both the Z-machine and Glulx

w = a_table-->1;
if (w == OOPS1__WD or OOPS2__WD or OOPS3__WD) {
    if (oops_from == 0) { L__M(##Miscellany, 14); continue; }
    if (nw == 1) { L__M(##Miscellany, 15); continue; }
    if (nw > 2) { L__M(##Miscellany, 16); continue; }

    ! So now we know: there was a previous mistake, and the player has
    ! attempted to correct a single word of it.

    for (i=0 : i<INPUT_BUFFER_LEN : i++) buffer2->i = a_buffer->i;
    #Ifdef TARGET_ZCODE;
    x1 = a_table->9;  ! Start of word following "oops"
    x2 = a_table->8;  ! Length of word following "oops"
    #Ifnot; ! TARGET_GLULX
    x1 = a_table-->6; ! Start of word following "oops"
    x2 = a_table-->5; ! Length of word following "oops"
    #Endif; ! TARGET_

    ! Repair the buffer to the text that was in it before the "oops"
    ! was typed:
    for (i=0 : i<64 : i++) a_buffer->i = oops_workspace->i;
    VM_Tokenise(a_buffer,a_table);

    ! Work out the position in the buffer of the word to be corrected:
    #Ifdef TARGET_ZCODE;
    w = a_table->(4*oops_from + 1); ! Start of word to go
    w2 = a_table->(4*oops_from);    ! Length of word to go
    #Ifnot; ! TARGET_GLULX
    w = a_table-->(3*oops_from);      ! Start of word to go
    w2 = a_table-->(3*oops_from - 1); ! Length of word to go
```

```
            #Endif; ! TARGET_

            ! Write spaces over the word to be corrected:
            for (i=0 : i<w2 : i++) a_buffer->(i+w) = ' ';

            if (w2 < x2) {
                ! If the replacement is longer than the original, move up...
                for (i=INPUT_BUFFER_LEN-1 : i>=w+x2 : i--)
                    a_buffer->i = a_buffer->(i-x2+w2);

                ! ...increasing buffer size accordingly.
                #Ifdef TARGET_ZCODE;
                a_buffer->1 = (a_buffer->1) + (x2-w2);
                #Ifnot; ! TARGET_GLULX
                a_buffer-->0 = (a_buffer-->0) + (x2-w2);
                #Endif; ! TARGET_
            }

            ! Write the correction in:
            for (i=0 : i<x2 : i++) a_buffer->(i+w) = buffer2->(i+x1);

            VM_Tokenise(a_buffer, a_table);
            #Ifdef TARGET_ZCODE; nw = a_table->1; #Ifnot; nw = a_table-->0; #Endif;

            return nw;
        }

        ! Undo handling

        if ((w == UNDO1__WD or UNDO2__WD or UNDO3__WD) && (nw==1)) {
            Perform_Undo();
            continue;
        }
        i = VM_Save_Undo();
        #ifdef PREVENT_UNDO; undo_flag = 0; #endif;
        #ifndef PREVENT_UNDO; undo_flag = 2; #endif;
        if (i == -1) undo_flag = 0;
        if (i == 0) undo_flag = 1;
        if (i == 2) {
            VM_RestoreWindowColours();
            VM_Style(SUBHEADER_VMSTY);
            SL_Location(); print "^";
            ! print (name) location, "^";
            VM_Style(NORMAL_VMSTY);
            L__M(##Miscellany, 13);
            continue;
        }
        return nw;
    }
];
```

§**10. Parser Proper.** The main parser routine is something of a leviathan, and it has traditionally been divided into 11 lettered parts:

(A) Get the input, do OOPS and AGAIN
(B) Is it a direction, and so an implicit GO? If so go to (K)
(C) Is anyone being addressed?
(D) Get the command verb: try all the syntax lines for that verb
(E) Break down a syntax line into analysed tokens
(F) Look ahead for advance warning for `multiexcept/multiinside`
(G) Parse each token in turn (calling `ParseToken` to do most of the work)
(H) Cheaply parse otherwise unrecognised conversation and return
(I) Print best possible error message
(J) Retry the whole lot
(K) Last thing: check for THEN and further instructions(s), return.

This lettering has been preserved here, with the code under each letter now being the body of "Parser Letter A", "Parser Letter B" and so on.

Note that there are three different places where a return can happen. The routine returns only when a sensible request has been made; for a fairly thorough description of its output, which is written into the `parser_results` array and also into several globals (see "OrderOfPlay.i6t").

```
[ Parser__parse
    syntax line num_lines line_address i j k token l m;
    cobj_flag = 0;
    parser_results-->ACTION_PRES = 0;
    parser_results-->NO_INPS_PRES = 0;
    parser_results-->INP1_PRES = 0;
    parser_results-->INP2_PRES = 0;
    meta = false;
```

§**11. Parser Letter A.** Get the input, do OOPS and AGAIN.

```
    if (held_back_mode == 1) {
        held_back_mode = 0;
        VM_Tokenise(buffer, parse);
        jump ReParse;
    }
.ReType;
    cobj_flag = 0;
    actors_location = ScopeCeiling(player);
    BeginActivity(READING_A_COMMAND_ACT); if (ForActivity(READING_A_COMMAND_ACT)==false) {
        Keyboard(buffer,parse);
        players_command = 100 + WordCount();
        num_words = WordCount();
    } if (EndActivity(READING_A_COMMAND_ACT)) jump ReType;
.ReParse;
    parser_inflection = name;
    ! Initially assume the command is aimed at the player, and the verb
    ! is the first word
    num_words = WordCount();
    wn = 1;
    #Ifdef LanguageToInformese;
    LanguageToInformese();
    ! Re-tokenise:
    VM_Tokenise(buffer,parse);
    #Endif; ! LanguageToInformese
    num_words = WordCount();
    k=0;
    #Ifdef DEBUG;
    if (parser_trace >= 2) {
        print "[ ";
        for (i=0 : i<num_words : i++) {
            #Ifdef TARGET_ZCODE;
            j = parse-->(i*2 + 1);
            #Ifnot; ! TARGET_GLULX
            j = parse-->(i*3 + 1);
            #Endif; ! TARGET_
            k = WordAddress(i+1);
            l = WordLength(i+1);
            print "~"; for (m=0 : m<l : m++) print (char) k->m; print "~ ";
            if (j == 0) print "?";
            else {
                #Ifdef TARGET_ZCODE;
                if (UnsignedCompare(j, HDR_DICTIONARY-->0) >= 0 &&
                    UnsignedCompare(j, HDR_HIGHMEMORY-->0) < 0)
                    print (address) j;
                else print j;
                #Ifnot; ! TARGET_GLULX
                if (j->0 == $60) print (address) j;
                else print j;
                #Endif; ! TARGET_
            }
            if (i ~= num_words-1) print " / ";
        }
        print " ]^";
    }
    #Endif; ! DEBUG
    verb_wordnum = 1;
    actor = player;
    actors_location = ScopeCeiling(player);
    usual_grammar_after = 0;
.AlmostReParse;
    scope_token = 0;
    action_to_be = NULL;
    ! Begin from what we currently think is the verb word
.BeginCommand;
    wn = verb_wordnum;
    verb_word = NextWordStopped();
    ! If there's no input here, we must have something like "person,".
    if (verb_word == -1) {
        best_etype = STUCK_PE;
        jump GiveError;
    }
```

```
! Now try for "again" or "g", which are special cases: don't allow "again" if nothing
! has previously been typed; simply copy the previous text across
if (verb_word == AGAIN2__WD or AGAIN3__WD) verb_word = AGAIN1__WD;
if (verb_word == AGAIN1__WD) {
    if (actor ~= player) {
        L__M(##Miscellany, 20);
        jump ReType;
    }
    #Ifdef TARGET_ZCODE;
    if (buffer3->1 == 0) {
        L__M(##Miscellany, 21);
        jump ReType;
    }
    #Ifnot; ! TARGET_GLULX
    if (buffer3-->0 == 0) {
        L__M(##Miscellany, 21);
        jump ReType;
    }
    #Endif; ! TARGET_
    for (i=0 : i<INPUT_BUFFER_LEN : i++) buffer->i = buffer3->i;
    VM_Tokenise(buffer,parse);
    num_words = WordCount();
    players_command = 100 + WordCount();
    jump ReParse;
}
! Save the present input in case of an "again" next time
if (verb_word ~= AGAIN1__WD)
    for (i=0 : i<INPUT_BUFFER_LEN : i++) buffer3->i = buffer->i;
if (usual_grammar_after == 0) {
    j = verb_wordnum;
    i = RunRoutines(actor, grammar);
    #Ifdef DEBUG;
    if (parser_trace >= 2 && actor.grammar ~= 0 or NULL)
        print " [Grammar property returned ", i, "]^";
    #Endif; ! DEBUG
    if ((i ~= 0 or 1) && (VM_InvalidDictionaryAddress(i))) {
        usual_grammar_after = verb_wordnum; i=-i;
    }
    if (i == 1) {
        parser_results-->ACTION_PRES = action;
        parser_results-->NO_INPS_PRES = 0;
        parser_results-->INP1_PRES = noun;
        parser_results-->INP2_PRES = second;
        if (noun) parser_results-->NO_INPS_PRES = 1;
        if (second) parser_results-->NO_INPS_PRES = 2;
        rtrue;
    }
    if (i ~= 0) { verb_word = i; wn--; verb_wordnum--; }
    else { wn = verb_wordnum; verb_word = NextWord(); }
}
else usual_grammar_after = 0;
```

§12. Parser Letter B. Is the command a direction name, and so an implicit GO? If so, go to (K).

```
#Ifdef LanguageIsVerb;
if (verb_word == 0) {
    i = wn; verb_word = LanguageIsVerb(buffer, parse, verb_wordnum);
    wn = i;
}
#Endif; ! LanguageIsVerb
! If the first word is not listed as a verb, it must be a direction
! or the name of someone to talk to
if (verb_word == 0 || ((verb_word->#dict_par1) & 1) == 0) {
    ! So is the first word an object contained in the special object "compass"
    ! (i.e., a direction)?  This needs use of NounDomain, a routine which
    ! does the object matching, returning the object number, or 0 if none found,
    ! or REPARSE_CODE if it has restructured the parse table so the whole parse
    ! must be begun again...
    wn = verb_wordnum; indef_mode = false; token_filter = 0; parameters = 0;
    @push actor; @push action; @push action_to_be;
    actor = player; meta = false; action = ##Go; action_to_be = ##Go;
    l = NounDomain(compass, 0, 0);
    @pull action_to_be; @pull action; @pull actor;
    if (l == REPARSE_CODE) jump ReParse;
    ! If it is a direction, send back the results:
    ! action=GoSub, no of arguments=1, argument 1=the direction.
    if ((l~=0) && (l ofclass K3_direction)) {
        parser_results-->ACTION_PRES = ##Go;
        parser_results-->NO_INPS_PRES = 1;
        parser_results-->INP1_PRES = l;
        jump LookForMore;
    }
} ! end of first-word-not-a-verb
```

§13. Parser Letter C. Is anyone being addressed?

```
! Only check for a comma (a "someone, do something" command) if we are
! not already in the middle of one.  (This simplification stops us from
! worrying about "robot, wizard, you are an idiot", telling the robot to
! tell the wizard that she is an idiot.)
if (actor == player) {
    for (j=2 : j<=num_words : j++) {
        i=NextWord();
        if (i == comma_word) jump Conversation;
    }
}
jump NotConversation;
! NextWord nudges the word number wn on by one each time, so we've now
! advanced past a comma.  (A comma is a word all on its own in the table.)
.Conversation;
j = wn - 1;
if (j == 1) {
    L__M(##Miscellany, 22);
    jump ReType;
}
! Use NounDomain (in the context of "animate creature") to see if the
! words make sense as the name of someone held or nearby
wn = 1; lookahead = HELD_TOKEN;
scope_reason = TALKING_REASON;
l = NounDomain(player,actors_location,6);
scope_reason = PARSING_REASON;
if (l == REPARSE_CODE) jump ReParse;
if (l == 0) {
    if (verb_word && ((verb_word->#dict_par1) & 1)) jump NotConversation;
    L__M(##Miscellany, 23);
    jump ReType;
}
.Conversation2;
! The object addressed must at least be "talkable" if not actually "animate"
! (the distinction allows, for instance, a microphone to be spoken to,
! without the parser thinking that the microphone is human).
if (l hasnt animate && l hasnt talkable) {
    L__M(##Miscellany, 24, l);
    jump ReType;
}
! Check that there aren't any mystery words between the end of the person's
! name and the comma (eg, throw out "dwarf sdfgsdgs, go north").
if (wn ~= j) {
    if (verb_word && ((verb_word->#dict_par1) & 1)) jump NotConversation;
    L__M(##Miscellany, 25);
    jump ReType;
}
! The player has now successfully named someone.  Adjust "him", "her", "it":
PronounNotice(l);
! Set the global variable "actor", adjust the number of the first word,
! and begin parsing again from there.
verb_wordnum = j + 1;
! Stop things like "me, again":
if (l == player) {
    wn = verb_wordnum;
    if (NextWordStopped() == AGAIN1__WD or AGAIN2__WD or AGAIN3__WD) {
        L__M(##Miscellany, 20);
        jump ReType;
    }
}
actor = l;
actors_location = ScopeCeiling(l);
#Ifdef DEBUG;
if (parser_trace >= 1)
    print "[Actor is ", (the) actor, " in ", (name) actors_location, "]^";
#Endif; ! DEBUG
jump BeginCommand;
```

§14. Parser Letter D. Get the verb: try all the syntax lines for that verb.

```
.NotConversation;
if (verb_word == 0 || ((verb_word->#dict_par1) & 1) == 0) {
    if (actor == player) {
        verb_word = UnknownVerb(verb_word);
        if (verb_word ~= 0) jump VerbAccepted;
    }
    best_etype = VERB_PE;
    jump GiveError;
}
.VerbAccepted;
! We now definitely have a verb, not a direction, whether we got here by the
! "take ..." or "person, take ..." method.  Get the meta flag for this verb:
meta = ((verb_word->#dict_par1) & 2)/2;
! You can't order other people to "full score" for you, and so on...
if (meta == 1 && actor ~= player) {
    best_etype = VERB_PE;
    meta = 0;
    jump GiveError;
}
! Now let i be the corresponding verb number...
i = DictionaryWordToVerbNum(verb_word);
! ...then look up the i-th entry in the verb table, whose address is at word
! 7 in the Z-machine (in the header), so as to get the address of the syntax
! table for the given verb...
#Ifdef TARGET_ZCODE;
syntax = (HDR_STATICMEMORY-->0)-->i;
#Ifnot; ! TARGET_GLULX
syntax = (#grammar_table)-->(i+1);
#Endif; ! TARGET_
! ...and then see how many lines (ie, different patterns corresponding to the
! same verb) are stored in the parse table...
num_lines = (syntax->0) - 1;
! ...and now go through them all, one by one.
! To prevent pronoun_word 0 being misunderstood,
pronoun_word = NULL; pronoun_obj = NULL;
#Ifdef DEBUG;
if (parser_trace >= 1)
    print "[Parsing for the verb '", (address) verb_word, "' (", num_lines+1, " lines)]^";
#Endif; ! DEBUG
best_etype = STUCK_PE; nextbest_etype = STUCK_PE;
multiflag = false;
! "best_etype" is the current failure-to-match error - it is by default
! the least informative one, "don't understand that sentence".
! "nextbest_etype" remembers the best alternative to having to ask a
! scope token for an error message (i.e., the best not counting ASKSCOPE_PE).
! multiflag is used here to prevent inappropriate MULTI_PE errors
! in addition to its unrelated duties passing information to action routines
```

§15. Parser Letter E. Break down a syntax line into analysed tokens.

```
line_address = syntax + 1;

for (line=0 : line<=num_lines : line++) {

    for (i=0 : i<32 : i++) {
        line_token-->i = ENDIT_TOKEN;
        line_ttype-->i = ELEMENTARY_TT;
        line_tdata-->i = ENDIT_TOKEN;
    }

    ! Unpack the syntax line from Inform format into three arrays; ensure that
    ! the sequence of tokens ends in an ENDIT_TOKEN.

    line_address = UnpackGrammarLine(line_address);

    #Ifdef DEBUG;
    if (parser_trace >= 1) {
        if (parser_trace >= 2) new_line;
        print "[line ", line; DebugGrammarLine();
        print "]^";
    }
    #Endif; ! DEBUG

    ! We aren't in "not holding" or inferring modes, and haven't entered
    ! any parameters on the line yet, or any special numbers; the multiple
    ! object is still empty.

    inferfrom = 0;
    parameters = 0;
    nsns = 0; special_word = 0;
    multiple_object-->0 = 0;
    multi_context = 0;
    etype = STUCK_PE;

    ! Put the word marker back to just after the verb

    wn = verb_wordnum+1;
```

§16. Parser Letter F. Look ahead for advance warning for `multiexcept`/`multiinside`.

There are two special cases where parsing a token now has to be affected by the result of parsing another token later, and these two cases (multiexcept and multiinside tokens) are helped by a quick look ahead, to work out the future token now. We can only carry this out in the simple (but by far the most common) case:

```
multiexcept <one or more prepositions> noun
```

and similarly for `multiinside`.

```
    advance_warning = -1; indef_mode = false;
    for (i=0,m=false,pcount=0 : line_token-->pcount ~= ENDIT_TOKEN : pcount++) {
        scope_token = 0;

        if (line_ttype-->pcount ~= PREPOSITION_TT) i++;

        if (line_ttype-->pcount == ELEMENTARY_TT) {
            if (line_tdata-->pcount == MULTI_TOKEN) m = true;
            if (line_tdata-->pcount == MULTIEXCEPT_TOKEN or MULTIINSIDE_TOKEN  && i == 1) {
                ! First non-preposition is "multiexcept" or
                ! "multiinside", so look ahead.

                #Ifdef DEBUG;
                if (parser_trace >= 2) print " [Trying look-ahead]^";
```

```
#Endif; ! DEBUG
! We need this to be followed by 1 or more prepositions.
pcount++;
if (line_ttype-->pcount == PREPOSITION_TT) {
    ! skip ahead to a preposition word in the input
    do {
        l = NextWord();
    } until ((wn > num_words) ||
            (l && (l->#dict_par1) & 8 ~= 0));
    if (wn > num_words) {
        #Ifdef DEBUG;
        if (parser_trace >= 2)
            print " [Look-ahead aborted: prepositions missing]^";
        #Endif;
        jump LineFailed;
    }
    do {
        if (PrepositionChain(l, pcount) ~= -1) {
            ! advance past the chain
            if ((line_token-->pcount)->0 & $20 ~= 0) {
                pcount++;
                while ((line_token-->pcount ~= ENDIT_TOKEN) &&
                    ((line_token-->pcount)->0 & $10 ~= 0))
                    pcount++;
            } else {
                pcount++;
            }
        } else {
            ! try to find another preposition word
            do {
                l = NextWord();
            } until ((wn >= num_words) ||
                    (l && (l->#dict_par1) & 8 ~= 0));
            if (l && (l->#dict_par1) & 8) continue;
            ! lookahead failed
            #Ifdef DEBUG;
            if (parser_trace >= 2)
                print " [Look-ahead aborted: prepositions don't match]^";
            #endif;
            jump LineFailed;
        }
        l = NextWord();
    } until (line_ttype-->pcount ~= PREPOSITION_TT);
    ! put back the non-preposition we just read
    wn--;
    if ((line_ttype-->pcount == ELEMENTARY_TT) &&
        (line_tdata-->pcount == NOUN_TOKEN)) {
        l = Descriptors();  ! skip past THE etc
        if (l~=0) etype=l;  ! don't allow multiple objects
        k = parser_results-->INP1_PRES; @push k; @push parameters;
        parameters = 1; parser_results-->INP1_PRES = 0;
```

```
                    l = NounDomain(actors_location, actor, NOUN_TOKEN);
                    @pull parameters; @pull k; parser_results-->INP1_PRES = k;
                    #Ifdef DEBUG;
                    if (parser_trace >= 2) {
                        print " [Advanced to ~noun~ token: ";
                        if (l == REPARSE_CODE) print "re-parse request]^";
                        else {
                            if (l == 1) print "but multiple found]^";
                            if (l == 0) print "error ", etype, "]^";
                            if (l >= 2) print (the) l, "]^";
                        }
                    }
                    #Endif; ! DEBUG
                    if (l == REPARSE_CODE) jump ReParse;
                    if (l >= 2) advance_warning = l;
                }
            }
            break;
        }
    }
}
! Slightly different line-parsing rules will apply to "take multi", to
! prevent "take all" behaving correctly but misleadingly when there's
! nothing to take.
take_all_rule = 0;
if (m && params_wanted == 1 && action_to_be == ##Take)
    take_all_rule = 1;
! And now start again, properly, forearmed or not as the case may be.
! As a precaution, we clear all the variables again (they may have been
! disturbed by the call to NounDomain, which may have called outside
! code, which may have done anything!).
inferfrom = 0;
parameters = 0;
nsns = 0; special_word = 0;
multiple_object-->0 = 0;
etype = STUCK_PE;
wn = verb_wordnum+1;
```

§17. Parser Letter G. Parse each token in turn (calling `ParseToken` to do most of the work).

The `pattern` gradually accumulates what has been recognised so far, so that it may be reprinted by the parser later on.

```
for (pcount=1 : : pcount++) {
    pattern-->pcount = PATTERN_NULL; scope_token = 0;
    token = line_token-->(pcount-1);
    lookahead = line_token-->pcount;
    #Ifdef DEBUG;
    if (parser_trace >= 2)
        print " [line ", line, " token ", pcount, " word ", wn, " : ", (DebugToken) token,
        "]^";
    #Endif; ! DEBUG
    if (token ~= ENDIT_TOKEN) {
        scope_reason = PARSING_REASON;
        AnalyseToken(token);
        l = ParseToken(found_ttype, found_tdata, pcount-1, token);
        while ((l >= GPR_NOUN) && (l < -1)) l = ParseToken(ELEMENTARY_TT, l + 256);
        scope_reason = PARSING_REASON;
        if (l == GPR_PREPOSITION) {
            if (found_ttype~=PREPOSITION_TT && (found_ttype~=ELEMENTARY_TT ||
                found_tdata~=TOPIC_TOKEN)) params_wanted--;
            l = true;
        }
        else
            if (l < 0) l = false;
            else
                if (l ~= GPR_REPARSE) {
                    if (l == GPR_NUMBER) {
                        if (nsns == 0) special_number1 = parsed_number;
                        else special_number2 = parsed_number;
                        nsns++; l = 1;
                    }
                    if (l == GPR_MULTIPLE) l = 0;
                    parser_results-->(parameters+INP1_PRES) = l;
                    parameters++;
                    pattern-->pcount = l;
                    l = true;
                }
        #Ifdef DEBUG;
        if (parser_trace >= 3) {
            print "  [token resulted in ";
            if (l == REPARSE_CODE) print "re-parse request]^";
            if (l == 0) print "failure with error type ", etype, "]^";
            if (l == 1) print "success]^";
        }
        #Endif; ! DEBUG
        if (l == REPARSE_CODE) jump ReParse;
        if (l == false) break;
    }
    else {
```

```
! If the player has entered enough already but there's still
! text to wade through: store the pattern away so as to be able to produce
! a decent error message if this turns out to be the best we ever manage,
! and in the mean time give up on this line

! However, if the superfluous text begins with a comma or "then" then
! take that to be the start of another instruction

if (wn <= num_words) {
    l = NextWord();
    if (l == THEN1__WD or THEN2__WD or THEN3__WD or comma_word) {
        held_back_mode = 1; hb_wn = wn-1;
    }
    else {
        for (m=0 : m<32 : m++) pattern2-->m = pattern-->m;
        pcount2 = pcount;
        etype = UPTO_PE;
        break;
    }
}

! Now, we may need to revise the multiple object because of the single one
! we now know (but didn't when the list was drawn up).

if (parameters >= 1 && parser_results-->INP1_PRES == 0) {
    l = ReviseMulti(parser_results-->INP2_PRES);
    if (l ~= 0) { etype = l; parser_results-->ACTION_PRES = action_to_be; break; }
}
if (parameters >= 2 && parser_results-->INP2_PRES == 0) {
    l = ReviseMulti(parser_results-->INP1_PRES);
    if (l ~= 0) { etype = l; break; }
}

! To trap the case of "take all" inferring only "yourself" when absolutely
! nothing else is in the vicinity...

if (take_all_rule == 2 && parser_results-->INP1_PRES == actor) {
    best_etype = NOTHING_PE;
    jump GiveError;
}

#Ifdef DEBUG;
if (parser_trace >= 1) print "[Line successfully parsed]^";
#Endif; ! DEBUG

! The line has successfully matched the text.  Declare the input error-free...

oops_from = 0;

! ...explain any inferences made (using the pattern)...

if (inferfrom ~= 0) {
    PrintInferredCommand(inferfrom);
    ClearParagraphing();
}

! ...copy the action number, and the number of parameters...

parser_results-->ACTION_PRES = action_to_be;
parser_results-->NO_INPS_PRES = parameters;

! ...reverse first and second parameters if need be...

if (action_reversed && parameters == 2) {
    i = parser_results-->INP1_PRES;
```

```
                parser_results-->INP1_PRES = parser_results-->INP2_PRES;
                parser_results-->INP2_PRES = i;
                if (nsns == 2) {
                    i = special_number1; special_number1 = special_number2;
                    special_number2 = i;
                }
            }
            ! ...and to reset "it"-style objects to the first of these parameters, if
            ! there is one (and it really is an object)...
            if (parameters > 0 && parser_results-->INP1_PRES >= 2)
                PronounNotice(parser_results-->INP1_PRES);
            ! ...and return from the parser altogether, having successfully matched
            ! a line.
            if (held_back_mode == 1) {
                wn=hb_wn;
                jump LookForMore;
            }
            rtrue;
        } ! end of if(token ~= ENDIT_TOKEN) else
    } ! end of for(pcount++)
    .LineFailed;
    ! The line has failed to match.
    ! We continue the outer "for" loop, trying the next line in the grammar.
    if (etype > best_etype) best_etype = etype;
    if (etype ~= ASKSCOPE_PE && etype > nextbest_etype) nextbest_etype = etype;
    ! ...unless the line was something like "take all" which failed because
    ! nothing matched the "all", in which case we stop and give an error now.
    if (take_all_rule == 2 && etype==NOTHING_PE) break;
} ! end of for(line++)
! The grammar is exhausted: every line has failed to match.
```

§18. Parser Letter H. Cheaply parse otherwise unrecognised conversation and return.

(Errors are handled differently depending on who was talking. If the command was addressed to somebody else (eg, DWARF, SFGH) then it is taken as conversation which the parser has no business in disallowing.)

The parser used to return the fake action `##NotUnderstood` when a command in the form PERSON, ARFLE BARFLE GLOOP is parsed, where a character is addressed but with an instruction which the parser can't understand. (If a command such as ARFLE BARFLE GLOOP is not an instruction to someone else, the parser prints an error and requires the player to type another command: thus `##NotUnderstood` was only returned when `actor` is not the player.) And I6 had elaborate object-oriented ways to deal with this, but we won't use any of that: we simply convert to a `##Answer` action, which communicates a snippet of words to another character, just as if the player had typed ANSWER ARFLE BARFLE GLOOP TO PERSON. For I7 purposes, the fake action `##NotUnderstood` does not exist.

```
.GiveError;
    etype = best_etype;
    if (actor ~= player) {
        if (usual_grammar_after ~= 0) {
            verb_wordnum = usual_grammar_after;
            jump AlmostReParse;
        }
        wn = verb_wordnum;
        special_word = NextWord();
        if (special_word == comma_word) {
            special_word = NextWord();
            verb_wordnum++;
        }
        parser_results-->ACTION_PRES = ##Answer;
        parser_results-->NO_INPS_PRES = 2;
        parser_results-->INP1_PRES = actor;
        parser_results-->INP2_PRES = 1; special_number1 = special_word;
        actor = player;
        consult_from = verb_wordnum; consult_words = num_words-consult_from+1;
        rtrue;
    }
```

§19. Parser Letter I. Print best possible error message.

```
    ! If the player was the actor (eg, in "take dfghh") the error must be printed,
    ! and fresh input called for.  In three cases the oops word must be jiggled.
    if ((etype ofclass Routine) || (etype ofclass String)) {
        if (ParserError(etype) ~= 0) jump ReType;
    } else {
        if (verb_wordnum == 0 && etype == CANTSEE_PE) etype = VERB_PE;
        players_command = 100 + WordCount(); ! The snippet variable ‘‘player’s command’’
        BeginActivity(PRINTING_A_PARSER_ERROR_ACT);
        if (ForActivity(PRINTING_A_PARSER_ERROR_ACT)) jump SkipParserError;
    }
    pronoun_word = pronoun__word; pronoun_obj = pronoun__obj;
    if (etype == STUCK_PE) {    L__M(##Miscellany, 27); oops_from = 1; }
    if (etype == UPTO_PE) {     L__M(##Miscellany, 28);
        for (m=0 : m<32 : m++) pattern-->m = pattern2-->m;
        pcount = pcount2; PrintCommand(0); L__M(##Miscellany, 56);
    }
    if (etype == NUMBER_PE)     L__M(##Miscellany, 29);
    if (etype == CANTSEE_PE) {  L__M(##Miscellany, 30); oops_from=saved_oops; }
    if (etype == TOOLIT_PE)     L__M(##Miscellany, 31);
    if (etype == NOTHELD_PE) {  L__M(##Miscellany, 32); oops_from=saved_oops; }
    if (etype == MULTI_PE)      L__M(##Miscellany, 33);
    if (etype == MMULTI_PE)     L__M(##Miscellany, 34);
    if (etype == VAGUE_PE)      L__M(##Miscellany, 35);
    if (etype == EXCEPT_PE)     L__M(##Miscellany, 36);
    if (etype == ANIMA_PE)      L__M(##Miscellany, 37);
    if (etype == VERB_PE)       L__M(##Miscellany, 38);
    if (etype == SCENERY_PE)    L__M(##Miscellany, 39);
    if (etype == ITGONE_PE) {
        if (pronoun_obj == NULL)
                                 L__M(##Miscellany, 35);
        else                    L__M(##Miscellany, 40);
    }
    if (etype == JUNKAFTER_PE)  L__M(##Miscellany, 41);
    if (etype == TOOFEW_PE)     L__M(##Miscellany, 42, multi_had);
```

```
if (etype == NOTHING_PE) {
    if (parser_results-->ACTION_PRES == ##Remove &&
        parser_results-->INP2_PRES ofclass Object) {
        noun = parser_results-->INP2_PRES; ! ensure valid for messages
        if (noun has animate) L__M(##Take, 6, noun);
        else if (noun hasnt container or supporter) L__M(##Insert, 2, noun);
        else if (noun has container && noun hasnt open) L__M(##Take, 9, noun);
        else if (children(noun)==0) L__M(##Search, 6, noun);
        else parser_results-->ACTION_PRES = 0;
        }
    if (parser_results-->ACTION_PRES ~= ##Remove) {
        if (multi_wanted==100)  L__M(##Miscellany, 43);
        else                    L__M(##Miscellany, 44);
    }
}
if (etype == ASKSCOPE_PE) {
    scope_stage = 3;
    if (indirect(scope_error) == -1) {
        best_etype = nextbest_etype;
        if (~~((etype ofclass Routine) || (etype ofclass String)))
            EndActivity(PRINTING_A_PARSER_ERROR_ACT);
        jump GiveError;
    }
}
if (etype == NOTINCONTEXT_PE) L__M(##Miscellany, 73);

.SkipParserError;
if ((etype ofclass Routine) || (etype ofclass String)) jump ReType;
say__p = 1;
EndActivity(PRINTING_A_PARSER_ERROR_ACT);
```

§**20. Parser Letter J.** Retry the whole lot.

```
! And go (almost) right back to square one...

jump ReType;

! ...being careful not to go all the way back, to avoid infinite repetition
! of a deferred command causing an error.
```

§21. Parser Letter K. Last thing: check for THEN and further instructions(s), return.

```
    ! At this point, the return value is all prepared, and we are only looking
    ! to see if there is a "then" followed by subsequent instruction(s).
.LookForMore;
    if (wn > num_words) rtrue;
    i = NextWord();
    if (i == THEN1__WD or THEN2__WD or THEN3__WD or comma_word) {
        if (wn > num_words) {
        held_back_mode = false;
        return;
        }
        i = WordAddress(verb_wordnum);
        j = WordAddress(wn);
        for (: i<j : i++) i->0 = ' ';
        i = NextWord();
        if (i == AGAIN1__WD or AGAIN2__WD or AGAIN3__WD) {
            ! Delete the words "then again" from the again buffer,
            ! in which we have just realised that it must occur:
            ! prevents an infinite loop on "i. again"
            i = WordAddress(wn-2)-buffer;
            if (wn > num_words) j = INPUT_BUFFER_LEN-1;
            else j = WordAddress(wn)-buffer;
            for (: i<j : i++) buffer3->i = ' ';
        }
        VM_Tokenise(buffer,parse);
        held_back_mode = true;
        return;
    }
    best_etype = UPTO_PE;
    jump GiveError;
```

§22. End of Parser Proper.

```
]; ! end of Parser__parse
```

§**23. Parse Token.** The main parsing routine above tried a sequence of "grammar lines" in turn, matching each against the text typed until one fitted. A grammar line is itself a sequence of "grammar tokens". Here we have to parse the tokens.

`ParseToken(type, data)` tries the match text beginning at the current word marker `wn` against a token of the given `type`, with the given `data`. The optional further arguments `token_n` and `token` supply the token number in the current grammar line (because some tokens do depend on what has happened before or is needed later) and the address of the dictionary word which makes up the `token`, in the case where it's a "preposition".

The return values are:
(a) `GPR_REPARSE` for "I have rewritten the command, please re-parse from scratch";
(b) `GPR_PREPOSITION` for "token accepted with no result";
(c) $-256 + x$ for "please parse `ParseToken(ELEMENTARY_TT, x)` instead";
(d) 0 for "token accepted, result is the multiple object list";
(e) 1 for "token accepted, result is the number in `parsed_number`";
(f) an object number for "token accepted with this object as result";
(g) -1 for "token rejected".

Strictly speaking `ParseToken` is a shell routine which saves the current state on the stack, and calling `ParseToken__` to do the actual work.

Once again the routine is traditionally divided into six letters, here named under paragraphs "Parse Token Letter A", and so on.

(A) Analyse the token; handle all tokens not involving object lists and break down others into elementary tokens
(B) Begin parsing an object list
(C) Parse descriptors (articles, pronouns, etc.) in the list
(D) Parse an object name
(E) Parse connectives (AND, BUT, etc.) and go back to (C)
(F) Return the conclusion of parsing an object list

```
[ ParseTokenStopped x y;
    if (wn>WordCount()) return GPR_FAIL;
    return ParseToken(x,y);
];

Global parsetoken_nesting = 0;
[ ParseToken given_ttype given_tdata token_n token  i t rv;
    if (parsetoken_nesting > 0) {
        ! save match globals
        @push match_from; @push token_filter; @push match_length;
        @push number_of_classes; @push oops_from;
        for (i=0: i<number_matched: i++) {
            t = match_list-->i; @push t;
            t = match_classes-->i; @push t;
            t = match_scores-->i; @push t;
        }
        @push number_matched;
    }

    parsetoken_nesting++;
    rv = ParseToken__(given_ttype, given_tdata, token_n, token);
    parsetoken_nesting--;

    if (parsetoken_nesting > 0) {
        ! restore match globals
        @pull number_matched;
        for (i=0: i<number_matched: i++) {
            @pull t; match_scores-->i = t;
            @pull t; match_classes-->i = t;
            @pull t; match_list-->i = t;
        }
        @pull oops_from; @pull number_of_classes;
        @pull match_length; @pull token_filter; @pull match_from;
    }
    return rv;
];
[ ParseToken__ given_ttype given_tdata token_n token
    l o i j k and_parity single_object desc_wn many_flag
    token_allows_multiple prev_indef_wanted;
```

§**24. Parse Token Letter A.** Analyse token; handle all not involving object lists, break down others.

```
    token_filter = 0;
    parser_inflection = name;
    switch (given_ttype) {
    ELEMENTARY_TT:
        switch (given_tdata) {
        SPECIAL_TOKEN:
            l = TryNumber(wn);
            special_word = NextWord();
            #Ifdef DEBUG;
            if (l ~= -1000)
                if (parser_trace >= 3) print "  [Read special as the number ", l, "]^";
            #Endif; ! DEBUG
            if (l == -1000) {
                #Ifdef DEBUG;
                if (parser_trace >= 3) print "  [Read special word at word number ", wn, "]^";
                #Endif; ! DEBUG
                l = special_word;
            }
            parsed_number = l;
            return GPR_NUMBER;
        NUMBER_TOKEN:
            l=TryNumber(wn++);
            if (l == -1000) {
                etype = NUMBER_PE;
                return GPR_FAIL;
            }
            #Ifdef DEBUG;
            if (parser_trace>=3) print "  [Read number as ", l, "]^";
            #Endif; ! DEBUG
            parsed_number = l;
            return GPR_NUMBER;
        CREATURE_TOKEN:
            if (action_to_be == ##Answer or ##Ask or ##AskFor or ##Tell)
                scope_reason = TALKING_REASON;
        TOPIC_TOKEN:
```

```
        consult_from = wn;
        if ((line_ttype-->(token_n+1) ~= PREPOSITION_TT) &&
        (line_token-->(token_n+1) ~= ENDIT_TOKEN))
            RunTimeError(13);
        do o = NextWordStopped();
        until (o == -1 || PrepositionChain(o, token_n+1) ~= -1);
        wn--;
        consult_words = wn-consult_from;
        if (consult_words == 0) return GPR_FAIL;
        if (action_to_be == ##Ask or ##Answer or ##Tell) {
            o = wn; wn = consult_from; parsed_number = NextWord();
            wn = o; return 1;
        }
        if (o==-1 && (line_ttype-->(token_n+1) == PREPOSITION_TT))
            return GPR_FAIL;    ! don't infer if required preposition is absent
        return GPR_PREPOSITION;
    }
PREPOSITION_TT:
    ! Is it an unnecessary alternative preposition, when a previous choice
    ! has already been matched?
    if ((token->0) & $10) return GPR_PREPOSITION;

    ! If we've run out of the player's input, but still have parameters to
    ! specify, we go into "infer" mode, remembering where we are and the
    ! preposition we are inferring...

    if (wn > num_words) {
        if (inferfrom==0 && parameters<params_wanted) {
            inferfrom = pcount; inferword = token;
            pattern-->pcount = REPARSE_CODE + VM_DictionaryAddressToNumber(given_tdata);
        }

        ! If we are not inferring, then the line is wrong...

        if (inferfrom == 0) return -1;

        ! If not, then the line is right but we mark in the preposition...

        pattern-->pcount = REPARSE_CODE + VM_DictionaryAddressToNumber(given_tdata);
        return GPR_PREPOSITION;
    }

    o = NextWord();

    pattern-->pcount = REPARSE_CODE + VM_DictionaryAddressToNumber(o);

    ! Whereas, if the player has typed something here, see if it is the
    ! required preposition... if it's wrong, the line must be wrong,
    ! but if it's right, the token is passed (jump to finish this token).

    if (o == given_tdata) return GPR_PREPOSITION;
    if (PrepositionChain(o, token_n) ~= -1) return GPR_PREPOSITION;
    return -1;

GPR_TT:
    l = indirect(given_tdata);
    #Ifdef DEBUG;
    if (parser_trace >= 3) print "  [Outside parsing routine returned ", l, "]^";
    #Endif; ! DEBUG
    return l;

SCOPE_TT:
    scope_token = given_tdata;
    scope_stage = 1;
    #Ifdef DEBUG;
    if (parser_trace >= 3) print "  [Scope routine called at stage 1]^";
    #Endif; ! DEBUG
    l = indirect(scope_token);
    #Ifdef DEBUG;
    if (parser_trace >= 3) print "  [Scope routine returned multiple-flag of ", l, "]^";
    #Endif; ! DEBUG
    if (l == 1) given_tdata = MULTI_TOKEN; else given_tdata = NOUN_TOKEN;
ATTR_FILTER_TT:
    token_filter = 1 + given_tdata;
    given_tdata = NOUN_TOKEN;
ROUTINE_FILTER_TT:
    token_filter = given_tdata;
    given_tdata = NOUN_TOKEN;
} ! end of switch(given_ttype)
token = given_tdata;
```

§25. Parse Token Letter B. Begin parsing an object list.

```
    ! There are now three possible ways we can be here:
    !     parsing an elementary token other than "special" or "number";
    !     parsing a scope token;
    !     parsing a noun-filter token (either by routine or attribute).
    !
    ! In each case, token holds the type of elementary parse to
    ! perform in matching one or more objects, and
    ! token_filter is 0 (default), an attribute + 1 for an attribute filter
    ! or a routine address for a routine filter.
    token_allows_multiple = false;
    if (token == MULTI_TOKEN or MULTIHELD_TOKEN or MULTIEXCEPT_TOKEN or MULTIINSIDE_TOKEN)
        token_allows_multiple = true;
    many_flag = false; and_parity = true; dont_infer = false;
```

§26. Parse Token Letter C. Parse descriptors (articles, pronouns, etc.) in the list.

```
    ! We expect to find a list of objects next in what the player's typed.
.ObjectList;
    #Ifdef DEBUG;
    if (parser_trace >= 3) print "  [Object list from word ", wn, "]^";
    #Endif; ! DEBUG
    ! Take an advance look at the next word: if it's "it" or "them", and these
    ! are unset, set the appropriate error number and give up on the line
    ! (if not, these are still parsed in the usual way - it is not assumed
    ! that they still refer to something in scope)
    o = NextWord(); wn--;
    pronoun_word = NULL; pronoun_obj = NULL;
    l = PronounValue(o);
    if (l ~= 0) {
        pronoun_word = o; pronoun_obj = l;
        if (l == NULL) {
            ! Don't assume this is a use of an unset pronoun until the
            ! descriptors have been checked, because it might be an
            ! article (or some such) instead
            for (l=1 : l<=LanguageDescriptors-->0 : l=l+4)
                if (o == LanguageDescriptors-->l) jump AssumeDescriptor;
            pronoun__word = pronoun_word; pronoun__obj = pronoun_obj;
            etype = VAGUE_PE;
            if (parser_trace >= 3) print "  [Stop: unset pronoun]^";
            return GPR_FAIL;
        }
    }
.AssumeDescriptor;
    if (o == ME1__WD or ME2__WD or ME3__WD) { pronoun_word = o; pronoun_obj = player; }
    allow_plurals = true; desc_wn = wn;
.TryAgain;
    ! First, we parse any descriptive words (like "the", "five" or "every"):
    l = Descriptors(token_allows_multiple);
    if (l ~= 0) { etype = l; return 0; }
.TryAgain2;
```

§27. Parse Token Letter D. Parse an object name.

```
! This is an actual specified object, and is therefore where a typing error
! is most likely to occur, so we set:

oops_from = wn;

! So, two cases.  Case 1: token not equal to "held" (so, no implicit takes)
! but we may well be dealing with multiple objects

! In either case below we use NounDomain, giving it the token number as
! context, and two places to look: among the actor's possessions, and in the
! present location.  (Note that the order depends on which is likeliest.)

if (token ~= HELD_TOKEN) {
    i = multiple_object-->0;
    #Ifdef DEBUG;
    if (parser_trace >= 3) print "  [Calling NounDomain on location and actor]^";
    #Endif; ! DEBUG
    l = NounDomain(actors_location, actor, token);
    if (l == REPARSE_CODE) return l;                  ! Reparse after Q&A
    if (indef_wanted == INDEF_ALL_WANTED && l == 0 && number_matched == 0)
        l = 1;  ! ReviseMulti if TAKE ALL FROM empty container

    if (token_allows_multiple && ~~multiflag) {
        if (best_etype==MULTI_PE) best_etype=STUCK_PE;
        multiflag = true;
    }
    if (l == 0) {
        if (indef_possambig) {
            ResetDescriptors();
            wn = desc_wn;
            jump TryAgain2;
        }
        if (etype == MULTI_PE or TOOFEW_PE && multiflag) etype = STUCK_PE;
        etype=CantSee();
        jump FailToken;
    } ! Choose best error

    #Ifdef DEBUG;
    if (parser_trace >= 3) {
        if (l > 1) print "  [ND returned ", (the) l, "]^";
        else {
            print "  [ND appended to the multiple object list:^";
            k = multiple_object-->0;
            for (j=i+1 : j<=k : j++)
                print "  Entry ", j, ": ", (The) multiple_object-->j,
                    " (", multiple_object-->j, ")^";
            print "  List now has size ", k, "]^";
        }
    }
    #Endif; ! DEBUG

    if (l == 1) {
        if (~~many_flag) many_flag = true;
        else {                                ! Merge with earlier ones
            k = multiple_object-->0;            ! (with either parity)
            multiple_object-->0 = i;
            for (j=i+1 : j<=k : j++) {
                if (and_parity) MultiAdd(multiple_object-->j);
                else            MultiSub(multiple_object-->j);
            }
            #Ifdef DEBUG;
            if (parser_trace >= 3)
                print "  [Merging ", k-i, " new objects to the ", i, " old ones]^";
            #Endif; ! DEBUG
        }
    }
    else {
        ! A single object was indeed found

        if (match_length == 0 && indef_possambig) {
            ! So the answer had to be inferred from no textual data,
            ! and we know that there was an ambiguity in the descriptor
            ! stage (such as a word which could be a pronoun being
            ! parsed as an article or possessive).  It's worth having
            ! another go.

            ResetDescriptors();
            wn = desc_wn;
            jump TryAgain2;
        }

        if ((token == CREATURE_TOKEN) && (CreatureTest(l) == 0)) {
            etype = ANIMA_PE;
            jump FailToken;
        } !  Animation is required

        if (~~many_flag) single_object = l;
        else {
            if (and_parity) MultiAdd(l); else MultiSub(l);
            #Ifdef DEBUG;
            if (parser_trace >= 3) print "  [Combining ", (the) l, " with list]^";
            #Endif; ! DEBUG
        }
    }
}

else {

! Case 2: token is "held" (which fortunately can't take multiple objects)
! and may generate an implicit take

    l = NounDomain(actor,actors_location,token);       ! Same as above...
    if (l == REPARSE_CODE) return l;
    if (l == 0) {
        if (indef_possambig) {
            ResetDescriptors();
            wn = desc_wn;
            jump TryAgain2;
        }
        etype = CantSee(); jump FailToken;            ! Choose best error
    }

    ! ...until it produces something not held by the actor.  Then an implicit
    ! take must be tried.  If this is already happening anyway, things are too
    ! confused and we have to give up (but saving the oops marker so as to get
    ! it on the right word afterwards).
```

```
        ! The point of this last rule is that a sequence like
        !
        !     > read newspaper
        !     (taking the newspaper first)
        !     The dwarf unexpectedly prevents you from taking the newspaper!
        !
        ! should not be allowed to go into an infinite repeat - read becomes
        ! take then read, but take has no effect, so read becomes take then read...
        ! Anyway for now all we do is record the number of the object to take.
        o = parent(l);
        if (o ~= actor) {
            #Ifdef DEBUG;
            if (parser_trace >= 3) print "  [Allowing object ", (the) l, " for now]^";
            #Endif; ! DEBUG
        }
        single_object = l;
    } ! end of if (token ~= HELD_TOKEN) else
    ! The following moves the word marker to just past the named object...
    wn = oops_from + match_length;
```

§28. Parse Token Letter E. Parse connectives (AND, BUT, etc.) and go back to (C).

```
    ! Object(s) specified now: is that the end of the list, or have we reached
    ! "and", "but" and so on?  If so, create a multiple-object list if we
    ! haven't already (and are allowed to).
.NextInList;
    o = NextWord();
    if (o == AND1__WD or AND2__WD or AND3__WD or BUT1__WD or BUT2__WD or BUT3__WD or comma_word) {
        #Ifdef DEBUG;
        if (parser_trace >= 3) print "  [Read connective '", (address) o, "']^";
        #Endif; ! DEBUG
        if (~~token_allows_multiple) {
            if (multiflag) jump PassToken; ! give UPTO_PE error
            etype=MULTI_PE;
            jump FailToken;
        }
        if (o == BUT1__WD or BUT2__WD or BUT3__WD) and_parity = 1-and_parity;
        if (~~many_flag) {
            multiple_object-->0 = 1;
            multiple_object-->1 = single_object;
            many_flag = true;
            #Ifdef DEBUG;
            if (parser_trace >= 3) print "  [Making new list from ", (the) single_object, "]^";
            #Endif; ! DEBUG
        }
        dont_infer = true; inferfrom=0;           ! Don't print (inferences)
        jump ObjectList;                          ! And back around
    }
    wn--;   ! Word marker back to first not-understood word
```

§29. Parse Token Letter F. Return the conclusion of parsing an object list.

```
    ! Happy or unhappy endings:
.PassToken;
    if (many_flag) {
        single_object = GPR_MULTIPLE;
        multi_context = token;
    }
    else {
        if (indef_mode == 1 && indef_type & PLURAL_BIT ~= 0) {
            if (indef_wanted < INDEF_ALL_WANTED && indef_wanted > 1) {
                multi_had = 1; multi_wanted = indef_wanted;
                etype = TOOFEW_PE;
                jump FailToken;
            }
        }
    }
    return single_object;
.FailToken;
    ! If we were only guessing about it being a plural, try again but only
    ! allowing singulars (so that words like "six" are not swallowed up as
    ! Descriptors)
    if (allow_plurals && indef_guess_p == 1) {
        #Ifdef DEBUG;
        if (parser_trace >= 4) print "   [Retrying singulars after failure ", etype, "]^";
        #Endif;
        prev_indef_wanted = indef_wanted;
        allow_plurals = false;
        wn = desc_wn;
        jump TryAgain;
    }
    if ((indef_wanted > 0 || prev_indef_wanted > 0) && (~~multiflag)) etype = MULTI_PE;
    return GPR_FAIL;
]; ! end of ParseToken__
```

§**30. Descriptors.** In grammatical terms, a descriptor appears at the front of an English noun phrase and clarifies the quantity or specific identity of what is referred to: for instance, *my* mirror, *the* dwarf, *that* woman. (Numbers, as in *four* duets, are also descriptors in linguistics: but the I6 parser doesn't handle them that way.)

Slightly unfortunately, the bitmap constants used for descriptors in the I6 parser have names in the form `*_BIT`, coinciding with the names of style bits in the list-writer: but they never occur in the same context.

The actual words used as descriptors are read from tables in the language definition. `ArticleDescriptors` uses this table to move current word marker past a run of one or more descriptors which refer to the definite or indefinite article.

```
Constant OTHER_BIT  =   1;     !  These will be used in Adjudicate()
Constant MY_BIT     =   2;     !  to disambiguate choices
Constant THAT_BIT   =   4;
Constant PLURAL_BIT =   8;
Constant LIT_BIT    =  16;
Constant UNLIT_BIT  =  32;

[ ResetDescriptors;
    indef_mode = 0; indef_type = 0; indef_wanted = 0; indef_guess_p = 0;
    indef_possambig = false;
    indef_owner = nothing;
    indef_cases = $$111111111111;
    indef_nspec_at = 0;
];

[ ArticleDescriptors  o x flag cto type n;
    if (wn > num_words) return 0;

    for (flag=true : flag :) {
        o = NextWordStopped(); flag = false;

    for (x=1 : x<=LanguageDescriptors-->0 : x=x+4)
            if (o == LanguageDescriptors-->x) {
                type = LanguageDescriptors-->(x+2);
                if (type == DEFART_PK or INDEFART_PK) flag = true;
            }
    }
    wn--;
    return 0;
];
```

§**31. Parsing Descriptors.** The `Descriptors()` routine parses the descriptors at the head of a noun phrase, leaving the current word marker `wn` at the first word of the noun phrase's body. It is allowed to set up for a plural only if `allow_p` is set; it returns a parser error number, or 0 if no error occurred.

```
[ Descriptors  o x flag cto type n;
    ResetDescriptors();
    if (wn > num_words) return 0;
    for (flag=true : flag :) {
        o = NextWordStopped(); flag = false;
    for (x=1 : x<=LanguageDescriptors-->0 : x=x+4)
            if (o == LanguageDescriptors-->x) {
                flag = true;
                type = LanguageDescriptors-->(x+2);
                if (type ~= DEFART_PK) indef_mode = true;
                indef_possambig = true;
                indef_cases = indef_cases & (LanguageDescriptors-->(x+1));
                if (type == POSSESS_PK) {
                    cto = LanguageDescriptors-->(x+3);
                    switch (cto) {
                    0: indef_type = indef_type | MY_BIT;
                    1: indef_type = indef_type | THAT_BIT;
                    default:
                        indef_owner = PronounValue(cto);
                        if (indef_owner == NULL) indef_owner = InformParser;
                    }
                }
                if (type == light)  indef_type = indef_type | LIT_BIT;
                if (type == -light) indef_type = indef_type | UNLIT_BIT;
            }
        if (o == OTHER1__WD or OTHER2__WD or OTHER3__WD) {
            indef_mode = 1; flag = 1;
            indef_type = indef_type | OTHER_BIT;
        }
        if (o == ALL1__WD or ALL2__WD or ALL3__WD or ALL4__WD or ALL5__WD) {
            indef_mode = 1; flag = 1; indef_wanted = INDEF_ALL_WANTED;
            if (take_all_rule == 1) take_all_rule = 2;
            indef_type = indef_type | PLURAL_BIT;
        }
        if (allow_plurals) {
            if (NextWordStopped() ~= -1) { wn--; n = TryNumber(wn-1); } else { n=0; wn--; }
            if (n == 1) { indef_mode = 1; flag = 1; }
            if (n > 1) {
                indef_guess_p = 1;
                indef_mode = 1; flag = 1; indef_wanted = n;
                indef_nspec_at = wn-1;
                indef_type = indef_type | PLURAL_BIT;
            }
        }
        if (flag == 1 && NextWordStopped() ~= OF1__WD or OF2__WD or OF3__WD or OF4__WD)
            wn--;  ! Skip 'of' after these
    }
    wn--;
    return 0;
];
[ SafeSkipDescriptors;
    @push indef_mode; @push indef_type; @push indef_wanted;
    @push indef_guess_p; @push indef_possambig; @push indef_owner;
    @push indef_cases; @push indef_nspec_at;

    Descriptors();

    @pull indef_nspec_at; @pull indef_cases;
    @pull indef_owner; @pull indef_possambig; @pull indef_guess_p;
    @pull indef_wanted; @pull indef_type; @pull indef_mode;
];
```

§**32. Preposition Chain.** A small utility for runs of prepositions.

```
[ PrepositionChain wd index;
    if (line_tdata-->index == wd) return wd;
    if ((line_token-->index)->0 & $20 == 0) return -1;
    do {
        if (line_tdata-->index == wd) return wd;
        index++;
    } until ((line_token-->index == ENDIT_TOKEN) || (((line_token-->index)->0 & $10) == 0));
    return -1;
];
```

§**33. Creature.** Will this object do for an I6 `creature` token? (In I7 terms, this affects the tokens "[someone]", "[somebody]", "[anyone]" and "[anybody]".)

```
[ CreatureTest obj;
    if (obj has animate) rtrue;
    if (obj hasnt talkable) rfalse;
    if (action_to_be == ##Ask or ##Answer or ##Tell or ##AskFor) rtrue;
    rfalse;
];
```

§34. Noun Domain. `NounDomain` does the most substantial part of parsing an object name. It is given two "domains" – usually a location and then the actor who is looking – and a context (i.e. token type), and returns:

(a) 0 if no match at all could be made,
(b) 1 if a multiple object was made,
(c) *k* if object *k* was the one decided upon,
(d) `REPARSE_CODE` if it asked a question of the player and consequently rewrote the player's input, so that the whole parser should start again on the rewritten input.

In case (c), `NounDomain` also sets the variable `length_of_noun` to the number of words in the input text matched to the noun. In case (b), the multiple objects are added to `multiple_object` by hand (not by `MultiAdd`, because we want to allow duplicates).

```
[ NounDomain domain1 domain2 context
    first_word i j k l answer_words marker;
    #Ifdef DEBUG;
    if (parser_trace >= 4) {
        print "   [NounDomain called at word ", wn, "^";
        print "   ";
        if (indef_mode) {
            print "seeking indefinite object: ";
            if (indef_type & OTHER_BIT)  print "other ";
            if (indef_type & MY_BIT)     print "my ";
            if (indef_type & THAT_BIT)   print "that ";
            if (indef_type & PLURAL_BIT) print "plural ";
            if (indef_type & LIT_BIT)    print "lit ";
            if (indef_type & UNLIT_BIT)  print "unlit ";
            if (indef_owner ~= 0) print "owner:", (name) indef_owner;
            new_line;
            print "   number wanted: ";
            if (indef_wanted == INDEF_ALL_WANTED) print "all"; else print indef_wanted;
            new_line;
            print "   most likely GNAs of names: ", indef_cases, "^";
        }
        else print "seeking definite object^";
    }
    #Endif; ! DEBUG

    match_length = 0; number_matched = 0; match_from = wn;

    SearchScope(domain1, domain2, context);

    #Ifdef DEBUG;
    if (parser_trace >= 4) print "   [ND made ", number_matched, " matches]^";
    #Endif; ! DEBUG

    wn = match_from+match_length;

    ! If nothing worked at all, leave with the word marker skipped past the
    ! first unmatched word...

    if (number_matched == 0) { wn++; rfalse; }

    ! Suppose that there really were some words being parsed (i.e., we did
    ! not just infer).  If so, and if there was only one match, it must be
    ! right and we return it...

    if (match_from <= num_words) {
        if (number_matched == 1) {
            i=match_list-->0;
            return i;
        }
        ! ...now suppose that there was more typing to come, i.e. suppose that
        ! the user entered something beyond this noun.  If nothing ought to follow,
        ! then there must be a mistake, (unless what does follow is just a full
        ! stop, and or comma)
        if (wn <= num_words) {
            i = NextWord(); wn--;
            if (i ~=  AND1__WD or AND2__WD or AND3__WD or comma_word
                or THEN1__WD or THEN2__WD or THEN3__WD
                or BUT1__WD or BUT2__WD or BUT3__WD) {
                if (lookahead == ENDIT_TOKEN) rfalse;
            }
        }
    }
    ! Now look for a good choice, if there's more than one choice...
    number_of_classes = 0;
    if (number_matched == 1) i = match_list-->0;
    if (number_matched > 1) {
        i = true;
        if (number_matched > 1)
            for (j=0 : j<number_matched-1 : j++)
                if (Identical(match_list-->j, match_list-->(j+1)) == false)
                    i = false;
        if (i) dont_infer = true;
        i = Adjudicate(context);
        if (i == -1) rfalse;
        if (i == 1) rtrue;       !  Adjudicate has made a multiple
                              !  object, and we pass it on
    }
    ! If i is non-zero here, one of two things is happening: either
    ! (a) an inference has been successfully made that object i is
    !     the intended one from the user's specification, or
    ! (b) the user finished typing some time ago, but we've decided
    !     on i because it's the only possible choice.
    ! In either case we have to keep the pattern up to date,
    ! note that an inference has been made and return.
    ! (Except, we don't note which of a pile of identical objects.)
    if (i ~= 0) {
        if (dont_infer) return i;
        if (inferfrom == 0) inferfrom=pcount;
        pattern-->pcount = i;
        return i;
    }
    ! If we get here, there was no obvious choice of object to make.  If in
    ! fact we've already gone past the end of the player's typing (which
    ! means the match list must contain every object in scope, regardless
    ! of its name), then it's foolish to give an enormous list to choose
    ! from - instead we go and ask a more suitable question...
    if (match_from > num_words) jump Incomplete;
    ! Now we print up the question, using the equivalence classes as worked
    ! out by Adjudicate() so as not to repeat ourselves on plural objects...
    BeginActivity(ASKING_WHICH_DO_YOU_MEAN_ACT);
    if (ForActivity(ASKING_WHICH_DO_YOU_MEAN_ACT)) jump SkipWhichQuestion;
    j = 1; marker = 0;
    for (i=1 : i<=number_of_classes : i++) {
        while (((match_classes-->marker) ~= i) && ((match_classes-->marker) ~= -i))
            marker++;
        if (match_list-->marker hasnt animate) j = 0;
    }
    if (j) L__M(##Miscellany, 45); else L__M(##Miscellany, 46);
    j = number_of_classes; marker = 0;
    for (i=1 : i<=number_of_classes : i++) {
        while (((match_classes-->marker) ~= i) && ((match_classes-->marker) ~= -i)) marker++;
        k = match_list-->marker;
        if (match_classes-->marker > 0) print (the) k; else print (a) k;
        if (i < j-1)  print (string) COMMA__TX;
        if (i == j-1) {
            #Ifdef SERIAL_COMMA;
            if (j ~= 2) print ",";
            #Endif; ! SERIAL_COMMA
            print (string) OR__TX;
        }
    }
    L__M(##Miscellany, 57);
    .SkipWhichQuestion; EndActivity(ASKING_WHICH_DO_YOU_MEAN_ACT);
    ! ...and get an answer:
.WhichOne;
    #Ifdef TARGET_ZCODE;
    for (i=2 : i<INPUT_BUFFER_LEN : i++) buffer2->i = ' ';
    #Endif; ! TARGET_ZCODE
    answer_words=Keyboard(buffer2, parse2);
    ! Conveniently, parse2-->1 is the first word in both ZCODE and GLULX.
    first_word = (parse2-->1);
    ! Take care of "all", because that does something too clever here to do
    ! later on:
    if (first_word == ALL1__WD or ALL2__WD or ALL3__WD or ALL4__WD or ALL5__WD) {
        if (context == MULTI_TOKEN or MULTIHELD_TOKEN or MULTIEXCEPT_TOKEN or MULTIINSIDE_TOKEN) {
            l = multiple_object-->0;
            for (i=0 : i<number_matched && l+i<MATCH_LIST_WORDS : i++) {
                k = match_list-->i;
                multiple_object-->(i+1+l) = k;
            }
            multiple_object-->0 = i+l;
            rtrue;
        }
        L__M(##Miscellany, 47);
        jump WhichOne;
    }
    ! Look for a comma, and interpret this as a fresh conversation command
    ! if so:
```

```
for (i=1 : i<=answer_words : i++)
    if (WordFrom(i, parse2) == comma_word) {
        VM_CopyBuffer(buffer, buffer2);
        jump RECONSTRUCT_INPUT;
    }
! If the first word of the reply can be interpreted as a verb, then
! assume that the player has ignored the question and given a new
! command altogether.
! (This is one time when it's convenient that the directions are
! not themselves verbs - thus, "north" as a reply to "Which, the north
! or south door" is not treated as a fresh command but as an answer.)
#Ifdef LanguageIsVerb;
if (first_word == 0) {
    j = wn; first_word = LanguageIsVerb(buffer2, parse2, 1); wn = j;
}
#Endif; ! LanguageIsVerb
if (first_word ~= 0) {
    j = first_word->#dict_par1;
    if ((0 ~= j&1) && ~~LanguageVerbMayBeName(first_word)) {
        VM_CopyBuffer(buffer, buffer2);
        jump RECONSTRUCT_INPUT;
    }
}
! Now we insert the answer into the original typed command, as
! words additionally describing the same object
! (eg, > take red button
!      Which one, ...
!      > music
! becomes "take music red button".  The parser will thus have three
! words to work from next time, not two.)
#Ifdef TARGET_ZCODE;
k = WordAddress(match_from) - buffer; l=buffer2->1+1;
for (j=buffer + buffer->0 - 1 : j>=buffer+k+l : j--) j->0 = 0->(j-l);
for (i=0 : i<l : i++) buffer->(k+i) = buffer2->(2+i);
buffer->(k+l-1) = ' ';
buffer->1 = buffer->1 + l;
if (buffer->1 >= (buffer->0 - 1)) buffer->1 = buffer->0;
#Ifnot; ! TARGET_GLULX
k = WordAddress(match_from) - buffer;
l = (buffer2-->0) + 1;
for (j=buffer+INPUT_BUFFER_LEN-1 : j>=buffer+k+l : j--) j->0 = j->(-l);
for (i=0 : i<l : i++) buffer->(k+i) = buffer2->(WORDSIZE+i);
buffer->(k+l-1) = ' ';
buffer-->0 = buffer-->0 + l;
if (buffer-->0 > (INPUT_BUFFER_LEN-WORDSIZE)) buffer-->0 = (INPUT_BUFFER_LEN-WORDSIZE);
#Endif; ! TARGET_
! Having reconstructed the input, we warn the parser accordingly
! and get out.
.RECONSTRUCT_INPUT;
num_words = WordCount();
wn = 1;
#Ifdef LanguageToInformese;
```

```
    LanguageToInformese();
    ! Re-tokenise:
    VM_Tokenise(buffer,parse);
    #Endif; ! LanguageToInformese
    num_words = WordCount();
    players_command = 100 + WordCount();
    actors_location = ScopeCeiling(player);
    FollowRulebook(Activity_after_rulebooks-->READING_A_COMMAND_ACT, true);

    return REPARSE_CODE;

    ! Now we come to the question asked when the input has run out
    ! and can't easily be guessed (eg, the player typed "take" and there
    ! were plenty of things which might have been meant).

.Incomplete;

    if (context == CREATURE_TOKEN) L__M(##Miscellany, 48);
    else                           L__M(##Miscellany, 49);

    #Ifdef TARGET_ZCODE;
    for (i=2 : i<INPUT_BUFFER_LEN : i++) buffer2->i=' ';
    #Endif; ! TARGET_ZCODE
    answer_words = Keyboard(buffer2, parse2);

    first_word=(parse2-->1);
    #Ifdef LanguageIsVerb;
    if (first_word==0) {
        j = wn; first_word=LanguageIsVerb(buffer2, parse2, 1); wn = j;
    }
    #Endif; ! LanguageIsVerb

    ! Once again, if the reply looks like a command, give it to the
    ! parser to get on with and forget about the question...

    if (first_word ~= 0) {
        j = first_word->#dict_par1;
        if (0 ~= j&1) {
            VM_CopyBuffer(buffer, buffer2);
            return REPARSE_CODE;
        }
    }

    ! ...but if we have a genuine answer, then:
    !
    ! (1) we must glue in text suitable for anything that's been inferred.

    if (inferfrom ~= 0) {
        for (j=inferfrom : j<pcount : j++) {
            if (pattern-->j == PATTERN_NULL) continue;
            #Ifdef TARGET_ZCODE;
            i = 2+buffer->1; (buffer->1)++; buffer->(i++) = ' ';
            #Ifnot; ! TARGET_GLULX
            i = WORDSIZE + buffer-->0;
            (buffer-->0)++; buffer->(i++) = ' ';
            #Endif; ! TARGET_

            #Ifdef DEBUG;
            if (parser_trace >= 5)
                print "[Gluing in inference with pattern code ", pattern-->j, "]^";
            #Endif; ! DEBUG
            ! Conveniently, parse2-->1 is the first word in both ZCODE and GLULX.
            parse2-->1 = 0;
            ! An inferred object.  Best we can do is glue in a pronoun.
            ! (This is imperfect, but it's very seldom needed anyway.)
            if (pattern-->j >= 2 && pattern-->j < REPARSE_CODE) {
                PronounNotice(pattern-->j);
                for (k=1 : k<=LanguagePronouns-->0 : k=k+3)
                    if (pattern-->j == LanguagePronouns-->(k+2)) {
                        parse2-->1 = LanguagePronouns-->k;
                        #Ifdef DEBUG;
                        if (parser_trace >= 5)
                             print "[Using pronoun '", (address) parse2-->1, "']^";
                        #Endif; ! DEBUG
                        break;
                    }
            }
            else {
                ! An inferred preposition.
                parse2-->1 = VM_NumberToDictionaryAddress(pattern-->j - REPARSE_CODE);
                #Ifdef DEBUG;
                if (parser_trace >= 5)
                    print "[Using preposition '", (address) parse2-->1, "']^";
                #Endif; ! DEBUG
            }
            ! parse2-->1 now holds the dictionary address of the word to glue in.
            if (parse2-->1 ~= 0) {
                k = buffer + i;
                #Ifdef TARGET_ZCODE;
                @output_stream 3 k;
                print (address) parse2-->1;
                @output_stream -3;
                k = k-->0;
                for (l=i : l<i+k : l++) buffer->l = buffer->(l+2);
                i = i + k; buffer->1 = i-2;
                #Ifnot; ! TARGET_GLULX
                k = Glulx_PrintAnyToArray(buffer+i, INPUT_BUFFER_LEN-i, parse2-->1);
                i = i + k; buffer-->0 = i - WORDSIZE;
                #Endif; ! TARGET_
            }
        }
    }
    ! (2) we must glue the newly-typed text onto the end.
    #Ifdef TARGET_ZCODE;
    i = 2+buffer->1; (buffer->1)++; buffer->(i++) = ' ';
    for (j=0 : j<buffer2->1 : i++,j++) {
        buffer->i = buffer2->(j+2);
        (buffer->1)++;
        if (buffer->1 == INPUT_BUFFER_LEN) break;
    }
    #Ifnot; ! TARGET_GLULX
    i = WORDSIZE + buffer-->0;
    (buffer-->0)++; buffer->(i++) = ' ';
```

```
    for (j=0 : j<buffer2-->0 : i++,j++) {
        buffer->i = buffer2->(j+WORDSIZE);
        (buffer-->0)++;
        if (buffer-->0 == INPUT_BUFFER_LEN) break;
    }
    #Endif; ! TARGET_

    ! (3) we fill up the buffer with spaces, which is unnecessary, but may
    !     help incorrectly-written interpreters to cope.

    #Ifdef TARGET_ZCODE;
    for (: i<INPUT_BUFFER_LEN : i++) buffer->i = ' ';
    #Endif; ! TARGET_ZCODE

    return REPARSE_CODE;

]; ! end of NounDomain
```

§35. Adjudicate. The `Adjudicate` routine tries to see if there is an obvious choice, when faced with a list of objects (the `match_list`) each of which matches the player's specification equally well. To do this it makes use of the `context` (the token type being worked on).

It counts up the number of obvious choices for the given context – all to do with where a candidate is, except for 6 (`animate`) which is to do with whether it is animate or not – and then:

(a) if only one obvious choice is found, that is returned;
(b) if we are in indefinite mode (don't care which) one of the obvious choices is returned, or if there is no obvious choice then an unobvious one is made;
(c) at this stage, we work out whether the objects are distinguishable from each other or not: if they are all indistinguishable from each other, then choose one, it doesn't matter which;
(d) otherwise, 0 (meaning, unable to decide) is returned (but remember that the equivalence classes we've just worked out will be needed by other routines to clear up this mess, so we can't economise on working them out).

`Adjudicate` returns -1 if an error occurred.

```
[ Adjudicate context i j k good_ones last n ultimate flag offset;
    #Ifdef DEBUG;
    if (parser_trace >= 4) {
        print "   [Adjudicating match list of size ", number_matched,
            " in context ", context, "^";
        print "   ";
        if (indef_mode) {
            print "indefinite type: ";
            if (indef_type & OTHER_BIT)  print "other ";
            if (indef_type & MY_BIT)     print "my ";
            if (indef_type & THAT_BIT)   print "that ";
            if (indef_type & PLURAL_BIT) print "plural ";
            if (indef_type & LIT_BIT)    print "lit ";
            if (indef_type & UNLIT_BIT)  print "unlit ";
            if (indef_owner ~= 0) print "owner:", (name) indef_owner;
            new_line;
            print "   number wanted: ";
            if (indef_wanted == INDEF_ALL_WANTED) print "all"; else print indef_wanted;
            new_line;
            print "   most likely GNAs of names: ", indef_cases, "^";
        }
        else print "definite object^";
    }
    #Endif; ! DEBUG
    j = number_matched-1; good_ones = 0; last = match_list-->0;
    for (i=0 : i<=j : i++) {
        n = match_list-->i;
        match_scores-->i = good_ones;
        ultimate = ScopeCeiling(n);
        if (context==HELD_TOKEN && parent(n)==actor)
        {   good_ones++; last=n; }
        if (context==MULTI_TOKEN && ultimate==ScopeCeiling(actor)
            && n~=actor && n hasnt concealed && n hasnt scenery)
        {   good_ones++; last=n; }
        if (context==MULTIHELD_TOKEN && parent(n)==actor)
        {   good_ones++; last=n; }
        if (context==MULTIEXCEPT_TOKEN or MULTIINSIDE_TOKEN)
        {   if (advance_warning==-1)
            {   if (context==MULTIEXCEPT_TOKEN)
                {   good_ones++; last=n;
                }
                if (context==MULTIINSIDE_TOKEN)
                {   if (parent(n)~=actor) { good_ones++; last=n; }
                }
            }
            else
            {   if (context==MULTIEXCEPT_TOKEN && n~=advance_warning)
                {   good_ones++; last=n; }
                if (context==MULTIINSIDE_TOKEN && n in advance_warning)
                {   good_ones++; last=n; }
            }
        }
        if (context==CREATURE_TOKEN && CreatureTest(n)==1)
        {   good_ones++; last=n; }
        match_scores-->i = 1000*(good_ones - match_scores-->i);
    }
    if (good_ones == 1) return last;
    ! If there is ambiguity about what was typed, but it definitely wasn't
    ! animate as required, then return anything; higher up in the parser
    ! a suitable error will be given.  (This prevents a question being asked.)
    if (context == CREATURE_TOKEN && good_ones == 0) return match_list-->0;
    if (indef_mode == 0) indef_type=0;
    ScoreMatchL(context);
    if (number_matched == 0) return -1;
    if (indef_mode == 0) {
        !  Is there now a single highest-scoring object?
        i = SingleBestGuess();
        if (i >= 0) {
            #Ifdef DEBUG;
            if (parser_trace >= 4) print "   Single best-scoring object returned.]^";
            #Endif; ! DEBUG
            return i;
        }
```

```
}
if (indef_mode == 1 && indef_type & PLURAL_BIT ~= 0) {
    if (context ~= MULTI_TOKEN or MULTIHELD_TOKEN or MULTIEXCEPT_TOKEN
                or MULTIINSIDE_TOKEN) {
        etype = MULTI_PE;
        return -1;
    }
    i = 0; offset = multiple_object-->0;
    for (j=BestGuess(): j~=-1 && i<indef_wanted && i+offset<MATCH_LIST_WORDS-1:
        j=BestGuess()) {
        flag = 0;
        BeginActivity(DECIDING_WHETHER_ALL_INC_ACT, j);
        if ((ForActivity(DECIDING_WHETHER_ALL_INC_ACT, j)) == 0) {
            if (j hasnt concealed && j hasnt worn) flag = 1;
            if (context == MULTIHELD_TOKEN or MULTIEXCEPT_TOKEN && parent(j) ~= actor)
                flag = 0;
            if (action_to_be == ##Take or ##Remove && parent(j) == actor)
                flag = 0;
            k = ChooseObjects(j, flag);
            if (k == 1)
                flag = 1;
            else {
                if (k == 2) flag = 0;
            }
        } else {
            flag = 0; if (RulebookSucceeded()) flag = 1;
        }
        EndActivity(DECIDING_WHETHER_ALL_INC_ACT, j);
        if (flag == 1) {
            i++; multiple_object-->(i+offset) = j;
            #Ifdef DEBUG;
            if (parser_trace >= 4) print "   Accepting it^";
            #Endif; ! DEBUG
        }
        else {
            i = i;
            #Ifdef DEBUG;
            if (parser_trace >= 4) print "   Rejecting it^";
            #Endif; ! DEBUG
        }
    }
    if (i < indef_wanted && indef_wanted < INDEF_ALL_WANTED) {
        etype = TOOFEW_PE; multi_wanted = indef_wanted;
        multi_had=i;
        return -1;
    }
    multiple_object-->0 = i+offset;
    multi_context = context;
    #Ifdef DEBUG;
    if (parser_trace >= 4)
        print "   Made multiple object of size ", i, "]^";
    #Endif; ! DEBUG
    return 1;
}
for (i=0 : i<number_matched : i++) match_classes-->i = 0;
n = 1;
for (i=0 : i<number_matched : i++)
    if (match_classes-->i == 0) {
        match_classes-->i = n++; flag = 0;
        for (j=i+1 : j<number_matched : j++)
            if (match_classes-->j == 0 && Identical(match_list-->i, match_list-->j) == 1) {
                flag=1;
                match_classes-->j = match_classes-->i;
            }
        if (flag == 1) match_classes-->i = 1-n;
    }
n--; number_of_classes = n;
#Ifdef DEBUG;
if (parser_trace >= 4) {
    print "   Grouped into ", n, " possibilities by name:^";
    for (i=0 : i<number_matched : i++)
        if (match_classes-->i > 0)
            print "   ", (The) match_list-->i, " (", match_list-->i, ")  ---  group ",
            match_classes-->i, "^";
}
#Endif; ! DEBUG
if (indef_mode == 0) {
    if (n > 1) {
        k = -1;
        for (i=0 : i<number_matched : i++) {
            if (match_scores-->i > k) {
                k = match_scores-->i;
                j = match_classes-->i; j = j*j;
                flag = 0;
            }
            else
                if (match_scores-->i == k) {
                    if ((match_classes-->i) * (match_classes-->i) ~= j)
                        flag = 1;
                }
        }
    if (flag) {
        #Ifdef DEBUG;
        if (parser_trace >= 4) print "   Unable to choose best group, so ask player.]^";
        #Endif; ! DEBUG
        return 0;
    }
    #Ifdef DEBUG;
    if (parser_trace >= 4) print "   Best choices are all from the same group.^";
    #Endif; ! DEBUG
    }
}
!  When the player is really vague, or there's a single collection of
!  indistinguishable objects to choose from, choose the one the player
```

```
    !  most recently acquired, or if the player has none of them, then
    !  the one most recently put where it is.
    if (n == 1) dont_infer = true;
    return BestGuess();
]; ! Adjudicate
```

§36. ReviseMulti. `ReviseMulti` revises the multiple object which already exists, in the light of information which has come along since then (i.e., the second parameter). It returns a parser error number, or else 0 if all is well. This only ever throws things out, never adds new ones.

```
[ ReviseMulti second_p  i low;
    #Ifdef DEBUG;
    if (parser_trace >= 4)
        print "   Revising multiple object list of size ", multiple_object-->0,
        " with 2nd ", (name) second_p, "^";
    #Endif; ! DEBUG
    if (multi_context == MULTIEXCEPT_TOKEN or MULTIINSIDE_TOKEN) {
        for (i=1,low=0 : i<=multiple_object-->0 : i++) {
            if ( (multi_context==MULTIEXCEPT_TOKEN && multiple_object-->i ~= second_p) ||
                (multi_context==MULTIINSIDE_TOKEN && multiple_object-->i in second_p)) {
                low++;
                multiple_object-->low = multiple_object-->i;
            }
        }
        multiple_object-->0 = low;
    }
    if (multi_context == MULTI_TOKEN && action_to_be == ##Take) {
        #Ifdef DEBUG;
        if (parser_trace >= 4) print "   Token 2 plural case: number with actor ", low, "^";
        #Endif; ! DEBUG
        if (take_all_rule == 2) {
            for (i=1,low=0 : i<=multiple_object-->0 : i++) {
                if (ScopeCeiling(multiple_object-->i) == ScopeCeiling(actor)) {
                    low++;
                    multiple_object-->low = multiple_object-->i;
                }
            }
            multiple_object-->0 = low;
        }
    }
    i = multiple_object-->0;
    #Ifdef DEBUG;
    if (parser_trace >= 4) print "   Done: new size ", i, "^";
    #Endif; ! DEBUG
    if (i == 0) return NOTHING_PE;
    return 0;
];
```

§**37. Match List.** The match list is an array, `match_list-->`, which holds the current best guesses at what object(s) a portion of the command refers to. The global `number_matched` is set to the current length of the `match_list`.

When the parser sees a possible match of object `obj` at quality level `q`, it calls `MakeMatch(obj, q)`. If this is the best quality match so far, then we wipe out all the previous matches and start a new list with this one. If it's only as good as the best so far, we add it to the list (provided we haven't run out of space, and provided it isn't in the list already). If it's worse, we ignore it altogether.

I6 tokens in the form `noun=Filter` or `Attribute` are "noun filter tokens", and mean that the match list should be filtered to accept only nouns which are acceptable to the given routine, or have the given attribute. Such a token is in force if `token_filter` is used. (I7 makes no use of this in the attribute case, which is deprecated nowadays.)

Quality is essentially the number of words in the command referring to the object: the idea is that "red panic button" is better than "red button" or "panic".

```
[ MakeMatch obj quality i;
    #Ifdef DEBUG;
    if (parser_trace >= 6) print "    Match with quality ",quality,"^";
    #Endif; ! DEBUG
    if (token_filter ~= 0 && ConsultNounFilterToken(obj) == 0) {
        #Ifdef DEBUG;
        if (parser_trace >= 6) print "    Match filtered out: token filter ", token_filter, "^";
        #Endif; ! DEBUG
        rtrue;
    }
    if (quality < match_length) rtrue;
    if (quality > match_length) { match_length = quality; number_matched = 0; }
    else {
        if (number_matched >= MATCH_LIST_WORDS) rtrue;
        for (i=0 : i<number_matched : i++)
            if (match_list-->i == obj) rtrue;
    }
    match_list-->number_matched++ = obj;
    #Ifdef DEBUG;
    if (parser_trace >= 6) print "    Match added to list^";
    #Endif; ! DEBUG
];
[ ConsultNounFilterToken obj;
    if (token_filter ofclass Routine) {
        noun = obj;
        return indirect(token_filter);
    }
    if (obj has (token_filter-1)) rtrue;
    rfalse;
];
```

§38. ScoreMatchL. `ScoreMatchL` scores the match list for quality in terms of what the player has vaguely asked for. Points are awarded for conforming with requirements like "my", and so on. Remove from the match list any entries which fail the basic requirements of the descriptors. (The scoring system used to evaluate the possibilities is discussed in detail in the DM4.)

```
Constant SCORE__CHOOSEOBJ = 1000;
Constant SCORE__IFGOOD = 500;
Constant SCORE__UNCONCEALED = 100;
Constant SCORE__BESTLOC = 60;
Constant SCORE__NEXTBESTLOC = 40;
Constant SCORE__NOTCOMPASS = 20;
Constant SCORE__NOTSCENERY = 10;
Constant SCORE__NOTACTOR = 5;
Constant SCORE__GNA = 1;
Constant SCORE__DIVISOR = 20;

Constant PREFER_HELD;
[ ScoreMatchL context its_owner its_score obj i j threshold met a_s l_s;
!   if (indef_type & OTHER_BIT ~= 0) threshold++;
    if (indef_type & MY_BIT ~= 0)    threshold++;
    if (indef_type & THAT_BIT ~= 0)  threshold++;
    if (indef_type & LIT_BIT ~= 0)   threshold++;
    if (indef_type & UNLIT_BIT ~= 0) threshold++;
    if (indef_owner ~= nothing)      threshold++;

    #Ifdef DEBUG;
    if (parser_trace >= 4) print "   Scoring match list: indef mode ", indef_mode, " type ",
    indef_type, ", satisfying ", threshold, " requirements:^";
    #Endif; ! DEBUG

    #ifdef PREFER_HELD;
    a_s = SCORE__BESTLOC; l_s = SCORE__NEXTBESTLOC;
    if (action_to_be == ##Take or ##Remove) {
        a_s = SCORE__NEXTBESTLOC; l_s = SCORE__BESTLOC;
    }
    context = context;  ! silence warning
    #ifnot;
    a_s = SCORE__NEXTBESTLOC; l_s = SCORE__BESTLOC;
    if (context == HELD_TOKEN or MULTIHELD_TOKEN or MULTIEXCEPT_TOKEN) {
        a_s = SCORE__BESTLOC; l_s = SCORE__NEXTBESTLOC;
    }
    #endif; ! PREFER_HELD

    for (i=0 : i<number_matched : i++) {
        obj = match_list-->i; its_owner = parent(obj); its_score=0; met=0;

        !      if (indef_type & OTHER_BIT ~= 0
        !          &&  obj ~= itobj or himobj or herobj) met++;
        if (indef_type & MY_BIT ~= 0 && its_owner == actor) met++;
        if (indef_type & THAT_BIT ~= 0 && its_owner == actors_location) met++;
        if (indef_type & LIT_BIT ~= 0 && obj has light) met++;
        if (indef_type & UNLIT_BIT ~= 0 && obj hasnt light) met++;
        if (indef_owner ~= 0 && its_owner == indef_owner) met++;

        if (met < threshold) {
            #Ifdef DEBUG;
            if (parser_trace >= 4)
                print "   ", (The) match_list-->i, " (", match_list-->i, ") in ",
                (the) its_owner, " is rejected (doesn't match descriptors)^";
            #Endif; ! DEBUG
            match_list-->i = -1;
        }
        else {
            its_score = 0;
            if (obj hasnt concealed) its_score = SCORE__UNCONCEALED;
            if (its_owner == actor) its_score = its_score + a_s;
            else
                if (its_owner == actors_location) its_score = its_score + l_s;
                else
                    if (its_owner ~= compass) its_score = its_score + SCORE__NOTCOMPASS;
            its_score = its_score + SCORE__CHOOSEOBJ * ChooseObjects(obj, 2);
            if (obj hasnt scenery) its_score = its_score + SCORE__NOTSCENERY;
            if (obj ~= actor) its_score = its_score + SCORE__NOTACTOR;
            !   A small bonus for having the correct GNA,
            !   for sorting out ambiguous articles and the like.
            if (indef_cases & (PowersOfTwo_TB-->(GetGNAOfObject(obj))))
                its_score = its_score + SCORE__GNA;
            match_scores-->i = match_scores-->i + its_score;
            #Ifdef DEBUG;
            if (parser_trace >= 4) print "     ", (The) match_list-->i, " (", match_list-->i,
            ") in ", (the) its_owner, " : ", match_scores-->i, " points^";
            #Endif; ! DEBUG
        }
    }
    for (i=0 : i<number_matched : i++) {
        while (match_list-->i == -1) {
            if (i == number_matched-1) { number_matched--; break; }
            for (j=i : j<number_matched-1 : j++) {
                match_list-->j = match_list-->(j+1);
                match_scores-->j = match_scores-->(j+1);
            }
            number_matched--;
        }
    }
];
```

§**39. BestGuess.** `BestGuess` makes the best guess it can out of the match list, assuming that everything in the match list is textually as good as everything else; however it ignores items marked as −1, and so marks anything it chooses. It returns −1 if there are no possible choices.

```
[ BestGuess  earliest its_score best i;
    earliest = 0; best = -1;
    for (i=0 : i<number_matched : i++) {
        if (match_list-->i >= 0) {
            its_score = match_scores-->i;
            if (its_score > best) { best = its_score; earliest = i; }
        }
    }
    #Ifdef DEBUG;
    if (parser_trace >= 4)
    if (best < 0) print "   Best guess ran out of choices^";
    else print "   Best guess ", (the) match_list-->earliest,
        " (", match_list-->earliest, ")^";
    #Endif; ! DEBUG
    if (best < 0) return -1;
    i = match_list-->earliest;
    match_list-->earliest = -1;
    return i;
];
```

§**40. SingleBestGuess.** `SingleBestGuess` returns the highest-scoring object in the match list if it is the clear winner, or returns −1 if there is no clear winner.

```
[ SingleBestGuess  earliest its_score best i;
    earliest = -1; best = -1000;
    for (i=0 : i<number_matched : i++) {
        its_score = match_scores-->i;
        if (its_score == best) earliest = -1;
        if (its_score > best) { best = its_score; earliest = match_list-->i; }
    }
    return earliest;
];
```

§**41. Identical.** `Identical` decides whether or not two objects can be distinguished from each other by anything the player can type. If not, it returns `true`. (This routine is critical to the handling of plurals, and the list-writer requires it to be an equivalence relation between objects: but it is, because it is equivalent to $O_1 \sim O_2$ if and only if $f(O_1) = f(O_2)$ for some function f.)

```
[ Identical o1 o2 p1 p2 n1 n2 i j flag;
    if (o1 == o2) rtrue;  ! This should never happen, but to be on the safe side
    if (o1 == 0 || o2 == 0) rfalse;  ! Similarly
    if (o1 ofclass K3_direction || o2 ofclass K3_direction) rfalse; ! Saves time

    !  What complicates things is that o1 or o2 might have a parsing routine,
    !  so the parser can't know from here whether they are or aren't the same.
    !  If they have different parsing routines, we simply assume they're
    !  different.  If they have the same routine (which they probably got from
    !  a class definition) then the decision process is as follows:
    !
    !     the routine is called (with self being o1, not that it matters)
    !       with noun and second being set to o1 and o2, and action being set
    !       to the fake action TheSame.  If it returns -1, they are found
    !       identical; if -2, different; and if >=0, then the usual method
    !       is used instead.

    if (o1.parse_name ~= 0 || o2.parse_name ~= 0) {
    if (o1.parse_name ~= o2.parse_name) rfalse;
    parser_action = ##TheSame; parser_one = o1; parser_two = o2;
    j = wn; i = RunRoutines(o1,parse_name); wn = j;
    if (i == -1) rtrue;
    if (i == -2) rfalse;
    }

    !  This is the default algorithm: do they have the same words in their
    !  "name" (i.e. property no. 1) properties.  (Note that the following allows
    !  for repeated words and words in different orders.)

    p1 = o1.&1; n1 = (o1.#1)/WORDSIZE;
    p2 = o2.&1; n2 = (o2.#1)/WORDSIZE;

    !  for (i=0 : i<n1 : i++) { print (address) p1-->i, " "; } new_line;
    !  for (i=0 : i<n2 : i++) { print (address) p2-->i, " "; } new_line;

    for (i=0 : i<n1 : i++) {
        flag = 0;
        for (j=0 : j<n2 : j++)
            if (p1-->i == p2-->j) flag = 1;
        if (flag == 0) rfalse;
    }

    for (j=0 : j<n2 : j++) {
        flag = 0;
        for (i=0 : i<n1 : i++)
            if (p1-->i == p2-->j) flag = 1;
        if (flag == 0) rfalse;
    }

    !  print "Which are identical!^";
    rtrue;
];
```

§42. Print Command. `PrintCommand` reconstructs the command as it presently reads, from the pattern which has been built up.

If `from` is 0, it starts with the verb: then it goes through the pattern.

The other parameter is `emptyf` – a flag: if 0, it goes up to `pcount`: if 1, it goes up to `pcount`-1.

Note that verbs and prepositions are printed out of the dictionary: and that since the dictionary may only preserve the first six characters of a word (in a V3 game), we have to hand-code the longer words needed. At present, I7 doesn't do this, but it probably should.

(Recall that pattern entries are 0 for "multiple object", 1 for "special word", 2 to `REPARSE_CODE-1` are object numbers and `REPARSE_CODE+n` means the preposition `n`.)

```
[ PrintInferredCommand from singleton_noun;
    singleton_noun = FALSE;
    if ((from ~= 0) && (from == pcount-1) &&
        (pattern-->from > 1) && (pattern-->from < REPARSE_CODE))
            singleton_noun = TRUE;
    if (singleton_noun) {
        BeginActivity(CLARIFYING_PARSERS_CHOICE_ACT, pattern-->from);
        if (ForActivity(CLARIFYING_PARSERS_CHOICE_ACT, pattern-->from) == 0) {
            print "("; PrintCommand(from); print ")^";
        }
        EndActivity(CLARIFYING_PARSERS_CHOICE_ACT, pattern-->from);
    } else {
        print "("; PrintCommand(from); print ")^";
    }
];
[ PrintCommand from i k spacing_flag;
    if (from == 0) {
        i = verb_word;
        if (LanguageVerb(i) == 0)
            if (PrintVerb(i) == 0) print (address) i;
        from++; spacing_flag = true;
    }
    for (k=from : k<pcount : k++) {
        i = pattern-->k;
        if (i == PATTERN_NULL) continue;
        if (spacing_flag) print (char) ' ';
        if (i == 0) { print (string) THOSET__TX; jump TokenPrinted; }
        if (i == 1) { print (string) THAT__TX;   jump TokenPrinted; }
        if (i >= REPARSE_CODE)
            print (address) VM_NumberToDictionaryAddress(i-REPARSE_CODE);
        else
            if (i ofclass K3_direction)
                print (LanguageDirection) i; ! the direction name as adverb
            else
                print (the) i;
    .TokenPrinted;
        spacing_flag = true;
    }
];
```

§43. CantSee. The `CantSee` routine returns a good error number for the situation where the last word looked at didn't seem to refer to any object in context.

The idea is that: if the actor is in a location (but not inside something like, for instance, a tank which is in that location) then an attempt to refer to one of the words listed as meaningful-but-irrelevant there will cause "you don't need to refer to that in this game" rather than "no such thing" or "what's 'it'?".

(The advantage of not having looked at "irrelevant" local nouns until now is that it stops them from clogging up the ambiguity-resolving process. Thus game objects always triumph over scenery.)

```
[ CantSee  i w e;
    saved_oops=oops_from;
    if (scope_token ~= 0) {
        scope_error = scope_token; return ASKSCOPE_PE;
    }
    wn--; w = NextWord();
    e = CANTSEE_PE;
    if (w == pronoun_word) {
        w = NextWordStopped(); wn--;
        if ((w == -1) || (line_token-->(pcount) ~= ENDIT_TOKEN)) {
            AnalyseToken(line_token-->(pcount-1));
            !DebugToken(pcount-1); print " ", found_ttype, "^";
            if (found_ttype == ROUTINE_FILTER_TT or ATTR_FILTER_TT)
                e = NOTINCONTEXT_PE;
            else {
                pronoun__word = pronoun_word; pronoun__obj = pronoun_obj;
                e = ITGONE_PE;
            }
        }
    }
    if (etype > e) return etype;
    return e;
];
```

§44. Multiple Object List. The `MultiAdd` routine adds object `o` to the multiple-object-list. This is only allowed to hold `MATCH_LIST_WORDS` minus one objects at most, at which point it ignores any new entries (and sets a global flag so that a warning may later be printed if need be).

The `MultiSub` routine deletes object `o` from the multiple-object-list. It returns 0 if the object was there in the first place, and 9 (because this is the appropriate error number in `Parser()`) if it wasn't.

The `MultiFilter` routine goes through the multiple-object-list and throws out anything without the given attribute `attr` set.

```
[ MultiAdd o i j;
    i = multiple_object-->0;
    if (i == MATCH_LIST_WORDS-1) { toomany_flag = 1; rtrue; }
    for (j=1 : j<=i : j++)
        if (o == multiple_object-->j) rtrue;
    i++;
    multiple_object-->i = o;
    multiple_object-->0 = i;
];
[ MultiSub o i j k;
    i = multiple_object-->0;
    for (j=1 : j<=i : j++)
        if (o == multiple_object-->j) {
            for (k=j : k<=i : k++) multiple_object-->k = multiple_object-->(k+1);
            multiple_object-->0 = --i;
            return 0;
        }
    return VAGUE_PE;
];

[ MultiFilter attr  i j o;
    .MFiltl;
    i = multiple_object-->0;
    for (j=1 : j<=i : j++) {
        o = multiple_object-->j;
        if (o hasnt attr) { MultiSub(o); jump Mfiltl; }
    }
];
```

§**45. Scope.** The scope of an actor is the set of objects which he can refer to in typed commands, which is normally the same as the set of visible objects; but this can be modified. This is how I7 handles tokens like "[any room]".

Scope determination is done by calling `SearchScope` to iterate through the objects in scope, and "visit" each one: which means, carry out some task for each as we get there. The task depends on the current value of `scope_reason`, which is `PARSING_REASON` when the parser is matching command text against object names.

The scope machinery is built on a number of levels, each making use only of lower levels:

(0) Either `NounDomain`, `TestScope` or `LoopOverScope` makes one or more calls to `SearchScope` (on level 1). The point of making multiple calls is to influence the order in which items in scope are visited, which improves the quality of "take all"-style multiple object lists, for instance.

(1) `SearchScope` searches for the objects in scope which are within first one domain, and then another: for instance, first within the current room but not within the current actor, and then within the current actor. It can be called either from level 0, or externally from the choose-objects machinery, but is not recursive. It works within the context of a given token in the parser (when called for `PARSING_REASON`) and in particular the `multiinside` token, and also handles testing commands, scope tokens, scope in darkness, and intervention by the I7 "deciding the scope of" activity. Most of its actual searches are delegated to `ScopeWithin` (level 2), but it also uses `DoScopeActionAndRecurse` (level 3) and `DoScopeAction` (level 4) as necessary.

(2) `ScopeWithin` iterates through the objects in scope which are within one supplied domain, but not within another. It can be called either from level 1, or independently from rules in the "deciding the scope of" activity via the I7 "place the contents of X in scope" phrase. It calls `DoScopeActionAndRecurse` (level 3) on any unconcealed objects it finds.

(3) `DoScopeActionAndRecurse` visits a given object by calling down to `DoScopeAction` (level 4), and recurses to all unconcealed object-tree contents and component parts of the object. The I7 phrase "place X in scope" uses this routine.

(4) `DoScopeAction` simply visits a single object, taking whatever action is needed there – which will depend on the `scope_reason`. The only use made by the parser of `TryGivenObject`, which tries to match command text against the name of a given object, is from here. The I7 phrase "place X in scope, but not its contents" uses this routine.

Two routines are provided for code external to the parser to modify the scope. They should be called only during scope deliberations – i.e., in `scope=...` tokens or in rules for the "deciding the scope of" activity. (At present, `AddToScope` is not used in I7 at all.) Note that this I7 form of `PlaceInScope` has a slightly different specification to its I6 library counterpart of the same name: it can place a room in scope. (In I6, room names were not normally parsed.)

```
[ PlaceInScope O opts; ! If opts is set, do not place contents in scope
    wn = match_from;
    if (opts == false) DoScopeActionAndRecurse(O);
    else DoScopeAction(O);
    return;
];
[ AddToScope obj;
    if (ats_flag >= 2) DoScopeActionAndRecurse(obj, 0, ats_flag-2);
    if (ats_flag == 1) { if (HasLightSource(obj)==1) ats_hls = 1; }
];
```

§**46. Scope Level 0.** The two ways of starting up the scope machinery other than via the parser code above.

```
[ TestScope obj act a al sr x y;
    x = parser_one; y = parser_two;
    parser_one = obj; parser_two = 0; a = actor; al = actors_location;
    sr = scope_reason; scope_reason = TESTSCOPE_REASON;
    if (act == 0) actor = player; else actor = act;
    actors_location = ScopeCeiling(actor);
    SearchScope(actors_location, actor, 0); scope_reason = sr; actor = a;
    actors_location = al; parser_one = x; x = parser_two; parser_two = y;
    return x;
];
[ LoopOverScope routine act x y a al;
    x = parser_one; y = scope_reason; a = actor; al = actors_location;
    parser_one = routine;
    if (act == 0) actor = player; else actor = act;
    actors_location = ScopeCeiling(actor);
    scope_reason = LOOPOVERSCOPE_REASON;
    SearchScope(actors_location, actor, 0);
    parser_one = x; scope_reason = y; actor = a; actors_location = al;
];
```

§**47. SearchScope.** Level 1. The method is:

(a) If the context is a `scope=...` token, then the search is delegated to "stage 2" of the scope routine. This was the old I6 way to override the searching behaviour: while users probably won't be using it any more, the template does, in order to give testing commands universal scope which is exempt from the activity below; and the NI compiler creates `scope=...` tokens to handle Understand grammar such as "[any room]". So the feature remains very much still in use.

(b) The "deciding the scope of" activity is given the chance to intervene. This is the I7 way to override the searching behaviour, and is the one taken by users.

(c) And otherwise:

   (1) The I6 `multiinside` token, used as the first noun of its grammar line, has as its scope all of the objects which are inside or on top of the *second* noun of the grammar line. This provides a neat scope for the ALL in a command like GET ALL FROM CUPBOARD, where the player clearly does not intend ALL to refer to the cupboard itself, for instance. The difficulty is that we don't yet know what the second object is, if we are parsing left to right. But the parser code above has taken care of all of that, and the `advance_warning` global is set to the object number of the second noun, or to -1 if that is not yet known. Note that we check that the contents are visible before adding them to scope, because otherwise an unscrupulous player could use such a command to detect the contents of an opaque locked box. If this rule applies, we skip (c.2), (c.3) and (c.4).

(2) For all other tokens except `creature`, searching scope for the room holding the current actor always catches the compass directions unless a definite article has already been typed. (Thus OPEN THE EAST would match an object called "east door", but not the compass direction "east".)
(3) The contents of `domain1` which are not contents of `domain2` are placed in scope, and so are any component parts of `domain1`. If `domain1` is a container or supporter, it is placed in scope itself.
(4) The contents and component parts of `domain2` are placed in scope. If `domain2` is a container or supporter, it is placed in scope itself.
(5) In darkness, the actor and his component parts are in scope. If the actor is inside or on top of something, then that thing is also in scope. (This avoids a situation where the player gets into an opaque box, then pulls it closed from the inside, plunging himself into darkness, then types OPEN BOX only to be told that he can't see any such thing.)

```
[ SearchScope domain1 domain2 context i;
    if (domain1 == 0) return;
    ! (a)
    if (scope_token) {
        scope_stage = 2;
        #Ifdef DEBUG;
        if (parser_trace >= 3) print "  [Scope routine called at stage 2]^";
        #Endif;
        if (indirect(scope_token) ~= 0) rtrue;
    }
    ! (b)
    BeginActivity(DECIDING_SCOPE_ACT, actor);
    if (ForActivity(DECIDING_SCOPE_ACT, actor) == false) {
        ! (c.1)
        if ((scope_reason == PARSING_REASON) && (context == MULTIINSIDE_TOKEN) &&
            (advance_warning ~= -1)) {
            if (IsSeeThrough(advance_warning) == 1)
                ScopeWithin(advance_warning, 0, context);
        } else {
            ! (c.2)
            if ((scope_reason == PARSING_REASON) && (context ~= CREATURE_TOKEN) &&
                (indef_mode == 0) && (domain1 == actors_location))
                    ScopeWithin(compass);
            ! (c.3)
            if (domain1 has supporter or container) DoScopeAction(domain1);
            ScopeWithin(domain1, domain2, context);
            ! (c.4)
            if (domain2) {
                if (domain2 has supporter or container) DoScopeAction(domain2);
                ScopeWithin(domain2, 0, context);
            }
        }
        ! (c.5)
        if (thedark == domain1 or domain2) {
            DoScopeActionAndRecurse(actor, actor, context);
            if (parent(actor) has supporter or container)
                DoScopeActionAndRecurse(parent(actor), parent(actor), context);
        }
    }
    EndActivity(DECIDING_SCOPE_ACT, actor);
];
```

§48. ScopeWithin. Level 2. `ScopeWithin` puts objects visible from within the `domain` into scope. An item belonging to the `domain` is placed in scope unless it is being concealed by the `domain`: and even then, if the `domain` is the current actor. Suppose Zorro conceals a book beneath his cloak: then the book is not in scope to his lady friend The Black Whip, but it is in scope to Zorro himself. (Thus an actor is not allowed to conceal anything from himself.)

Note that the `domain` object itself, and its component parts if any, are not placed in scope by this routine, though nothing prevents some other code doing so.

```
[ ScopeWithin domain nosearch context obj next_obj;
    if (domain == 0) rtrue;

    ! Look through the objects in the domain, avoiding "objectloop" in case
    ! movements occur.
    obj = child(domain);
    while (obj) {
        next_obj = sibling(obj);
        if ((domain == actor) || (TestConcealment(domain, obj) == false))
            DoScopeActionAndRecurse(obj, nosearch, context);
        obj = next_obj;
    }
];
```

§49. DoScopeActionAndRecurse. Level 3. In all cases, the `domain` itself is visited. There are then three possible forms of recursion:

(a) To unconcealed objects which are inside, on top of, carried or worn by the `domain`: this is called "searching" in traditional I6 language and is suppressed if `domain` is the special value `nosearch`.
(b) To unconcealed component parts of the `domain`.
(c) To any other objects listed in the `add_to_scope` property array, or supplied by the `add_to_scope` property routine, if it has one. (I7 does not usually use `add_to_scope`, but it remains a useful hook in the parser, so it retains its old I6 library interpretation.)

```
[ DoScopeActionAndRecurse domain nosearch context i ad n obj next_obj;
    DoScopeAction(domain);

    ! (a)
    if ((domain ~= nosearch) &&
        ((domain ofclass K1_room or K8_person) || (IsSeeThrough(domain) == 1))) {
        obj = child(domain);
        while (obj) {
            next_obj = sibling(obj);
            if ((domain == actor) || (TestConcealment(domain, obj) == false))
                DoScopeActionAndRecurse(obj, nosearch, context);
            obj = next_obj;
        }
    }

    ! (b)
    if (domain provides component_child) {
        obj = domain.component_child;
        while (obj) {
            next_obj = obj.component_sibling;
            if ((domain == actor) || (TestConcealment(domain, obj) == false))
                DoScopeActionAndRecurse(obj, 0, context);
            obj = next_obj;
        }
    }
    ! (c)
    ad = domain.&add_to_scope;
    if (ad ~= 0) {
        ! Test if the property value is not an object.
        #Ifdef TARGET_ZCODE;
        i = (UnsignedCompare(ad-->0, top_object) > 0);
        #Ifnot; ! TARGET_GLULX
        i = (((ad-->0)->0) ~= $70);
        #Endif; ! TARGET_
        if (i) {
            ats_flag = 2+context;
            RunRoutines(domain, add_to_scope);
            ats_flag = 0;
        }
        else {
            n = domain.#add_to_scope;
            for (i=0 : (WORDSIZE*i)<n : i++)
                if (ad-->i)
                    DoScopeActionAndRecurse(ad-->i, 0, context);
        }
    }
];
```

§**50. DoScopeAction.** Level 4. This is where we take whatever action is to be performed as the "visit" to each scoped object, and it's the bottom at last of the scope mechanism.

```
[ DoScopeAction item;
    #Ifdef DEBUG;
    if (parser_trace >= 6)
        print "[DSA on ", (the) item, " with reason = ", scope_reason,
            " p1 = ", parser_one, " p2 = ", parser_two, "]^";
    #Endif; ! DEBUG
    @push parser_one; @push scope_reason;
    switch(scope_reason) {
        TESTSCOPE_REASON: if (item == parser_one) parser_two = 1;
        LOOPOVERSCOPE_REASON: if (parser_one ofclass Routine) indirect(parser_one, item);
        PARSING_REASON, TALKING_REASON: MatchTextAgainstObject(item);
    }
    @pull scope_reason; @pull parser_one;
];
```

§51. Parsing Object Names. We now reach the final major block of code in the parser: the part which tries to match a given object's name(s) against the text at word position `match_from` in the player's command, and calls `MakeMatch` if it succeeds. There are basically four possibilities: ME, a pronoun such as IT, a name which doesn't begin misleadingly with a number, and a name which does. In the latter two cases, we pass the job down to `TryGivenObject`.

```
[ MatchTextAgainstObject item i;
    if (match_from <= num_words) { ! If there's any text to match, that is
        wn = match_from;
        i = NounWord();
        if ((i == 1) && (player == item)) MakeMatch(item, 1); ! "me"
        if ((i >= 2) && (i < 128) && (LanguagePronouns-->i == item)) MakeMatch(item, 1);
    }
    ! Construing the current word as the start of a noun, can it refer to the
    ! object?
    wn = match_from;
    if (TryGivenObject(item) > 0)
        if (indef_nspec_at > 0 && match_from ~= indef_nspec_at) {
            ! This case arises if the player has typed a number in
            ! which is hypothetically an indefinite descriptor:
            ! e.g. "take two clubs".  We have just checked the object
            ! against the word "clubs", in the hope of eventually finding
            ! two such objects.  But we also backtrack and check it
            ! against the words "two clubs", in case it turns out to
            ! be the 2 of Clubs from a pack of cards, say.  If it does
            ! match against "two clubs", we tear up our original
            ! assumption about the meaning of "two" and lapse back into
            ! definite mode.
            wn = indef_nspec_at;
            if (TryGivenObject(item) > 0) {
                match_from = indef_nspec_at;
                ResetDescriptors();
            }
            wn = match_from;
        }
];
```

§**52. TryGivenObject.** `TryGivenObject` tries to match as many words as possible in what has been typed to the given object, `obj`. If it manages any words matched at all, it calls `MakeMatch` to say so, then returns the number of words (or 1 if it was a match because of inadequate input).

```
[ TryGivenObject obj nomatch threshold k w j;
    #Ifdef DEBUG;
    if (parser_trace >= 5) print "    Trying ", (the) obj, " (", obj, ") at word ", wn, "^";
    #Endif; ! DEBUG
    if (nomatch && obj == 0) return 0;
! if (nomatch) print "*** TryGivenObject *** on ", (the) obj, " at wn = ", wn, "^";
    dict_flags_of_noun = 0;
!  If input has run out then always match, with only quality 0 (this saves
!  time).
    if (wn > num_words) {
        if (nomatch) return 0;
        if (indef_mode ~= 0)
            dict_flags_of_noun = $$01110000;  ! Reject "plural" bit
        MakeMatch(obj,0);
        #Ifdef DEBUG;
        if (parser_trace >= 5) print "    Matched (0)^";
        #Endif; ! DEBUG
        return 1;
    }
!  Ask the object to parse itself if necessary, sitting up and taking notice
!  if it says the plural was used:
    if (obj.parse_name~=0) {
        parser_action = NULL; j=wn;
        k = RunRoutines(obj,parse_name);
        if (k > 0) {
            wn=j+k;
        .MMbyPN;
            if (parser_action == ##PluralFound)
                dict_flags_of_noun = dict_flags_of_noun | 4;
            if (dict_flags_of_noun & 4) {
                if (~~allow_plurals) k = 0;
                else {
                    if (indef_mode == 0) {
                        indef_mode = 1; indef_type = 0; indef_wanted = 0;
                    }
                    indef_type = indef_type | PLURAL_BIT;
                    if (indef_wanted == 0) indef_wanted = INDEF_ALL_WANTED;
                }
            }
            #Ifdef DEBUG;
            if (parser_trace >= 5) print "    Matched (", k, ")^";
            #Endif; ! DEBUG
            if (nomatch == false) MakeMatch(obj,k);
            return k;
        }
        if (k == 0) jump NoWordsMatch;
    }
    ! The default algorithm is simply to count up how many words pass the
    ! Refers test:

    parser_action = NULL;

    w = NounWord();

    if (w == 1 && player == obj) { k=1; jump MMbyPN; }

    if (w >= 2 && w < 128 && (LanguagePronouns-->w == obj)) { k = 1; jump MMbyPN; }

    if (Refers(obj, wn-1) == 0) {
        .NoWordsMatch;
        if (indef_mode ~= 0) { k = 0; parser_action = NULL; jump MMbyPN; }
        rfalse;
    }

    threshold = 1;
    dict_flags_of_noun = (w->#dict_par1) & $$01110100;
    w = NextWord();
    while (Refers(obj, wn-1)) {
        threshold++;
        if (w)
        dict_flags_of_noun = dict_flags_of_noun | ((w->#dict_par1) & $$01110100);
        w = NextWord();
    }

    k = threshold;
    jump MMbyPN;
];
```

§**53. Refers.** `Refers` works out whether the word at number wnum can refer to the object `obj`, returning true or false. The standard method is to see if the word is listed under the `name` property for the object, but this is more complex in languages other than English.

```
[ Refers obj wnum   wd k l m;
    if (obj == 0) rfalse;

    #Ifdef LanguageRefers;
    k = LanguageRefers(obj,wnum); if (k >= 0) return k;
    #Endif; ! LanguageRefers

    k = wn; wn = wnum; wd = NextWordStopped(); wn = k;

    if (parser_inflection >= 256) {
        k = indirect(parser_inflection, obj, wd);
        if (k >= 0) return k;
        m = -k;
    }
    else
        m = parser_inflection;
    k = obj.&m; l = (obj.#m)/WORDSIZE-1;
    for (m=0 : m<=l : m++)
        if (wd == k-->m) rtrue;
    rfalse;
];

[ WordInProperty wd obj prop k l m;
    k = obj.&prop; l = (obj.#prop)/WORDSIZE-1;
    for (m=0 : m<=l : m++)
        if (wd == k-->m) rtrue;
    rfalse;
];
```

§**54. NounWord.** `NounWord` (which takes no arguments) returns:

(a) 0 if the next word is not in the dictionary or is but does not carry the "noun" bit in its dictionary entry,
(b) 1 if it is a word meaning "me",
(c) the index in the pronoun table (plus 2) of the value field of a pronoun, if it is a pronoun,
(d) the address in the dictionary if it is a recognised noun.

```
[ NounWord i j s;
    i = NextWord();
    if (i == 0) rfalse;
    if (i == ME1__WD or ME2__WD or ME3__WD) return 1;
    s = LanguagePronouns-->0;
    for (j=1 : j<=s : j=j+3)
        if (i == LanguagePronouns-->j)
            return j+2;
    if ((i->#dict_par1)&128 == 0) rfalse;
    return i;
];
```

§**55. TryNumber.** `TryNumber` takes word number `wordnum` and tries to parse it as an (unsigned) decimal number or the name of a small number, returning

(a) −1000 if it is not a number
(b) the number, if it has between 1 and 4 digits
(c) 10000 if it has 5 or more digits.

(The danger of allowing 5 digits is that Z-machine integers are only 16 bits long, and anyway this routine isn't meant to be perfect: it only really needs to be good enough to handle numeric descriptors such as those in TAKE 31 COINS or DROP FOUR DAGGERS. In particular, it is not the way I7 "[number]" tokens are parsed.)

```
[ TryNumber wordnum   i j c num len mul tot d digit;
    i = wn; wn = wordnum; j = NextWord(); wn = i;
    j = NumberWord(j); ! Test for verbal forms ONE to TWENTY
    if (j >= 1) return j;
    #Ifdef TARGET_ZCODE;
    i = wordnum*4+1; j = parse->i; num = j+buffer; len = parse->(i-1);
    #Ifnot; ! TARGET_GLULX
    i = wordnum*3; j = parse-->i; num = j+buffer; len = parse-->(i-1);
    #Endif; ! TARGET_
    if (len >= 4) mul=1000;
    if (len == 3) mul=100;
    if (len == 2) mul=10;
    if (len == 1) mul=1;
    tot = 0; c = 0; len = len-1;
    for (c=0 : c<=len : c++) {
        digit=num->c;
        if (digit == '0') { d = 0; jump digok; }
        if (digit == '1') { d = 1; jump digok; }
        if (digit == '2') { d = 2; jump digok; }
        if (digit == '3') { d = 3; jump digok; }
        if (digit == '4') { d = 4; jump digok; }
        if (digit == '5') { d = 5; jump digok; }
        if (digit == '6') { d = 6; jump digok; }
        if (digit == '7') { d = 7; jump digok; }
        if (digit == '8') { d = 8; jump digok; }
        if (digit == '9') { d = 9; jump digok; }
        return -1000;
    .digok;
        tot = tot+mul*d; mul = mul/10;
    }
    if (len > 3) tot=10000;
    return tot;
];
```

§56. Extended TryNumber. The same, but recognising verbal forms up to 30.

```
[ I7_ExtendedTryNumber wordnum i j;
    i = wn; wn = wordnum; j = NextWordStopped(); wn = i;
    switch (j) {
        'twenty-one': return 21;
        'twenty-two': return 22;
        'twenty-three': return 23;
        'twenty-four': return 24;
        'twenty-five': return 25;
        'twenty-six': return 26;
        'twenty-seven': return 27;
        'twenty-eight': return 28;
        'twenty-nine': return 29;
        'thirty': return 30;
        default: return TryNumber(wordnum);
    }
];
```

§57. Gender. `GetGender` returns 0 if the given animate object is female, and 1 if male, and is abstracted as a routine in case something more elaborate is ever needed.

For GNAs – gender/noun/animation combinations – see the *Inform Designer's Manual*, 4th edition.

```
[ GetGender person;
    if (person hasnt female) rtrue;
    rfalse;
];
[ GetGNAOfObject obj case gender;
    if (obj hasnt animate) case = 6;
    if (obj has male) gender = male;
    if (obj has female) gender = female;
    if (obj has neuter) gender = neuter;
    if (gender == 0) {
        if (case == 0) gender = LanguageAnimateGender;
        else gender = LanguageInanimateGender;
    }
    if (gender == female)   case = case + 1;
    if (gender == neuter)   case = case + 2;
    if (obj has pluralname) case = case + 3;
    return case;
];
```

§58. Noticing Plurals.

```
[ DetectPluralWord at n i w swn outcome;
    swn = wn; wn = at;
    for (i=0:i<n:i++) {
        w = NextWordStopped();
        if (w == 0 or THEN1__WD or COMMA_WORD or -1) break;
        if ((w->#dict_par1) & $$00000100) {
            parser_action = ##PluralFound;
            outcome = true;
        }
    }
    wn = swn;
    return outcome;
];
```

§59. Pronoun Handling.

```
[ SetPronoun dword value x;
    for (x=1 : x<=LanguagePronouns-->0 : x=x+3)
        if (LanguagePronouns-->x == dword) {
            LanguagePronouns-->(x+2) = value; return;
        }
    RunTimeError(14);
];
[ PronounValue dword x;
    for (x=1 : x<=LanguagePronouns-->0 : x=x+3)
        if (LanguagePronouns-->x == dword)
            return LanguagePronouns-->(x+2);
    return 0;
];
[ ResetVagueWords obj; PronounNotice(obj); ];
[ PronounNotice obj x bm;
    if (obj == player) return;
    bm = PowersOfTwo_TB-->(GetGNAOfObject(obj));
    for (x=1 : x<=LanguagePronouns-->0 : x=x+3)
        if (bm & (LanguagePronouns-->(x+1)) ~= 0)
            LanguagePronouns-->(x+2) = obj;
];
[ PronounNoticeHeldObjects x;
#IFNDEF MANUAL_PRONOUNS;
    objectloop(x in player) PronounNotice(x);
#ENDIF;
    x = 0; ! To prevent a "not used" error
    rfalse;
];
```

§60. Yes/No Questions.

```
[ YesOrNo i j;
    for (::) {
        #Ifdef TARGET_ZCODE;
        if (location == nothing || parent(player) == nothing) read buffer parse;
        else read buffer parse DrawStatusLine;
        j = parse->1;
        #Ifnot; ! TARGET_GLULX;
        KeyboardPrimitive(buffer, parse);
        j = parse-->0;
        #Endif; ! TARGET_
        if (j) { ! at least one word entered
            i = parse-->1;
            if (i == YES1__WD or YES2__WD or YES3__WD) rtrue;
            if (i == NO1__WD or NO2__WD or NO3__WD) rfalse;
        }
        L__M(##Quit, 1); print "> ";
    }
];
```

§61. Number Words. Not much of a parsing routine: we look through an array of pairs of number words (single words) and their numeric equivalents.

```
[ NumberWord o i n;
    n = LanguageNumbers-->0;
    for (i=1 : i<=n : i=i+2)
        if (o == LanguageNumbers-->i) return LanguageNumbers-->(i+1);
    return 0;
];
```

§62. Choose Objects. This material, the final body of code in the parser, is an I7 addition. The I6 parser leaves it to the user to provide a `ChooseObjects` routine to decide between possibilities when the situation is ambiguous. For I7 use, we provide a `ChooseObjects` which essentially runs the "does the player mean" rulebook to decide, though this is not obvious from the code below because it is hidden in the `CheckDPMR` routine – which is defined in the Standard Rules, not here.

```
!Constant COBJ_DEBUG;

! the highest value returned by CheckDPMR (see the Standard Rules)
Constant HIGHEST_DPMR_SCORE = 4;

Array alt_match_list --> (MATCH_LIST_WORDS+1);

#ifdef TARGET_GLULX;
[ COBJ__Copy words from to  i;
    for (i=0: i<words: i++)
        to-->i = from-->i;
];
#ifnot;
[ COBJ__Copy words from to  bytes;
    bytes = words * 2;
    @copy_table from to bytes;
];
```

```
#endif;
! swap alt_match_list with match_list/number_matched
[ COBJ__SwapMatches i x;
    ! swap the counts
    x = number_matched;
    number_matched = alt_match_list-->0;
    alt_match_list-->0 = x;
    ! swap the values
    if (x < number_matched) x = number_matched;
    for (i=x: i>0: i--) {
        x = match_list-->(i-1);
        match_list-->(i-1) = alt_match_list-->i;
        alt_match_list-->i = x;
    }
];
[ ChooseObjects obj code  l i swn spcount;
    if (code<2) rfalse;
    if (cobj_flag == 1) {
        .CodeOne;
        if (parameters > 0) {
            #ifdef COBJ_DEBUG;
            print "[scoring ", (the) obj, " (second)]^";
            #endif;
            return ScoreDabCombo(parser_results-->INP1_PRES, obj);
        } else {
            #ifdef COBJ_DEBUG;
            print "[scoring ", (the) obj, " (first) in ",
                alt_match_list-->0, " combinations]^";
            #endif;
            l = 0;
            for (i=1: i<=alt_match_list-->0: i++) {
                spcount = ScoreDabCombo(obj, alt_match_list-->i);
                if (spcount == HIGHEST_DPMR_SCORE) {
                    #ifdef COBJ_DEBUG;
                    print "[scored ", spcount, " - best possible]^";
                    #endif;
                    return spcount;
                }
                if (spcount>l) l = spcount;
            }
            return l;
        }
    }
    if (cobj_flag == 2) {
        .CodeTwo;
        #ifdef COBJ_DEBUG;
        print "[scoring ", (the) obj, " (simple); parameters = ", parameters,
            " aw = ", advance_warning, "]^";
        #endif;
        @push action_to_be;
        if (parameters==0) {
            if (advance_warning > 0)
                l = ScoreDabCombo(obj, advance_warning);
            else
                l = ScoreDabCombo(obj, 0);
        } else {
            l = ScoreDabCombo(parser_results-->INP1_PRES, obj);
        }
        @pull action_to_be;
        return l;
    }
    #ifdef COBJ_DEBUG;
    print "[choosing a cobj strategy: ";
    #endif;
    swn = wn;
    spcount = pcount;
    while (line_ttype-->pcount == PREPOSITION_TT) pcount++;
    if (line_ttype-->pcount == ELEMENTARY_TT) {
        while (wn <= num_words) {
            l = NextWordStopped(); wn--;
            if ( (l ~= -1 or 0) && (l->#dict_par1) &8 ) { wn++; continue; } ! if preposition
            if (l == ALL1__WD or ALL2__WD or ALL3__WD or ALL4__WD or ALL5__WD) { wn++; continue; }
            SafeSkipDescriptors();
            ! save the current match state
            @push match_length; @push token_filter; @push match_from;
            alt_match_list-->0 = number_matched;
            COBJ__Copy(number_matched, match_list, alt_match_list+WORDSIZE);
            ! now get all the matches for the second noun
            match_length = 0; number_matched = 0; match_from = wn;
            token_filter = 0;
            SearchScope(actor, actors_location, line_tdata-->pcount);
            #ifdef COBJ_DEBUG;
            print number_matched, " possible second nouns]^";
            #endif;
            wn = swn;
            cobj_flag = 1;
            ! restore match variables
            COBJ__SwapMatches();
            @pull match_from; @pull token_filter; @pull match_length;
            pcount = spcount;
            jump CodeOne;
        }
    }
    pcount = spcount;
    wn = swn;
    #ifdef COBJ_DEBUG;
    print "nothing interesting]^";
    #endif;
    cobj_flag = 2;
    jump CodeTwo;
];
[ ScoreDabCombo a b  result;
    @push action; @push act_requester; @push noun; @push second;
    action = action_to_be;
    act_requester = player;
    if (action_reversed) { noun = b; second = a; }
    else { noun = a; second = b; }
    result = CheckDPMR();
    @pull second; @pull noun; @pull act_requester; @pull action;
    #ifdef COBJ_DEBUG;
    print "[", (the) a, " / ", (the) b, " => ", result, "]^";
    #endif;
    return result;
];
```

§**63. Default Topic.** A default value for the I7 sort-of-kind “topic”, which never matches.

```
[ DefaultTopic; return GPR_FAIL; ];
```

ListWriter Template B/lwt

Purpose

A flexible object-lister taking care of plurals, inventory information, various formats and so on.

§**1. Specification.** The list-writer is called by one of the following function calls:

(1) `WriteListOfMarkedObjects(style)`, where the set of objects listed is understood to be exactly those with the `workflag2` attribute set, and
   (a) the `style` is a sum of `*_BIT` constants as defined in "Definitions.i6t".

(2) `WriteListFrom(obj, style, depth, noactivity, iterator)`, where only the first two parameters are compulsory:
   (a) the set of objects listed begins with `obj`;
   (b) the `style` is a sum of `*_BIT` constants as defined in "Definitions.i6t";
   (c) the `depth` is the recursion depth within the list – ordinarily 0, but by supplying a higher value, we can simulate a sublist of another list;
   (d) `noactivity` is a flag which forces the list-writer to ignore the "listing the contents of" activity (in cases where it would otherwise consult this): by default this is `false`;
   (e) `iterator` is an iterator function which provides the objects in sequence.

`WriteListOfMarkedObjects` is simply a front-end which supplies suitable parameters for `WriteListFrom`.

The iterator function is by default `ObjectTreeIterator`. This defines the sequence of objects as being the children of the parent of `obj`, in object tree sequence (that is: `child(parent(obj))` is first). Moreover, when using `ObjectTreeIterator`, the "listing the contents of" activity is carried out, unless `noactivity` is set.

We also provide the iterator function `MarkedListIterator`, which defines the sequence of objects as being the list in the word array `MarkedObjectArray` with length `MarkedObjectLength`. Here the "listing the contents of" activity is never used, since the objects are not necessarily the contents of any single thing. This of course is the iterator used by `WriteListOfMarkedObjects(style)`: it works by filling this array with all the objects having `workflag2` set.

The full specification for iterator functions is given below.

The list-writer automatically groups adjacent and indistinguishable terms in the sequence into plurals, and carries out the "printing the plural name" activity to handle these. Doing this alone would result in text such as "You can see a cake, three coins, an onion, and two coins here", where the five coins are mentioned in two clauses because they happen not to be adjacent in the list. The list-writer therefore carries out the activity "grouping together" to see if the user would like to tie certain objects together into a single entry: what this does is to set suitable `list_together` properties for the objects. NI has already given plural objects a similar `list_together` property. The net effect is that any entries in the list with a non-zero value of `list_together` must be adjacent to each other.

We could achieve that by sorting the list in order of `list_together` value, but that would result in drastic movements, whereas we want to upset the original ordering as little as possible. So instead we use a process called *coalescing* the list. This is such that for every value $L \neq 0$ of `list_together`, every entry with that value is moved back in the list to follow the first entry with that value. Thus if the original order is $x_1, x_2, ..., x_N$ then x_j still precedes x_k in coalesced order unless there exists $i < j < k$ such that $L(i) = L(k) \neq 0$ and $L(j) \neq L(i)$. This is as stable as it can be while achieving the "interval" property that non-zero L values occur in contiguous runs.

We therefore obtain text such as “You can see a cake, five coins, the tiles W, X, Y and Z from a Scrabble set, and an onion here.” where the coins and the Scrabble tiles have been coalesced together in the list.

It’s important to note that the default `ObjectTreeIterator` has the side-effect of actually reordering the object tree: it rearranges the children being listed so that they appear in the tree in coalesced order. The `MarkedListIterator` used by `WriteListOfMarkedObjects` has the same side-effect if the marked objects all happen to share a common parent. It might seem odd for a list-writer to have side effects at all, but the idea is that occasional coalescing improves the quality of play in many small ways – for instance, the sequence of matches to TAKE ALL is tidier – and that coalescing has a small speed cost, so we want to do it as little as possible. (The latter was more of a consideration for I6: interpreters are faster nowadays.)

This specification is somewhat stronger than that of the I6 library’s traditional list-writer, because
(i) it supports arbitrary lists of objects, not just children of specific parents, while still allowing coalesced and grouped lists,
(ii) it can be used recursively in all cases,
(iii) it uses its own memory, rather than borrowing memory from the parser, so that it can safely be used while the parser is working, and
(iv) it manages this memory more flexibly and without silently failing by buffer overruns on unexpectedly large lists.

The I7 version of `WriteListFrom`, when using `ObjectTreeIterator`, differs from the I6 version in that the object `o` is required to be the `child` of its `parent`, that is, to be the eldest child. (So in effect it’s a function of parents, not children, but we retain the form for familiarity’s sake.) In practice this was invariably the way `WriteListFrom` was used even in I6 days.

Finally, the `ISARE_BIT` is differently interpreted in I7: instead of printing something like “ are ducks and drakes”, as it would have done in I6, the initial space is now suppressed and we instead print “are ducks and drakes”.

§**2. Memory.** The list-writer needs to dynamically allocate temporary array space of a known size, in such a way that the array is effectively on the local stack frame: if only either the Z-machine or Glulx supported a stack in memory, this would be no problem, but they do not and we must therefore use the following.

The size of the stack is such that it can support a list which includes every object and recurses in turn to most other objects: in practice, this never happens. It would be nice to allocate more just in case, but the Z-machine is desperately short of array memory.

```
Constant REQUISITION_STACK_SIZE = 3*{-value:Data::Instances::count(K_object)};
Array requisition_stack --> REQUISITION_STACK_SIZE;
Global requisition_stack_pointer = 0;

[ RequisitionStack len top addr;
    top = requisition_stack_pointer + len;
    if (top > REQUISITION_STACK_SIZE) return false;
    addr = requisition_stack + requisition_stack_pointer*WORDSIZE;
    ! print "Allocating ", addr, " at pointer ", requisition_stack_pointer, "^";
    requisition_stack_pointer = top;
    return addr;
];

[ FreeStack addr;
    if (addr == 0) return;
    requisition_stack_pointer = (addr - requisition_stack)/WORDSIZE;
];
```

§**3. WriteListOfMarkedObjects.** This routine will use the `MarkedListIterator`. That means it has to create an array containing the object numbers of every object with the `workflag2` attribute set, placing the address of this array in `MarkedObjectArray` and the length in `MarkedObjectLength`. Note that we preserve any existing marked list on the stack (using the assembly-language instructions `@push` and `@pull`) for the duration of our use.

While the final order of this list will depend on what it looks like after coalescing, the initial order is also important. If all of the objects have a common parent in the object tree, then we coalesce those objects and place the list in object tree order. But if not, we place the list in object number order (which is essentially the order of traversal of the initial state of the object tree: thus objects in Room A will all appear before objects in Room B if A was created before B in the source text).

```
Global MarkedObjectArray = 0;
Global MarkedObjectLength = 0;

[ WriteListOfMarkedObjects style
    obj common_parent first mixed_parentage length;

    objectloop (obj ofclass Object && obj has workflag2) {
        length++;
        if (first == nothing) { first = obj; common_parent = parent(obj); }
        else { if (parent(obj) ~= common_parent) mixed_parentage = true; }
    }
    if (mixed_parentage) common_parent = nothing;

    if (length == 0) {
        if (style & ISARE_BIT ~= 0) print (string) IS3__TX, " ", (string) NOTHING__TX;
        else if (style & CFIRSTART_BIT ~= 0) print (string) NOTHING2__TX;
        else print (string) NOTHING__TX;
    } else {
        @push MarkedObjectArray; @push MarkedObjectLength;
        MarkedObjectArray = RequisitionStack(length);
        MarkedObjectLength = length;
        if (MarkedObjectArray == 0) return RunTimeProblem(RTP_LISTWRITERMEMORY);

        if (common_parent) {
            ObjectTreeCoalesce(child(common_parent));
            length = 0;
            objectloop (obj in common_parent) ! object tree order
                if (obj has workflag2) MarkedObjectArray-->length++ = obj;
        } else {
            length = 0;
            objectloop (obj ofclass Object) ! object number order
                if (obj has workflag2) MarkedObjectArray-->length++ = obj;
        }

        WriteListFrom(first, style, 0, false, MarkedListIterator);

        FreeStack(MarkedObjectArray);
        @pull MarkedObjectLength; @pull MarkedObjectArray;
    }
    return;
];
```

§4. About Iterator Functions. Suppose `Iter` is an iterator function and that we have a "raw list" $x_1, x_2, ..., x_n$ of objects. Of these, the iterator function will choose a sublist of "qualifying" objects. It is called with arguments

```
    Iter(obj, depth, L, function)
```

where the `obj` is required to be x_j for some j and the function is one of the `*_ITF` constants defined below:

(a) On `START_ITF`, we return x_1, or `nothing` if the list is empty.

(b) On `SEEK_ITF`, we return the smallest $k \geq j$ such that x_k qualifies, or `nothing` if none of $x_j, x_{j+1}, ..., x_n$ qualify.

(c) On `ADVANCE_ITF`, we return the smallest $k > j$ such that x_k qualifies, or `nothing` if none of $x_{j+1}, x_{j+2}..., x_n$ qualify.

(d) On `COALESCE_ITF`, we coalesce the entire list (not merely the qualifying entries) and return the new x_1, or `nothing` if the list is empty.

Thus, given any x_i, we can produce the sublist of qualifying entries by performing `START_ITF` on x_i, then `SEEK_ITF`, to produce q_1; then `ADVANCE_ITF` to produce q_2, and so on until `nothing` is returned, when there are no more qualifying entries. `SEEK_ITF` and `ADVANCE_ITF` always return qualifying objects, or `nothing`; but `START_ITF` and `COALESCE_ITF` may return a non-qualifying object, since there's no reason why x_1 should qualify in any ordering.

The iterator can make its own choice of which entries qualify, and of what the raw list is; `depth` is supplied to help in that decision. But if `L` is non-zero then the iterator is required to disqualify any entry whose `list_together` value is not `L`.

```
Constant SEEK_ITF = 0;
Constant ADVANCE_ITF = 1;
Constant COALESCE_ITF = 2;
Constant START_ITF = 3;

! Constant DBLW; ! Uncomment this to provide debugging information at run-time
```

§5. Marked List Iterator. Here the raw list is provided by the `MarkedObjectArray`, which is convenient for coalescing, but not so helpful for translating the `obj` parameter into the i such that it is x_i. We simply search from the beginning to do this, which combined with other code using the iterator makes for some unnecessary $O(n^2)$ calculations. But it saves memory and nuisance, and the iterator is used only with printing to the screen, which never needs to be very rapid anyway (because the player can only read very slowly).

```
[ MarkedListIterator obj depth required_lt function i;
    if (obj == nothing) return nothing;
    switch(function) {
        START_ITF: return MarkedObjectArray-->0;
        COALESCE_ITF: return MarkedListCoalesce();
        SEEK_ITF, ADVANCE_ITF:
            for (i=0: i<MarkedObjectLength: i++)
                if (MarkedObjectArray-->i == obj) {
                    if (function == ADVANCE_ITF) i++;
                    for (:i<MarkedObjectLength: i++) {
                        obj = MarkedObjectArray-->i;
                        if ((required_lt) && (obj.list_together ~= required_lt)) continue;
                        if ((c_style & WORKFLAG_BIT) && (depth==0) && (obj hasnt workflag))
                            continue;
                        if ((c_style & CONCEAL_BIT) &&
                            ((obj has concealed) || (obj has scenery))) continue;
                        return obj;
                    }
                    return nothing;
                }
    }
    return nothing;
];
```

§6. Coalesce Marked List. The return value is the new first entry in the raw list.

```
[ MarkedListCoalesce o i lt l swap m;
    for (i=0: i<MarkedObjectLength: i++) {
        lt = (MarkedObjectArray-->i).list_together;
        if (lt ~= 0) {
            ! Find first object in list after contiguous run with this list_together value:
            for (i++: (i<MarkedObjectLength)&&((MarkedObjectArray-->i).list_together==lt): i++) ;
            ! If the contiguous run extends to end of list, the list is now perfect:
            if (i == MarkedObjectLength) return MarkedObjectArray-->0;
            ! And otherwise we look to see if any future entries belong in the earlier run:
            for (l=i+1: l<MarkedObjectLength: l++)
                if ((MarkedObjectArray-->l).list_together == lt) {
                    ! Yes, they do: so we perform a rotation to insert it before element i:
                    swap = MarkedObjectArray-->l;
                    for (m=l: m>i: m--) MarkedObjectArray-->m = MarkedObjectArray-->(m-1);
                    MarkedObjectArray-->i = swap;
                    ! And now the run is longer:
                    i++;
                    if (i == MarkedObjectLength) return MarkedObjectArray-->0;
                }
            i--;
        }
    }
    return MarkedObjectArray-->0;
];
```

§7. Object Tree Iterator. Here the raw list is the list of all children of a given parent: since the argument `obj` is required to be a member of the raw list, we can use `parent(obj)` to find it. Now seeking and advancing are fast, but coalescing is slower.

```
Global list_filter_routine;

[ ObjectTreeIterator obj depth required_lt function;
    if ((obj == nothing) || (parent(obj) == nothing)) return nothing;
    if (function == START_ITF) return child(parent(obj));
    if (function == COALESCE_ITF) return ObjectTreeCoalesce(obj);
    if (function == ADVANCE_ITF) obj = sibling(obj);
    for (:: obj = sibling(obj)) {
        if (obj == nothing) return nothing;
        if ((required_lt) && (obj.list_together ~= required_lt)) continue;
        if ((c_style & WORKFLAG_BIT) && (depth==0) && (obj hasnt workflag)) continue;
        if (obj hasnt list_filter_permits) continue;
        if ((c_style & CONCEAL_BIT) &&
            ((obj has concealed) || (obj has scenery))) continue;
        return obj;
    }
];
```

§**8. Coalesce Object Tree.** Again, the return value is the new first entry in the raw list.

```
[ ObjectTreeCoalesce obj memb lt later;
    #Ifdef DBLW; print "^^Sorting out: "; DiagnoseSortList(obj); #Endif;
    .StartAgain;
    for (memb=obj: memb~=nothing: memb=sibling(memb)) {
        lt = memb.list_together;
        if (lt ~= 0) {
            ! Find first object in list after contiguous run with this list_together value:
            for (memb=sibling(memb): (memb) && (memb.list_together == lt): memb = sibling(memb)) ;
            ! If the contiguous run extends to end of list, the list is now perfect:
            if (memb == 0) return obj;
            ! And otherwise we look to see if any future entries belong in the earlier run:
            for (later=sibling(memb): later: later=sibling(later))
                if (later.list_together == lt) {
                    ! Yes, they do: so we perform a regrouping of the list and start again:
                    obj = GroupChildren(parent(obj), list_together, lt);
                    #Ifdef DBLW; print "^^Sorted to: "; DiagnoseSortList(obj); #Endif;
                    jump StartAgain;
                }
        }
    }
    return obj;
];
#Ifdef DBLW;
[ DiagnoseSortList obj memb;
    for (memb=child(obj): memb~=nothing: memb=sibling(memb)) print memb, " --> "; new_line;
];
#Endif;
```

§**9. WriteListFrom.** And here we go at last. Or at any rate we initialise the quartet of global variables detailing the current list-writing process, and begin.

```
[ WriteListFrom first style depth noactivity iter a ol;
    @push c_iterator; @push c_style; @push c_depth; @push c_margin;
    if (iter) c_iterator = iter; else c_iterator = ObjectTreeIterator;
    c_style = style; c_depth = depth;
    c_margin = 0; if (style & EXTRAINDENT_BIT) c_margin = 1;
    objectloop (a ofclass Object) {
        give a list_filter_permits;
        if ((list_filter_routine) && (list_filter_routine(a) == false))
            give a ~list_filter_permits;
    }
    first = c_iterator(first, depth, 0, START_ITF);
    if (first == nothing) {
        print (string) NOTHING__TX;
        if (style & NEWLINE_BIT ~= 0) new_line;
    } else {
        if ((noactivity) || (iter)) {
            WriteListR(first, c_depth, true);
            say__p = 1;
        } else {
            objectloop (ol provides list_together) ol.list_together = 0;
            CarryOutActivity(LISTING_CONTENTS_ACT, parent(first));
        }
    }
    @pull c_margin; @pull c_depth; @pull c_style; @pull c_iterator;
];
```

§**10. Standard Contents Listing Rule.** The default for the listing contents activity is to call this rule in its "for" stage: note that this suppresses the use of the activity, to avoid an infinite regress. The activity is used only for the default `ObjectTreeIterator`, so there is no need to specify which is used.

```
[ STANDARD_CONTENTS_LISTING_R;
    WriteListFrom(child(parameter_object), c_style, c_depth, true);
];
```

§**11. Partitioning.** Given qualifying objects $x_1, ..., x_j$, we partition them into classes of the equivalence relation $x_i \sim x_j$ if and only

(i) they both have a `plural` property (not necessarily the same), and
(ii) neither will cause the list-maker to recurse downwards to show the objects inside or on top of them, and
(iii) if they cause the list-maker to add information about whether they are worn, lighted or open, then it will add the same information about both, and
(iv) they are considered identical by the parser, in that they respond to the same syntaxes of name: so that it is impossible to find a TAKE X command such that X matches one and not the other.

The equivalence classes are numbered consecutively upwards from 1 to n, in order of first appearance in the list. For each object x_i, `partition_classes->(i-1)` is the number of its equivalence class. For each equivalence class number c, `partition_class_sizes->c` is the number of objects in this class.

```
#Ifdef DBLW;
Global DBLW_no_classes; Global DBLW_no_objs;
[ DebugPartition partition_class_sizes partition_classes first depth i k o;
    print "[Length of list is ", DBLW_no_objs, " with ", k, " plural.]^";
    print "[Partitioned into ", DBLW_no_classes, " equivalence classes.]^";
    for (i=1: i<=DBLW_no_classes : i++) {
        print "Class ", i, " has size ", partition_class_sizes->i, "^";
    }
    for (k=0, o=first: k<DBLW_no_objs : k++, o = c_iterator(o, depth, lt_value, ADVANCE_ITF)) {
        print "Entry ", k, " has class ", partition_classes->k,
            " represented by ", o, " with L=", o.list_together, "^";
    }
];
#Endif;
```

§12. Partition List. The following creates the `partition_classes` and `partition_class_sizes` accordingly. We return n, the number of classes.

```
[ PartitionList first no_objs depth partition_classes partition_class_sizes
    i k l n m;
    for (i=0: i<no_objs: i++) partition_classes->i = 0;
    n = 1;
    for (i=first, k=0: k<no_objs: i=c_iterator(i, depth, lt_value, ADVANCE_ITF), k++)
        if (partition_classes->k == 0) {
            partition_classes->k = n; partition_class_sizes->n = 1;
            for (l=c_iterator(i, depth, lt_value, ADVANCE_ITF), m=k+1:
                (l~=0) && (m<no_objs):
                l=c_iterator(l, depth, lt_value, ADVANCE_ITF), m++) {
                if ((partition_classes->m == 0) && (ListEqual(i, l))) {
                    if (partition_class_sizes->n < 255) (partition_class_sizes->n)++;
                    partition_classes->m = n;
                }
            }
            if (n < 255) n++;
        }
    n--;
    #Ifdef DBLW;
    DBLW_no_classes = n; DBLW_no_objs = no_objs;
    DebugPartition(partition_class_sizes, partition_classes, first, depth);
    #Endif;
    return n;
];
```

§13. Equivalence Relation. The above algorithm will fail unless `ListEqual` is indeed reflexive, symmetric and transitive, which ultimately depends on the care with which `Identical` is implemented, which in turn hangs on the `parse_noun` properties compiled by NI. But this seems to be sound.

```
[ ListEqual o1 o2;
    if ((o1.plural == 0) || (o2.plural == 0)) rfalse;
    if (child(o1) ~= 0 && WillRecurs(o1) ~= 0) rfalse;
    if (child(o2) ~= 0 && WillRecurs(o2) ~= 0) rfalse;
    if (c_style & (FULLINV_BIT + PARTINV_BIT) ~= 0) {
        if ((o1 hasnt worn && o2 has worn) || (o2 hasnt worn && o1 has worn)) rfalse;
        if ((o1 hasnt light && o2 has light) || (o2 hasnt light && o1 has light)) rfalse;
        if (o1 has container) {
            if (o2 hasnt container) rfalse;
            if ((o1 has open && o2 hasnt open) || (o2 has open && o1 hasnt open))
                rfalse;
        }
        else if (o2 has container)
            rfalse;
    }
    return Identical(o1, o2);
];

[ WillRecurs o;
    if (c_style & ALWAYS_BIT ~= 0) rtrue;
    if (c_style & RECURSE_BIT == 0) rfalse;
    if ((o has supporter) || ((o has container) && (o has open or transparent))) rtrue;
    rfalse;
];
```

§14. Grouping. A “group” is a maximally-sized run of one or more adjacent partition classes in the list whose first members have a common value of `list_together` which is a routine or string, and which is not equal to `lt_value`, the current grouping value. (As we see below, it’s by setting `lt_value` that we restrict attention to a particular group: if we reacted to that as a `list_together` value here, then we would simply go around in circles, and never be able to see the individual objects in the group.)

For instance, suppose we have objects with `list_together` values as follows, where R_1 and R_2 are addresses of routines:

`coin` (R_1), `coin` (R_1), `box` (R_2), `statuette` (0), `coin` (R_1), `box` (R_2)

Then the partition is 1, 1, 2, 3, 1, 2, so that the final output will be something like “three coins, two boxes and a statuette” – with three grouped terms. Here each partition class is the only member in its group, so the number of groups is the same as the number of classes. (The above list is not coalesced, so the R_1 values, for instance, are not contiguous: we need to work in general with non-coalesced lists because not every iterator function will be able to coalesce fully.)

But if we have something like this:

`coin` (R_1), `Q` (R_2), `W` (R_2), `coin` (R_1), `statuette` (0), `E` (R_2), `R` (R_2)

then the partition is 1, 2, 3, 1, 4, 5, 6 and we have six classes in all. But classes 2 and 3 are grouped together, as are classes 5 and 6, so we end up with a list having just four groups: “two coins, the letters Q and W from a Scrabble set, a statuette and the letters E and R from a Scrabble set”. (Again, this list has not been fully coalesced: if it had been, it would be reordered `coin`, `coin`, `Q`, `W`, `E`, `R`, `statuette`, with partition 1, 1, 2, 3, 4, 5, 6, and three groups: “two coins, the letters Q, W, E and R from a Scrabble set and a statuette”.)

The reason we do not group together classes with a common non-zero `list_together` which is *not* a routine or string is that low values of `list_together` are used in coalescing the list into a pleasing order (say, moving all of the key-like items together) but not in grouping: see `list_together` in the *Inform Designer’s Manual*, 4th edition.

We calculate the number of groups by starting with the number of classes and then subtracting one each time two adjacent classes share `list_together` in the above sense.

```
[ NumberOfGroupsInList o no_classes depth partition_classes partition_class_sizes
    no_groups cl memb k current_lt lt;
    no_groups = no_classes;
    for (cl=1, memb=o, k=0: cl<=no_classes: cl++) {
        ! Advance to first member of class number cl
        while (partition_classes->k ~= cl) {
            k++; memb = c_iterator(memb, depth, lt_value, ADVANCE_ITF);
        }
        if (memb) { ! In case of accidents, but should always happen
            lt = memb.list_together;
            if ((lt ~= lt_value) && (lt ofclass Routine or String) && (lt == current_lt))
                no_groups--;
            current_lt = lt;
        }
    }
    #Ifdef DBLW; print "[There are ", no_groups, " groups.]^"; #Endif;
    return no_groups;
];
```

§**15. Write List Recursively.** The big one: `WriteListR` is the heart of the list-writer.

```
[ WriteListR o depth from_start
    partition_classes partition_class_sizes
    cl memb index k2 l m no_classes q groups_to_do current_lt;
    if (o == nothing) return; ! An empty list: no output
    if (from_start) {
        o = c_iterator(o, depth, 0, COALESCE_ITF); ! Coalesce list and choose new start
    }
    o = c_iterator(o, depth, 0, SEEK_ITF); ! Find first entry in list from o
    if (o == nothing) return;
    ! Count index = length of list
    for (memb=o, index=0: memb: memb=c_iterator(memb, depth, lt_value, ADVANCE_ITF)) index++;
    if (c_style & ISARE_BIT ~= 0) {
        if (index == 1 && o hasnt pluralname) print (string) IS3__TX;
        else                              print (string) ARE3__TX;
        if (c_style & NEWLINE_BIT ~= 0)   print ":^";
        else                              print (char) ' ';
        c_style = c_style - ISARE_BIT;
    }
    partition_classes = RequisitionStack(index/WORDSIZE + 2);
    partition_class_sizes = RequisitionStack(index/WORDSIZE + 2);
    if ((partition_classes == 0) || (partition_class_sizes == 0))
        return RunTimeProblem(RTP_LISTWRITERMEMORY);
    no_classes =
        PartitionList(o, index, depth, partition_classes, partition_class_sizes);
    groups_to_do =
        NumberOfGroupsInList(o, no_classes, depth, partition_classes, partition_class_sizes);
    for (cl=1, memb=o, index=0, current_lt=0: groups_to_do>0: cl++) {
        ! Set memb to first object of partition class cl
        while (partition_classes->index ~= cl) {
            index++; memb=c_iterator(memb, depth, lt_value, ADVANCE_ITF);
            if (memb==0) { print "*** Error in list-writer ***^"; break; }
        }
        #Ifdef DBLW;
        ! DebugPartition(partition_class_sizes, partition_classes, o, depth);
        print "^[Class ", cl, " of ", no_classes, ": first object ", memb,
            " (", memb.list_together, "); groups_to_do ", groups_to_do, ",
            current_lt=", current_lt, " listing_size=", listing_size,
            " lt_value=", lt_value, " memb.list_together=", memb.list_together, "]^";
        #Endif;
        if ((memb.list_together == lt_value) ||
            (~~(memb.list_together ofclass Routine or String))) current_lt = 0;
        else {
            if (memb.list_together == current_lt) continue;
            ! Otherwise this class begins a new group
            @push listing_size;
            q = memb; listing_size = 1; l = index; m = cl;
            while (m < no_classes && q.list_together == memb.list_together) {
                m++;
                while (partition_classes->l ~= m) {
                    l++; q = c_iterator(q, depth, lt_value, ADVANCE_ITF);
                }
                if (q.list_together == memb.list_together) listing_size++;
            }
            if (listing_size > 1) {
                ! The new group contains more than one partition class
                WriteMultiClassGroup(cl, memb, depth, partition_class_sizes);
                current_lt = memb.list_together;
                jump GroupComplete;
            }
            current_lt = 0;
            @pull listing_size;
        }
        WriteSingleClassGroup(cl, memb, depth, partition_class_sizes->cl);
        .GroupComplete;
        groups_to_do--;
        if (c_style & ENGLISH_BIT ~= 0) {
            if (groups_to_do == 1) {
                if (cl <= 1) print (string) LISTAND2__TX;
                else print (string) LISTAND__TX;
            }
            if (groups_to_do > 1) print (string) COMMA__TX;
        }
    }
    FreeStack(partition_class_sizes);
    FreeStack(partition_classes);
]; ! end of WriteListR
```

§16. Write Multiple Class Group. The text of a single group which contains more than one partition class. We carry out the "grouping together" activity, so that the user can add text fore and aft – this is how groups of objects such as "X, Y and Z" can be fluffed up to "the letters X, Y and Z from a Scrabble set" – but the default is simple: we print the grouped items by calling `WriteListR` once again, but this time starting from X, and where `lt_value` is set to the common `list_together` value of X, Y and Z. (That restricts the list to just the objects in this group.) Because `lt_value` is set to this value, the grouping code won't then group X, Y and Z again, and they will instead be individual classes in the new list – so each will end up being sent, in turn, to `WriteSingleClassGroup` below, and *that* is where they are printed.

```
[ WriteMultiClassGroup cl memb depth partition_class_sizes q k2 l;
    ! Save the style, because the activity below is allowed to change it
    q = c_style;
    if (c_style & INDENT_BIT ~= 0) PrintSpaces(2*(depth+c_margin));
    BeginActivity(GROUPING_TOGETHER_ACT, memb);
    if (ForActivity(GROUPING_TOGETHER_ACT, memb)) {
        c_style = c_style &~ NEWLINE_BIT;
    } else {
        if (memb.list_together ofclass String) {
            ! Set k2 to the number of objects covered by the group
            k2 = 0;
            for (l=0 : l<listing_size : l++) k2 = k2 + partition_class_sizes->(l+cl);
            EnglishNumber(k2); print " ";
            print (string) memb.list_together;
            if (c_style & ENGLISH_BIT ~= 0) print " (";
            if (c_style & INDENT_BIT ~= 0)  print ":^";
        } else {
            inventory_stage = 1;
            parser_one = memb; parser_two = depth + c_margin;
            if (RunRoutines(memb, list_together) == 1) jump Omit__Sublist2;
        }
        c_margin++;
        @push lt_value; @push listing_together; @push listing_size;
        lt_value = memb.list_together; listing_together = memb;
        #Ifdef DBLW; print "^^DOWN lt_value = ", lt_value, " listing_together = ", memb, "^^";
        @push DBLW_no_classes; @push DBLW_no_objs; #Endif;
        WriteListR(memb, depth, false);
        #Ifdef DBLW; print "^^UP^^"; @pull DBLW_no_objs; @pull DBLW_no_classes; #Endif;
        @pull listing_size; @pull listing_together; @pull lt_value;
        c_margin--;
        if (memb.list_together ofclass String) {
            if (q & ENGLISH_BIT ~= 0) print ")";
        } else {
            inventory_stage = 2;
            parser_one = memb; parser_two = depth+c_margin;
            RunRoutines(memb, list_together);
        }
        .Omit__Sublist2;
    }
    EndActivity(GROUPING_TOGETHER_ACT, memb);
    ! If the NEWLINE_BIT has been forced by the activity, act now
    ! before it vanishes...
    if (q & NEWLINE_BIT ~= 0 && c_style & NEWLINE_BIT == 0) new_line;
    ! ...when the original style is restored again:
    c_style = q;
];
```

§17. Write Single Class Group. The text of a single group which contains exactly one partition class. Because of the way the multiple-class case recurses, every class ends up in this routine sooner or later; this is the place where the actual name of an object is printed, at long last.

```
[ WriteSingleClassGroup cl memb depth size q;
    q = c_style;
    if (c_style & INDENT_BIT) PrintSpaces(2*(depth+c_margin));
    if (size == 1) {
        if (c_style & NOARTICLE_BIT ~= 0) print (name) memb;
        else {
            if (c_style & DEFART_BIT) {
                if ((cl == 1) && (c_style & CFIRSTART_BIT)) print (The) memb;
                else print (the) memb;
            } else {
                if ((cl == 1) && (c_style & CFIRSTART_BIT)) print (CIndefArt) memb;
                else print (a) memb;
            }
        }
    } else {
        if (c_style & DEFART_BIT) {
            if ((cl == 1) && (c_style & CFIRSTART_BIT)) PrefaceByArticle(memb, 0, size);
            else PrefaceByArticle(memb, 1, size);
        }
        @push listing_size; listing_size = size;
        CarryOutActivity(PRINTING_A_NUMBER_OF_ACT, memb);
        @pull listing_size;
    }
    if ((size > 1) && (memb hasnt pluralname)) {
        give memb pluralname;
        WriteAfterEntry(memb, depth);
        give memb ~pluralname;
    } else WriteAfterEntry(memb, depth);
    c_style = q;
];
```

§18. Write After Entry. Each entry can be followed by supplementary, usually parenthetical, information: exactly what, depends on the style. The extreme case is when the style, and the object, call for recursion to list the object-tree contents: this is achieved by calling `WriteListR`, using the `ObjectTreeIterator` (whatever the iterator used at the top level) and increasing the depth by 1.

```
[ WriteAfterEntry o depth
    p recurse_flag parenth_flag eldest_child child_count combo;
    inventory_stage = 2;
    if (c_style & PARTINV_BIT) {
        BeginActivity(PRINTING_ROOM_DESC_DETAILS_ACT);
        if (ForActivity(PRINTING_ROOM_DESC_DETAILS_ACT) == false) {
        combo = 0;
        if (o has light && location hasnt light) combo=combo+1;
        if (o has container && o hasnt open)     combo=combo+2;
        if ((o has container && (o has open || o has transparent))
            && (child(o)==0))                    combo=combo+4;
        if (combo) L__M(##ListMiscellany, combo, o);
        }
        EndActivity(PRINTING_ROOM_DESC_DETAILS_ACT);
    }   ! end of PARTINV_BIT processing
    if (c_style & FULLINV_BIT) {
        if (o has light && o has worn) { L__M(##ListMiscellany, 8, o);  parenth_flag = true; }
        else {
            if (o has light)           { L__M(##ListMiscellany, 9, o);  parenth_flag = true; }
            if (o has worn)            { L__M(##ListMiscellany, 10, o); parenth_flag = true; }
        }
        if (o has container)
            if (o has openable) {
                if (parenth_flag) {
                    #Ifdef SERIAL_COMMA; print ","; #Endif;
                    print (string) AND__TX;
                } else           L__M(##ListMiscellany, 11, o);
                if (o has open)
                    if (child(o)) L__M(##ListMiscellany, 12, o);
                    else          L__M(##ListMiscellany, 13, o);
                else
                    if (o has lockable && o has locked) L__M(##ListMiscellany, 15, o);
                    else                                L__M(##ListMiscellany, 14, o);
                parenth_flag = true;
            }
            else
                if (child(o)==0 && o has transparent)
                    if (parenth_flag) L__M(##ListMiscellany, 16, o);
                    else              L__M(##ListMiscellany, 17, o);
        if (parenth_flag) print ")";
    }   ! end of FULLINV_BIT processing
    child_count = 0;
    eldest_child = nothing;
    objectloop (p in o)
        if ((c_style & CONCEAL_BIT == 0) || (p hasnt concealed && p hasnt scenery))
            if (p has list_filter_permits) {
                child_count++;
                if (eldest_child == nothing) eldest_child = p;
            }
    if (child_count && (c_style & ALWAYS_BIT)) {
        if (c_style & ENGLISH_BIT) L__M(##ListMiscellany, 18, o);
        recurse_flag = true;
    }
    if (child_count && (c_style & RECURSE_BIT)) {
        if (o has supporter) {
            if (c_style & ENGLISH_BIT) {
                if (c_style & TERSE_BIT) L__M(##ListMiscellany, 19, o);
                else                     L__M(##ListMiscellany, 20, o);
                if (o has animate)       print (string) WHOM__TX;
                else                     print (string) WHICH__TX;
            }
            recurse_flag = true;
        }
        if (o has container && (o has open || o has transparent)) {
            if (c_style & ENGLISH_BIT) {
                if (c_style & TERSE_BIT) L__M(##ListMiscellany, 21, o);
                else                     L__M(##ListMiscellany, 22, o);
                if (o has animate)       print (string) WHOM__TX;
                else                     print (string) WHICH__TX;
            }
            recurse_flag = true;
        }
    }
    if (recurse_flag && (c_style & ENGLISH_BIT))
        if (child_count > 1 || eldest_child has pluralname) print (string) ARE2__TX;
        else                                                print (string) IS2__TX;
    if (c_style & NEWLINE_BIT) new_line;
    if (recurse_flag) {
        o = child(o);
        @push lt_value; @push listing_together; @push listing_size;
        @push c_iterator;
        c_iterator = ObjectTreeIterator;
```

```
        lt_value = 0;   listing_together = 0;   listing_size = 0;
        WriteListR(o, depth+1, true);
        @pull c_iterator;
        @pull listing_size; @pull listing_together; @pull lt_value;
        if (c_style & TERSE_BIT) print ")";
    }
];
```

OutOfWorld Template B/oowt

Purpose

To implement some of the out of world actions.

§1. Perform Undo.

```
[ Perform_Undo;
    #ifdef PREVENT_UNDO; L__M(##Miscellany, 70); return; #endif;
    if (turns == 1) { L__M(##Miscellany, 11); return; }
    if (undo_flag == 0) { L__M(##Miscellany, 6); return; }
    if (undo_flag == 1) { L__M(##Miscellany, 7); return; }
    if (VM_Undo() == 0) L__M(##Miscellany, 7);
];
```

§2. Announce Score Rule.

```
[ ANNOUNCE_SCORE_R;
    if (actor ~= player) rfalse;
    #ifdef NO_SCORING; L__M(##Score, 2);
    #ifnot; GL__M(##Score); PrintRank();
    #endif;
];
```

§3. Switch Score Notification On Rule.

```
[ SWITCH_SCORE_NOTIFY_ON_R;
    if (actor ~= player) rfalse;
    #ifdef NO_SCORING; ANNOUNCE_SCORE_R();
    #ifnot; notify_mode=1; #endif;
];
```

§4. Standard Report Switching Score Notification On Rule.

```
[ REP_SWITCH_NOTIFY_ON_R;
    if (actor ~= player) rfalse;
    #ifndef NO_SCORING; GL__M(##NotifyOn); #endif;
];
```

§5. Switch Score Notification Off Rule.

```
[ SWITCH_SCORE_NOTIFY_OFF_R;
    if (actor ~= player) rfalse;
    #ifdef NO_SCORING; ANNOUNCE_SCORE_R();
    #ifnot; notify_mode=0; #endif;
];
```

§6. Standard Report Switching Score Notification Off Rule.

```
[ REP_SWITCH_NOTIFY_OFF_R;
    if (actor ~= player) rfalse;
    #ifndef NO_SCORING; GL__M(##NotifyOff); #endif;
];
```

§7. Prefer Sometimes Abbreviated Room Descriptions Rule.

```
[ PREFER_SOMETIMES_ABBREVIATED_R;
    if (actor ~= player) rfalse;
    lookmode=1;
]; ! Brief
```

§8. Standard Report Prefer Sometimes Abbreviated Room Descriptions Rule.

```
[ REP_PREFER_SOMETIMES_ABBR_R;
    if (actor ~= player) rfalse;
    print (string) Story; GL__M(##LMode1);
]; ! Brief
```

§9. Prefer Unabbreviated Room Descriptions Rule.

```
[ PREFER_UNABBREVIATED_R;
    if (actor ~= player) rfalse;
    lookmode=2;
]; ! Verbose
```

§10. Standard Report Prefer Unabbreviated Room Descriptions Rule.

```
[ REP_PREFER_UNABBREVIATED_R;
    if (actor ~= player) rfalse;
    print (string) Story; GL__M(##LMode2);
]; ! Verbose
```

§11. Prefer Abbreviated Room Descriptions Rule.

```
[ PREFER_ABBREVIATED_R;
    if (actor ~= player) rfalse;
    lookmode=3;
]; ! Superbrief
```

§12. Standard Report Prefer Abbreviated Room Descriptions Rule.

```
[ REP_PREFER_ABBREVIATED_R;
    if (actor ~= player) rfalse;
    print (string) Story; GL__M(##LMode3);
]; ! Superbrief
```

§13. Announce Pronoun Meanings Rule.

```
[ ANNOUNCE_PRONOUN_MEANINGS_R x y c d;
    if (actor ~= player) rfalse;
    GL__M(##Pronouns, 1);
    c = (LanguagePronouns-->0)/3;
    if (player ~= selfobj) c++;
    if (c==0) return GL__M(##Pronouns, 4);
    for (x = 1, d = 0 : x <= LanguagePronouns-->0: x = x+3) {
        print "~", (address) LanguagePronouns-->x, "~ ";
        y = LanguagePronouns-->(x+2);
        if (y == NULL) GL__M(##Pronouns, 3);
        else { GL__M(##Pronouns, 2); print (the) y; }
        d++;
        if (d < c-1) print ", ";
        if (d == c-1) print (string) LISTAND__TX;
    }
    if (player ~= selfobj) {
        print "~", (address) ME1__WD, "~ "; GL__M(##Pronouns, 2);
        c = player; player = selfobj;
        print (the) c; player = c;
    }
    ".";
];
```

WorldModel Template B/wmt

Purpose

Testing and changing the fundamental spatial relations.

§**1. The Core Tree.** Whereas I6 traditionally has a simple object tree hierarchy for containment, support, carrying and so on, the I7 template also uses the `component_*` properties to provide a second tree which defines the "part of" relation. These two trees interact in a subtle way, and it took a very long time to work out the simplest way to express this.

The *core* of an object is the root of its subtree in the component relation tree. So for a television set with a control panel which has a button on, the core for the button (and also for the panel and the set) will be the set. When X is a part of Y, it must be spatially in the same position as Y, and so the spatial location of X is determined by the position in the object tree of its core. (In effect, the spatial situation can be found by contracting together all nodes corresponding to objects which are parts of each other, but it would waste memory to construct such a tree: `CoreOfParentOfCoreOf` simulates what the `parent` operation would be in such a tree if it existed, wasting a little time instead.)

The *holder* of an object is its component parent, if it is part of something, or its object-tree parent if it has one. It is illegal for both of these to be non-`nothing`, so this is unambiguous.

It is just possible for `HolderOf` to be called before the player has been placed into the model world, in cases where Inform is checking a past tense condition in the opening pre-turn. We don't want to return `nothing` then because that would make it true that

```
    HolderOf(player) == ContainerOf(player)
```

and similar conditions – thus, it would appear that in the immediate past the player had been on the holder of the player, which is not in fact the case. So we return `thedark` as a typesafe but impossible value here.

```
[ HolderOf o;
    if (InitialSituation-->DONE_INIS == false) return thedark;
    if (o && (o.component_parent)) return o.component_parent;
    if (o && (parent(o))) return parent(o);
    return nothing;
];
[ ParentOf o;
    if (o) o = parent(o);
    return o;
];
[ CoreOf o;
    while (o && (o provides component_parent) && (o.component_parent)) o = o.component_parent;
    return o;
];
[ CoreOfParentOfCoreOf o;
    while (o && (o provides component_parent) && (o.component_parent)) o = o.component_parent;
    if (o) o = parent(o);
    while (o && (o provides component_parent) && (o.component_parent)) o = o.component_parent;
    return o;
];
```

§**2. Climbing the Core Tree.** `LocationOf` returns the room in which an object can be found, or `nothing` if it is out of play. For this purpose a backdrop has no location, there being no canonical choice to make, while a two-sided door is in its "front side" (see below). Directions and regions are always out of play. For a room, `LocationOf` is necessarily itself.

`CommonAncestor` finds the nearest object indirectly containing `o1` and `o2`, or returns `nothing` if there is no common ancestor. (This is a port of `CommonAncestor` from the I6 library, adapted to work on the core tree.)

```
[ LocationOf o;
    if (~~(O ofclass K1_room or K2_thing)) return nothing;
    if (O ofclass K4_door) return FrontSideOfDoor(O);
    if (O ofclass K7_backdrop) return nothing;
    while (o) {
        if (o ofclass K1_room) return o;
        o = CoreOfParentOfCoreOf(o);
    }
    return nothing;
];
[ CommonAncestor o1 o2 i j;
    o1 = CoreOf(o1);
    o2 = CoreOf(o2);
    for (i=o1: i: i = CoreOfParentOfCoreOf(i))
        for (j=o2: j: j = CoreOfParentOfCoreOf(j))
            if (j == i) return j;
    return nothing;
];
[ IndirectlyContains o1 o2;
    if ((o1 == nothing) || (o2 == nothing)) rfalse;
    if ((o1 ofclass K1_room) && (o2 ofclass K4_door)) {
        if (o1 == FrontSideOfDoor(o2)) rtrue;
        if (o1 == BackSideOfDoor(o2)) rtrue;
        rfalse;
    }
    if (o2 ofclass K7_backdrop) rfalse;
    for (o2 = HolderOf(o2) : o2 && o2 ~= thedark : o2 = HolderOf(o2)) if (o2 == o1) rtrue;
    rfalse;
];
```

§3. To Decide Whether In. A curiosity, this: the I6 definition of "To decide whether in (obj - object)". As can be seen, there are three possible interpretations of "in", depending on the kind of object, so this could all be done with three different definitions in the Standard Rules, so that run-time type checking would decide which to apply: but that would produce a little overhead and make the code for this rather common operation a bit complicated. Besides, this way we can produce a respectable run-time problem message when the phrase is misapplied.

Note that "in X" is not equivalent to "the player is in X": the latter uses direct containment, whereas we use indirect containment. Thus "in the Hall" and "in the laundry-basket" are both true if the player is in a laundry-basket in the Hall.

```
[ WhetherIn obj;
    if (obj has enterable) {
        if (IndirectlyContains(obj, player)) rtrue;
        rfalse;
    }
    if (obj ofclass K9_region) return TestRegionalContainment(real_location, obj);
    if (obj ofclass K1_room) {
        if (obj == real_location) rtrue;
        rfalse;
    }
    RunTimeProblem(RTP_NOTINAROOM, obj);
    rfalse;
];
```

§4. Containment Relation. This is the single most important relation in I7: direct containment. It is complicated by the fact that "A is in B" is represented differently at run-time when A is a room and B is a region, and when it isn't.

For each A there is at most one B such that "A is in B" is true. Because of this we test the relation with a function of one variable which turns A into B, rather than a function of two variables returning true or false. `ContainerOf` is that function.

I7 frequently needs to compile loops over all A such that "A is in B". In simpler relations (such as the support relation below) it can do this efficiently by iterating through the object-tree children of B, but for containment we have to provide an iterator function `TestContainmentRange`, as otherwise we would get the wrong result when B is a region.

```
[ ContainerOf A p;
    if (A ofclass K1_room) return A.map_region;
    p = parent(A);
    if (p == nothing) return nothing;
    if (p ofclass K5_container) return p;
    if (p ofclass K1_room) return p;
    if (p ofclass K9_region) return p;
    return nothing;
];
[ TestContainmentRange obj e f;
    if (obj ofclass K9_region) {
        objectloop (f ofclass K1_room && f.map_region == obj)
            if (f > e) return f;
        return nothing;
    }
    if (obj ofclass K5_container or K1_room) {
        if (e == nothing) return child(obj);
        return sibling(e);
    }
    return nothing;
];
```

§**5. Support Relation.** This then is simpler, with no need for an iterator governing searches.

```
[ SupporterOf obj p;
    p = parent(obj);
    if (p == nothing) return nothing;
    if (p ofclass K6_supporter) return p;
    return nothing;
];
```

§**6. Carrying Relation.** Only people may carry, and something worn is not in this sense carried.

```
[ CarrierOf obj p;
    p = parent(obj);
    if (p && (p ofclass K8_person) && (obj hasnt worn)) return p;
    return nothing;
];
```

§**7. Wearing Relation.** Only people may wear.

```
[ WearerOf obj p;
    p = parent(obj);
    if (p && (p ofclass K8_person) && (obj has worn)) return p;
    return nothing;
];
```

§**8. Having Relation.** A person has something if and only if he either wears or carries it.

```
[ OwnerOf obj p;
    p = parent(obj);
    if (p && (p ofclass K8_person)) return p;
    return nothing;
];
```

§9. Making Parts. Note that MakePart removes the part-to-be from the object tree before attaching it: it cannot, of course, be the core of the resulting object.

```
[ MakePart P Of First;
    if (parent(P)) remove P; give P ~worn;
    if (Of == nothing) { DetachPart(P); return; }
    if (P.component_parent) DetachPart(P);
    P.component_parent = Of;
    First = Of.component_child;
    Of.component_child = P; P.component_sibling = First;
];
[ DetachPart P From Daddy O;
    Daddy = P.component_parent; P.component_parent = nothing;
    if (Daddy == nothing) { P.component_sibling = nothing; return; }
    if (Daddy.component_child == P) {
        Daddy.component_child = P.component_sibling;
        P.component_sibling = nothing; return;
    }
    for (O = Daddy.component_child: O: O = O.component_sibling)
        if (O.component_sibling == P) {
            O.component_sibling = P.component_sibling;
            P.component_sibling = nothing; return;
        }
];
```

§10. Movements. Note that an object is detached from its component parent, if it has one, when moved.

```
[ MoveObject F T opt going_mode was;
    if (F == nothing) return RunTimeProblem(RTP_CANTMOVENOTHING);
    if (F ofclass K7_backdrop) {
        if (T ofclass K9_region) {
            give F ~absent; F.found_in = T.regional_found_in;
            if (TestRegionalContainment(LocationOf(player), T)) move F to LocationOf(player);
            else remove F;
            return; }
        return RunTimeProblem(RTP_BACKDROP, F, T);
    }
    if (~~(F ofclass K2_thing)) return RunTimeProblem(RTP_NOTTHING, F, T);
    if (T ofclass K9_region) return RunTimeProblem(RTP_NOTBACKDROP, F, T);
    if (F has worn) {
        give F ~worn;
        if (F in T) return;
    }
    DetachPart(F);
    if (going_mode == false) {
        if (F == player) { PlayerTo(T, opt); return; }
        if ((IndirectlyContains(F, player)) && (LocationOf(player) ~= LocationOf(T))) {
            was = parent(player);
            move player to real_location;
            move F to T;
            PlayerTo(was, true);
            return;
        }
    }
    move F to T;
];
[ RemoveFromPlay F;
    if (F == nothing) return RunTimeProblem(RTP_CANTREMOVENOTHING);
    if (F == player) return RunTimeProblem(RTP_CANTREMOVEPLAYER);
    if (F ofclass K4_door) return RunTimeProblem(RTP_CANTREMOVEDOORS);
    give F ~worn; DetachPart(F);
    if (F ofclass K7_backdrop) give F absent;
    remove F;
];
```

§11. On Stage. The following implements the "on-stage" and "off-stage" adjectives provided by the Standard Rules. Here, as above, note that the I6 attribute `absent` marks a floating object (see below) which has been removed from play; in I7 only doors and backdrops are allowed to float, and only backdrops are allowed to be removed from play.

```
[ OnStage O set x;
    if (O ofclass K1_room) rfalse;
    if (set < 0) {
        while (metaclass(O) == Object) {
            if (O ofclass K1_room) rtrue;
            if (O ofclass K9_region) rfalse;
            if (O ofclass K4_door) rtrue;
            if (O ofclass K7_backdrop) { if (O has absent) rfalse; rtrue; }
            x = O.component_parent; if (x) { O = x; continue; }
            x = parent(O); if (x) { O = x; continue; }
            rfalse;
        }
    }
    x = OnStage(O, -1);
    if ((x) && (set == false)) RemoveFromPlay(O);
    if ((x == false) && (set)) MoveObject(O, real_location);
    rfalse;
];
```

§12. Moving the Player. Note that the player object can only be moved by this routine: this allows us to maintain the invariant for `real_location` and `location` (for which, see "Light.i6t") and to ensure that multiply-present objects can be witnessed where they need to be.

```
[ PlayerTo newplace flag;
    @push actor; actor = player;
    move player to newplace;
    location = LocationOf(newplace);
    real_location = location;
    MoveFloatingObjects();
    SilentlyConsiderLight();
    DivideParagraphPoint();
    if (flag == 0) <Look>;
    if (flag == 1) give location visited;
    if (flag == 2) AbbreviatedRoomDescription();
    @pull actor;
];
```

§13. Move During Going. The following routine preserves the invariant for `real_location`, but gets `location` wrong since it doesn't adjust for light. Nor are floating objects moved. It should be used only in the course of other operations which get the rest of this right. (I7 uses it only for the "going" action, where these various operations are each handled by different named rules to increase the flexibility of the system.)

```
[ MoveDuringGoing F T;
    MoveObject(F, T, 0, true);
    if (actor == player) {
        location = LocationOf(player);
        real_location = location;
    }
];
```

§14. Being Everywhere. The following is used as the `found_in` property for any backdrop which is "everywhere", that is, which is found in every room.

```
[ FoundEverywhere; rtrue; ];
```

§15. Changing the Player. It is very important that nobody simply change the value of the `player` variable, because so much else must be updated when the identity of the player changes: the light situation, floating objects, the `real_location` and so on. Because of this, the NI compiler contains code which compiles an assertion of the proposition "is(`player`, X)" into a function call `ChangePlayer(X)` rather than a variable assignment `player = X`. So we cannot catch out the system by writing "now the player is Mr Henderson".

We must ensure that if `player` is initially X, is changed to Y, and is then changed back to X, that both X and Y end exactly as they began – hence the flummery below with using `remove_proper` to ensure that `proper` is left with the correct value. Note that:

(1) at any given time exactly one person has the I6 `concealed` attribute: the current player;

(2) the `selfobj` is the default initial value of `player`, and because it has as its actual printed name "yourself", we need to override this when something else takes over as player: we change to "your former self", in fact. No such device is needed for other people being changed from because they are explicitly given printed names – say "Mr Darcy", "the sous-chef", etc. – in the source text.

```
[ ChangePlayer obj flag i;
    if (~~(obj ofclass K8_person)) return RunTimeProblem(RTP_CANTCHANGE, obj);
    if (~~(OnStage(obj, -1))) return RunTimeProblem(RTP_CANTCHANGEOFFSTAGE, obj);
    if (obj == player) return;

    give player ~concealed;
    if (player has remove_proper) give player ~proper;
    if (player == selfobj) {
        player.saved_short_name = player.short_name; player.short_name = FORMER__TX;
    }
    player = obj;
    if (player == selfobj) {
        player.short_name = player.saved_short_name;
    }
    if (player hasnt proper) give player remove_proper; ! when changing out again
    give player concealed proper;

    location = LocationOf(player); real_location = location;
    MoveFloatingObjects();
    SilentlyConsiderLight();
];
```

§16. Floating Objects. A single object can only be in one position in the object tree, yet backdrops and doors must be present in multiple rooms. This is accomplished by making them, in I6 jargon, "floating objects": objects which move in the tree whenever the player does, so that – from the player's perspective – they are always present when they should be. In I6, almost anything can be made a floating object, but in I7 this is strictly and only used for backdrops and two-sided doors.

There are several conceptual problems with this scheme: chiefly that it assumes that the only witness to the spatial arrangement of objects is the player. In I6 that was usually true, but in I7, where every person can undertake actions, it really isn't true any longer: if the objects float to follow the player, it means that they are not present with other people who might need to interact with them. This is why the accessibility rules are somewhat hacked for backdrops and doors (see "Light.i6t"). In fact we generally achieve the illusion we want, but this is largely because it is difficult to catch out all the exceptions for backdrops and doors, and because in practice authors tend not to set things up so that the presence or absence of backdrops much affects what non-player characters do.

All the same, the scheme is not logically defensible, and this is why we do not allow the user to create new categories of floating objects in I7.

The I6 implementation of `MoveFloatingObjects` acted on the `location` rather than the `real_location`, which (a) meant that multiply-present objects could include `thedark` – the generic Darkness place – as one possible location, but (b) assumed that the only sense by which the player could witness an object was sight. In I7, `thedark` is not a valid room, and we are a bit more careful about the senses.

```
[ MoveFloatingObjects i k l m address flag;
    if (real_location == nothing) return;
    objectloop (i) {
        address = i.&found_in;
        if (address ~= 0 && i hasnt absent) {
            if (ZRegion(address-->0) == 2) {
                m = address-->0;
                .TestPropositionally;
                if (m.call(real_location) ~= 0) move i to real_location;
                else remove i;
            }
            else {
                k = i.#found_in;
                for (l=0 : l<k/WORDSIZE : l++) {
                    m = address-->l;
                    if (ZRegion(m) == 2) jump TestPropositionally;
                    if (m == real_location || m in real_location) {
                        if (i notin real_location) move i to real_location;
                        flag = true;
                    }
                }
                if (flag == false) { if (parent(i)) remove i; }
            }
        }
    }
];
[ MoveBackdrop bd D x address;
    if (~~(bd ofclass K7_backdrop)) return RunTimeProblem(RTP_BACKDROPONLY, bd);
    if (bd.#found_in > WORDSIZE) {
        address = bd.&found_in;
        address-->0 = D;
    } else bd.found_in = D;
    give bd ~absent;
    MoveFloatingObjects();
];
```

§17. Wearing Clothes. An object X is worn by a person P if and only if (i) the object tree `parent` of X is P, and (ii) X has the `worn` attribute. In fact for I7 purposes we are careful to ensure that (ii) happens only when (i) does in any case: if X is moved in the object tree, or made a part of something, or removed from play, then `worn` is removed.

```
[ WearObject X P opt;
    if (X == false) rfalse;
    if (X notin P) MoveObject(X, P, opt);
    give X worn;
];
```

§18. Map Connections. `MapConnection` returns the room which is in the given direction (as an object) from the given room: it is used, among other things, to test the "mapped east of" and similar relations. (The same relations are asserted true with `AssertMapConnection` and false with `AssertMapUnconnection`.)

`RoomOrDoorFrom` returns either the room or door which is in the given direction, and is thus simpler (since it doesn't have to investigate what is through such a door).

Both routines return values which are type-safe in I7 provided the kind of value they are assigned to is "object": neither returns any specific kind of object without fail. (`MapConnection` is always either a door or `nothing`, but `nothing` is not a typesafe value for "door".)

Note that map connections via doors are immutable.

```
[ MapConnection from_room dir
    in_direction through_door;
    if ((from_room ofclass K1_room) && (dir ofclass K3_direction)) {
        in_direction = Map_Storage-->
            ((from_room.IK1_Count)*No_Directions + dir.IK3_Count);
        if (in_direction ofclass K1_room) return in_direction;
        if (in_direction ofclass K4_door) {
            @push location;
            location = from_room;
            through_door = in_direction.door_to();
            @pull location;
            if (through_door ofclass K1_room) return through_door;
        }
    }
    return nothing;
];
[ DoorFrom obj dir rv;
    rv = RoomOrDoorFrom(obj, dir);
    if (rv ofclass K4_door) return rv;
    return nothing;
];
[ RoomOrDoorFrom obj dir use_doors in_direction sl through_door;
    if ((obj ofclass K1_room) && (dir ofclass K3_direction)) {
        in_direction = Map_Storage-->
            ((obj.IK1_Count)*No_Directions + dir.IK3_Count);
        if (in_direction ofclass K1_room or K4_door) return in_direction;
    }
    return nothing;
];
[ AssertMapConnection r1 dir r2 in_direction;
    SignalMapChange();
    in_direction = Map_Storage-->
        ((r1.IK1_Count)*No_Directions + dir.IK3_Count);
    if ((in_direction == 0) || (in_direction ofclass K1_room)) {
        Map_Storage-->((r1.IK1_Count)*No_Directions + dir.IK3_Count) = r2;
        return;
    }
    if (in_direction ofclass K4_door) {
        RunTimeProblem(RTP_EXITDOOR, r1, dir);
        return;
    }
    RunTimeProblem(RTP_NOEXIT, r1, dir);
];
[ AssertMapUnconnection r1 dir r2 in_direction;
    SignalMapChange();
    in_direction = Map_Storage-->
        ((r1.IK1_Count)*No_Directions + dir.IK3_Count);
    if (r1 ofclass K4_door) {
        RunTimeProblem(RTP_EXITDOOR, r1, dir);
        return;
    }
    if (in_direction == r2)
        Map_Storage-->((r1.IK1_Count)*No_Directions + dir.IK3_Count) = 0;
    return;
];
```

§**19. Adjacency Relation.** A relation between two rooms which, note, does not see connections through doors.

```
[ TestAdjacency R1 R2 i row;
    row = (R1.IK1_Count)*No_Directions;
    for (i=0: i<No_Directions: i++, row++)
        if (Map_Storage-->row == R2) rtrue;
    rfalse;
];
```

§**20. Regional Containment Relation.** This tests whether an object with a definite physical location is in a given region. (For a two-sided door which straddles regions, the front side is what counts; for a backdrop, a direction or another region, the answer is always no.) We rely on the fact that every room object has a `map_region` property which is the smallest region containing it, if any does, and `nothing` otherwise; region objects are then in the object tree such that parenthood corresponds to spatial containment. (Note that in I7, a region must either completely contain another region, or else have no overlap with it.)

```
[ TestRegionalContainment obj region o;
    if ((obj == nothing) || (region == nothing)) rfalse;
    if (~~(obj ofclass K1_room)) obj = LocationOf(obj);
    if (obj == nothing) rfalse;
    o = obj.map_region;
    while (o) {
        if (o == region) rtrue;
        o = parent(o);
    }
    rfalse;
];
```

§21. Doors. There are two sorts of door: one-sided and two-sided.

A two-sided door is in two rooms at once: one is called the front side, the other the back side. The front side is the one declared first in the source. Note that a one-sided door is also an I6 object of class `K4_door`, and then the front side is the room holding it, while the back side is `nothing`.

The `door_to` property calculates the room which the door leads to; that of course depends on which side of it the player is standing. Similarly, `door_dir` calculates the direction it leads in. Unlike the I6 setup, where `door_dir` returned the direction property (`n_to`, `s_to`, etc.), here in I7's template it returns the direction object (`n_obj`, `s_obj`, etc.)

```
[ FrontSideOfDoor D; if (~~(D ofclass K4_door)) rfalse;
    if (D provides found_in) return (D.&found_in)-->0; ! Two-sided
    return parent(D); ! One-sided
];
[ BackSideOfDoor D; if (~~(D ofclass K4_door)) rfalse;
    if (D provides found_in) return (D.&found_in)-->1; ! Two-sided
    return nothing; ! One-sided
];
[ OtherSideOfDoor D from_room rv;
    if (D ofclass K4_door) {
        @push location;
        location = LocationOf(from_room);
        rv = D.door_to();
        @pull location;
    }
    return rv;
];
[ DirectionDoorLeadsIn D from_room rv dir;
    if (D ofclass K4_door) {
        @push location;
        location = LocationOf(from_room);
        rv = D.door_dir();
        @pull location;
    }
    return rv;
];
```

§22. Visibility Relation. We use `TestScope` to decide whether there is a line of sight from A to B; it's a relation which cannot be asserted true or false.

```
[ TestVisibility A B;
    if (~~OffersLight(parent(CoreOf(A)))) rfalse;
    if (suppress_scope_loops) rtrue;
    return TestScope(B, A);
];
```

§23. Touchability Relation. We use `ObjectIsUntouchable` to decide whether there is physical access from A to B; it's a relation which cannot be asserted true or false.

```
[ TestTouchability A B;
    if (TestScope(B,A) == false) rfalse;
    if (ObjectIsUntouchable(B, 1, 0, A)) rfalse;
    rtrue;
];
```

§24. Concealment Relation. An activity determines whether one object conceals another; it's a relation which cannot be asserted true or false.

```
[ TestConcealment A B;
    if (A ofclass K2_thing && B ofclass K2_thing) {
        particular_possession = B;
        if (CarryOutActivity(DECIDING_CONCEALED_POSSESS_ACT, A)) rtrue;
    }
    rfalse;
];
```

Light Template B/light

Purpose

The determination of light, visibility and physical access.

§1. Darkness. "Darkness" is not really a place: but in I6 it has to be an object so that the location-name on the status line can be "Darkness". In I7 we use it as little as possible: note that it has no properties.

```
Object thedark "(darkness object)";
```

§2. Light Measurement. These two routines, `OffersLight` and `HasLightSource`, are largely unchanged from their I6 definitions; see the *Inform Designer's Manual*, 4th edition, for a commentary. In terms of how they are used in I7, `OffersLight` is called only by the two rules below; `HasLightSource` is called also in determining the scope, that is, what is visible to the player.

```
[ OffersLight obj j;
    while (obj) {
        if (obj has light) rtrue;
        objectloop (j in obj) if (HasLightSource(j)) rtrue;
        if ((obj has container) && (obj hasnt open) && (obj hasnt transparent)) rfalse;
        if ((obj provides component_parent) && (obj.component_parent))
            obj = obj.component_parent;
        else
            obj = parent(obj);
    }
    rfalse;
];
[ HasLightSource i j ad sr po;
    if (i == 0) rfalse;
    if (i has light) rtrue;
    if ((IsSeeThrough(i)) && (~~(HidesLightSource(i))))
        objectloop (j in i)
            if (HasLightSource(j)) rtrue;
    ad = i.&add_to_scope;
    if (parent(i) ~= 0 && ad ~= 0) {
        if (metaclass(ad-->0) == Routine) {
            ats_hls = 0; ats_flag = 1;
            sr = scope_reason; po = parser_one;
            scope_reason = LOOPOVERSCOPE_REASON; parser_one = 0;
            RunRoutines(i, add_to_scope);
            scope_reason = sr; parser_one = po;
            ats_flag = 0; if (ats_hls == 1) rtrue;
        }
        else {
            for (j=0 : (WORDSIZE*j)<i.#add_to_scope : j++)
                if ((ad-->j) && (HasLightSource(ad-->j) == 1)) rtrue;
        }
    }
    if (ComponentHasLight(i)) rtrue;
    rfalse;
];
[ ComponentHasLight o obj next_obj;
    if (o provides component_child) {
        obj = o.component_child;
        while (obj) {
            next_obj = obj.component_sibling;
            if (obj has light) rtrue;
            if (HasLightSource(obj)) rtrue;
            if ((obj provides component_child) && (ComponentHasLight(obj))) rtrue;
            obj = next_obj;
        }
    }
    rfalse;
];
[ HidesLightSource obj;
    if (obj == player) rfalse;
    if (obj has transparent or supporter) rfalse;
    if (obj has animate) rfalse;
    if (obj has container) return (obj hasnt open);
    return (obj hasnt enterable);
];
```

§**3. Invariant.** The following routines maintain two variables about the light condition of the player:

(a) If on the most recent check the player was in light, then `location` equals `real_location` and `lightflag` is false.

(b) If on the most recent check the player was in darkness, then `location` equals `thedark` and `lightflag` is true.

Note that they are not allowed to alter `real_location`, whose definition has nothing to do with light.

```
Global lightflag = false;
```

§**4. Adjust Light Rule.** This rule fires at least once a turn, and more often when the player moves location, since that's likely to invalidate any previous assumptions. It compares the state of light now with the last time it ran, and gives instructions on what to do in each of the four possibilities.

```
[ ADJUST_LIGHT_R previous_light_condition;
    previous_light_condition = lightflag;
    lightflag = OffersLight(parent(player));
    if ((previous_light_condition == false) && (lightflag == false)) {
        location = thedark;
        rfalse;
    }
    if ((previous_light_condition == false) && (lightflag == true)) {
        location = real_location;
        CarryOutActivity(PRINTING_NEWS_OF_LIGHT_ACT);
        rfalse;
    }
    if ((previous_light_condition == true) && (lightflag == false)) {
        location = thedark;
        DivideParagraphPoint();
        BeginActivity(PRINTING_NEWS_OF_DARKNESS_ACT);
        if (ForActivity(PRINTING_NEWS_OF_DARKNESS_ACT) == false) L__M(##Miscellany, 9);
        EndActivity(PRINTING_NEWS_OF_DARKNESS_ACT);
        rfalse;
    }
    if ((previous_light_condition == true) && (lightflag == true)) {
        location = real_location;
        rfalse;
    }
    rfalse;
];
```

§**5. Silent Light Consideration.** The Adjust Light Rule makes a fuss when light changes: it prints messages, for instance. This rule is silent instead, and simply does the minimum necessary to maintain the light invariant. It is used in only four circumstances:

(a) To determine the initial light condition at start of play.
(b) When the player moves from one room to another via the "going" action.
(c) When the player moves via `PlayerTo`, which is used by "now the player is in ...".
(d) When the player changes from one persona to another.

Perhaps case (b) is surprising. Why not simply use the adjust light rule, as we do on an instance of "move player to ...", for instance? The answer is that the going action is just about to print details of the new location anyway, so it would be redundant to have the adjust light rule print out details as well.

```
[ SilentlyConsiderLight;
    lightflag = OffersLight(parent(player));
    if (lightflag) location = real_location; else location = thedark;
    rfalse;
];
```

§6. Translucency. `IsSeeThrough` is used at various places: roughly speaking, it determines whether `obj` being in scope means that the object-tree contents of `obj` are in scope.

```
[ IsSeeThrough obj;
    if ((obj has supporter)
        || (obj has transparent)
        || (obj has animate)
        || ((obj has container) && (obj has open)))
        rtrue;
rfalse;
];
```

§7. Visibility Parent. The idea of `VisibilityParent` is that it takes us from a given position in the object tree, `o`, to the next visible position above. Note that

(1) A container has an inside and an outside: this routine calculates from the "inside of `o`", which is why it returns `nothing` from an opaque closed container;

(2) Component parts are (for purposes of this routine) attached to the outside surface of a container, so that if `o` is part of a closed opaque container then the visibility parent of `o` is its actual parent.

```
[ VisibilityParent o;
    if (o && (o has container) && (o hasnt open) && (o hasnt transparent)) return nothing;
    if (o) o = CoreOfParentOfCoreOf(o);
    return o;
];
```

§8. Find Visibility Levels. The following routine sets the pair of variables `visibility_ceiling`, the highest visible point in the object tree above the player – or `thedark` if the player cannot see at all – and `visibility_levels`, the number of steps of `VisibilityParent` needed to reach this ceiling; or 0 if the player cannot see at all.

```
[ FindVisibilityLevels lc up;
    if (location == thedark) {
        visibility_ceiling = thedark;
        visibility_levels = 0;
    } else {
        visibility_ceiling = player;
        while (true) {
            up = VisibilityParent(visibility_ceiling);
            if (up == 0) break;
            visibility_ceiling = up;
            lc++;
        }
        visibility_levels = lc;
    }
];
```

§9. Scope Ceiling. Scope is almost the same thing as visibility, but not quite, and the following routine does not quite duplicate the calculation of `FindVisibilityLevels`. The difference arises in the first step, where we take the `parent` of `pos`, not the core of the `parent` of the core of `pos`: this makes a difference if `pos` is inside a container which is itself part of something else.

```
[ ScopeCeiling pos c;
    if (pos == player && location == thedark) return thedark;
    c = parent(pos);
    if (c == 0) return pos;
    while (VisibilityParent(c)) c = VisibilityParent(c);
    return c;
];
```

§10. Object Is Untouchable. The following routine imitates the I6 library one of the same name, but works instead by delegating the decision to the accessibility rulebook.

It is not easy to answer the question of whether someone other than the player, but in another room, can touch a multiply-present object (a two-sided door or a backdrop) and we err on the side of caution by saying yes in such cases. The question would only be asked in the case of a "try" action deliberately caused by the author, or by an instruction made by the player to someone not in the same room: in the first case, we will assume that the author knows what he is doing, and in the second case, the circumstances in which saying yes would be a wrong call are *highly* improbable.

```
[ ObjectIsUntouchable item silent_flag flag2 p save_sp decision;
    if ((p ~= player) && (LocationOf(p) ~= LocationOf(player)) &&
        ((item ofclass K4_door) || (item ofclass K7_backdrop))) {
        decision = false;
    } else {
        untouchable_object = item; untouchable_silence = silent_flag;
        touch_persona = p; if (p == actor) touch_persona = 0;
        save_sp = say__p; say__p = 0;
        if (ProcessRulebook(ACCESSIBILITY_RB, 0, true)) {
            if (RulebookSucceeded()) decision = false;
            else decision = true;
        } else decision = false;
        if (say__p == false) say__p = save_sp;
    }
    untouchable_silence = 0;
    return decision;
];
```

§11. Access Through Barriers Rule.

```
[ ACCESS_THROUGH_BARRIERS_R ancestor i j external p;
    p = touch_persona; if (p == 0) p = actor;
    ancestor = CommonAncestor(p, untouchable_object);
    if ((ancestor == 0) && (LocationOf(untouchable_object) == nothing)
        && ((untouchable_object ofclass K4_door or K7_backdrop) == false)) {
        if (touch_persona == 0) GL__M(##Take,8,untouchable_object);
        RulebookFails();
        rtrue;
    }
    ! First, a barrier between the player and the ancestor.
    if (CoreOf(p) ~= ancestor) {
        i = parent(CoreOf(p)); j = CoreOf(i); external = false;
        if (j ~= i) { i = j; external = true; }
        while (i~=ancestor && i) {
            if ((external == false)
                && (ProcessRulebook(REACHING_OUTSIDE_RB, i))
                && (RulebookFailed())) rtrue; ! Barrier
            i = parent(CoreOf(i)); j = CoreOf(i); external = false;
            if (j ~= i) { i = j; external = true; }
        }
    }
    ! Second, a barrier between the item and the ancestor.
    if (CoreOf(untouchable_object) ~= ancestor) {
        ! We can always get to the core of the item.
        i = CoreOf(untouchable_object);
        ! This will be on the inside of its parent, if its parent is a
        ! container, so there should be no exemption.
        i = parent(i); external = false;
        ! j = CoreOf(i); if (j ~= i) { i = j; external = true; }
        while (i~=ancestor && i) {
            if ((external == false) &&
                (ProcessRulebook(REACHING_INSIDE_RB, i)) &&
                (RulebookFailed())) rtrue; ! Barrier
            i = CoreOf(i);
            if (i == ancestor) break;
            i = parent(i); j = CoreOf(i); external = false;
            if (j ~= i) { i = j; external = true; }
        }
    }
    RulebookSucceeds(); ! No barrier
    rtrue;
];
```

§12. Can't Reach Inside Closed Containers Rule.

```
[ CANT_REACH_INSIDE_CLOSED_R;
    if (parameter_object has container && parameter_object hasnt open) {
        if (touch_persona == 0) GL__M(##Take,9,parameter_object);
        RulebookFails(); rtrue;
    }
    rfalse;
];
```

§13. Can't Reach Outside Closed Containers Rule.

```
[ CANT_REACH_OUTSIDE_CLOSED_R;
    if (parameter_object has container && parameter_object hasnt open) {
        if (touch_persona == 0) GL__M(##Take,9,parameter_object);
        RulebookFails(); rtrue;
    }
    rfalse;
];
```

§14. Can't Reach Inside Rooms Rule.

```
[ CANT_REACH_INSIDE_ROOMS_R;
    if (parameter_object && parameter_object ofclass K1_room) {
        if (touch_persona == 0) GL__M(##Take,14,parameter_object);
        RulebookFails(); rtrue;
    }
    rfalse;
];
```

Tests Template B/testt

Purpose

The command grammar and I6 implementation for testing commands such as TEST, ACTIONS and PURLOIN.

§1. Abstract Command. The code below is compiled only if the symbol `DEBUG` is defined, which it always is for normal runs in the Inform user interface, but not for Release runs.

Not all of these commands are documented; this is intentional. They may be changed in name or function. This is all of the testing commands except for the GLKLIST command, which is in Glulx.i6t (and does not exist when the target VM is the Z-machine).

We take the commands in alphabetical order, beginning with ABSTRACT, which moves an object to a new position in the object tree.

```
[ XAbstractSub;
    if (XTestMove(noun, second)) return;
    move noun to second;
    "[Abstracted.]";
];
[ XTestMove obj dest;
    if ((obj <= InformLibrary) || (obj == LibraryMessages))
        "[Can't move ", (name) obj, ": it's a system object.]";
    if (obj.component_parent)
        "[Can't move ", (name) obj, ": it's part of ",
        (the) obj.component_parent, ".]";
    while (dest) {
        if (dest == obj) "[Can't move ", (name) obj, ": it would contain itself.]";
        dest = CoreOfParentOfCoreOf(dest);
    }
    rfalse;
];
```

§2. Actions Command. ACTIONS turns tracing of actions on.

```
[ ActionsOnSub; trace_actions = 1; say__p = 1; "Actions listing on."; ];
[ ActionsOffSub; trace_actions = 0; say__p = 1; "Actions listing off."; ];
```

§3. Gonear Command. GONEAR teleports the player to the vicinity of some named item.

```
[ GonearSub;
    PlayerTo(LocationOf(noun));
];
```

§**4. Purloin Command.** To PURLOIN is to acquire something without reference to any rules on accessibility.

```
[ XPurloinSub;
    if (XTestMove(noun, player)) return;
    move noun to player; give noun moved ~concealed;
    say__p = 1;
    "[Purloined.]";
];
```

§**5. Random Command.** RANDOM forces the random-number generator to a predictable seed value.

```
[ PredictableSub;
    VM_Seed_RNG(-100);
    say__p = 1;
    "[Random number generator now predictable.]";
];
```

§**6. Relations Command.** RELATIONS lists the current state of the mutable relations.

```
[ ShowRelationsSub;
IterateRelations(ShowOneRelation);
];
[ ShowOneRelation rel;
if ((rel-->RR_PERMISSIONS) & (RELS_SHOW)) {
    (rel-->RR_HANDLER)(rel, RELS_SHOW);
}
];
```

§**7. Rules Command.** RULES changes the level of rule tracing.

```
[ RulesOnSub;
    debug_rules = 1; say__p = 1;
    "Rules tracing now switched on. Type ~rules off~ to switch it off again,
    or ~rules all~ to include even rules which do not apply.";
];
[ RulesAllSub;
    debug_rules = 2; say__p = 1;
    "Rules tracing now switched to ~all~. Type ~rules off~ to switch it off again.";
];
[ RulesOffSub;
    debug_rules = 0; say__p = 1;
    "Rules tracing now switched off. Type ~rules~ to switch it on again.";
];
```

§**8. Scenes Command.** SCENES switches scene-change tracing on or off, and also shows the current position.

```
[ ScenesOnSub;
    debug_scenes = 1;
    ShowSceneStatus(); say__p = 1;
    "(Scene monitoring now switched on. Type ~scenes off~ to switch it off again.)";
];
[ ScenesOffSub;
    debug_scenes = 0; say__p = 1;
    "(Scene monitoring now switched off. Type ~scenes~ to switch it on again.)";
];
```

§**9. Scope Command.** SCOPE prints a numbered list of all objects in scope to the player.

```
Global x_scope_count;
[ ScopeSub;
    x_scope_count = 0;
    LoopOverScope(Print_ScL, noun);
    if (x_scope_count == 0) "Nothing is in scope.";
];
[ Print_ScL obj; print_ret ++x_scope_count, ": ", (a) obj, " (", obj, ")"; ];
```

§**10. Showheap Command.** SHOWHEAP is for debugging the memory heap, and is intended for Inform maintainers rather than users.

```
[ ShowHeapSub;
    DebugHeap();
];
```

§**11. ShowMe Command.** SHOWME is probably the most useful testing command: it shows the state of the current room, or a named item.

```
[ ShowMeSub t_0 na;
    t_0 = noun;
    if (noun == nothing) noun = real_location;
    if (ShowMeRecursively(noun, 0, (noun == real_location))) {
        if (noun == real_location)
            print "* denotes things which are not in scope^";
    }
    if (t_0 ofclass K2_thing) {
        print "location:"; ShowRLocation(noun, true); print "^";
    }
    {-call:Plugins::Showme::compile_SHOWME_details}
];
[ ShowRLocation obj top;
    if (obj ofclass K1_room) return;
    print " ";
    if (parent(obj)) {
        if (obj has worn) print "worn by ";
        else {
            if (parent(obj) has animate) print "carried by ";
            if (parent(obj) has container) print "in ";
            if (parent(obj) ofclass K1_room) print "in ";
            if (parent(obj) has supporter) print "on ";
        }
        print (the) parent(obj);
        ShowRLocation(parent(obj));
    } else {
        if (obj.component_parent) {
            if (top == false) print ", which is ";
            print "part of ", (the) obj.component_parent;
            ShowRLocation(obj.component_parent);
        }
        else print "out of play";
    }
];
[ ShowMeRecursively obj depth f c i k;
    spaces(2*depth);
    if (f && (depth > 0) && (TestScope(obj, player) == false)) { print "*"; c = true; }
    print (name) obj;
    if (depth > 0) {
        if (obj.component_parent) print " (part of ", (name) obj.component_parent, ")";
        if (obj has worn) print " (worn)";
    }
    if (obj provides KD_Count) {
        k = KindHierarchy-->((obj.KD_Count)*2);
        if ((k ~= K2_thing) || (depth==0)) {
            print " - ";
            if (k == K4_door or K5_container) {
                if (obj has transparent) print "transparent ";
                if (obj has locked) print "locked ";
                else if (obj has open) print "open ";
                else print "closed ";
            }
            print (I7_Kind_Name) k;
        }
    }
    print "^";
    if (obj.component_child) c = c | ShowMeRecursively(obj.component_child, depth+2, f);
    if ((depth>0) && (obj.component_sibling))
        c = c | ShowMeRecursively(obj.component_sibling, depth, f);
    if (child(obj)) c = c | ShowMeRecursively(child(obj), depth+2, f);
    if ((depth>0) && (sibling(obj))) c = c | ShowMeRecursively(sibling(obj), depth, f);
    return c;
];
[ AllowInShowme pr;
    if (pr == workflag or concealed or mentioned) rfalse;
    rtrue;
];
```

§12. Showverb Command. SHOWVERB is a holdover from old I6 days, but still quite useful. It writes out the I6 command verb grammar for the supplied command.

```
[ ShowVerbSub address lines meta i x;
    wn = 2; x = NextWordStopped();
    if (x == 0 || ((x->#dict_par1) & 1) == 0)
        "Try typing ~showverb~ and then the name of a verb.";
    meta = ((x->#dict_par1) & 2)/2;
    i = DictionaryWordToVerbNum(x);
    address = VM_CommandTableAddress(i);
    lines = address->0;
    address++;
    print "Verb ";
    if (meta) print "meta ";
    VM_PrintCommandWords(i);
    new_line;
    if (lines == 0) "has no grammar lines.";
    for (: lines>0 : lines--) {
        address = UnpackGrammarLine(address);
        print "    "; DebugGrammarLine(); new_line;
    }
    ParaContent();
];
[ DebugGrammarLine pcount;
    print " * ";
    for (: line_token-->pcount ~= ENDIT_TOKEN : pcount++) {
        if ((line_token-->pcount)->0 & $10) print "/ ";
        print (DebugToken) line_token-->pcount, " ";
    }
    print "-> ", (DebugAction) action_to_be;
    if (action_reversed) print " reverse";
];
[ DebugToken token;
    AnalyseToken(token);
    switch (found_ttype) {
    ILLEGAL_TT:
        print "<illegal token number ", token, ">";
    ELEMENTARY_TT:
        switch (found_tdata) {
        NOUN_TOKEN:          print "noun";
        HELD_TOKEN:          print "held";
        MULTI_TOKEN:         print "multi";
        MULTIHELD_TOKEN:     print "multiheld";
        MULTIEXCEPT_TOKEN:   print "multiexcept";
        MULTIINSIDE_TOKEN:   print "multiinside";
        CREATURE_TOKEN:      print "creature";
        SPECIAL_TOKEN:       print "special";
        NUMBER_TOKEN:        print "number";
        TOPIC_TOKEN:         print "topic";
        ENDIT_TOKEN:         print "END";
        }
    PREPOSITION_TT:
        print "’", (address) found_tdata, "’";
    ROUTINE_FILTER_TT:
        print "noun=Routine(", found_tdata, ")";
    ATTR_FILTER_TT:
        print (DebugAttribute) found_tdata;
    SCOPE_TT:
        print "scope=Routine(", found_tdata, ")";
    GPR_TT:
        print "Routine(", found_tdata, ")";
    }
];
```

§13. Test Command. TEST runs a short script of commands from the source text.

```
#Iftrue ({-value:NUMBER_CREATED(test_scenario)} > 0);
[ TestScriptSub;
    switch(special_word) {
{-call:Plugins::Parsing::TestScripts::compile_switch}
    default:
        print ">--> The following tests are available:^";
{-call:Plugins::Parsing::TestScripts::compile_printout}
    }
];
#ifdef TARGET_GLULX;
Constant TEST_STACK_SIZE = 128;
#ifnot;
Constant TEST_STACK_SIZE = 48;
#endif;
Array test_stack --> TEST_STACK_SIZE;
Global test_sp = 0;
[ TestStart T R l k;
    if (test_sp >= TEST_STACK_SIZE) ">--> Testing too many levels deep";
    test_stack-->test_sp = T;
    test_stack-->(test_sp+1) = 0;
    test_stack-->(test_sp+3) = l;
    test_sp = test_sp + 4;
    if ((R-->0) && (R-->0 ~= real_location)) {
        print "(first moving to ", (name) R-->0, ")^";
        PlayerTo(R-->0, 1);
    }
    k=1;
    while (R-->k) {
        if (R-->k notin player) {
            print "(first acquiring ", (the) R-->k, ")^";
            move R-->k to player;
        }
        k++;
    }
    print "(Testing.)^"; say__p = 1;
];
[ TestKeyboardPrimitive a_buffer a_table p i j l spaced ch;
    if (test_sp == 0) {
        test_stack-->2 = 1;
        return VM_ReadKeyboard(a_buffer, a_table);
    }
    else {
        p = test_stack-->(test_sp-4);
        i = test_stack-->(test_sp-3);
        l = test_stack-->(test_sp-1);
        print "[";
        print test_stack-->2;
        print "] ";
        test_stack-->2 = test_stack-->2 + 1;
        style bold;
        while ((i < l) && (p->i ~= '/')) {
            ch = p->i;
            if (spaced || (ch ~= ' ')) {
                if ((p->i == '[') && (p->(i+1) == '/') && (p->(i+2) == ']')) {
                    ch = '/'; i = i+2;
                }
                a_buffer->(j+WORDSIZE) = ch;
                print (char) ch;
                i++; j++;
                spaced = true;
            } else i++;
        }
        style roman;
        print "^";
        #ifdef TARGET_ZCODE;
        a_buffer->1 = j;
        #ifnot; ! TARGET_GLULX
        a_buffer-->0 = j;
        #endif;
        VM_Tokenise(a_buffer, a_table);
        if (p->i == '/') i++;
        if (i >= l) {
            test_sp = test_sp - 4;
        } else test_stack-->(test_sp-3) = i;
    }
];
#IFNOT;
[ TestScriptSub;
    ">--> No test scripts exist for this game.";
];
#ENDIF;
```

§14. Trace Command. Another holdover from I6: TRACE sets the level of parser tracing, on a scale of 0 (off, the default) to 5.

```
[ TraceOnSub; parser_trace=1; say__p = 1; "[Trace on.]"; ];
[ TraceLevelSub;
    parser_trace = parsed_number; say__p = 1;
    print "[Parser tracing set to level ", parser_trace, ".]^";
];
[ TraceOffSub; parser_trace=0; say__p = 1; "Trace off."; ];
```

§15. Tree Command. TREE prints out the I6 object tree, though this is not always very helpful in I7 terms. It should arguably be withdrawn, but doesn't seem to do any harm.

```
[ XTreeSub i;
    if (noun == 0) {
        objectloop (i)
            if (i ofclass Object && parent(i) == 0) XObj(i);
    }
    else XObj(noun,1);
];
[ XObj obj f;
    if (parent(obj) == 0) print (name) obj; else print (a) obj;
    print " (", obj, ") ";
    if (f == 1 && parent(obj) ~= 0)
        print "(in ", (name) parent(obj), " ", parent(obj), ")";
    new_line;
    if (child(obj) == 0) rtrue;
    if (obj == Class)
        WriteListFrom(child(obj), NEWLINE_BIT+INDENT_BIT+ALWAYS_BIT+NOARTICLE_BIT, 1);
    else
        WriteListFrom(child(obj), NEWLINE_BIT+INDENT_BIT+ALWAYS_BIT+FULLINV_BIT, 1);
];
```

§16. Grammar. In the old I6 parser, testing commands had their own scope hardwired in to the code: this worked by comparing the verb command word directly against `'scope'` and the like. That would go wrong if the testing commands were translated into other languages, and was a crude design at best. The following scope token is better: using this token instead of `multi` provides a noun with universal scope (but restricted to I7 objects, so I6 pseudo-objects like `compass` are not picked up) and able to accept multiple objects.

```
[ testcommandnoun obj o2;
    switch (scope_stage) {
        1: rtrue; ! allow multiple objects
        2: objectloop (obj)
            if ((obj ofclass Object) && (obj provides KD_Count))
                PlaceInScope(obj, true);
        3: print "There seems to be no such object anywhere in the model world.^";
    }
];
{-testing-command:abstract}
    * scope=testcommandnoun 'to' scope=testcommandnoun -> XAbstract;
```

```
{-testing-command:actions}
    *                                         -> ActionsOn
    * 'on'                                    -> ActionsOn
    * 'off'                                   -> ActionsOff;
{-testing-command:gonear}
    * scope=testcommandnoun                   -> Gonear;
{-testing-command:purloin}
    * scope=testcommandnoun                   -> XPurloin;
{-testing-command:random}
    *                                         -> Predictable;
{-testing-command:relations}
    *                                         -> ShowRelations;
{-testing-command:rules}
    *                                         -> RulesOn
    * 'all'                                   -> RulesAll
    * 'on'                                    -> RulesOn
    * 'off'                                   -> RulesOff;
{-testing-command:scenes}
    *                                         -> ScenesOn
    * 'on'                                    -> ScenesOn
    * 'off'                                   -> ScenesOff;
{-testing-command:scope}
    *                                         -> Scope
    * scope=testcommandnoun                   -> Scope;
{-testing-command:showheap}
    *                                         -> ShowHeap;
{-testing-command:showme}
    *                                         -> ShowMe
    * scope=testcommandnoun                   -> ShowMe;
{-testing-command:showverb}
    * special                                 -> Showverb;
{-testing-command:test}
    *                                         -> TestScript
    * special                                 -> TestScript;
{-testing-command:trace}
    *                                         -> TraceOn
    * number                                  -> TraceLevel
    * 'on'                                    -> TraceOn
    * 'off'                                   -> TraceOff;
{-testing-command:tree}
    *                                         -> XTree
    * scope=testcommandnoun                   -> XTree;
```

Language Template B/langt

Purpose

The fundamental definitions needed by the parser and the verb library in order to specify the language of play – that is, the language used for communications between the story file and the player.

§1. Vocabulary.

```
Constant AGAIN1__WD     = 'again';
Constant AGAIN2__WD     = 'g//';
Constant AGAIN3__WD     = 'again';
Constant OOPS1__WD      = 'oops';
Constant OOPS2__WD      = 'o//';
Constant OOPS3__WD      = 'oops';
Constant UNDO1__WD      = 'undo';
Constant UNDO2__WD      = 'undo';
Constant UNDO3__WD      = 'undo';

Constant ALL1__WD       = 'all';
Constant ALL2__WD       = 'each';
Constant ALL3__WD       = 'every';
Constant ALL4__WD       = 'everything';
Constant ALL5__WD       = 'both';
Constant AND1__WD       = 'and';
Constant AND2__WD       = 'and';
Constant AND3__WD       = 'and';
Constant BUT1__WD       = 'but';
Constant BUT2__WD       = 'except';
Constant BUT3__WD       = 'but';
Constant ME1__WD        = 'me';
Constant ME2__WD        = 'myself';
Constant ME3__WD        = 'self';
Constant OF1__WD        = 'of';
Constant OF2__WD        = 'of';
Constant OF3__WD        = 'of';
Constant OF4__WD        = 'of';
Constant OTHER1__WD     = 'another';
Constant OTHER2__WD     = 'other';
Constant OTHER3__WD     = 'other';
Constant THEN1__WD      = 'then';
Constant THEN2__WD      = 'then';
Constant THEN3__WD      = 'then';

Constant NO1__WD        = 'n//';
Constant NO2__WD        = 'no';
Constant NO3__WD        = 'no';
Constant YES1__WD       = 'y//';
Constant YES2__WD       = 'yes';
Constant YES3__WD       = 'yes';

Constant AMUSING__WD    = 'amusing';
Constant FULLSCORE1__WD = ’fullscore’;
Constant FULLSCORE2__WD = ’full’;
Constant QUIT1__WD      = ’q//’;
Constant QUIT2__WD      = ’quit’;
Constant RESTART__WD    = ’restart’;
Constant RESTORE__WD    = ’restore’;
```

§2. Pronouns.

```
Array LanguagePronouns table

! word        possible GNAs                   connected
!             to follow:                      to:
!             a     i
!             s  p  s  p
!             mfnmfnmfnmfn

    ’it’      $$001000111000                    NULL
    ’him’     $$100000000000                    NULL
    ’her’     $$010000000000                    NULL
    ’them’    $$000111000111                    NULL;
```

§3. Descriptors.

```
Array LanguageDescriptors table

! word        possible GNAs   descriptor      connected
!             to follow:      type:           to:
!             a     i
!             s  p  s  p
!             mfnmfnmfnmfn

    ’my’      $$111111111111    POSSESS_PK      0
    ’this’    $$111111111111    POSSESS_PK      0
    ’these’   $$000111000111    POSSESS_PK      0
    ’that’    $$111111111111    POSSESS_PK      1
    ’those’   $$000111000111    POSSESS_PK      1
    ’his’     $$111111111111    POSSESS_PK      ’him’
    ’her’     $$111111111111    POSSESS_PK      ’her’
    ’their’   $$111111111111    POSSESS_PK      ’them’
    ’its’     $$111111111111    POSSESS_PK      ’it’
    ’the’     $$111111111111    DEFART_PK       NULL
    ’a//’     $$111000111000    INDEFART_PK     NULL
    ’an’      $$111000111000    INDEFART_PK     NULL
    ’some’    $$000111000111    INDEFART_PK     NULL
    ’lit’     $$111111111111    light           NULL
    ’lighted’ $$111111111111    light           NULL
    ’unlit’   $$111111111111    (-light)        NULL;
```

§4. Numbers.

```
Array LanguageNumbers table
    'one' 1 'two' 2 'three' 3 'four' 4 'five' 5
    'six' 6 'seven' 7 'eight' 8 'nine' 9 'ten' 10
    'eleven' 11 'twelve' 12 'thirteen' 13 'fourteen' 14 'fifteen' 15
    'sixteen' 16 'seventeen' 17 'eighteen' 18 'nineteen' 19 'twenty' 20
    'twenty-one' 21 'twenty-two' 22 'twenty-three' 23 'twenty-four' 24
    'twenty-five' 25 'twenty-six' 26 'twenty-seven' 27 'twenty-eight' 28
    'twenty-nine' 29 'thirty' 30
;
[ LanguageNumber n f;
    if (n == 0)    { print "zero"; rfalse; }
    if (n < 0)     { print "minus "; n = -n; }
#Iftrue (WORDSIZE == 4);
    if (n >= 1000000000) {
        if (f == 1) print ", ";
        print (LanguageNumber) n/1000000000, " million"; n = n%1000000000; f = 1;
    }
    if (n >= 1000000) {
        if (f == 1) print ", ";
        print (LanguageNumber) n/1000000, " million"; n = n%1000000; f = 1;
    }
#Endif;
    if (n >= 1000) {
        if (f == 1) print ", ";
        print (LanguageNumber) n/1000, " thousand"; n = n%1000; f = 1;
    }
    if (n >= 100)  {
        if (f == 1) print ", ";
        print (LanguageNumber) n/100, " hundred"; n = n%100; f = 1;
    }
    if (n == 0) rfalse;
    #Ifdef DIALECT_US;
    if (f == 1) print " ";
    #Ifnot;
    if (f == 1) print " and ";
    #Endif;
    switch (n) {
    1:    print "one";
    2:    print "two";
    3:    print "three";
    4:    print "four";
    5:    print "five";
    6:    print "six";
    7:    print "seven";
    8:    print "eight";
    9:    print "nine";
    10:   print "ten";
    11:   print "eleven";
    12:   print "twelve";
    13:   print "thirteen";
    14:   print "fourteen";
    15:   print "fifteen";
    16:   print "sixteen";
    17:   print "seventeen";
    18:   print "eighteen";
    19:   print "nineteen";
    20 to 99: switch (n/10) {
        2:  print "twenty";
        3:  print "thirty";
        4:  print "forty";
        5:  print "fifty";
        6:  print "sixty";
        7:  print "seventy";
        8:  print "eighty";
        9:  print "ninety";
        }
        if (n%10 ~= 0) print "-", (LanguageNumber) n%10;
    }
];
```

§5. Time.

```
[ LanguageTimeOfDay hours mins i;
    i = hours%12;
    if (i == 0) i = 12;
    if (i < 10) print " ";
    print i, ":", mins/10, mins%10;
    if ((hours/12) > 0) print " pm"; else print " am";
];
```

§6. Directions.

```
[ LanguageDirection d;
    print (name) d;
];
```

§7. Translation.

```
[ LanguageToInformese; ];
```

§8. Articles.

```
Constant LanguageAnimateGender   = male;
Constant LanguageInanimateGender = neuter;
Constant LanguageContractionForms = 2;     ! English has two:
                                          ! 0 = starting with a consonant
                                          ! 1 = starting with a vowel
[ LanguageContraction text;
    if (text->0 == 'a' or 'e' or 'i' or 'o' or 'u'
                or 'A' or 'E' or 'I' or 'O' or 'U') return 1;
    return 0;
];
Array LanguageArticles -->
!   Contraction form 0:     Contraction form 1:
!   Cdef   Def    Indef     Cdef   Def    Indef
    "The " "the " "a "      "The " "the " "an "          ! Articles 0
    "The " "the " "some "   "The " "the " "some ";       ! Articles 1
                !             a           i
                !             s     p     s     p
                !             m f n m f n m f n m f n
Array LanguageGNAsToArticles --> 0 0 0 1 1 1 0 0 0 1 1 1;
```

§9. Commands. `LanguageVerbLikesAdverb` is called by `PrintCommand` when printing an `UPTO_PE` error or an inference message. Words which are intransitive verbs, i.e., which require a direction name as an adverb ("walk west"), not a noun ("I only understood you as far as wanting to touch the ground"), should cause the routine to return `true`.

`LanguageVerbMayBeName` is called by `NounDomain` when dealing with the player's reply to a "Which do you mean, the short stick or the long stick?" prompt from the parser. If the reply is another verb (for example, LOOK) then then previous ambiguous command is discarded unless it is one of these words which could be both a verb and an adjective in a `name` property.

```
[ LanguageVerb i;
    switch (i) {
    'i//','inv','inventory':
            print "take inventory";
    'l//':   print "look";
    'x//':   print "examine";
    'z//':   print "wait";
    default: rfalse;
    }
    rtrue;
];
[ LanguageVerbLikesAdverb w;
    if (w == 'look' or 'go' or 'push' or 'walk')
        rtrue;
    rfalse;
];
[ LanguageVerbMayBeName w;
    if (w == 'long' or 'short' or 'normal'
                    or 'brief' or 'full' or 'verbose')
        rtrue;
    rfalse;
];
```

§10. Short Texts.

```
Constant NKEY__TX       = "N = next subject";
Constant PKEY__TX       = "P = previous";
Constant QKEY1__TX      = "  Q = resume game";
Constant QKEY2__TX      = "Q = previous menu";
Constant RKEY__TX       = "RETURN = read subject";
Constant NKEY1__KY      = 'N';
Constant NKEY2__KY      = 'n';
Constant PKEY1__KY      = 'P';
Constant PKEY2__KY      = 'p';
Constant QKEY1__KY      = 'Q';
Constant QKEY2__KY      = 'q';
Constant SCORE__TX      = "Score: ";
Constant MOVES__TX      = "Moves: ";
Constant TIME__TX       = "Time: ";
Global CANTGO__TX     = "You can't go that way.";
Global FORMER__TX     = "your former self";
Global YOURSELF__TX   = "yourself";
Constant YOU__TX        = "You";
Constant DARKNESS__TX   = "Darkness";
Constant THOSET__TX     = "those things";
Constant THAT__TX       = "that";
Constant OR__TX         = " or ";
Constant NOTHING__TX    = "nothing";
Constant NOTHING2__TX    = "Nothing";
Global IS__TX         = " is";
Global ARE__TX        = " are";
Global IS2__TX        = "is ";
Global ARE2__TX       = "are ";
Global IS3__TX        = "is";
Global ARE3__TX       = "are";
Constant AND__TX        = " and ";
#ifdef SERIAL_COMMA;
Constant LISTAND__TX   = ", and ";
Constant LISTAND2__TX  = " and ";
#ifnot;
Constant LISTAND__TX   = " and ";
Constant LISTAND2__TX  = " and ";
#endif; ! SERIAL_COMMA
Constant WHOM__TX       = "whom ";
Constant WHICH__TX      = "which ";
Constant COMMA__TX      = ", ";
```

§11. Printed Inflections.

```
[ ThatorThose obj;      ! Used in the accusative
    if (obj == player)            { print "you"; return; }
    if (obj has pluralname)       { print "those"; return; }
    if (obj has animate) {
        if (obj has female)       { print "her"; return; }
        else
            if (obj hasnt neuter) { print "him"; return; }
    }
    print "that";
];
[ ItorThem obj;
    if (obj == player)            { print "yourself"; return; }
    if (obj has pluralname)       { print "them"; return; }
    if (obj has animate) {
        if (obj has female)       { print "her"; return; }
        else
            if (obj hasnt neuter) { print "him"; return; }
    }
    print "it";
];
[ IsorAre obj;
    if (obj has pluralname || obj == player) print "are"; else print "is";
];
[ HasorHave obj;
    if (obj has pluralname || obj == player) print "have"; else print "has";
];
[ CThatorThose obj;     ! Used in the nominative
    if (obj == player)            { print "You"; return; }
    if (obj has pluralname)       { print "Those"; return; }
    if (obj has animate) {
        if (obj has female)       { print "She"; return; }
        else
            if (obj hasnt neuter) { print "He"; return; }
    }
    print "That";
];
[ CTheyreorThats obj;
    if (obj == player)             { print "You're"; return; }
    if (obj has pluralname)        { print "They're"; return; }
    if (obj has animate) {
        if (obj has female)        { print "She's"; return; }
        else if (obj hasnt neuter) { print "He's"; return; }
    }
    print "That's";
];
[ HisHerTheir o; if (o has pluralname) { print "their"; return; }
    if (o has female) { print "her"; return; }
    if (o has neuter) { print "its"; return; }
    print "his";
];
```

```
[ HimHerItself o; if (o has pluralname) { print "theirselves"; return; }
    if (o has female) { print "herself"; return; }
    if (o has neuter) { print "itself"; return; }
    print "himself";
];
```

§**12. Long Texts.** The messages here are expected eventually to move into I7 tables, where they will be more easily dealt with. But for now, the old-fashioned way:

```
[ LanguageLM n x1 x2;
say__p = 1;
Answer,Ask:
            "There is no reply.";
! Ask:      see Answer
Attack:   "Violence isn't the answer to this one.";
Burn:     "This dangerous act would achieve little.";
Buy:      "Nothing is on sale.";
Climb:    "I don't think much is to be achieved by that.";
Close: switch (n) {
        1:  print_ret (ctheyreorthats) x1, " not something you can close.";
        2:  print_ret (ctheyreorthats) x1, " already closed.";
        3:  "You close ", (the) x1, ".";
        4: print (The) actor, " closes ", (the) x1, ".^";
        5: print (The) x1, " close"; if (x1 hasnt pluralname) print "s";
            print ".^";
    }
Consult: switch (n) {
        1: "You discover nothing of interest in ", (the) x1, ".";
        2: print (The) actor, " looks at ", (the) x1, ".^";
    }
Cut:      "Cutting ", (thatorthose) x1, " up would achieve little.";
Disrobe: switch (n) {
        1:  "You're not wearing ", (thatorthose) x1, ".";
        2:  "You take off ", (the) x1, ".";
        3: print (The) actor, " takes off ", (the) x1, ".^";
    }
Drink:    "There's nothing suitable to drink here.";
Drop: switch (n) {
        1:  if (x1 has pluralname) print (The) x1, " are "; else print (The) x1, " is ";
            "already here.";
        2:  "You haven't got ", (thatorthose) x1, ".";
        3:  print "(first taking ", (the) x1, " off)^"; say__p = 0; return;
        4:  "Dropped.";
        5: "There is no more room on ", (the) x1, ".";
        6: "There is no more room in ", (the) x1, ".";
        7: print (The) actor, " puts down ", (the) x1, ".^";
    }
Eat: switch (n) {
        1:  print_ret (ctheyreorthats) x1, " plainly inedible.";
        2:  "You eat ", (the) x1, ". Not bad.";
        3: print (The) actor, " eats ", (the) x1, ".^";
    }
Enter: switch (n) {
        1:  print "But you're already ";
            if (x1 has supporter) print "on "; else print "in ";
            print_ret (the) x1, ".";
        2:  if (x1 has pluralname) print "They're"; else print "That's";
            print " not something you can ";
            switch (verb_word) {
            'stand':  "stand on.";
            'sit':    "sit down on.";
            'lie':    "lie down on.";
            default:  "enter.";
            }
        3:  "You can't get into the closed ", (name) x1, ".";
        4:  "You can only get into something free-standing.";
        5:  print "You get ";
            if (x1 has supporter) print "onto "; else print "into ";
            print_ret (the) x1, ".";
        6:  print "(getting ";
            if (x1 has supporter) print "off "; else print "out of ";
            print (the) x1; print ")^"; say__p = 0; return;
        7:  ! say__p = 0;
            if (x1 has supporter) "(getting onto ", (the) x1, ")";
            if (x1 has container) "(getting into ", (the) x1, ")";
            "(entering ", (the) x1, ")";
        8: print (The) actor, " gets into ", (the) x1, ".^";
        9:  print (The) actor, " gets onto ", (the) x1, ".^";
    }
Examine: switch (n) {
        1:  "Darkness, noun.  An absence of light to see by.";
        2:  "You see nothing special about ", (the) x1, ".";
        3:  print (The) x1, " ", (isorare) x1, " currently switched ";
            if (x1 has on) "on."; else "off.";
        4: print (The) actor, " looks closely at ", (the) x1, ".^";
        5: "You see nothing unexpected in that direction.";
    }
Exit: switch (n) {
        1:  "But you aren't in anything at the moment.";
        2:  "You can't get out of the closed ", (name) x1, ".";
        3:  print "You get ";
            if (x1 has supporter) print "off "; else print "out of ";
            print_ret (the) x1, ".";
        4:  print "But you aren't ";
            if (x1 has supporter) print "on "; else print "in ";
            print_ret (the) x1, ".";
        5: print (The) actor, " gets off ", (the) x1, ".^";
        6: print (The) actor, " gets out of ", (the) x1, ".^";
    }
GetOff:   "But you aren't on ", (the) x1, " at the moment.";
Give: switch (n) {
        1:  "You aren't holding ", (the) x1, ".";
        2:  "You juggle ", (the) x1, " for a while, but don't achieve much.";
        3:  print (The) x1;
            if (x1 has pluralname) print " don't"; else print " doesn't";
            " seem interested.";
        4:  print (The) x1;
            if (x1 has pluralname) print " aren't";
            else print " isn't";
            " able to receive things.";
        5: "You give ", (the) x1, " to ", (the) second, ".";
        6: print (The) actor, " gives ", (the) x1, " to you.^";
        7: print (The) actor, " gives ", (the) x1, " to ", (the) second, ".^";
    }
Go: switch (n) {
        1:  print "You'll have to get ";
            if (x1 has supporter) print "off "; else print "out of ";
            print_ret (the) x1, " first.";
        2:  print_ret (string) CANTGO__TX;   ! "You can't go that way."
        6:  print "You can't, since ", (the) x1;
            if (x1 has pluralname) " lead nowhere."; else " leads nowhere.";
        7: "You'll have to say which compass direction to go in.";
        8: print (The) actor, " goes up";
        9: print (The) actor, " goes down";
        10: print (The) actor, " goes ", (name) x1;
        11: print (The) actor, " arrives from above";
        12: print (The) actor, " arrives from below";
        13: print (The) actor, " arrives from the ", (name) x1;
        14: print (The) actor, " arrives";
        15: print (The) actor, " arrives at ", (the) x1, " from above";
        16: print (The) actor, " arrives at ", (the) x1, " from below";
        17: print (The) actor, " arrives at ", (the) x1, " from the ", (name) x2;
        18: print (The) actor, " goes through ", (the) x1;
        19: print (The) actor, " arrives from ", (the) x1;
        20: print "on ", (the) x1;
        21: print "in ", (the) x1;
        22: print ", pushing ", (the) x1, " in front, and you along too";
        23: print ", pushing ", (the) x1, " in front";
        24: print ", pushing ", (the) x1, " away";
        25: print ", pushing ", (the) x1, " in";
        26: print ", taking you along";
        27: print "(first getting off ", (the) x1, ")^"; say__p = 0; return;
        28: print "(first opening ", (the) x1, ")^"; say__p = 0; return;
    }
Insert: switch (n) {
        1:  "You need to be holding ", (the) x1, " before you can put ", (itorthem) x1,
            " into something else.";
        2:  print_ret (Cthatorthose) x1, " can't contain things.";
        3:  print_ret (The) x1, " ", (isorare) x1, " closed.";
        4:  "You'll need to take ", (itorthem) x1, " off first.";
        5:  "You can't put something inside itself.";
        6:  print "(first taking ", (itorthem) x1, " off)^"; say__p = 0; return;
        7:  "There is no more room in ", (the) x1, ".";
        8:  "Done.";
        9:  "You put ", (the) x1, " into ", (the) second, ".";
    10:  print (The) actor, " puts ", (the) x1, " into ", (the) second, ".^";
    }
Inv: switch (n) {
        1:  "You are carrying nothing.";
        2:  print "You are carrying";
        3:  print ":^";
        4:  print ".^";
        5: print (The) x1, " looks through ", (HisHerTheir) x1, " possessions.^";
    }
Jump:     "You jump on the spot, fruitlessly.";
Kiss:     "Keep your mind on the game.";
Listen:   "You hear nothing unexpected.";
ListMiscellany: switch (n) {
        1:  print " (providing light)";
        2:  print " (closed)";
        4:  print " (empty)";
        6:  print " (closed and empty)";
        3:  print " (closed and providing light)";
        5:  print " (empty and providing light)";
        7:  #ifdef SERIAL_COMMA;
            print " (closed, empty, and providing light)";
            #ifnot;
            print " (closed, empty and providing light)";
            #endif;
        8:  print " (providing light and being worn";
        9:  print " (providing light";
        10: print " (being worn";
        11: print " (";
        12: print "open";
        13: print "open but empty";
        14: print "closed";
        15: print "closed and locked";
        16: print " and empty";
        17: print " (empty)";
        18: print " containing ";
        19: print " (on ";
        20: print ", on top of ";
        21: print " (in ";
        22: print ", inside ";
    }
LMode1:   " is now in its normal ~brief~ printing mode, which gives long descriptions
            of places never before visited and short descriptions otherwise.";
LMode2:   " is now in its ~verbose~ mode, which always gives long descriptions
            of locations (even if you've been there before).";
LMode3:   " is now in its ~superbrief~ mode, which always gives short descriptions
            of locations (even if you haven't been there before).";
Lock: switch (n) {
        1:  if (x1 has pluralname) print "They don't "; else print "That doesn't ";
            "seem to be something you can lock.";
        2:  print_ret (ctheyreorthats) x1, " locked at the moment.";
        3:  "First you'll have to close ", (the) x1, ".";
        4:  if (x1 has pluralname) print "Those don't "; else print "That doesn't ";
            "seem to fit the lock.";
        5:  "You lock ", (the) x1, ".";
        6: print (The) actor, " locks ", (the) x1, ".^";
    }
Look: switch (n) {
        1:  print " (on ", (the) x1, ")";
        2:  print " (in ", (the) x1, ")";
        3:  print " (as ", (object) x1, ")";
        4:  print "On ", (the) x1, " ";
            WriteListFrom(child(x1),
            ENGLISH_BIT+RECURSE_BIT+PARTINV_BIT+TERSE_BIT+CONCEAL_BIT+ISARE_BIT);
            ".";
        5,6:
            if (x1 ~= location) {
                if (x1 has supporter) print "On "; else print "In ";
                print (the) x1, " you";
            }
            else print "You";
            print " can ";
            if (n == 5) print "also ";
            print "see ";
            WriteListFrom(child(x1),
            ENGLISH_BIT+RECURSE_BIT+PARTINV_BIT+TERSE_BIT+CONCEAL_BIT+WORKFLAG_BIT);
            if (x1 ~= location) "."; else " here.";
        7:  "You see nothing unexpected in that direction.";
        8:  if (x1 has supporter) print " (on "; else print " (in ";
            print (the) x1, ")";
        9: print (The) actor, " looks around.^";
    }
LookUnder: switch (n) {
        1:  "But it's dark.";
        2:  "You find nothing of interest.";
        3: print (The) actor, " looks under ", (the) x1, ".^";
    }
Mild:     "Quite.";
Miscellany: switch (n) {
        1:  "(considering the first sixteen objects only)^";
        2:  "Nothing to do!";
        3:  print " You have died ";
        4:  print " You have won ";
        5:  print "^Would you like to RESTART, RESTORE a saved game";
            #Ifdef DEATH_MENTION_UNDO;
            print ", UNDO your last move";
            #Endif;
            #ifdef SERIAL_COMMA;
            print ",";
            #endif;
            " or QUIT?";
        6:  "[Your interpreter does not provide ~undo~.  Sorry!]";
            #Ifdef TARGET_ZCODE;
        7:  "~Undo~ failed.  [Not all interpreters provide it.]";
            #Ifnot; ! TARGET_GLULX
        7:  "[You cannot ~undo~ any further.]";
            #Endif; ! TARGET_
        8:  "Please give one of the answers above.";
        9:  "It is now pitch dark in here!";
        10: "I beg your pardon?";
        11: "[You can't ~undo~ what hasn't been done!]";
```

```
12: "[Can't ~undo~ twice in succession. Sorry!]";
13: "[Previous turn undone.]";
14: "Sorry, that can't be corrected.";
15: "Think nothing of it.";
16: "~Oops~ can only correct a single word.";
17: "It is pitch dark, and you can't see a thing.";
18: print "yourself";
19: "As good-looking as ever.";
20: "To repeat a command like ~frog, jump~, just say ~again~, not ~frog, again~.";
21: "You can hardly repeat that.";
22: "You can't begin with a comma.";
23: "You seem to want to talk to someone, but I can't see whom.";
24: "You can't talk to ", (the) x1, ".";
25: "To talk to someone, try ~someone, hello~ or some such.";
26: "(first taking ", (the) x1, ")";
27: "I didn't understand that sentence.";
28: print "I only understood you as far as wanting to ";
29: "I didn't understand that number.";
30: "You can't see any such thing.";
31: "You seem to have said too little!";
32: "You aren't holding that!";
33: "You can't use multiple objects with that verb.";
34: "You can only use multiple objects once on a line.";
35: "I'm not sure what ~", (address) pronoun_word, "~ refers to.";
36: "You excepted something not included anyway!";
37: "You can only do that to something animate.";
    #Ifdef DIALECT_US;
38: "That's not a verb I recognize.";
    #Ifnot;
38: "That's not a verb I recognise.";
    #Endif;
39: "That's not something you need to refer to in the course of this game.";
40: "You can't see ~", (address) pronoun_word, "~ (", (the) pronoun_obj,
    ") at the moment.";
41: "I didn't understand the way that finished.";
42: if (x1 == 0) print "None"; else print "Only ", (number) x1;
    print " of those ";
    if (x1 == 1) print "is"; else print "are";
    " available.";
43: "Nothing to do!";
44: "There are none at all available!";
45: print "Who do you mean, ";
46: print "Which do you mean, ";
47: "Sorry, you can only have one item here. Which exactly?";
48: print "Whom do you want";
    if (actor ~= player) print " ", (the) actor;
    print " to "; PrintCommand(); print "?^";
49: print "What do you want";
    if (actor ~= player) print " ", (the) actor;
    print " to "; PrintCommand(); print "?^";
50: print "Your score has just gone ";
    if (x1 > 0) print "up"; else { x1 = -x1; print "down"; }
    print " by ", (number) x1, " point";
```

```
                if (x1 > 1) print "s";
            51: "(Since something dramatic has happened, your list of commands has been cut short.)";
            52: "^Type a number from 1 to ", x1, ", 0 to redisplay or press ENTER.";
            53: "^[Please press SPACE.]";
            54: "[Comment recorded.]";
            55: "[Comment NOT recorded.]";
            56: print ".^";
            57: print "?^";
            58: print (The) actor, " ", (IsOrAre) actor, " unable to do that.^";
            59: "You must supply a noun.";
            60: "You may not supply a noun.";
            61: "You must name an object.";
            62: "You may not name an object.";
            63: "You must name a second object.";
            64: "You may not name a second object.";
            65: "You must supply a second noun.";
            66: "You may not supply a second noun.";
            67: "You must name something more substantial.";
            68: print "(", (The) actor, " first taking ", (the) x1, ")^";
            69: "(first taking ", (the) x1, ")";
            70: "The use of UNDO is forbidden in this game.";
            71: print (string) DARKNESS__TX;
            72: print (The) x1;
                if (x1 has pluralname) print " have"; else print " has";
                " better things to do.";
            73: "That noun did not make sense in this context.";
            74: print "[That command asks to do something outside of play, so it can
                only make sense from you to me. ", (The) x1, " cannot be asked to do this.]^";
            75:  print " The End ";
    }
No,Yes:   "That was a rhetorical question.";
NotifyOff:
                "Score notification off.";
NotifyOn: "Score notification on.";
Open: switch (n) {
            1:  print_ret (ctheyreorthats) x1, " not something you can open.";
            2:  if (x1 has pluralname) print "They seem "; else print "It seems ";
                "to be locked.";
            3:  print_ret (ctheyreorthats) x1, " already open.";
            4:  print "You open ", (the) x1, ", revealing ";
                if (WriteListFrom(child(x1), ENGLISH_BIT+TERSE_BIT+CONCEAL_BIT) == 0) "nothing.";
                ".";
            5:  "You open ", (the) x1, ".";
            6: print (The) actor, " opens ", (the) x1, ".^";
            7: print (The) x1, " open";
                if (x1 hasnt pluralname) print "s";
                print ".^";
    }
Pronouns: switch (n) {
            1:  print "At the moment, ";
            2:  print "means ";
            3:  print "is unset";
            4:  "no pronouns are known to the game.";
```

```
        5:  ".";
    }
Pull,Push,Turn: switch (n) {
        1:  if (x1 has pluralname) print "Those are "; else print "It is ";
            "fixed in place.";
        2:  "You are unable to.";
        3:  "Nothing obvious happens.";
        4:  "That would be less than courteous.";
        5: print (The) actor, " pulls ", (the) x1, ".^";
        6: print (The) actor, " pushes ", (the) x1, ".^";
        7: print (The) actor, " turns ", (the) x1, ".^";
    }
! Push: see Pull
PushDir: switch (n) {
        1:  print (The) x1, " cannot be pushed from place to place.^";
        2:  "That's not a direction.";
        3:  "Not that way you can't.";
    }
PutOn: switch (n) {
        1:  "You need to be holding ", (the) x1, " before you can put ",
                (itorthem) x1, " on top of something else.";
        2:  "You can't put something on top of itself.";
        3:  "Putting things on ", (the) x1, " would achieve nothing.";
        4:  "You lack the dexterity.";
        5:  print "(first taking ", (itorthem) x1, " off)^"; say__p = 0; return;
        6:  "There is no more room on ", (the) x1, ".";
        7:  "Done.";
        8:  "You put ", (the) x1, " on ", (the) second, ".";
        9:  print (The) actor, " puts ", (the) x1, " on ", (the) second, ".^";
    }
Quit: switch (n) {
        1:  print "Please answer yes or no.";
        2:  print "Are you sure you want to quit? ";
    }
Remove: switch (n) {
        1:  if (x1 has pluralname) print "They are"; else print "It is";
            " unfortunately closed.";
        2:  if (x1 has pluralname) print "But they aren't"; else print "But it isn't";
            " there now.";
        3:  "Removed.";
    }
Restart: switch (n) {
        1:  print "Are you sure you want to restart? ";
        2:  "Failed.";
    }
Restore: switch (n) {
        1:  "Restore failed.";
        2:  "Ok.";
    }
Rub:      "You achieve nothing by this.";
Save: switch (n) {
        1:  "Save failed.";
        2:  "Ok.";
    }
Score: switch (n) {
        1:  if (deadflag) print "In that game you scored "; else print "You have so far scored ";
            print score, " out of a possible ", MAX_SCORE, ", in ", turns, " turn";
            if (turns ~= 1) print "s";
            return;
        2:  "There is no score in this story.";
        3: print ", earning you the rank of ";
    }
ScriptOff: switch (n) {
        1:  "Transcripting is already off.";
        2:  "^End of transcript.";
        3:  "Attempt to end transcript failed.";
    }
ScriptOn: switch (n) {
        1:  "Transcripting is already on.";
        2:  "Start of a transcript of";
        3:  "Attempt to begin transcript failed.";
    }
Search: switch (n) {
        1:  "But it's dark.";
        2:  "There is nothing on ", (the) x1, ".";
        3:  print "On ", (the) x1, " ";
            WriteListFrom(child(x1), ENGLISH_BIT+TERSE_BIT+CONCEAL_BIT+ISARE_BIT);
            ".";
        4:  "You find nothing of interest.";
        5:  "You can't see inside, since ", (the) x1, " ", (isorare) x1, " closed.";
        6:  print_ret (The) x1, " ", (isorare) x1, " empty.";
        7:  print "In ", (the) x1, " ";
            WriteListFrom(child(x1), ENGLISH_BIT+TERSE_BIT+CONCEAL_BIT+ISARE_BIT);
            ".";
        8: print (The) actor, " searches ", (the) x1, ".^";
    }
SetTo:    "No, you can't set ", (thatorthose) x1, " to anything.";
Show: switch (n) {
        1:  "You aren't holding ", (the) x1, ".";
        2:  print_ret (The) x1, " ", (isorare) x1, " unimpressed.";
    }
Sing:     "Your singing is abominable.";
Sleep:    "You aren't feeling especially drowsy.";
Smell:    "You smell nothing unexpected.";
            #Ifdef DIALECT_US;
Sorry:    "Oh, don't apologize.";
            #Ifnot;
Sorry:    "Oh, don't apologise.";
            #Endif;
Squeeze: switch (n) {
        1:  "Keep your hands to yourself.";
        2:  "You achieve nothing by this.";
        3: print (The) actor, " squeezes ", (the) x1, ".^";
    }
Strong:   "Real adventurers do not use such language.";
Swing:    "There's nothing sensible to swing here.";
```

```
SwitchOff: switch (n) {
        1:  print_ret (ctheyreorthats) x1, " not something you can switch.";
        2:  print_ret (ctheyreorthats) x1, " already off.";
        3:  "You switch ", (the) x1, " off.";
        4: print (The) actor, " switches ", (the) x1, " off.^";
    }
SwitchOn: switch (n) {
        1:  print_ret (ctheyreorthats) x1, " not something you can switch.";
        2:  print_ret (ctheyreorthats) x1, " already on.";
        3:  "You switch ", (the) x1, " on.";
        4: print (The) actor, " switches ", (the) x1, " on.^";
    }
Take: switch (n) {
        1:  "Taken.";
        2:  "You are always self-possessed.";
        3:  "I don't suppose ", (the) x1, " would care for that.";
        4:  print "You'd have to get ";
            if (x1 has supporter) print "off "; else print "out of ";
            print_ret (the) x1, " first.";
        5:  "You already have ", (thatorthose) x1, ".";
        6:  if (noun has pluralname) print "Those seem "; else print "That seems ";
            "to belong to ", (the) x1, ".";
        7:  if (noun has pluralname) print "Those seem "; else print "That seems ";
            "to be a part of ", (the) x1, ".";
        8:  print_ret (Cthatorthose) x1, " ", (isorare) x1,
            "n't available.";
        9:  print_ret (The) x1, " ", (isorare) x1, "n't open.";
        10: if (x1 has pluralname) print "They're "; else print "That's ";
            "hardly portable.";
        11: if (x1 has pluralname) print "They're "; else print "That's ";
            "fixed in place.";
        12: "You're carrying too many things already.";
        13: print "(putting ", (the) x1, " into ", (the) x2,
            " to make room)^"; say__p = 0; return;
        14: "You can't reach into ", (the) x1, ".";
        15: "You cannot carry ", (the) x1, ".";
        16: print (The) actor, " picks up ", (the) x1, ".^";
    }
Taste:    "You taste nothing unexpected.";
Tell: switch (n) {
        1:  "You talk to yourself a while.";
        2:  "This provokes no reaction.";
    }
Think:    "What a good idea.";
ThrowAt: switch (n) {
        1:  "Futile.";
        2:  "You lack the nerve when it comes to the crucial moment.";
    }
Tie: "You would achieve nothing by this.";
Touch: switch (n) {
        1:  "Keep your hands to yourself!";
        2:  "You feel nothing unexpected.";
        3:  "If you think that'll help.";
        4: print (The) actor, " touches ", (himheritself) x1, ".^";
        5: print (The) actor, " touches you.^";
        6: print (The) actor, " touches ", (the) x1, ".^";
    }
! Turn: see Pull.
Unlock:  switch (n) {
        1:  if (x1 has pluralname) print "They don't "; else print "That doesn't ";
            "seem to be something you can unlock.";
        2:  print_ret (ctheyreorthats) x1, " unlocked at the moment.";
        3:  if (x1 has pluralname) print "Those don't "; else print "That doesn't ";
            "seem to fit the lock.";
        4:  "You unlock ", (the) x1, ".";
        5: print (The) actor, " unlocks ", (the) x1, ".^";
    }
Verify: switch (n) {
        1:  "The game file has verified as intact.";
        2:  "The game file did not verify as intact, and may be corrupt.";
    }
Wait: switch (n) {
        1:  "Time passes.";
        2: print (The) actor, " waits.^";
    }
Wake:     "The dreadful truth is, this is not a dream.";
WakeOther:"That seems unnecessary.";
Wave: switch (n) {
        1:  "But you aren't holding ", (thatorthose) x1, ".";
        2:  "You look ridiculous waving ", (the) x1, ".";
        3: print (The) actor, " waves ", (the) x1, ".^";
    }
WaveHands:"You wave, feeling foolish.";
Wear: switch (n) {
        1:  "You can't wear ", (thatorthose) x1, "!";
        2:  "You're not holding ", (thatorthose) x1, "!";
        3:  "You're already wearing ", (thatorthose) x1, "!";
        4:  "You put on ", (the) x1, ".";
        5: print (The) actor, " puts on ", (the) x1, ".^";
    }
! Yes:  see No.
];
```

§13. Printing Mechanism. The following routine produces a "library message" – though the library is no more, the terminology lives on.

The `L__M` routine is designed to reach up into I7 to offer it a chance to intervene, but then go back to the I6 method if it doesn't. The Standard Rules ordinarily define these three routines as stubs which always return false, so by default there's no intervention. (This is the hook for the new model of library messages which will be introduced in future builds.)

We divide into three cases because `##Miscellany` and `##ListMiscellany` are fake actions, not actions, in I6. This means it would not be type-safe to store them in I7 variables whose kind of value is "action name", and that would make any single I7 routine handling all three kinds of message quite difficult to write.

```
[ L__M act n x1 x2 rv flag;
    @push sw__var;
    sw__var = act;
    if (n == 0) n = 1;
    @push action;
    lm_act = act;
    lm_n = n;
    lm_o = x1;
    lm_o2 = x2;
    switch (act) {
        ##Miscellany: rv = (+ whether or not intervened in miscellaneous message +);
        ##ListMiscellany: rv = (+ whether or not intervened in miscellaneous list message +);
        default: rv = (+ whether or not intervened in action message +);
    }
    action = sw__var;
    if (rv == false) rv = RunRoutines(LibraryMessages, before);
    @pull action;
    if (rv == false) LanguageLM(n, x1, x2);
    @pull sw__var;
];
```

MStack Template B/stackt

Purpose

To allocate space on the memory stack for frames of variables to be used by rulebooks, activities and actions.

§**1. The Memory Stack.** The M-Stack, or memory stack, is a sequence of frames, piled upwards. If we had an accessible stack in memory, we could use that, but neither the Z-machine nor Glulx has such a stack, alas, alas, alas. The following is not a very good solution, but it just about works.

```
Constant MAX_MSTACK_FRAME = 2 + {-value:max_frame_size_needed};
Constant MSTACK_CAPACITY = 20;
Constant MSTACK_SIZE = MSTACK_CAPACITY*MAX_MSTACK_FRAME;

Array MStack --> MSTACK_SIZE;
Global MStack_Top = 0; ! Topmost word currently used
```

§**2. Create Frame.** A frame is created by calling the following function with two arguments: `creator`, a function which initialises a block of variables, and an ID number identifying the owner.

The `creator` function is called with the address at which to initialise the variables as its first argument, and the value 1 as the second argument. (The idea is that the same function can be used later to deallocate the variables, and then the second argument will be −1.) The `creator` function returns the extent of the block of memory it has used, in words. Thus is required to be strictly less than `MAX_MSTACK_FRAME` minus 1.

```
[ Mstack_Create_Frame creator id extent;
    if (creator == 0) rfalse;
    extent = creator.call(MStack_Top+2, 1);
    if (extent == 0) rfalse;
    if (MStack_Top + MAX_MSTACK_FRAME >= MSTACK_SIZE + 2) {
        RunTimeProblem(RTP_MSTACKMEMORY, MSTACK_SIZE);
        Mstack_Backtrace();
        rfalse;
    }
    MStack_Top++;
    MStack-->MStack_Top = id;
    MStack_Top++;
    MStack_Top = MStack_Top + extent;
    MStack-->MStack_Top = -(extent+2);
    rtrue;
];
```

§**3. Destroy Frame.** As sketched above, the same creator function and ID number are passed to the following routine to destroy the frame again. It takes the stack down to the level of the most recently created frame with this ID number: note that each action, for instance, has its own ID number for this purpose, but can be taking place several times in a nested fashion – one taking action might have caused another taking action which caused a third, for instance, so that there are three incomplete taking actions at once. In that case, there will be three independent sets of taking action variables on the M-stack, all with the same ID number. We remove the topmost one: the implication of that is that frames must always be destroyed in reverse order of creation.

In practice, I7 uses frames such that the frame sought should always be the topmost one in any case, and so that frames are always explicitly destroyed, not wiped by being undercut when an earlier-created frame is destroyed.

```
[ Mstack_Destroy_Frame creator id pos;
    pos = Mstack_Seek_Frame(id);
    if (pos == 0) rfalse; ! Not found: do nothing
    MStack_Top = pos - 2; ! Clear mstack down to just below this frame
    if (creator) creator.call(pos, -1);
    rtrue;
];
```

§**4. Seek Frame.** We return the position on the M-stack of the most recently created frame with the given ID number (see above), or 0 if no such frame exists; the size is stored in the global variable `MStack_Frame_Extent`. (Because word 0 on the stack is used as a sentinel – all frames are placed above it – no frame can actually begin at word 0 on the stack, so 0 is safe to use as an exception.)

```
Global MStack_Frame_Extent = 0;

[ Mstack_Seek_Frame id pos;
    pos = MStack_Top;
    while ((pos > 0) && (MStack-->pos ~= 0)) {
        MStack_Frame_Extent = MStack-->pos;
        pos = pos + MStack_Frame_Extent;
        MStack_Frame_Extent = (-2) - MStack_Frame_Extent;
        if (MStack-->(pos+1) == id) return pos+2;
    }
    MStack_Frame_Extent = 0;
    return 0; ! Not found
];
```

§**5. Backtrace.** Purely for debugging purposes, and giving feedback if the stack runs out of memory:

```
[ Mstack_Backtrace pos k;
    print "Mstack backtrace: size ", MStack_Top+1, " words^";
    pos = MStack_Top;
    while (MStack-->pos ~= 0) {
        MStack_Frame_Extent = MStack-->pos;
        pos = pos + MStack_Frame_Extent;
        MStack_Frame_Extent = (-2) - MStack_Frame_Extent;
        print "Block at ", pos+2,
            " owner ID ", MStack-->(pos+1), " size ", MStack_Frame_Extent, "^";
        for (k=0: k<MStack_Frame_Extent: k++) print MStack-->(pos+2+k), " ";
        print "^";
    }
];
```

§**6. Access to Variables.** An M-stack variable is identified by a combination of ID number and offset: for instance ID 20007, offset 1, is the variable "room gone to" belonging to the going action. The following routine converts that into an address on the M-stack, in the topmost block with the given ID number (since "room gone to", for instance, always means its value in the most current going action of those now under way). Typechecking in the compiler should mean that it is impossible to produce either error message below: NI will only compile valid uses of `MstVO` ("M-stack variable offset") where the seek succeeds and the offset is within range.

```
[ MstVO id off pos;
    pos = Mstack_Seek_Frame(id);
    if (pos == 0) {
        print "Variable unavailable for this action, activity or rulebook: ",
            "internal ID number ",
            id, "/", off, "^";
        rfalse;
    }
    if ((off<0) || (off >= MStack_Frame_Extent)) {
        print "Variable stack offset wrong: ", id, "/", off, " at ", pos, "^";
        rfalse;
    }
    return pos+off;
];
```

§**7. Access to Nonexistent Variables.** A long-standing point where I7 is not as strict in type-checking as it might be occurs when checking rule preambles like "Before going to a dead end...". Such a preamble must be checked whatever the current action is – in many cases, it will not be a going action at all; which means that "room gone to", a value implied by the "to" clause, will not exist. If the type-checking were stricter, it would be a nuisance for authors, and instead we relax a little by accessing such variables using a more forgiving routine. Here, if a variable does not exist, we return 0 to mean that it can be read at M-stack position 0: this is the sentinel word, which is not part of any frame, and which contains 0. Thus the variable reads as if it is 0, the default for the kind of value "object", which is the KOV for action variables such as "room gone to".

The routine may only be used where the variable is being read, and never where it is to be written, of course: that would corrupt the sentinel.

```
[ MstVON id off pos;
    pos = Mstack_Seek_Frame(id);
    if (pos == 0) {
        return 0; ! word position 0 on the M-stack
    }
    if ((off<0) || (off >= MStack_Frame_Extent)) {
        print "Variable stack offset wrong: ", id, "/", off, " at ", pos, "^";
        rfalse;
    }
    return pos+off;
];
```

§**8. Rulebook Variables.** Each rulebook has a slate of variables, usually empty, with ID number the same as the rulebook's own ID number. (Rulebook IDs number upwards from 0 in order of creation in the source text.) The associated creator functions, usually null, are stored in an array if there is no problem about memory usage, but with a switch statement if `MEMORY_ECONOMY` is in force; this costs a very small amount of time, but saves 1K of readable memory.

```
{-array:Code::Rulebooks::rulebook_var_creators}
[ MStack_CreateRBVars rb cr;
{-call:Code::Rulebooks::rulebook_var_creators_lookup}
    if (cr == 0) return;
    Mstack_Create_Frame(cr, rb);
];
[ MStack_DestroyRBVars rb cr;
{-call:Code::Rulebooks::rulebook_var_creators_lookup}
    if (cr == 0) return;
    Mstack_Destroy_Frame(cr, rb);
];
```

§**9. Activity Variables.** Exactly the same goes for activity variables except that here the ID number is $10000 + N$, where N is the allocation ID of the activity. (This would fail if there were more than 10,000 rulebooks, but this is very difficult to see happening.)

```
{-array:Code::Activities::activity_var_creators}
[ MStack_CreateAVVars av cr;
    cr = activity_var_creators-->av;
    if (cr == 0) return;
    Mstack_Create_Frame(cr, av + 10000);
];
[ MStack_DestroyAVVars av cr;
    cr = activity_var_creators-->av;
    if (cr == 0) return;
    Mstack_Destroy_Frame(cr, av + 10000);
];
```

Chronology Template B/chrt

Purpose

To record information now which will be needed later, when a condition phrased in the perfect tense is tested.

§**1. Scheme I.** If source text contains a condition like "if the well has been dry, ...", then we need to keep a chronological record by testing at every turn whether or not the well is dry: we log whether this is true now, whether it has ever been true, for how many consecutive turns it has been true (to a maximum of 127), and how many times it has become true having just been false (again, to a maximum of 127 times). All this information is packed into a single word, arranged as a 15-bit bitmap: the least significant bit is the state of the condition now (true or false), the next 7 bits are the number of false-to-true "trips" observed over time, and the top 7 bits are the number of consecutive turns on which the condition has been true ("consecutives"). The 16th and most significant bit is unused, so that the state is always positive even in a 16-bit virtual machine – a convenience since we then don't need to worry about the effect of signed division and remainder on the bitmap.

There is no need to store a flag for "has this condition ever been true", because this is equivalent the number of trips being greater than zero. This might look wrong for a condition which is true at start of play – say, if the well was always dry – because then its state has never changed from false to true. But in fact when the VM starts up the state word is initially always 0: it's only when the startup rulebook fires the update chronological records rule (see below) that we first test whether the well is dry, and that forces a trip from false to true if the well is dry at start of play. Therefore, if T is the number of trips for the condition then $T = 0$ if and only if the condition has been false from the very start of play. If the condition is true now, then $T = 1$ if and only if it has always been so.

§**2. Present and Past.** Each individual condition has its own unique "PT number", counting upwards from 0 in order of compilation by NI, and a "chronological record" is a word array with a state word as described above for each PT number.

However, we keep not one but two chronological records: one for the situation now, called the "present chronological record", and one for the situation as it was just before interesting things most recently happened, called the "past chronological record". If one of those interesting things was that the well ran dry, then in our example the state word for the condition "if the well has been dry" would be different in the two chronological records. We keep two records in order to be able to detect conditions in four different tenses:

(1) Present tense ("if the well is dry"): none of this machinery is used, because we can just test directly instead, so a present tense condition has no PT-number.
(2) Past tense ("if the well was dry"): we look at the flag bit in the past chronological record.
(3) Perfect tense ("if the well has been dry"): we look at whether or not $T > 0$ in the present chronological record.
(4) Past perfect tense ("if the well had been dry"): we look at whether or not $T > 0$ in the past chronological record.

It's a somewhat ambiguous matter of context in English when the reference time is for past tenses. If somebody comes out into the sunshine and says, "But it was raining," when does he mean? We would reasonably guess earlier that day, and probably only an hour or two ago, but that's a contextual judgement made on the basis of our own experience of how rapidly weather changes. The context for Inform source text is always that of actions, so our convention is that the reference time for past tenses is the point just before the current action began. Such key moments – when things are just about to happen – are called "chronology points". There is a CP just before each action begins; there is a CP in the startup process; and there is a

CP at the end of each turn, for good measure. CPs happen when somebody calls `ChronologyPoint()`. The action machinery does this directly, while the startup CP and the CP at end of turn happen in the course of the update chronological records rule (below).

§3. Chronology Point. This is when the time of reference for past tenses is now: so it is where the past becomes the present. Soon, exciting things will happen, and the present will go on developing, while the past will remain as it was; until things calm down again and we come to another chronology point.

```
[ ChronologyPoint pt;
    for (pt=0:pt<NO_PAST_TENSE_CONDS:pt++)
        past_chronological_record-->pt = present_chronological_record-->pt;
];
```

§4. Update Chronological Records Rule. It might seem odd that a routine to, supposedly, update something would only call another routine called `TestSinglePastState`: but in this setting, any test updates the state, because it changes the number of times something has been found true, and so on.

```
[ UPDATE_CHRONOLOGICAL_RECORDS_R pt;
    for (pt=0: pt<NO_PAST_TENSE_CONDS: pt++) TestSinglePastState(false, pt, true, -1);
    ChronologyPoint();
    rfalse;
];
```

§5. Test Single Past State. `TestSinglePastState` is called with four arguments:

(a) `past_flag` is true if we want to test a condition like "if the well was dry" or "if the well had been dry", which concern only the state as it was at the last chronology point – in other words, the `past_chronological_record`, which we must not change; whereas `past_flag` is false if we want to test a present tense like "if the well is dry" or "if the well has been dry", because then we deal with the `present_chronological_record` which must be kept continuously updated.
(b) `pt` is the PT number for the condition.
(c) `turn_end` is true if we are making the test at the end of a turn, as part of the update chronological records rule, and false otherwise. (For these purposes the start of play is a turn end since the UCRR runs then, too.)
(d) `wanted` describes what information the function should return, as detailed in its code below.

```
{-call:Code::Chronology::past_actions_i6_routines}
{-call:Code::Chronology::chronology_extents_i6_escape}
[ TestSinglePastState past_flag pt turn_end wanted
    old new trips consecutives ct_0 ct_1 I7BASPL;
    if (past_flag) {
        new = (past_chronological_record-->pt) & 1;
        trips = ((past_chronological_record-->pt) & $$11111110)/2;
        consecutives = ((past_chronological_record-->pt) & $$111111100000000)/256;
    } else {
        old = (present_chronological_record-->pt) & 1;
        trips = ((present_chronological_record-->pt) & $$11111110)/2;
        consecutives = ((present_chronological_record-->pt) & $$111111100000000)/256;
! Test cases for conditions by PT number: each sets "new" to whether it is true or false now
{-call:Code::Chronology::past_tenses_i6_escape}
        if (new == false) {
            consecutives = 0;
        } else {
            if (old == false) { trips++; if (trips > 127) trips = 127; }
            if (turn_end) { consecutives++; if (consecutives > 127) consecutives = 127; }
        }
        present_chronological_record-->pt = new + 2*trips + 256*consecutives;
    }
    ! print pt, ": old=", old, " new=", new, " trips=", trips, " consec=", consecutives,
    ! " wanted=", wanted, "^";
    switch(wanted) {
        0: if (new) return new;
        1: if (new) return trips;
        2: if (new) return consecutives+1; ! Plus one because we count the current turn
        4: return new;
        5: return trips;
        6: return consecutives;
    }
    return 0;
];
```

§**6. Scheme II.** Actions discussed in the past tense – "if we have taken the ball" or, more subtly, "Instead of waiting for the third turn" (which also refers to the past) – are handled in a related but simpler way. NI counts such references, just as with other past tense conditions above, but this time stores a state in two simple word arrays: `TimesActionHasHappened` and `TurnsActionHasBeenHappening`.

It might reasonably be asked why we don't simply use all of the clever machinery above: why have an entirely different system here? One reason is that actions are events and not continuous states of being. "The well is dry" could be true for any extent of time, but "taking the ball" either happens at a given moment or doesn't: it is not continuously true. We are therefore in the business of counting events, not measuring durations. Another reason is that the point of reference for past tenses is different. It makes no sense to say "if we were looking" because that would mean at a time just before the current action, when actions were probably not happening at all; while "if we had looked" and "if we have looked" would almost always be identical in meaning for the same reason. So we need a much simpler system with just one possible past tense; we don't need to keep two different states; and we use the extra storage to enable us to count the number of times, and the number of turns, to at least 31767 instead of stopping at 127. (We also generate more legible code to test past tense actions.)

§**7. Past Action Routines.** Actions can be quite complicated to test even in the present, and this is much more conveniently done in a routine with its own NI-generated stack frame; so each past tense action has its own testing routine, with a name like `PAPR_41`, which returns true or false depending on whether the action is going on now. The word array `PastActionsI6Routines` then contains a null-terminated list of these routines.

§**8. Track Actions.** The routine `TrackActions` then updates the two arrays, and is the equivalent for past tense actions of `ChronologyPoint`. It's called twice for each in-world action: `TrackActions(false)` just as a new action is about to begin, and then `TrackActions(true)` just after it has finished – note in particular that if action B happens in the middle of action A, for instance because of a try phrase, then the `TrackActions(true)` call for B happens when A is back as the current action. In fact, that's the point: because it tells us that B was not the main action for the turn, but only a phase which has now passed again.

For each of the action patterns we track, we need to count the number of times such an action has been tried (*not* the number of times it succeeded), and also the number of consecutive turns on which it has been "the" action. The count of the number of tries is unambiguous and easy to keep: it is incremented at the start-of-action call only, and only if the action matches. But the consecutive turn count is more problematic. The user wants to think of actions and turns as synonymous, and to write conditions like "Before going for the third turn", and we want to play along because that's a natural phrasing: but actions and turns are not synonymous at all. The rules are therefore:

(1) At the start-of-action call, provided the action is not a silent one, the consecutive turns count is either incremented or zeroed, depending on whether the action matches. Silent actions are ignored because they are almost always knock-on actions tried in the course of other actions, and are certainly not the "main" action of the turn. But the count can be incremented only once in the course of each turn, so that "Instead of taking something for the third turn: ..." cannot happen on the very first turn because TAKE ALL causes three or more take actions.
(2) At the readjustment call, provided the action is not a silent one, the consecutive turns count is zeroed if the action does not match.

Why do we zero the count in rule (2)? Well, suppose that the player types OPEN BOX, so that the opening action takes place, and that it causes a further (non-silent) action, say examining the lid: and suppose that the action pattern we track the turn count for is "examining something". Then the following calls take place:

(a) `TrackActions(false)` while the action is "opening the box". Turn count zeroed by rule (1).
(b) `TrackActions(false)` while the action is "examining the lid". Turn count incremented by rule (1).
(c) `TrackActions(true)` while the action is "opening the box". Turn count zeroed by rule (2).
(d) `TrackActions(true)` while the action is "opening the box". Turn count zeroed by rule (2).

Thus rule (2) prevents us from counting this turn as the first of a sequence of turns in which examining something was the action, because it wasn't the main action of the turn.

A further complication is that we need to record the qualification, or not, even of silent actions, because testing rules like

```
Instead of taking the top hat less than three times...
```

works by checking that (a) we are currently taking the top hat, and (b) have done so fewer than three times before. (b) uses the turn count described above; but (a) cannot simply look to see if that count is positive, since the action might be happening silently and thus not have contributed to the count; so we also record a flag to hold whether the action seems to be happening in this turn, silent or not.

And a still further complication is that out-of-world actions should never affect counts of turns for in-world actions. Thus,

```
Every turn jumping for three turns: say "A demon appears!"
```

must not have the count to three broken by the player typing SAVE, which causes an out-of-world action but doesn't use a turn. The simplest way to deal with that would be to make `TrackActions` do nothing when `oow` is set, but then we would get rules like this wrong:

```
Check requesting the pronoun meanings for the first time: ...
```

because it *does* make sense for OOW actions to have a count of how many times they've happened, even though they don't occur in simulated time. An OOW action can cause its own `ActionCurrentlyHappeningFlag` to be set, but it can't cause any other action's flag to be cleared, and nor can it zero the turns count for another action.

```
[ TrackActions readjust oow ct_0 ct_1 i;
    for (i=0: PastActionsI6Routines-->i: i++) {
        if ((PastActionsI6Routines-->i).call()) {
            ! Yes, the current action matches action pattern i:
            if (readjust) continue;
            (TimesActionHasHappened-->i)++;
            if (LastTurnActionHappenedOn-->i ~= turns + 5) {
                LastTurnActionHappenedOn-->i = turns + 5;
                ActionCurrentlyHappeningFlag->i = 1;
                if (keep_silent == false)
                    (TurnsActionHasBeenHappening-->i)++;
            }
        } else {
            ! No, the current action doesn't match action pattern i:
            if (oow == false) {
                if (keep_silent == false) { TurnsActionHasBeenHappening-->i = 0; }
                if (LastTurnActionHappenedOn-->i ~= turns + 5)
                    ActionCurrentlyHappeningFlag->i = 0;
            }
        }
    }
];
```

§**9. Storage.** The necessary array allocation.

```
Array TimesActionHasHappened-->(NO_PAST_TENSE_ACTIONS+1);
Array TurnsActionHasBeenHappening-->(NO_PAST_TENSE_ACTIONS+1);
Array LastTurnActionHappenedOn-->(NO_PAST_TENSE_ACTIONS+1);
Array ActionCurrentlyHappeningFlag->(NO_PAST_TENSE_ACTIONS+1);

Array past_chronological_record-->(NO_PAST_TENSE_CONDS+1);
Array present_chronological_record-->(NO_PAST_TENSE_CONDS+1);
```

Printing Template B/print

Purpose

To manage the line skips which space paragraphs out, and to handle the printing of names of objects, pieces of text and numbers.

§**1. Paragraph Control.** Ah, yes: the paragraph breaking algorithm. In *TEX: The Program*, Donald Knuth writes at §768: "It's sort of a miracle whenever `\halign` and `\valign` work, because they cut across so many of the control structures of TEX." It's sort of a miracle whenever Inform 7's paragraph breaking system works, too. Most users probably imagine that it's implemented by having I7 look at where the cursor currently is (at the start of a line or not) and whether a line has just been skipped. In fact, the virtual machines simply do not offer facilities like that, and so we have to use our own book-keeping. Given the huge number of ways in which text can be printed, this is a delicate business. For some years now, "spacing bugs" – those where a spurious extra skipped line appears in a paragraph break, or where, conversely, no line is skipped at all – have been the least welcome in the Inform bugs database.

The basic method is to set `say__p`, the paragraph flag, when we print any matter; every so often we reach a "divide paragraph" point – for instance when one rule has finished and before another is about to start – and at those positions we look for `say__p`, and print a skipped line (and clear `say__p` again) if we find it. Thus:

```
> WAIT
The clock ticks ominously. ...first rule
     ...skipped line printed at a Divide Paragraph point
Mme Tourmalet rises from her chair and slips out. second rule
     ...skipped line printed at a Divide Paragraph point
>
```

A divide paragraph point occurs between any two rules in an action rulebook, but not an activity rulebook: many activities exist to print text, such as the names of objects, and there would be wild spacing accidents if paragraphs were divided there. Inform places DPPs elsewhere, too: and the text substitution "[conditional paragraph break]" allows the user to place one anywhere.

A traditional layout convention handed down from Infocom makes an exception of the first paragraph to appear after the prompt, but only in one situation. Ordinarily, the first paragraph of any turn appears straight after the prompt:

```
> EXAMINE DOG
Mme Tourmalet's borzoi looks as if it means fashion, not business.
```

The command is echoed on screen as the player types, but this doesn't set the paragraph flag, which is still clear when the text "Mme Tourmalet's..." begins to be printed. The single exception occurs when the command calls for the player to go to a new location, when a skipped line is printed before the room description for the new room. Thus:

```
> SOUTH
     ...the "going look break"
Rocky Beach
```

(Note that this is not inherent in the looking action:

```
> LOOK
Rocky Beach
```

...which obeys the standard paragraphing conventions.)

So much for automatic paragraph breaks. However, we need a variety of different ways explicitly to control paragraphs, in order to accommodate traditional layout conventions handed down from Infocom.

The simplest exceptional kind of paragraph break is a "command clarification break", in which a single new-line is printed but there is no skipped line: as the name implies, it's traditionally used when a command such as OPEN DOOR is clarified. For example:

```
(first unlocking the oak door) ...now a command clarification break
You open the oak door.
```

This is not quite the same thing as a "run paragraph on" break, in which we also deliberately suppress the skipped line, but make an exception for the skipped line which ought to appear last before the prompt: the idea is to merge two or more paragraphs together.

```
> TAKE ALL
marmot: Taken. ...we run paragraph on here
weasel: Taken. ...and also here
    ...despite which the final skip does occur
> ...before the next prompt
```

A more complicated case is "special look spacing", used for the break which occurs after the (boldface) short name of a room description is printed. This is tricky because it is sometimes followed directly by a long description, and we don't want a skipped line:

```
Villa Christiane ...a special look spacing break
The walled garden of a villa in Cap d'Agde.
    ...a Divide Paragraph break
Mme Tourmalet's borzoi lazes in the long grass.
```

But sometimes it is followed directly by a subsequent paragraph, and again we want no skip:

```
Villa Christiane ...a special look spacing break
Mme Tourmalet's borzoi lazes in the long grass.
```

And sometimes it is the only content of the room description and is followed only by the prompt:

```
Villa Christiane ...a special look spacing break
    ...a break inserted before the prompt
>
```

To recap, we have five kinds of paragraph break:

(a) Standard breaks at "divide paragraph points", used between rules.
(b) The "going look break", used before the room description after going to a new room.
(c) A "command clarification break", used after text clarifying a command.
(d) A "run paragraph on" break, used to merge multiple paragraphs into a single block of text.
(e) The "special look spacing" break, used after the boldface headline of a room description.

We now have to implement all of these behaviours. The code, while very simple, is highly prone to behaving unexpectedly if changes are made, simply because of the huge number of circumstances in which paragraphs are printed: so change nothing without very careful testing.

§**2. State.** The current state is stored in a combination of two global variables:

(1) `say__p`, the "say paragraph" flag, which is set if a paragraph break needs to be printed before the next text can begin;

(2) `say__pc`, originally named as the "paragraph completed" flag, but which is now a bitmap:

- (2a) `PARA_COMPLETED` is set if a standard paragraph break has been made since the last time the flag was cleared;
- (2b) `PARA_PROMPTSKIP` is set to indicate that the current printing position does not follow a skipped line, and that further material is expected which will run on from the previous paragraph, but that if no further material turns up then a skipped line would be needed before the next prompt;
- (2c) `PARA_SUPPRESSPROMPTSKIP` is set to indicate that, despite `PARA_PROMPTSKIP` being set, no skipped line is needed before the prompt after all;
- (2d) `PARA_NORULEBOOKBREAKS` suppresses divide paragraph points in between rules in rulebooks; it treats all rulebooks, and in particular action rulebooks, the way activity rulebooks are treated. (The flag is used for short periods only and never across turn boundaries, prompts and so on.)
- (2e) `PARA_CONTENTEXPECTED` is set after a paragraph division as a signal that if any contents looks likely to be printed soon then `say__p` needs to be set, because a successor paragraph will then have started. This is checked by calling `ParaContent()` – while it's slow to have to call this routine so often, that's better than compiling inline code with the same effect, because minimising compiled code size is more important, and speed is never a big deal when printing.

Not all printing is to the screen: sometimes the output is to a file, or to memory, and in that case we want to start the switched output at a clear paragraphing state and then go back to the screen afterwards without any sign of change. The correct way to do this is to push the `say__p` and `say__pc` variables onto the VM stack and call `ClearParagraphing()` before starting to print to the new stream, and then pull the variables back again before resuming printing to the old stream.

In no other case should any code alter `say__pc` except via the routines below.

```
!Constant TRACE_I7_SPACING;

[ ClearParagraphing;
    say__p = 0; say__pc = 0;
];

[ DivideParagraphPoint;
    #ifdef TRACE_I7_SPACING; print "[DPP", say__p, say__pc, "]"; #endif;
    if (say__p) {
        new_line; say__p = 0; say__pc = say__pc | PARA_COMPLETED;
        if (say__pc & PARA_PROMPTSKIP) say__pc = say__pc - PARA_PROMPTSKIP;
        if (say__pc & PARA_SUPPRESSPROMPTSKIP) say__pc = say__pc - PARA_SUPPRESSPROMPTSKIP;
    }
    #ifdef TRACE_I7_SPACING; print "[-->", say__p, say__pc, "]"; #endif;
    say__pc = say__pc | PARA_CONTENTEXPECTED;
];

[ ParaContent;
    if (say__pc & PARA_CONTENTEXPECTED) {
        say__pc = say__pc - PARA_CONTENTEXPECTED;
        say__p = 1;
    }
];

[ GoingLookBreak;
    if (say__pc & PARA_COMPLETED == 0) new_line;
    ClearParagraphing();
];

[ CommandClarificationBreak;
```

```
    new_line;
    ClearParagraphing();
];
[ RunParagraphOn;
    #ifdef TRACE_I7_SPACING; print "[RPO", say__p, say__pc, "]"; #endif;
    say__p = 0;
    say__pc = say__pc | PARA_PROMPTSKIP;
    say__pc = say__pc | PARA_SUPPRESSPROMPTSKIP;
];
[ SpecialLookSpacingBreak;
    #ifdef TRACE_I7_SPACING; print "[SLS", say__p, say__pc, "]"; #endif;
    say__p = 0;
    say__pc = say__pc | PARA_PROMPTSKIP;
];
[ EnsureBreakBeforePrompt;
    if ((say__p) ||
        ((say__pc & PARA_PROMPTSKIP) && ((say__pc & PARA_SUPPRESSPROMPTSKIP)==0)))
        new_line;
    ClearParagraphing();
];
[ PrintSingleParagraph matter;
    say__p = 1;
    say__pc = say__pc | PARA_NORULEBOOKBREAKS;
    PrintText(matter);
    DivideParagraphPoint();
    say__pc = 0;
];
```

§**3. Say Number.** The global variable `say__n` is set to the numerical value of any quantity printed, and this is used for the text substitution "[s]", so that "You have been awake for [turn count] turn[s]." will expand correctly.

```
[ STextSubstitution;
    if (say__n ~= 1) print "s";
];
```

§**4. Prompt.** This is the text printed just before we wait for the player's command: it prompts him to type.

```
[ PrintPrompt i;
    style roman;
    EnsureBreakBeforePrompt();
    PrintText( (+ command prompt +) );
    ClearBoxedText();
    ClearParagraphing();
    enable_rte = true;
];
```

§5. Boxed Quotations. These appear once only, and happen outside of the paragraphing scheme: they are normally overlaid as windows on top of the regular text. We can request one at any time, but it will appear only at prompt time, when the screen is fairly well guaranteed not to be scrolling. (Only fairly well since it's just possible that *Border Zone*-like tricks with real-time play might be going on, but whatever happens, there is at least a human-appreciable pause in which the quotation can be read before being taken away again.)

```
Global pending_boxed_quotation; ! a routine to overlay the quotation on screen

[ DisplayBoxedQuotation Q;
    pending_boxed_quotation = Q;
];

[ ClearBoxedText i;
    if (pending_boxed_quotation) {
        for (i=0: Runtime_Quotations_Displayed-->i: i++)
            if (Runtime_Quotations_Displayed-->i == pending_boxed_quotation) {
                pending_boxed_quotation = 0;
                return;
            }
        Runtime_Quotations_Displayed-->i = pending_boxed_quotation;

        ClearParagraphing();
        pending_boxed_quotation();
        ClearParagraphing();

        pending_boxed_quotation = 0;
    }
];
```

§6. Score Notification. This doesn't really deserve to be at I6 level at all, but since for traditional reasons we need to use conditional compilation on `NO_SCORING`, and since we want a fancy text style for Glulx, ...

```
[ NotifyTheScore;
#Ifndef NO_SCORING;
    if (notify_mode == 1) {
        DivideParagraphPoint();
        VM_Style(NOTE_VMSTY);
        print "["; L__M(##Miscellany, 50, score-last_score); print ".]^";
        VM_Style(NORMAL_VMSTY);
    }
#Endif;
];
```

§7. Status Line. Status line printing happens on the upper screen window, and outside of the paragraph control system.

Support for version 6 of the Z-machine is best described as grudging. It requires a heavily rewritten `DrawStatusLine` equivalent, to be found in "ZMachine.i6t".

```
#Ifdef TARGET_ZCODE;
#Iftrue (#version_number == 6);
[ DrawStatusLine; Z6_DrawStatusLine(); ];
#Endif;
#Endif;
```

```
#Ifndef DrawStatusLine;
[ DrawStatusLine width posb;
    @push say__p; @push say__pc;
    BeginActivity(CONSTRUCTING_STATUS_LINE_ACT);
    VM_StatusLineHeight(1); VM_MoveCursorInStatusLine(1, 1);
    if (statuswin_current) {
        width = VM_ScreenWidth(); posb = width-15;
        spaces width;
        ClearParagraphing();
        if (ForActivity(CONSTRUCTING_STATUS_LINE_ACT) == false) {
            VM_MoveCursorInStatusLine(1, 2);
            switch(metaclass(left_hand_status_line)) {
                String: print (string) left_hand_status_line;
                Routine: left_hand_status_line();
            }
            VM_MoveCursorInStatusLine(1, posb);
            switch(metaclass(right_hand_status_line)) {
                String: print (string) right_hand_status_line;
                Routine: right_hand_status_line();
            }
        }
        VM_MoveCursorInStatusLine(1, 1); VM_MainWindow();
    }
    ClearParagraphing();
    EndActivity(CONSTRUCTING_STATUS_LINE_ACT);
    @pull say__pc; @pull say__p;
];
#Endif;
```

§8. Status Line Utilities. Two convenient routines for the default values of `right_hand_status_line` and `left_hand_status_line` respectively. `SL_Location` also implements the text substitution "[player's surroundings]".

```
[ SL_Score_Moves;
    if (not_yet_in_play) return;
    #ifdef NO_SCORING; print sline2; #ifnot; print sline1, "/", sline2; #endif;
];
[ SL_Location;
    if (not_yet_in_play) return;
    if (location == thedark) {
        BeginActivity(PRINTING_NAME_OF_DARK_ROOM_ACT);
        if (ForActivity(PRINTING_NAME_OF_DARK_ROOM_ACT) == false)
            L__M(##Miscellany, 71);
        EndActivity(PRINTING_NAME_OF_DARK_ROOM_ACT);
    } else {
        FindVisibilityLevels();
        if (visibility_ceiling == location) print (name) location;
        else print (The) visibility_ceiling;
    }
];
```

§9. Banner. Note that NI always compiles `Story` and `Headline` texts, but does not always compile a `Story_Author`.

```
[ Banner;
BeginActivity(PRINTING_BANNER_TEXT_ACT);
if (ForActivity(PRINTING_BANNER_TEXT_ACT) == false) {
        VM_Style(HEADER_VMSTY);
        print (string) Story;
        VM_Style(NORMAL_VMSTY);
        new_line;
        print (string) Headline;
        #ifdef Story_Author;
        print " by ", (string) Story_Author;
        #endif; ! Story_Author
        new_line;
        VM_Describe_Release();
        print " / Inform 7 build ", (string) NI_BUILD_COUNT, " ";
        print "(I6/v"; inversion;
        print " lib ", (string) LibRelease, ") ";
        #Ifdef STRICT_MODE;
        print "S";
        #Endif; ! STRICT_MODE
        #Ifdef DEBUG;
        print "D";
        #Endif; ! DEBUG
        new_line;
    }
    EndActivity(PRINTING_BANNER_TEXT_ACT);
];
```

§10. Print Decimal Number. `DecimalNumber` is a trivial function which just prints a number, in decimal digits. It is left over from the I6 library's support routines for Glulx, where it was intended as a stub to pass to the Glulx `Glulx_PrintAnything` routine (which I7 does not use). In I7, however, it's also used as the default printing routine for new kinds of value.

```
[ DecimalNumber num; print num; ];
```

§11. Print English Number. Another traditional name, this: in fact it prints the number as text in whatever is the current language of play.

```
[ EnglishNumber n; LanguageNumber(n); ];
```

§12. Print Text. The routine for printing an I7 "text" value, which might text with or without substitutions.

```
[ PrintText x;
    if (x ofclass String) print (string) x;
    if (x ofclass Routine) (x)();
];
[ I7_String x; PrintText(x); ]; ! An alternative name now used only by extensions
```

§13. Print Or Run. This utility remains from the old I6 library: it essentially treats a property as textual and prints it where possible. Where the `no_break` flag is set, we expect the text to form only a small part of a paragraph, and it's inappropriate to break here: for instance, for printing the "printed name" of an object. Where the flag is clear, however, the text is expected to form its own paragraph.

Where `PrintOrRun` is used in breaking mode, which is only for a very few properties in I7 (indeed at present only `initial` and `description`), the routine called is given the chance to decide whether to print or not. It should return `true` or `false` according to whether it did so; this allows us to divide the paragraph or not accordingly.

```
[ PrintOrRun obj prop no_break  routine_return_value;
    !print "(", obj, ".", prop, ";", say__p, say__pc, ")";
    if (prop == 0) {
        print (name) prop; routine_return_value = true;
    } else {
        switch (metaclass(obj.prop)) {
            nothing:
                routine_return_value = false;
            String:
                if (obj.prop == EMPTY_TEXT_VALUE) break;
                print (string) obj.prop; !if (no_break == false) new_line;
                routine_return_value = true;
            Routine:
                routine_return_value = RunRoutines(obj, prop);
                !print "[", routine_return_value, "]";
        }
    }
    if (routine_return_value) {
        say__p = 1;
        if (no_break == false) {
            new_line;
            !print "(DP->", say__p, say__pc, ")";
            DivideParagraphPoint();
            !print "(to", say__p, say__pc, ")";
        }
    }
    !print "(-->", say__p, say__pc, ")";
    return routine_return_value;
];
```

§14. Short Name Storage. None of the following functions should be called for the Z-machine if the short name exceeds the size of the following buffer: whereas the Glulx implementation of `VM_PrintToBuffer` will safely truncate overlong text, that's impossible for the Z-machine, and horrible results will follow.

`CPrintOrRun` is a variation on `PrintOrRun`, simplified by not needing to handle entire paragraphs (so, no fuss about dividing) but complicated by having to capitalise the first letter. We do this by writing to the buffer and then altering the first character.

```
Array StorageForShortName buffer 250;
[ CPrintOrRun obj prop  v length i;
    if ((obj ofclass String or Routine) || (prop == 0))
        VM_PrintToBuffer (StorageForShortName, 160, obj);
    else {
        if (obj.prop == NULL) rfalse;
        if (metaclass(obj.prop) == Routine or String)
            VM_PrintToBuffer(StorageForShortName, 160, obj, prop);
        else return RunTimeError(2, obj, prop);
    }
    length = StorageForShortName-->0;
    StorageForShortName->WORDSIZE = VM_LowerToUpperCase(StorageForShortName->WORDSIZE);
    for (i=WORDSIZE: i<length+WORDSIZE: i++) print (char) StorageForShortName->i;
    if (i>WORDSIZE) say__p = 1;
    return;
];
[ Cap str nocaps;
    if (nocaps) print (string) str;
    else CPrintOrRun(str, 0);
];
```

§15. Object Names I. We now begin the work of printing object names. In the lowest level of this process we print just the name itself (without articles attached), and we do it by carrying out an activity.

```
[ PSN__ o;
    if (o == 0) { print (string) NOTHING__TX; rtrue; }
    switch (metaclass(o)) {
        Routine:  print "<routine ", o, ">"; rtrue;
        String:   print "<string ~", (string) o, "~>"; rtrue;
        nothing:  print "<illegal object number ", o, ">"; rtrue;
    }
    CarryOutActivity(PRINTING_THE_NAME_ACT, o);
];
```

§16. Standard Name Printing Rule. In its initial state, the "printing the name of" activity has just one rule: the following "for" rule.

```
Global caps_mode = false;
[ STANDARD_NAME_PRINTING_R obj;
    obj = parameter_object;
    if (obj == 0) {
        print (string) NOTHING__TX; return;
    }
    switch (metaclass(obj)) {
        Routine:  print "<routine ", obj, ">"; return;
        String:   print "<string ~", (string) obj, "~>"; return;
        nothing:  print "<illegal object number ", obj, ">"; return;
    }
    if (obj == player) {
        if (indef_mode == NULL && caps_mode) print (string) YOU__TX;
        else print (string) YOURSELF__TX;
        return;
    }
    #Ifdef LanguagePrintShortName;
    if (LanguagePrintShortName(obj)) return;
    #Endif; ! LanguagePrintShortName
    if (indef_mode && obj.&short_name_indef ~= 0 &&
        PrintOrRun(obj, short_name_indef, true) ~= 0) return;
    if (caps_mode &&
        obj.&cap_short_name ~= 0 && PrintOrRun(obj, cap_short_name, true) ~= 0) {
        caps_mode = false;
        return;
    }
    if (obj.&short_name ~= 0 && PrintOrRun(obj, short_name, true) ~= 0) return;
    print (object) obj;
];
```

§17. Object Names II. The second level of the system for printing object names handles the placing of articles in front of them: *the* red herring, *an* elephant, *Some* bread. The following routine allows us to choose:

(a) `obj`, the object whose name is to be printed;
(b) `acode`, the kind of article needed: capitalised definite (0), lower case uncapitalised definite (1), or uncapitalised indefinite (2);
(c) `pluralise`, a flag forcing to a plural form (e.g., "some" being the pluralised form of an indefinite article in English);
(d) `capitalise`, a flag forcing us to capitalise the article – it's by setting this that we can achieve the fourth option missing from (b), viz., capitalised indefinite. (All of this is a legacy design from a time when the I6 library did not support capitalised indefinite articles.)

The routine then looks after issues such as which contraction form to use: for instance, in English, whether to use "a" or "an" for the indefinite singular depends on the text of the object's name.

```
Global short_name_case;
[ PrefaceByArticle obj acode pluralise capitalise  i artform findout artval;
    if (obj provides articles) {
        artval=(obj.&articles)-->(acode+short_name_case*LanguageCases);
        if (capitalise)
```

```
            print (Cap) artval, " ";
        else
            print (string) artval, " ";
        if (pluralise) return;
        print (PSN__) obj; return;
    }
    i = GetGNAOfObject(obj);
    if (pluralise) {
        if (i < 3 || (i >= 6 && i < 9)) i = i + 3;
    }
    i = LanguageGNAsToArticles-->i;
    artform = LanguageArticles
        + 3*WORDSIZE*LanguageContractionForms*(short_name_case + i*LanguageCases);
    #Iftrue (LanguageContractionForms == 2);
    if (artform-->acode ~= artform-->(acode+3)) findout = true;
    #Endif; ! LanguageContractionForms
    #Iftrue (LanguageContractionForms == 3);
    if (artform-->acode ~= artform-->(acode+3)) findout = true;
    if (artform-->(acode+3) ~= artform-->(acode+6)) findout = true;
    #Endif; ! LanguageContractionForms
    #Iftrue (LanguageContractionForms == 4);
    if (artform-->acode ~= artform-->(acode+3)) findout = true;
    if (artform-->(acode+3) ~= artform-->(acode+6)) findout = true;
    if (artform-->(acode+6) ~= artform-->(acode+9)) findout = true;
    #Endif; ! LanguageContractionForms
    #Iftrue (LanguageContractionForms > 4);
    findout = true;
    #Endif; ! LanguageContractionForms
    #Ifdef TARGET_ZCODE;
    if (standard_interpreter ~= 0 && findout) {
        StorageForShortName-->0 = 160;
        @output_stream 3 StorageForShortName;
        if (pluralise) print (number) pluralise; else print (PSN__) obj;
        @output_stream -3;
        acode = acode + 3*LanguageContraction(StorageForShortName + 2);
    }
    #Ifnot; ! TARGET_GLULX
    if (findout) {
        if (pluralise)
            Glulx_PrintAnyToArray(StorageForShortName, 160, EnglishNumber, pluralise);
        else
            Glulx_PrintAnyToArray(StorageForShortName, 160, PSN__, obj);
        acode = acode + 3*LanguageContraction(StorageForShortName);
    }
    #Endif; ! TARGET_
    Cap (artform-->acode, ~~capitalise); ! print article
    if (pluralise) return;
    print (PSN__) obj;
];
```

§18. Object Names III. The routines accessible from outside this segment.

```
[ IndefArt obj i;
    if (obj == 0) { print (string) NOTHING__TX; rtrue; }
    i = indef_mode; indef_mode = true;
    if (obj has proper) { indef_mode = NULL; print (PSN__) obj; indef_mode = i; return; }
    if ((obj provides article) && (obj.article ~= EMPTY_TEXT_VALUE)) {
        PrintOrRun(obj, article, true); print " ", (PSN__) obj; indef_mode = i;
        return;
    }
    PrefaceByArticle(obj, 2); indef_mode = i;
];
[ CIndefArt obj i;
    if (obj == 0) { CPrintOrRun(NOTHING__TX, 0); rtrue; }
    i = indef_mode; indef_mode = true;
    if (obj has proper) {
        indef_mode = NULL;
        caps_mode = true;
        print (PSN__) obj;
        indef_mode = i;
        caps_mode = false;
        return;
    }
    if ((obj provides article) && (obj.article ~= EMPTY_TEXT_VALUE)) {
        CPrintOrRun(obj, article); print " ", (PSN__) obj; indef_mode = i;
        return;
    }
    PrefaceByArticle(obj, 2, 0, 1); indef_mode = i;
];
[ DefArt obj i;
    i = indef_mode; indef_mode = false;
    if ((~~obj ofclass Object) || obj has proper) {
        indef_mode = NULL; print (PSN__) obj; indef_mode = i;
        return;
    }
    PrefaceByArticle(obj, 1); indef_mode = i;
];
[ CDefArt obj i;
    i = indef_mode; indef_mode = false;
    if ((obj ofclass Object) && (obj has proper || obj == player)) {
        indef_mode = NULL;
        caps_mode = true;
        print (PSN__) obj;
        indef_mode = i;
        caps_mode = false;
        return;
    }
    if ((~~obj ofclass Object) || obj has proper) {
        indef_mode = NULL; print (PSN__) obj; indef_mode = i;
        return;
    }
    PrefaceByArticle(obj, 0); indef_mode = i;
];
```

```
[ PrintShortName obj i;
    i = indef_mode; indef_mode = NULL;
    PSN__(obj); indef_mode = i;
];
```

§19. Say One Of. These routines are described in the Extensions chapter of the Inform documentation.

```
[ I7_SOO_PAR oldval count; if (count <= 1) return count; return random(count); ];
[ I7_SOO_RAN oldval count v; if (count <= 1) return count;
    v = oldval; while (v == oldval) v = random(count); return v; ];
[ I7_SOO_STI oldval count v; if (oldval) return oldval; return I7_SOO_PAR(oldval, count); ];
[ I7_SOO_CYC oldval count; oldval++; if (oldval > count) oldval = 1; return oldval; ];
[ I7_SOO_STOP oldval count; oldval++; if (oldval > count) oldval = count; return oldval; ];
[ I7_SOO_TAP oldval count tn rn c; if (count <= 1) return count; tn = count*(count+1)/2;
    rn = random(tn); for (c=1:c<=count:c++) { rn = rn - c; if (rn<=0) return (count-c+1); } ];
[ I7_SOO_TRAN oldval count; if (oldval<count) return oldval+1;
    return count + 1 + I7_SOO_RAN(oldval%(count+1), count); ];
[ I7_SOO_TPAR oldval count; if (oldval<count) return oldval+1;
    return count + 1 + I7_SOO_PAR(oldval%(count+1), count); ];

Array I7_SOO_SHUF->32;

[ I7_SOO_SHU oldval count sd ct v i j s ssd scope cc base;
    base = count+1;
    v = oldval%base; oldval = oldval/base; ct = oldval%base; sd = oldval/base;
    if (count > 32) return I7_SOO_PAR(oldval, count);
    if (count <= 1) v = count;
    else {
        !print "^In v=", v, " ct=", ct, " sd=", sd, "^";
        cc = base*base;
        scope = (MAX_POSITIVE_NUMBER-1)/cc;
        !print "Scope = ", scope, "^";
        if (sd == 0) { sd = random(scope); ct=0; }
        for (i=0:i<count:i++) I7_SOO_SHUF->i = i;
        ssd = sd;
        for (i=0:i<count-1:i++) {
            j = (sd)%(count-i)+i; sd = (sd*31973)+17; if (sd<0) sd=-sd;
            s = I7_SOO_SHUF->j; I7_SOO_SHUF->j = I7_SOO_SHUF->i; I7_SOO_SHUF->i = s;
        }
        !for (i=0:i<count:i++) print I7_SOO_SHUF->i, " "; print "^";
        v = (I7_SOO_SHUF->ct)+1;
        ct++; if (ct >= count) { ct = 0; ssd = 0; }
    }
    !print "Out v=", v, " ct=", ct, " ssd=", sd, "^";
    !print "Return ", v + ct*base + ssd*base*base, "^";
    return v + ct*base + ssd*base*base;
];
```

RTP Template B/rtpt

Purpose

To issue run-time problem messages, and to perform some run-time type checking which may issue such messages.

§**1. Reporting.** All RTPs are produced by calling the following routine, which takes one compulsory argument: `n`, the RTP number. When I7 is being used with the Inform user interface, it's important that these numbers correspond to the explanatory web pages stored within the interface application: those in turn are created by the `inrtps` utility. The interface knows to display these pages because it parses the printed output during play to look for the text layout produced by the following routine: so do not reformat RTPs without ensuring that corresponding changes have been made to the Inform user interface applications.

The arguments `par1`, `par2` and `par3` are optional parameters clarifying the message; `ln` is an optional parameter specifying a paragraph number in the original source text (though in fact I7 does not use this at present).

```
[ RunTimeProblem n par1 par2 par3 ln   i c;
    if (enable_rte == false) return;
    enable_rte = false;
    print "^*** Run-time problem P", n;
    if (ln) print " (at paragraph ", ln, " in the source text)";
    print ": ";
    switch(n) {
        RTP_BACKDROP:
            print "Tried to move ", (the) par1, " (a backdrop) to ", (the) par2,
                ", which is not a region.^";
        RTP_CANTCHANGE:
            print "Tried to change player to ", (the) par1,
                ", which is not a player-character.^";
        RTP_NOEXIT:
            print "Tried to change ", (the) par2, " exit of ", (the) par1,
                ", but it didn't seem to have such an exit to change.^";
        RTP_EXITDOOR:
            print "Tried to change ", (the) par2, " exit of ", (the) par1,
                ", but it led to a door, not a room.^";
        RTP_IMPREL:
            print "Tried to access an inappropriate relation for ", (the) par1,
                ", violating '", (string) par2-->RR_DESCRIPTION, "'.^";
        RTP_RULESTACK:
            print "Too many procedural rules acting all at once.^";
        RTP_TOOMANYRULEBOOKS:
            print "Too many rulebooks in simultaneous use.^";
        RTP_TOOMANYEVENTS:
            print "Too many timed events are going on at once.^";
        RTP_BADPROPERTY:
            print "Tried to access non-existent property for ", (the) par1, ".^";
        RTP_UNPROVIDED:
```

```
    print "Since ", (the) par1, " is not allowed the property ~",
            (string) par2, "~, it is against the rules to try to use it.^";
RTP_UNSET:
    print "Although ", (the) par1, " is allowed to have the property ~",
        (string) par2, "~, no value was ever given, so it can't now be used.^";
RTP_TOOMANYACTS:
    print "Too many activities are going on at once.^";
RTP_CANTABANDON:
    print "Tried to abandon an activity which wasn't going on.^";
RTP_CANTEND:
    print "Tried to end an activity which wasn't going on.^";
RTP_CANTMOVENOTHING:
    print "You can't move nothing.^";
RTP_CANTREMOVENOTHING:
    print "You can't remove nothing from play.^";
RTP_DIVZERO:
    print "You can't divide by zero.^";
RTP_BADVALUEPROPERTY:
    print "Tried to access property for a value which didn't fit: ",
        "if this were a number it would be ", par1, ".^";
RTP_NOTBACKDROP:
    print "Tried to move ", (the) par1, " (not a backdrop) to ", (the) par2,
        ", which is a region.^";
RTP_TABLE_NOCOL:
    print "Attempt to look up a non-existent column in the table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_NOCORR:
    print "Attempt to look up a non-existent correspondence in the table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_NOROW:
    print "Attempt to look up a non-existent row in the table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_NOENTRY:
    print "Attempt to look up a non-existent entry at column ", par2,
        ", row ", par3, " of the table '", (PrintTableName) par1, "'.^";
RTP_TABLE_NOTABLE:
    print "Attempt to blank out a row from a non-existent table (value ",
        par1, ").^";
RTP_TABLE_NOMOREBLANKS:
    print "Attempt to choose a blank row in a table with none left: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_NOROWS:
    print "Attempt to choose a random row in an entirely blank table: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_CANTRUNTHROUGH:
    print "Attempt to repeat through a table in a tricky column order: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_CANTSORT:
    print "Attempt to sort a table whose ordering must remain fixed: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_CANTSAVE:
    print "Attempt to save a table to a file whose data is unstable: table '",
        (PrintTableName) par1, "'.^";
```

```
RTP_TABLE_WONTFIT:
    print "File being read has too many rows or columns to fit into table: table '",
        (PrintTableName) par1, "'.^";
RTP_TABLE_BADFILE:
    print "File being read is not a previously saved table: table '",
        (PrintTableName) par1, "'.^";
RTP_NOTINAROOM:
    print "Attempt to test if the current location is '",
        (the) par1, "', which is not a room or region.^";
RTP_BADTOPIC:
    print "Attempt to see if a snippet of text matches something which
        is not a topic.^";
RTP_ROUTELESS:
    print "Attempt to find route or count steps through an implicit
        relation.^";
RTP_PROPOFNOTHING:
    print "Attempt to use a property of the 'nothing' non-object: property ",
        (PrintPropertyName) par2, "^";
RTP_DECIDEONWRONGKIND:
    print "Attempt to 'decide on V' where V is the wrong kind of object.^";
RTP_DECIDEONNOTHING:
    print "Attempt to 'decide on nothing'.^";
RTP_LOWLEVELERROR:
    print "Low level error.^";
RTP_DONTIGNORETURNSEQUENCE:
    print "Attempt to ignore the turn sequence rules.^";
RTP_SAYINVALIDSNIPPET:
    print "Attempt to say a snippet value which is currently invalid: words ",
        par1, " to ", par2, ".^";
RTP_SPLICEINVALIDSNIPPET:
    print "Attempt to splice a snippet value which is currently invalid: words ",
        par1, " to ", par2, ".^";
RTP_INCLUDEINVALIDSNIPPET:
    print "Attempt to match a snippet value which is currently invalid: words ",
        par1, " to ", par2, ".^";
RTP_LISTWRITERMEMORY:
    print "The list-writer has run out of memory.^";
RTP_CANTREMOVEPLAYER:
    print "Attempt to remove the player from play.^";
RTP_CANTREMOVEDOORS:
    print "Attempt to remove a door from play.^";
RTP_CANTCHANGEOFFSTAGE:
    print "Attempt to change the player to a person off-stage.^";
RTP_MSTACKMEMORY:
    print "The memory stack is exhausted.^";
RTP_TYPECHECK:
    print "Phrase applied to an incompatible kind of value.^";
RTP_FILEIOERROR:
    print "Error handling external file.^";
RTP_HEAPERROR:
    print "Memory allocation proved impossible.^";
RTP_LISTRANGEERROR:
    print "Attempt to use list item which does not exist.^";
```

```
        RTP_REGEXPSYNTAXERROR:
            print "Syntax error in regular expression.^";
        RTP_NOGLULXUNICODE:
            print "This interpreter does not support Unicode.^";
        RTP_BACKDROPONLY:
            print "Only backdrops can be moved to multiple places.^";
        RTP_NOTBACKDROP:
            print "Tried to move ", (the) par1, " (not a thing) to ", (the) par2,
                ", but only things can move around.^";
        RTP_SCENEHASNTSTARTED:
            print "The scene ", (PrintSceneName) par1,
                " hasn't started, so you can't ask when it did.^";
        RTP_SCENEHASNTENDED:
            print "The scene ", (PrintSceneName) par1,
                " hasn't ended, so you can't ask when it did.^";
        RTP_NEGATIVEROOT:
            print "You can't take the square root of a negative number.^";
        RTP_CANTITERATE:
            print "You can't implicitly repeat through the values of this kind: ",
                "a problem arising from a description which started out here - ~",
                (string) par1, "~.^";
        RTP_WRONGASSIGNEDKIND:
            print "Attempt to set a variable to the wrong kind of object: ",
                "you wrote '", (string) par2, "', which sets the value to ", (the) par1,
                " - but that doesn't have the kind '", (string) par3, "'.^";
        }
    print "^";
];
```

§2. Low-Level Errors. The following is a residue from the old I6 library, and most of its possible messages can't be seen in I7 use – for instance I7 has no timers or daemons, so error 4 is out of the question. But we retain the routine because the veneer code added by I6 requires it to be present.

```
Constant MAX_TIMERS = 0;
[ RunTimeError n p1 p2;
    #Ifdef DEBUG;
    print "** Library error ", n, " (", p1, ",", p2, ") **^** ";
    switch (n) {
    1:    print "preposition not found (this should not occur)";
    2:    print "Property value not routine or string: ~", (property) p2, "~ of ~", (name) p1,
                "~ (", p1, ")";
    3:    print "Entry in property list not routine or string: ~", (property) p2, "~ list of ~",
                (name) p1, "~ (", p1, ")";
    4:    print "Too many timers/daemons are active simultaneously.
                The limit is the library constant MAX_TIMERS (currently ",
                MAX_TIMERS, ") and should be increased";
    5:    print "Object ~", (name) p1, "~ has no ~time_left~ property";
    7:    print "The object ~", (name) p1, "~ can only be used as a player object if it has
                the ~number~ property";
    8:    print "Attempt to take random entry from an empty table array";
    9:    print p1, " is not a valid direction property number";
    10:   print "The player-object is outside the object tree";
    11:   print "The room ~", (name) p1, "~ has no ~description~ property";
    12:   print "Tried to set a non-existent pronoun using SetPronoun";
    13:   print "A 'topic' token can only be followed by a preposition";
    default: print "(unexplained)";
    }
    print " **^";
    #Ifnot;
    print "** Library error ", n, " (", p1, ",", p2, ") **^";
    #Endif; ! DEBUG
    RunTimeProblem(RTP_LOWLEVELERROR);
];
```

§**3. Argument Type Checking Failed.** This is called when run-time type checking for the argument of a phrase fails, so that no definition for the phrase can be applied.

```
[ ArgumentTypeFailed file line arg;
    RunTimeProblem(RTP_TYPECHECK, 0, 0, 0, line);
];
```

§**4. Return Type Checking Failed.** Similarly, though in a more restricted set of circumstances. NI can usually prove that the value returned by a phrase matches its supposed kind, but because of the deliberately weak type-checking within the kind of value "object" it will allow a broader kind of object to be returned when the requirement is for a narrower one. In such cases, it checks the result with the following routine. The value `V` is a valid "object", but that means it can be `nothing`, and we must check that it matches a given I7 kind `K` (where `nothing` is not valid).

```
[ CheckKindReturned V K;
    if (V ofclass K) return V;
    if (v == nothing) RunTimeProblem(RTP_DECIDEONNOTHING);
    else RunTimeProblem(RTP_DECIDEONWRONGKIND);
    return V;
];
```

§**5. Whether Provides.** This routine defines the phrase "if O provides P": there are three tests to pass, and if any of the three fail, we return `false`. (The `issue_rtp` flag, causing RTPs to be issued depending on which test fails, is never set when the routine is simply testing the condition.)

Firstly, P has to be a property known to I7. Secondly, there has to be permission either for this individual object to have it, or for its kind to have it, or its kind's kind, and so on. Thirdly, the object has to actually have the property in question at an I6 level – having permission to have it doesn't mean it actually does have.

```
[ WhetherProvides obj either_or p issue_rtp  off i textual a l;
    if (metaclass(obj) ~= Object) rfalse;
    if (p<0) p = ~p;
    if (either_or) {
        if (p < FBNA_PROP_NUMBER) off = attributed_property_offsets-->p;
        else off = valued_property_offsets-->p;
    } else off = valued_property_offsets-->p;
    if (off<0) {
        if (issue_rtp) RunTimeProblem(RTP_BADPROPERTY, obj);
        rfalse;
    }
    textual = property_metadata-->off; off++;
    if (ScanPropertyMetadata(obj, off)) jump PermissionFound;
    if (obj provides KD_Count) {
        l = obj.KD_Count;
        while (l > 0) {
            a = l*2;
            if (ScanPropertyMetadata(KindHierarchy-->a, off)) jump PermissionFound;
            l = KindHierarchy-->(a+1);
        }
    }
    if (issue_rtp) RunTimeProblem(RTP_UNPROVIDED, obj, textual);
    rfalse;
    .PermissionFound;
        if (either_or) rtrue;
        if (obj provides p) rtrue;
        if (issue_rtp) RunTimeProblem(RTP_UNSET, obj, textual);
        rfalse;
];
[ PrintPropertyName  p  off textual;
    if (p<0) p = ~p;
    off = valued_property_offsets-->p;
    textual = property_metadata-->off;
    print (string) textual;
];
```

§**6. Scan Property Metadata.** The `property_metadata` table is a series of zero-terminated lists of objects (or class objects, representing I7 kinds). Each list corresponds to a single property; the position in the table is called the "offset" for the property. The following searches from a given offset.

```
[ ScanPropertyMetadata obj off i;
    for (i=off: property_metadata-->i >= 0: i++)
        if (obj == property_metadata-->i) rtrue;
    rfalse;
];
```

§**7. Get Either-Or Property.** If `p` represents a property, then `~p` (its bitwise negation) represents its logical negation. The bitwise negation will change the sign bit; since all properties are ordinarily positive, we can detect bitwise negation by looking for negative property numbers. (This could fail on Glulx story files above 1GB in size, but that's about 1000 times the size of the record-sized file created thus far.)

FBNA stands for First Boolean Not to be an Attribute, in the I6 sense. Some either/or (i.e., boolean) properties are stored as I6 attributes, others as I6 properties whose values are always `true` or `false`.

Note that we allow either/or properties to be *read* for any object, regardless of permissions, returning `false` if the object does not have the property. This is so that a description such as "open things" can be applied against any object without run-time errors, even though it is only normally valid for doors and containers.

```
[ GetEitherOrProperty o p;
    if (o == nothing) rfalse;
    if (p<0) p = ~p;
    if (WhetherProvides(o, true, p, false)) {
        if (p<FBNA_PROP_NUMBER) { if (o has p) rtrue; rfalse; }
        if ((o provides p) && (o.p)) rtrue;
    }
    rfalse;
];
```

§**8. Set Either-Or Property.** An attempt to *write* an either/or property which is not provided will, however, always produce a run-time problem.

```
[ SetEitherOrProperty o p negate adj;
    if (p<0) { p = ~p; negate = ~negate; }
    if (adj) {
        (adj)(o);
    } else if (WhetherProvides(o, true, p, true)) {
        if (negate) {
            if (p<FBNA_PROP_NUMBER) give o ~p; else o.p = false;
        } else {
            if (p<FBNA_PROP_NUMBER) give o p; else o.p = true;
        }
    }
];
```

§**9. Value Property.** Some value properties belong to other values (those created in tables), and these are detected by being properties of the special `ValuePropertyHolder` pseudo-object – an I6 object which is not part of the world model, and not a valid I7 "object" value, but which is used in order that properties belonging to values are still I6 property numbers. `ValuePropertyHolder.P` is the table column address for this property; `obj` is then a value for the kind of value created by the table, so it is used as an index into the table column to get the address of the memory location storing the property value.

The `door_to` property, relevant only for doors, is called rather than read: this enables it to be an I6 routine returning the other side of the door from the one which the player is on.

```
[ GProperty K V pr obj;
    if (K == OBJECT_TY) obj = V; else obj = KOV_representatives-->K;
    if (obj == 0) { RunTimeProblem(RTP_PROPOFNOTHING, obj, pr); rfalse; }
    if (obj provides pr) {
        if (K == OBJECT_TY) {
            if (pr == door_to) return obj.pr();
            if (WhetherProvides(V, false, pr, true)) return obj.pr;
            rfalse;
        }
        if (obj ofclass KO_kind)
            WhetherProvides(V, false, pr, true); ! to force a run-time problem
        if ((V < 1) || (V > obj.value_range)) {
            RunTimeProblem(RTP_BADVALUEPROPERTY); return 0; }
        return (obj.pr)-->(V+COL_HSIZE);
    } else {
        if (obj ofclass KO_kind)
            WhetherProvides(V, false, pr, true); ! to force a run-time problem
    }
    rfalse;
];
```

§**10. Write Value Property.** This routine's name must consist of the read-value-property routine's name with the prefix `Write`, as that is how a reference to such a property is converted from an rvalue to an lvalue.

```
[ WriteGProperty K V pr val obj;
    if (K == OBJECT_TY) obj = V; else obj = KOV_representatives-->K;
    if (obj == 0) { RunTimeProblem(RTP_PROPOFNOTHING, obj, pr); rfalse; }
    if (K == OBJECT_TY) {
        if (WhetherProvides(V, false, pr, true)) obj.pr = val;
    } else {
        if ((V < 1) || (V > obj.value_range))
            return RunTimeProblem(RTP_BADVALUEPROPERTY);
        if (obj provides pr) { (obj.pr)-->(V+COL_HSIZE) = val; }
    }
];
```

§**11. Printing Property Names.** Inform doesn't print property names prettily; it more or less prints them only as decimal numbers.

```
[ PROPERTY_TY_Say v;
    print "property ", v;
];
```

Utilities Template B/utilt

Purpose

Miscellaneous utility routines for some fundamental I6 needs.

§1. Saying Phrases.

```
[ SayPhraseName closure;
    if (closure == 0) print "nothing";
    else print (string) closure-->2;
];
```

§2. Kinds.

```
[ KindAtomic kind;
    if ((kind >= 0) && (kind < BASE_KIND_HWM)) return kind;
    return kind-->0;
];
[ KindBaseArity kind;
    if ((kind >= 0) && (kind < BASE_KIND_HWM)) return 0;
    return kind-->1;
];
[ KindBaseTerm kind n;
    if ((kind >= 0) && (kind < BASE_KIND_HWM)) return UNKNOWN_TY;
    return kind-->(2+n);
];
```

§3. DigitToValue.

This takes a ZSCII or Unicode character code and returns its value as a digit, or returns -1 if it isn't a digit.

```
[ DigitToValue c n;
    n = c-'0';
    if ((n<0) || (n>9)) return -1;
    return n;
];
```

§**4. GenerateRandomNumber.** The following uses the virtual machine's RNG (via the I6 built-in function `random`) to produce a uniformly random integer in the range n to m inclusive, where n and m are allowed to be either way around; so that a random number between 17 and 4 is the same thing as a random number between 4 and 17, and there is therefore no pair of n and m corresponding to an empty range of values.

```
[ GenerateRandomNumber n m s;
    if (n==m) return n;
    if (n>m) { s = n; n = m; m = s; }
    n--;
    return random(m-n) + n;
];
Constant R_DecimalNumber = GenerateRandomNumber;
Constant R_PrintTimeOfDay = GenerateRandomNumber;
```

§**5. GroupChildren.** The following runs through the child-objects of `par` in the I6 object tree, and moves those having a given property value together, to become the eldest children. It preserves the ordering in between those objects, and also in between those not having that property value. The property is required to be a common property.

We do this by temporarily moving objects into and out of `in_obj` and `out_obj`, objects which in all other circumstances never have children in the tree.

```
[ GroupChildren par prop value;
    while (child(par) ~= 0) {
        if (child(par).prop ~= value) move child(par) to out_obj;
        else move child(par) to in_obj;
    }
    while (child(in_obj) ~= 0)  move child(in_obj) to par;
    while (child(out_obj) ~= 0) move child(out_obj) to par;
    return child(par);
];
```

§**6. PrintSpaces.** Which prints a row of n spaces, for $n \geq 0$.

```
[ PrintSpaces n;
    while (n > 0) {
        print " ";
        n = n - 1;
    }
];
```

§7. RunRoutines. This function may not be very well-named, but the idea is to take a property of a given object and either to print it (and return `true`) if it's a string, and call it (and pass along its return value) if it's a routine. If the object does not provide the property, we act on the default value for the property if it has one, and otherwise do nothing (and return `false`).

The I6 pseudo-object `thedark` is used to give the impression that Darkness is a room in its own right, which is not really true. Note that it is not permitted to have properties other than the three named here: all other properties are redirected to the current location's object.

Properties with numbers under `INDIV_PROP_START` are "common properties". These come along with a table of default values, so that it is meaningful (in I6, anyway) to read them even when they are not provided (so that the address, returned by the `.&` operator, is zero).

```
[ RunRoutines obj prop;
    if (obj == thedark) obj = real_location;
    if ((obj.&prop == 0) && (prop >= INDIV_PROP_START)) rfalse;
    return obj.prop();
];
```

§8. SwapWorkflags. Recall that we have two general-purpose temporary attributes for each object: `workflag` and `workflag2`. The following swaps their values over for every object at once.

```
[ SwapWorkflags obj lst;
    objectloop (obj ofclass Object) {
        lst = false;
        if (obj has workflag2) lst = true;
        give obj ~workflag2;
        if (obj has workflag) give obj workflag2;
        give obj ~workflag;
        if (lst) give obj workflag;
    }
];
```

§9. TestUseOption. This routine, compiled by NI, returns `true` if the supplied argument is the number of a use option in force for the current run of NI, and `false` otherwise.

```
{-routine:Config::Inclusions::UseOptions::TestUseOption}
```

§10. IntegerDivide. We can't simply use I6's `/` operator, as that translates directly into a virtual machine opcode which crashes on divide by zero.

```
[ IntegerDivide A B;
    if (B == 0) { RunTimeProblem(RTP_DIVZERO); rfalse; }
    return A/B;
];
```

§11. IntegerRemainder. Similarly.

```
[ IntegerRemainder A B;
    if (B == 0) { RunTimeProblem(RTP_DIVZERO); rfalse; }
    return A%B;
];
```

§**12. UnsignedCompare.** Comparison of I6 integers is normally signed, that is, treating the word as a twos-complement signed number, so that `$FFFF` is less than `0`, for instance. If we want to construe words as being unsigned integers, or as addresses, we need to compare them with the following routine, which returns 1 if $x > y$, 0 if $x = y$ and -1 if $x < y$.

```
[ UnsignedCompare x y u v;
    if (x == y) return 0;
    if (x < 0 && y >= 0) return 1;
    if (x >= 0 && y < 0) return -1;
    u = x&~WORD_HIGHBIT; v= y&~WORD_HIGHBIT;
    if (u > v) return 1;
    return -1;
];
```

§**13. ZRegion.** I7 contains many relics from I6, but here's a relic from I5: a routine which used to determine the metaclass of a value, before that concept was given a name. In I6 code, it can be implemented simply using `metaclass`, as the following shows. (The name is from "region of the Z-machine".)

```
[ ZRegion addr;
    switch (metaclass(addr)) {
        nothing: return 0;
        Object, Class: return 1;
        Routine: return 2;
        String: return 3;
    }
];
```

§**14. Library Messages.** This is another fossil, and probably soon to change.

```
[ GL__M a b c d;
    if ((actor ~= player) || (untouchable_silence)) rtrue;
    return L__M(a,b,c,d); ];
[ AGL__M a b c d;
    if (untouchable_silence) rtrue;
    return L__M(a,b,c,d); ];
```

Number Template B/numt

Purpose

Support for parsing integers.

§1. Understanding. In our target virtual machines, numbers are stored in twos-complement form, so that a 16-bit VM can hold the range of integers $-2^{15} = -32768$ to $2^{15} - 1 = +32767$, while a 32-bit VM can hold $-2^{31} = -2147483648$ to $2^{31} - 1 = +2147483647$: the token below accepts exactly those ranges.

```
[ DECIMAL_TOKEN wnc wna r n wa wl sign base digit digit_count original_wn group_wn;
    wnc = wn; original_wn = wn; group_wn = wn;
{-call:Plugins::Parsing::Tokens::Values::number}
    wn = wnc;
    r = ParseTokenStopped(ELEMENTARY_TT, NUMBER_TOKEN);
    if ((r == GPR_NUMBER) && (parsed_number ~= 10000)) return r;
    wn = wnc;
    wa = WordAddress(wn);
    wl = WordLength(wn);
    sign = 1; base = 10; digit_count = 0;
    if (wa->0 ~= '-' or '$' or '0' or '1' or '2' or '3' or '4'
        or '5' or '6' or '7' or '8' or '9')
        return GPR_FAIL;
    if (wa->0 == '-') { sign = -1; wl--; wa++; }
    if (wl == 0) return GPR_FAIL;
    n = 0;
    while (wl > 0) {
        if (wa->0 >= 'a') digit = wa->0 - 'a' + 10;
        else digit = wa->0 - '0';
        digit_count++;
        switch (base) {
            2:  if (digit_count == 17) return GPR_FAIL;
            10:
                #Iftrue (WORDSIZE == 2);
                if (digit_count == 6) return GPR_FAIL;
                if (digit_count == 5) {
                    if (n > 3276) return GPR_FAIL;
                    if (n == 3276) {
                        if (sign == 1 && digit > 7) return GPR_FAIL;
                        if (sign == -1 && digit > 8) return GPR_FAIL;
                    }
                }
                #Ifnot; ! i.e., if (WORDSIZE == 4)
                if (digit_count == 11) return GPR_FAIL;
                if (digit_count == 10) {
                    if (n > 214748364) return GPR_FAIL;
                    if (n == 214748364) {
                        if (sign == 1 && digit > 7) return GPR_FAIL;
                        if (sign == -1 && digit > 8) return GPR_FAIL;
                    }
                }
                #Endif;
            16: if (digit_count == 5) return GPR_FAIL;
        }
        if (digit >= 0 && digit < base) n = base*n + digit;
        else return GPR_FAIL;
        wl--; wa++;
    }
    parsed_number = n*sign; wn++;
    return GPR_NUMBER;
];
```

§**2. Truth states.** And although truth states are not strictly speaking numbers, this seems as good a point as any to parse them:

```
[ TRUTH_STATE_TOKEN original_wn wd;
    original_wn = wn;
{-call:Plugins::Parsing::Tokens::Values::truth_state}
    wn = original_wn;
    wd = NextWordStopped();
    if (wd == 'true') { parsed_number = 1; return GPR_NUMBER; }
    if (wd == 'false') { parsed_number = 0; return GPR_NUMBER; }
    wn = original_wn;
    return GPR_FAIL;
];
```

Time Template B/timet

Purpose

Support for parsing and printing times of day.

§**1. Rounding.** The following rounds a numerical value `t1` to the nearest unit of `t2`; for instance, if `t2` is 5 then it rounds to the nearest 5.

```
[ RoundOffTime t1 t2; return ((t1+t2/2)/t2)*t2; ];
```

§**2. Square Root.** This is an old algorithm for extracting binary square roots, taking 2 bits at a time. We start out with `one` at the highest bit which isn't the sign bit; that used to be worked out as `WORD_HIGHBIT/2`, but this caused unexpected portability problems (exposing a minor bug in Inform and also glulxe) because of differences in how C compilers handle signed division of constants in the case where the dividend is -2^{31}, the unique number which cannot be negated in 32-bit twos complement arithmetic.

```
[ SquareRoot num
    op res one;
    op = num;
    if (num < 0) { RunTimeProblem(RTP_NEGATIVEROOT); return 1; }
    ! "one" starts at the highest power of four <= the argument.
    for (one = WORD_NEXTTOHIGHBIT: one > op: one = one/4) ;
    while (one ~= 0) {
        !print "Round: op = ", op, " res = ", res, ", res**2 = ", res*res, " one = ", one, "^";
        if (op >= res + one) {
            op = op - res - one;
            res = res + one*2;
        }
        res = res/2;
        one = one/4;
    }
    !print "Res is ", res, "^";
    return res;
];
```

§**3. Cube Root.** The following is an iterative scheme for finding cube roots by Newton-Raphson approximation, not a great method but which, on the narrow ranges of integers we deal with, is good enough. The square root is used only as a sighting shot.

```
[ CubeRoot num x y n;
    if (num < 0) x = -SquareRoot(-num); else x = SquareRoot(num);
    for (n=0: (y ~= x) && (n++ < 100): y = x, x = (2*x + num/x/x)/3) ;
    return x;
];
```

§4. Digital Printing. For instance, "2:06 am".

```
[ PrintTimeOfDay t h aop;
    if (t<0) { print "<no time>"; return; }
    if (t >= TWELVE_HOURS) { aop = "pm"; t = t - TWELVE_HOURS; } else aop = "am";
    h = t/ONE_HOUR; if (h==0) h=12;
    print h, ":";
    if (t%ONE_HOUR < 10) print "0"; print t%ONE_HOUR, " ", (string) aop;
];
```

§5. Analogue Printing. For instance, "six minutes past two".

```
[ PrintTimeOfDayEnglish t h m dir aop;
    h = (t/ONE_HOUR) % 12; m = t%ONE_HOUR; if (h==0) h=12;
    if (m==0) { print (number) h, " o'clock"; return; }
    dir = "past";
    if (m > HALF_HOUR) { m = ONE_HOUR-m; h = (h+1)%12; if (h==0) h=12; dir = "to"; }
    switch(m) {
        QUARTER_HOUR: print "quarter"; HALF_HOUR: print "half";
        default: print (number) m;
            if (m%5 ~= 0) {
                if (m == 1) print " minute"; else print " minutes";
            }
    }
    print " ", (string) dir, " ", (number) h;
];
```

§6. Understanding. This I6 grammar token converts words in the player's command to a valid I7 time, and is heavily based on the one presented as a solution to an exercise in the DM4.

```
[ TIME_TOKEN first_word second_word at length flag
    illegal_char offhour hr mn i original_wn;
    original_wn = wn;
{-call:Plugins::Parsing::Tokens::Values::time}
    wn = original_wn;
    first_word = NextWordStopped();
    switch (first_word) {
        'midnight': parsed_number = 0; return GPR_NUMBER;
        'midday', 'noon': parsed_number = TWELVE_HOURS;
        return GPR_NUMBER;
    }
    ! Next try the format 12:02
    at = WordAddress(wn-1); length = WordLength(wn-1);
    for (i=0: i<length: i++) {
        switch (at->i) {
            ':': if (flag == false && i>0 && i<length-1) flag = true;
            else illegal_char = true;
            '0', '1', '2', '3', '4', '5', '6', '7', '8', '9': ;
            default: illegal_char = true;
        }
    }
    if (length < 3 || length > 5 || illegal_char) flag = false;
    if (flag) {
        for (i=0: at->i~=':': i++, hr=hr*10) hr = hr + at->i - '0';
        hr = hr/10;
        for (i++: i<length: i++, mn=mn*10) mn = mn + at->i - '0';
        mn = mn/10;
        second_word = NextWordStopped();
        parsed_number = HoursMinsWordToTime(hr, mn, second_word);
        if (parsed_number == -1) return GPR_FAIL;
        if (second_word ~= 'pm' or 'am') wn--;
        return GPR_NUMBER;
    }
    ! Lastly the wordy format
    offhour = -1;
    if (first_word == 'half') offhour = HALF_HOUR;
    if (first_word == 'quarter') offhour = QUARTER_HOUR;
    if (offhour < 0) offhour = TryNumber(wn-1);
    if (offhour < 0 || offhour >= ONE_HOUR) return GPR_FAIL;
    second_word = NextWordStopped();
    switch (second_word) {
        ! "six o'clock", "six"
        'o^clock', 'am', 'pm', -1:
            hr = offhour; if (hr > 12) return GPR_FAIL;
        ! "quarter to six", "twenty past midnight"
        'to', 'past':
            mn = offhour; hr = TryNumber(wn);
            if (hr <= 0) {
                switch (NextWordStopped()) {
                    'noon', 'midday': hr = 12;
                    'midnight': hr = 0;
                    default: return GPR_FAIL;
                }
            }
            if (hr >= 13) return GPR_FAIL;
            if (second_word == 'to') {
                mn = ONE_HOUR-mn; hr--; if (hr<0) hr=23;
            }
            wn++; second_word = NextWordStopped();
        ! "six thirty"
        default:
            hr = offhour; mn = TryNumber(--wn);
            if (mn < 0 || mn >= ONE_HOUR) return GPR_FAIL;
            wn++; second_word = NextWordStopped();
    }
    parsed_number = HoursMinsWordToTime(hr, mn, second_word);
    if (parsed_number < 0) return GPR_FAIL;
    if (second_word ~= 'pm' or 'am' or 'o^clock') wn--;
    return GPR_NUMBER;
];
[ HoursMinsWordToTime hour minute word x;
    if (hour >= 24) return -1;
    if (minute >= ONE_HOUR) return -1;
    x = hour*ONE_HOUR + minute; if (hour >= 13) return x;
    x = x % TWELVE_HOURS; if (word == 'pm') x = x + TWELVE_HOURS;
    if (word ~= 'am' or 'pm' && hour == 12) x = x + TWELVE_HOURS;
    return x;
];
```

§**7. Relative Time Token.** "Time" is an interesting kind of value since it can hold two conceptually different ways of thinking about time: absolute times, such as "12:03 PM", and also relative times, like "ten minutes". For parsing purposes, these are completely different from each other, and the time token above handles only absolute times; we need the following for relative ones.

```
[ RELATIVE_TIME_TOKEN first_word second_word offhour mult mn original_wn;
    original_wn = wn;
    wn = original_wn;
    first_word = NextWordStopped(); wn--;
    if (first_word == 'an' or 'a//') mn=1; else mn=TryNumber(wn);
    if (mn == -1000) {
        first_word = NextWordStopped();
        if (first_word == 'half') offhour = HALF_HOUR;
        if (first_word == 'quarter') offhour = QUARTER_HOUR;
        if (offhour > 0) {
            second_word = NextWordStopped();
            if (second_word == 'of') second_word = NextWordStopped();
            if (second_word == 'an') second_word = NextWordStopped();
            if (second_word == 'hour') {
                parsed_number = offhour;
                return GPR_NUMBER;
            }
        }
        return GPR_FAIL;
    }
    wn++;
    first_word = NextWordStopped();
    switch (first_word) {
        'minutes', 'minute': mult = 1;
        'hours', 'hour': mult = 60;
        default: return GPR_FAIL;
    }
    parsed_number = mn*mult;
    if (mult == 60) {
        mn=TryNumber(wn);
        if (mn ~= -1000) {
            wn++;
            first_word = NextWordStopped();
            if (first_word == 'minutes' or 'minute')
                parsed_number = parsed_number + mn;
            else wn = wn - 2;
        }
    }
    return GPR_NUMBER;
];
```

§8. During Scene Matching.

```
[ DuringSceneMatching prop sc;
    for (sc=0: sc<NUMBER_SCENES_CREATED: sc++)
        if ((scene_status-->sc == 1) && (prop(sc+1))) rtrue;
    rfalse;
];
```

§9. Scene Questions.

```
[ SceneUtility sc task;
    if (sc <= 0) return 0;
    if (task == 1 or 2) {
        if (scene_endings-->(sc-1) == 0) return RunTimeProblem(RTP_SCENEHASNTSTARTED, sc);
    } else {
        if (scene_endings-->(sc-1) <= 1) return RunTimeProblem(RTP_SCENEHASNTENDED, sc);
    }
    switch (task) {
        1: return (the_time - scene_started-->(sc-1))%(TWENTY_FOUR_HOURS);
        2: return scene_started-->(sc-1);
        3: return (the_time - scene_ended-->(sc-1))%(TWENTY_FOUR_HOURS);
        4: return scene_ended-->(sc-1);
    }
];
```

Tables Template B/tabt

Purpose

To read, write, search and allocate rows in the Table data structure.

§**1. Format.** The I7 Table structure is not to be confused with the I6 `table` form of array: it is essentially a two-dimensional array which has some metadata at the top of each column.

The run-time representation for a Table is the address `T` of an I6 `table` array: that is, `T-->0` holds the number of columns (which is at most 99) and `T-->i` is the address of column number `i`. Columns are therefore numbered from 1 to `T-->0`, but they are also identified by an ID number of 100 or more, with each different column name having its own ID number. (This is so that multiple tables can share columns with the same name, and correlate them: it also means that NI's type-checking machinery can know the kind of value of a table entry from the name of the column alone.)

Each column `C` is also a `table` array, with `C-->1` holding the unique ID number for the column's name, `C-->2` holding the blank entry flags offset and `C-->3` up to `C-->(C-->0)` holding the entries.

`C-->1` also contains four upper bit flags. These are also defined in "Tables.w" in the NI source, and the values must agree.

```
Constant TB_COLUMN_SIGNED      $4000;
Constant TB_COLUMN_TOPIC       $2000;
Constant TB_COLUMN_DONTSORTME  $1000;
Constant TB_COLUMN_NOBLANKBITS $0800;
Constant TB_COLUMN_CANEXCHANGE $0400;
Constant TB_COLUMN_ALLOCATED   $0200;
Constant TB_COLUMN_NUMBER      $01ff; ! Mask to remove upper bit flags
Constant COL_HSIZE 2; ! Column header size: two words (ID/flags, blank bits)
```

§**2. Find Column.** Columns can be referenced either by their physical column numbers – from 1 to, potentially, 99 – or else by unique ID numbers associated with column names. For instance, if a table has a column called "liquid capacity", then all references to its "liquid capacity entry" are via the ID number associated with this column name, which will be $\geq$ 100 and on the other hand $\leq$ `TB_COLUMN_NUMBER`. At present, this is only 511, so there can be at most 411 different column names across all the tables present in the source text. (It is just about possible to imagine this being a problem on a very large work, so we will probably one day revise the above to make use of the larger word-size in Glulx and so raise this limit. But so far nobody has got even close to it.)

```
[ TableFindCol tab col f i no_cols n;
    no_cols = tab-->0;
    for (i=1: i<=no_cols: i++)
        if (col == ((tab-->i)-->1) & TB_COLUMN_NUMBER) return i;
    if (f) { RunTimeProblem(RTP_TABLE_NOCOL, tab); return 0; }
    return 0;
];
```

§3. Number of Rows. The columns in a table can be assumed all to have the same height (i.e., number of rows): thus the number of rows in `T` can be calculated by looking at column 1, thus...

```
[ TableRows tab first_col;
    first_col = tab-->1; if (first_col == 0) return 0;
    return (first_col-->0) - COL_HSIZE;
];
```

§4. Blanks. Each table entry is stored in a single word in memory: indeed, column `C` row `R` is at address `(T-->C)-->(R+COL_HSIZE)`.

But this is not sufficient storage in all cases, because each entry can be either a value or can be designated "blank". Since, for some columns at least, the possible values include every number, we find that we have to store $2^{16}+1$ possibilities given only a 16-bit memory word. (Well, or $2^{32}+1$ with a 32-bit word, depending on the virtual machine.) This cannot be done.

We therefore need, at least in some cases, an additional bit of storage for each table entry which indicates whether or not it is blank. If we provided such a bit for every table entry, that would be a fairly simple system to implement, but it would also be wasteful of memory, with an overhead of about 5% in practice: and memory in the virtual machine is in very short supply. The reason such a system would be wasteful is that many columns are known to hold values which are in a narrow range; for instance, a time of day cannot exceed 1440, and there will never be more than 10,000 rulebooks or scenes, and so on. For such columns it would be more efficient and indeed faster to indicate blankness by using an exceptional value in the memory cell which is such that it cannot be valid for the kind of value stored in the column. We therefore provide a "blanks bitmap" for only some columns.

This leads us to define the following dummy value, chosen so that it is both impossible for most kinds of value – which is easy to arrange – and also unlikely for even those kinds of value where it is legal. For instance, -1 would be impossible for enumerative kinds of value such as rulebooks and scenes, but it would be a poor choice for the dummy value because it occurs pretty often as an integer. Instead we use the constant `IMPROBABLE_VALUE`, whose value depends on the word size of the virtual machine, and which is declared in "Definitions.i6t".

An entry is therefore blank if and only if either
(a) its column has no blanks bitmap and the stored entry is `TABLE_NOVALUE`, or
(b) its column does have a blanks bitmap, the blanks bit for this entry is set, and the stored entry is also `TABLE_NOVALUE`.

To look up the blanks bitmap is a little slower than to access the stored entry directly. Most of the time, entries accessed will be non-blank: so it is efficient to have a system where we can quickly determine this. If we look at the entry and find that it is not `TABLE_NOVALUE`, then we know it is not a blank. If we find that it is `TABLE_NOVALUE`, on the other hand, then quite often the column has no blanks bitmap and again we have a quick answer: it's blank. Only if the column also has a blanks bitmap do we need to check that we haven't got a false negative. (The more improbable `TABLE_NOVALUE` is as a stored value, the rarer it is that we have to check the blanks bitmap for a non-blank entry.)

```
Constant TABLE_NOVALUE = IMPROBABLE_VALUE;
```

§5. Masks. The blanks bitmaps are stored as bytes; we therefore need a quick way to test or set whether bit number i of a byte is zero, where $0 \le i \le 7$. I6 provides no very useful operators here, whereas memory lookup is cheap, so we use two arrays of bitmaps:

```
Array CheckTableEntryIsBlank_LU
    -> $$00000001
        $$00000010
        $$00000100
        $$00001000
        $$00010000
        $$00100000
        $$01000000
        $$10000000;
Array CheckTableEntryIsNonBlank_LU
    -> $$11111110
        $$11111101
        $$11111011
        $$11110111
        $$11101111
        $$11011111
        $$10111111
        $$01111111;
```

§6. Testing Blankness. The following routine is the one which checks that there is no false negative: it should be used when we know that the table entry is `TABLE_NOVALUE` and we need to check the blank bit, if there is one, to make sure the entry is indeed blank.

The second word in the column table header, `C-->2`, holds the address of the blanks bitmap: this in turn contains one bit for each row, starting with the least significant bit of the first byte. If the table contains a number of rows which isn't a multiple of 8, the spare bits at the end of the last byte in the blanks bitmap are wasted, but this is an acceptable overhead in practice.

```
[ CheckTableEntryIsBlank tab col row i at;
    if (col >= 100) col = TableFindCol(tab, col);
    if (col == 0) rtrue;
    if ((tab-->col)-->(row+COL_HSIZE) ~= TABLE_NOVALUE) {
        print "*** CTEIB on nonblank value ", tab, " ", col, " ", row, " ***^";
    }
    if (((tab-->col)-->1) & TB_COLUMN_NOBLANKBITS) rtrue;
    row--;
    at = ((tab-->col)-->2) + (row/8);
    if ((TB_Blanks->at) & (CheckTableEntryIsBlank_LU->(row%8))) rtrue;
    rfalse;
];
```

§7. Force Entry Blank. We blank a table cell by storing `TABLE_NOVALUE` in its entry word and also setting the relevant bit in the blanks bitmap, if there is one.

We need to be careful if the column holds a kind of value where values are pointers to blocks of allocated memory, because if so then overwriting such a value might lead to a memory leak. So in such cases we call `BlkFree` to free the memory block. (Note that each memory block is pointed to by one and only one I7 value at any given time: we are using them as values, not pointers to values. So if this reference is deleted, it's by definition the only one.) `TABLE_NOVALUE` is chosen such that it cannot be an address of a memory block, which is convenient here. (The value 0 means "no memory block allocated yet".)

```
[ ForceTableEntryBlank tab col row i at oldv flags;
    if (col >= 100) col = TableFindCol(tab, col);
    if (col == 0) rtrue;
    flags = (tab-->col)-->1;
    oldv = (tab-->col)-->(row+COL_HSIZE);
    if ((flags & TB_COLUMN_ALLOCATED) && (oldv ~= 0 or TABLE_NOVALUE))
        BlkFree(oldv);
    (tab-->col)-->(row+COL_HSIZE) = TABLE_NOVALUE;
    if (flags & TB_COLUMN_NOBLANKBITS) return;
    row--;
    at = ((tab-->col)-->2) + (row/8);
    (TB_Blanks->at) = (TB_Blanks->at) | (CheckTableEntryIsBlank_LU->(row%8));
];
```

§8. Force Entry Non-Blank. To unblank a cell, we need to clear the relevant bit in the bitmap. We then go on to write a new value in to the entry – thus overwriting the `TABLE_NOVALUE` value – but that isn't done here; the expectation is that whoever calls this routine is just about to write a new entry anyway.

The exception is again for columns holding a kind of value pointing to a memory block, where we create a suitable initialised but uninteresting memory block for the KOV in question, and set the entry to that.

```
[ ForceTableEntryNonBlank tab col row i at oldv flags tc kov;
    if (col >= 100) col=TableFindCol(tab, col);
    if (col == 0) rtrue;
    if (((tab-->col)-->1) & TB_COLUMN_NOBLANKBITS) return;
    flags = (tab-->col)-->1;
    oldv = (tab-->col)-->(row+COL_HSIZE);
    if ((flags & TB_COLUMN_ALLOCATED) &&
        (oldv == 0 or TABLE_NOVALUE)) {
        kov = UNKNOWN_TY;
        tc = ((tab-->col)-->1) & TB_COLUMN_NUMBER;
        kov = TC_KOV(tc);
        if (kov ~= UNKNOWN_TY) {
            i = kov;
            if (KindBaseArity(i) > 0) i = KindAtomic(i); else i = 0;
            (tab-->col)-->(row+COL_HSIZE) = BlkValueCreate(kov, 0, i);
        }
    }
    row--;
    at = ((tab-->col)-->2) + (row/8);
    (TB_Blanks->at) = (TB_Blanks->at) & (CheckTableEntryIsNonBlank_LU->(row%8));
];
```

§9. Swapping Blank Bits. When sorting a table, we obviously need to swap rows from time to time; if any of its columns have blanks bitmaps, then the relevant bits in them need to be swapped to match, and the following routine performs this operation for two rows in a given column.

```
[ TableSwapBlankBits tab row1 row2 col at1 at2 bit1 bit2;
    if (col >= 100) col=TableFindCol(tab, col);
    if (col == 0) rtrue;
    if (((tab-->col)-->1) & TB_COLUMN_NOBLANKBITS) return;
    row1--;
    at1 = ((tab-->col)-->2) + (row1/8);
    row2--;
    at2 = ((tab-->col)-->2) + (row2/8);
    bit1 = ((TB_Blanks->at1) & (CheckTableEntryIsBlank_LU->(row1%8)));
    bit2 = ((TB_Blanks->at2) & (CheckTableEntryIsBlank_LU->(row2%8)));
    if (bit1) bit1 = true;
    if (bit2) bit2 = true;
    if (bit1 == bit2) return;
    if (bit1) {
        (TB_Blanks->at1)
            = (TB_Blanks->at1) & (CheckTableEntryIsNonBlank_LU->(row1%8));
        (TB_Blanks->at2)
            = (TB_Blanks->at2) | (CheckTableEntryIsBlank_LU->(row2%8));
    } else {
        (TB_Blanks->at1)
            = (TB_Blanks->at1) | (CheckTableEntryIsBlank_LU->(row1%8));
        (TB_Blanks->at2)
            = (TB_Blanks->at2) & (CheckTableEntryIsNonBlank_LU->(row2%8));
    }
];
```

§10. Moving Blank Bits Down. Another common table operation is to compress it by moving all the blank rows down to the bottom, so that non-blank rows occur in a contiguous block at the top: this means table sorting can be done without having to refer continually to the blanks bitmaps. The following operation is useful for keeping the blanks bitmaps up to date when blank rows are moved down.

```
[ TableMoveBlankBitsDown tab row1 row2 col at atp1 bit rx;
    if (col >= 100) col=TableFindCol(tab, col);
    if (col == 0) rtrue;
    if (((tab-->col)-->1) & TB_COLUMN_NOBLANKBITS) return;
    row1--; row2--;
    ! Read blank bit for row1:
    at = ((tab-->col)-->2) + (row1/8);
    bit = ((TB_Blanks->at) & (CheckTableEntryIsBlank_LU->(row1%8)));
    if (bit) bit = true;
    ! Loop through, setting each blank bit to the next:
    for (rx=row1:rx<row2:rx++) {
        atp1 = ((tab-->col)-->2) + ((rx+1)/8);
        at = ((tab-->col)-->2) + (rx/8);
        if ((TB_Blanks->atp1) & (CheckTableEntryIsBlank_LU->((rx+1)%8))) {
            (TB_Blanks->at)
                = (TB_Blanks->at) | (CheckTableEntryIsBlank_LU->(rx%8));
        } else {
            (TB_Blanks->at)
```

```
                = (TB_Blanks->at) & (CheckTableEntryIsNonBlank_LU->(rx%8));
        }
    }
    ! Write bit to blank bit for row2:
    at = ((tab-->col)-->2) + (row2/8);
    if (bit) {
        (TB_Blanks->at)
            = (TB_Blanks->at) | (CheckTableEntryIsBlank_LU->(row2%8));
    } else {
        (TB_Blanks->at)
            = (TB_Blanks->at) & (CheckTableEntryIsNonBlank_LU->(row2%8));
    }
];
```

§11. Table Row Corresponding. `TableRowCorr(T, C, V)` returns the first row on which value `V` appears in column `C` of table `T`, or prints an error if it doesn't.

`ExistsTableRowCorr(T, C, V)` returns the first row on which `V` appears in column `C` of table `T`, or 0 if `V` does not occur at all. If the column is a topic, then we match the entry as a snippet against the value as a general parsing routine.

```
[ TableRowCorr tab col lookup_value lookup_col i j f;
    if (col >= 100) col=TableFindCol(tab, col, true);
    lookup_col = tab-->col;
    j = lookup_col-->0 - COL_HSIZE;
    f=0;
    if (((tab-->col)-->1) & TB_COLUMN_ALLOCATED) f=1;
    for (i=1:i<=j:i++) {
        if ((lookup_value == TABLE_NOVALUE) &&
            (CheckTableEntryIsBlank(tab,col,i))) continue;
        if (f) {
            if (BlkValueCompare(lookup_col-->(i+COL_HSIZE), lookup_value) == 0)
                return i;
        } else {
            if (lookup_col-->(i+COL_HSIZE) == lookup_value) return i;
        }
    }
    return RunTimeProblem(RTP_TABLE_NOCORR, tab);
];
[ ExistsTableRowCorr tab col entry i k v f kov;
    if (col >= 100) col=TableFindCol(tab, col);
    if (col == 0) rfalse;
    f=0;
    if (((tab-->col)-->1) & TB_COLUMN_TOPIC) f=1;
    else if (((tab-->col)-->1) & TB_COLUMN_ALLOCATED) f=2;
    k = TableRows(tab);
    for (i=1:i<=k:i++) {
        v = (tab-->col)-->(i+COL_HSIZE);
        if ((v == TABLE_NOVALUE) && (CheckTableEntryIsBlank(tab,col,i))) continue;
        switch (f) {
            1: if ((v)(entry/100, entry%100) ~= GPR_FAIL) return i;
            2: if (BlkValueCompare(v, entry) == 0) return i;
            default: if (v == entry) return i;
        }
    }
    ! print "Giving up^";
    return 0;
];
```

§12. Table Look Up Corresponding Row. `TableLookUpCorr(T, C1, C2, V)` finds the first row on which value `V` appears in column `C2`, and returns the corresponding value in `C1`, or prints an error if the value `V` cannot be found or has no corresponding value in `C1`.

`ExistsTableLookUpCorr(T, C1, C2, V)` returns `true` if the operation `TableLookUpCorr(T, C1, C2, V)` can be done, `false` otherwise.

```
[ TableLookUpCorr tab col1 col2 lookup_value write_flag write_value cola1 cola2 i j v f;
    if (col1 >= 100) col1=TableFindCol(tab, col1, true);
    if (col2 >= 100) col2=TableFindCol(tab, col2, true);
    cola1 = tab-->col1;
    cola2 = tab-->col2;
    j = cola2-->0;
    f=0;
    if (((tab-->col2)-->1) & TB_COLUMN_ALLOCATED) f=1;
    if (((tab-->col2)-->1) & TB_COLUMN_TOPIC) f=2;
    for (i=1+COL_HSIZE:i<=j:i++) {
        v = cola2-->i;
        if ((v == TABLE_NOVALUE) && (CheckTableEntryIsBlank(tab,col2,i-COL_HSIZE))) continue;
        if (f == 1) {
            if (BlkValueCompare(v, lookup_value) ~= 0) continue;
        } else if (f == 2) {
            if ((v)(lookup_value/100, lookup_value%100) == GPR_FAIL) continue;
        } else {
            if (v ~= lookup_value) continue;
        }
        if (write_flag) {
            ForceTableEntryNonBlank(tab,col1,i-COL_HSIZE);
            switch (write_flag) {
                2: cola1-->i = cola1-->i + write_value;
                3: cola1-->i = cola1-->i - write_value;
                default: cola1-->i = write_value;
            }
            rfalse;
        }
        v = cola1-->i;
        if ((v == TABLE_NOVALUE) &&
            (CheckTableEntryIsBlank(tab,col1,i-COL_HSIZE))) continue;
        return v;
    }
    return RunTimeProblem(RTP_TABLE_NOCORR, tab);
];
[ ExistsTableLookUpCorr tab col1 col2 lookup_value cola1 cola2 i j f;
    if (col1 >= 100) col1=TableFindCol(tab, col1, false);
    if (col2 >= 100) col2=TableFindCol(tab, col2, false);
    if (col1*col2 == 0) rfalse;
    cola1 = tab-->col1; cola2 = tab-->col2;
    j = cola2-->0;
    f=0;
    if (((tab-->col2)-->1) & TB_COLUMN_ALLOCATED) f=1;
    if (((tab-->col2)-->1) & TB_COLUMN_TOPIC) f=2;
    for (i=1+COL_HSIZE:i<=j:i++) {
        if (f == 1) {
            if (BlkValueCompare(cola2-->i, lookup_value) ~= 0) continue;
        } else if (f == 2) {
            if ((cola2-->i)(lookup_value/100, lookup_value%100) == GPR_FAIL) continue;
        } else {
            if (cola2-->i ~= lookup_value) continue;
        }
        if ((cola1-->i == TABLE_NOVALUE) &&
            (CheckTableEntryIsBlank(tab,col1,i-COL_HSIZE))) continue;
        rtrue;
    }
    rfalse;
];
```

§13. Table Look Up Entry. `TableLookUpEntry(T, C, R)` returns the value at column `C`, row `R`, printing an error if that doesn't exist.

`ExistsTableLookUpEntry(T, C, R)` returns true if a value exists at column `C`, row `R`, false otherwise.

```
[ TableLookUpEntry tab col index write_flag write_value v;
    if (col >= 100) col=TableFindCol(tab, col, true);
    if ((index < 1) || (index > TableRows(tab)))
        return RunTimeProblem(RTP_TABLE_NOROW, tab, index);
    if (write_flag) {
        switch(write_flag) {
            1: ForceTableEntryNonBlank(tab,col,index);
                (tab-->col)-->(index+COL_HSIZE) = write_value;
            2: ForceTableEntryNonBlank(tab,col,index);
                (tab-->col)-->(index+COL_HSIZE) =
                    ((tab-->col)-->(index+COL_HSIZE)) + write_value;
            3: ForceTableEntryNonBlank(tab,col,index);
                (tab-->col)-->(index+COL_HSIZE) =
                    ((tab-->col)-->(index+COL_HSIZE)) - write_value;
            4: ForceTableEntryBlank(tab,col,index);
            5: ForceTableEntryNonBlank(tab,col,index);
                return ((tab-->col)-->(index+COL_HSIZE));
        }
        rfalse;
    }
    v = ((tab-->col)-->(index+COL_HSIZE));
    if ((v == TABLE_NOVALUE) && (CheckTableEntryIsBlank(tab,col,index)))
        return RunTimeProblem(RTP_TABLE_NOENTRY, tab, col, index);
    return v;
];
[ ExistsTableLookUpEntry tab col index v;
    if (col >= 100) col=TableFindCol(tab, col);
    if (col == 0) rfalse;
    if ((index<1) || (index > TableRows(tab))) rfalse;
    v = ((tab-->col)-->(index+COL_HSIZE));
    if ((v == TABLE_NOVALUE) && (CheckTableEntryIsBlank(tab,col,index)))
        rfalse;
    rtrue;
];
```

§14. Blank Rows. `TableRowIsBlank(T, R)` returns true if row `R` of table `T` is blank. (`R` must be a legal row number.)

`TableBlankOutRow(T, R)` fills row `R` of table `T` with blanks. (`R` must be a legal row number.)

`TableBlankRows(T)` returns the number of blank rows in `T`.

`TableFilledRows(T)` returns the number of non-blank rows in `T`.

`TableBlankRow(T)` finds the first blank row in `T`.

```
[ TableRowIsBlank tab j k;
    for (k=1:k<=tab-->0:k++) {
        if (((tab-->k)-->(j+COL_HSIZE)) ~= TABLE_NOVALUE) rfalse;
        if (CheckTableEntryIsBlank(tab, k, j) == false) rfalse;
    }
    rtrue;
];
[ TableBlankOutRow tab row k;
    if (tab==0) return RunTimeProblem(RTP_TABLE_NOTABLE, tab);
    for (k=1:k<=tab-->0:k++)
        ForceTableEntryBlank(tab, k, row);
];
[ TableBlankOutColumn tab col n k;
    if (tab==0) return RunTimeProblem(RTP_TABLE_NOTABLE, tab);
    n = TableRows(tab);
    for (k=1:k<=n:k++)
        ForceTableEntryBlank(tab, col, k);
];
[ TableBlankOutAll tab n k;
    if (tab==0) return RunTimeProblem(RTP_TABLE_NOTABLE, tab);
    n = TableRows(tab);
    for (k=1:k<=n:k++)
        TableBlankOutRow(tab, k);
];
[ TableBlankRows tab i j c;
    i = TableRows(tab); !print i, " rows^";
    for (j=1:j<=i:j++)
        if (TableRowIsBlank(tab, j)) c++;
    !print c, " blank^";
    return c;
];
[ TableFilledRows tab;
    return TableRows(tab) - TableBlankRows(tab);
];
[ TableBlankRow tab i j;
    i = TableRows(tab);
    for (j=1:j<=i:j++)
```

```
        if (TableRowIsBlank(tab, j)) return j;
    RunTimeProblem(RTP_TABLE_NOMOREBLANKS, tab);
    return i;
];
```

§15. Random Row. `TableRandomRow(T)` chooses a random non-blank row in `T`.

```
[ TableRandomRow tab i j k;
    i = TableRows(tab);
    j = TableFilledRows(tab);
    if (j==0) return RunTimeProblem(RTP_TABLE_NOROWS, tab);
    if (j>1) j = random(j);
    for (k=1:k<=i:k++) {
        if (TableRowIsBlank(tab, k) == false) j--;
        if (j==0) return k;
    }
];
```

§16. Swap Rows. `TableSwapRows(T, R1, R2)` exchanges rows `R1` and `R2`.

```
[ TableSwapRows tab i j k l v1 v2;
    if (i==j) return;
    l = tab-->0;
    for (k=1:k<=l:k++) {
        v1 = (tab-->k)-->(i+COL_HSIZE);
        v2 = (tab-->k)-->(j+COL_HSIZE);
        (tab-->k)-->(i+COL_HSIZE) = v2;
        (tab-->k)-->(j+COL_HSIZE) = v1;
        if ((v1 == TABLE_NOVALUE) || (v2 == TABLE_NOVALUE))
            TableSwapBlankBits(tab, i, j, k);
    }
];
```

§17. Compare Rows. `TableCompareRows(T, C, R1, R2, D)` returns:

(a) +1 if the entry at row R1 of column C is > the entry at row R2,
(b) 0 if they are equal, and
(c) −1 if entry at R1 < entry at R2.

When $D = +1$, a blank value is greater than all other values, so that in an ascending sort the blanks come last; when $D = -1$, a blank value is less than all others, so that once again blanks are last. Finally, a wholly blank row is always placed after a row in which the entry in C is blank but where other entries are not.

```
[ TableCompareRows tab col row1 row2 dir val1 val2 bl1 bl2 f;
    if (col >= 100) col=TableFindCol(tab, col, false);
    val1 = (tab-->col)-->(row1+COL_HSIZE);
    val2 = (tab-->col)-->(row2+COL_HSIZE);
    if (val1 == TABLE_NOVALUE) bl1 = CheckTableEntryIsBlank(tab,col,row1);
    if (val2 == TABLE_NOVALUE) bl2 = CheckTableEntryIsBlank(tab,col,row2);
    if ((val1 == val2) && (bl1 == bl2)) {
        if (val1 ~= TABLE_NOVALUE) return 0;
        if (bl1 == false) return 0;
        ! The two entries are both blank:
```

```
        if (TableRowIsBlank(tab, row1)) {
            if (TableRowIsBlank(tab, row2)) return 0;
            return -1*dir;
        }
        if (TableRowIsBlank(tab, row2)) return dir;
        return 0;
    }
    if (bl1) return dir;
    if (bl2) return -1*dir;
    f = ((tab-->col)-->1);
    if (f & TB_COLUMN_ALLOCATED) {
        if (BlkValueCompare(val2, val1) < 0) return 1;
        return -1;
    } else if (f & TB_COLUMN_SIGNED) {
        if (val1 > val2) return 1;
        return -1;
    } else {
        if (UnsignedCompare(val1, val2) > 0) return 1;
        return -1;
    }
];
```

§18. Move Row Down.

```
[ TableMoveRowDown tab r1 r2 rx k l m v f;
    if (r1==r2) return;
    l = tab-->0;
    for (k=1:k<=l:k++) {
        f = false;
        m = (tab-->k)-->(r1+COL_HSIZE);
        if (m == TABLE_NOVALUE) f = true;
        for (rx=r1:rx<r2:rx++) {
            v = (tab-->k)-->(rx+COL_HSIZE+1);
            (tab-->k)-->(rx+COL_HSIZE) = v;
            if (v == TABLE_NOVALUE) f = true;
        }
        (tab-->k)-->(r2+COL_HSIZE) = m;
        if (f) TableMoveBlankBitsDown(tab, r1, r2, k);
    }
];
```

§19. Shuffle. `TableShuffle(T)` sorts `T` into random row order.

```
[ TableShuffle tab i to;
    TableMoveBlanksToBack(tab, 1, TableRows(tab));
    to = TableFilledRows(tab);
    for (i=2:i<=to:i++) TableSwapRows(tab, i, random(i));
];
```

§20. Next Row. `TableNextRow(T, C, R, D)` is used when scanning through a table in order of the values in column `C`: ascending order if `D = 1`, descending if `D = -1`. The current position is row `R` of column `C`, or `R = 0` if we have not yet found the first row. The return value is the row number for the next value, or 0 if we are already at the final value. Note that if there are several equal values in the column, they will be run through in turn, in order of their physical row numbers - ascending if `D = 1`, descending if `D = -1`, so that using the routine with `D = -1` always produces the exact reverse ordering from using it with `D = 1` and the same parameters. Rows with blank entries in `C` are skipped.

```
    for (R=TableNextRow(T,C,0,D): R : R=TableNextRow(T,C,R,D)) ...
```

will perform a loop of valid row numbers in order of column `C`.

```
[ TableNextRow tab col row dir i k val v dv min_dv min_at signed_arithmetic f;
    if (col >= 100) col=TableFindCol(tab, col, false);
    f = ((tab-->col)-->1);
    if (f & TB_COLUMN_ALLOCATED) RunTimeProblem(RTP_TABLE_CANTRUNTHROUGH, tab);
    signed_arithmetic = f & TB_COLUMN_SIGNED;
    #Iftrue (WORDSIZE == 2);
    if (row == 0) {
        if (signed_arithmetic) {
            if (dir == 1) val = $8000; else val = $7fff;
        } else {
            if (dir == 1) val = 0; else val = $ffff;
        }
    } else val = (tab-->col)-->(row+COL_HSIZE);
    if (signed_arithmetic) min_dv = $7fff; else min_dv = $ffff;
    #ifnot; ! WORDSIZE == 4
    if (row == 0) {
        if (signed_arithmetic) {
            if (dir == 1) val = $80000000; else val = $7fffffff;
        } else {
            if (dir == 1) val = 0; else val = $ffffffff;
        }
    } else val = (tab-->col)-->(row+COL_HSIZE);
    if (signed_arithmetic) min_dv = $7fffffff; else min_dv = $ffffffff;
    #endif;
    k = TableRows(tab);
    if (dir == 1) {
        for (i=1:i<=k:i++) {
            v = (tab-->col)-->(i+COL_HSIZE);
            if ((v == TABLE_NOVALUE) && (CheckTableEntryIsBlank(tab,col,i)))
                continue;
            dv = dir*v;
            if (signed_arithmetic)
            f = (((dv > dir*val) || ((v == val) && (i>row))) &&
                (dv < min_dv));
            else
            f = (((UnsignedCompare(dv, dir*val) > 0) || ((v == val) && (i>row))) &&
                (UnsignedCompare(dv, min_dv) < 0));
            if (f) { min_dv = dv; min_at = i; }
        }
    } else {
        for (i=k:i>=1:i--) {
            v = (tab-->col)-->(i+COL_HSIZE);
            if ((v == TABLE_NOVALUE) && (CheckTableEntryIsBlank(tab,col,i)))
```

```
                continue;
            dv = dir*v;
            if (signed_arithmetic)
            f = (((dv > dir*val) || ((v == val) && (i<row))) &&
                (dv < min_dv));
            else
            f = (((UnsignedCompare(dv, dir*val) > 0) || ((v == val) && (i<row))) &&
                (UnsignedCompare(dv, min_dv) < 0));
            if (f) { min_dv = dv; min_at = i; }
        }
    }
    return min_at;
];
```

§21. Move Blanks to Back.

```
[ TableMoveBlanksToBack tab fromrow torow i fbl lnbl blc;
    if (torow < fromrow) return;
    fbl = 0; lnbl = 0;
    for (i=fromrow: i<=torow: i++)
        if (TableRowIsBlank(tab, i)) {
            if (fbl == 0) fbl = i;
            blc++;
        } else {
            lnbl = i;
        }
    if ((fbl>0) && (lnbl>0) && (fbl < lnbl)) {
        TableMoveRowDown(tab, fbl, lnbl); ! Move first blank just past last nonblank
        TableMoveBlanksToBack(tab, fbl, lnbl-1);
    }
    return torow-blc; ! Final non-blank row
];
```

§22. Sort. This is really only a front-end: it calls the sorting code at “Sort.i6t”.

```
[ TableSort tab col dir test_flag algorithm i j k f;
    for (i=1:i<=tab-->0:i++) {
        j = tab-->i; ! Address of column table
        if ((j-->1) & TB_COLUMN_DONTSORTME)
            return RunTimeProblem(RTP_TABLE_CANTSORT, tab);
    }
    if (col >= 100) col=TableFindCol(tab, col, false);
    k = TableRows(tab);
    k = TableMoveBlanksToBack(tab, 1, k);
    if (test_flag) {
        print "After moving blanks to back:^"; TableColumnDebug(tab, col);
    }
    SetSortDomain(TableSwapRows, TableCompareRows);
    SortArray(tab, col, dir, k, test_flag, algorithm);
    if (test_flag) {
        print "Final state:^"; TableColumnDebug(tab, col);
    }
];
```

§23. Print Table Name. NI fills this in: it's used to say the "table" kind of value.

```
[ PrintTableName T;
    switch(T) {
{-call:Data::Tables::compile_print_table_names}
        default: print "** No such table **";
    }
];
```

§24. Print Table to File. This is how we serialise a table to an external file, though the writing is done by printing characters in the standard way; it's just that the output stream will be an external file rather than the screen when this routine is called.

```
[ TablePrint tab i j k row col v tc kov;
    for (i=1:i<=tab-->0:i++) {
        j = tab-->i; ! Address of column table
        if (((j-->1) & TB_COLUMN_CANEXCHANGE) == 0)
            rtrue;
    }
    k = TableRows(tab);
    k = TableMoveBlanksToBack(tab, 1, k);
    print "! ", (PrintTableName) tab, " (", k, ")^";
    for (row=1:row<=k:row++) {
        for (col=1:col<=tab-->0:col++) {
            tc = ((tab-->col)-->1) & TB_COLUMN_NUMBER;
            kov = KindAtomic(TC_KOV(tc));
            if (kov == UNKNOWN_TY) kov = NUMBER_TY;
            v = (tab-->col)-->(row+COL_HSIZE);
            if ((v == TABLE_NOVALUE) && (CheckTableEntryIsBlank(tab,col,row)))
                print "-- ";
            else {
                if (BlkValueWriteToFile(v, kov) == false) print v;
                print " ";
            }
        }
        print "^";
    }
    rfalse;
];
```

§25. Read Table from File. And this is how we unserialise again. It makes sense only on Glulx.

```
#ifdef TARGET_GLULX;
[ TableRead tab auxf row maxrow col ch v sgn dg j tc kov;
    for (col=1:col<=tab-->0:col++) {
        j = tab-->col; ! Address of column table
        if (((j-->1) & TB_COLUMN_CANEXCHANGE) == 0)
            return RunTimeProblem(RTP_TABLE_CANTSAVE, tab);
    }
    maxrow = TableRows(tab);
    !print maxrow, " rows available.^";
    for (row=1: row<=maxrow: row++) {
        TableBlankOutRow(tab, row);
    }
    for (row=1: row<=maxrow: row++) {
        !print "Reading row ", row, "^";
        ch = FileIO_GetC(auxf);
        if (ch == '!') {
            while (ch ~= -1 or 10 or 13) ch = FileIO_GetC(auxf);
            while (ch == 10 or 13) ch = FileIO_GetC(auxf);
        }
        for (col=1: col<=tab-->0: col++) {
            if (ch == -1) { row++; jump NoMore; }
            if (ch == 10 or 13) break;
            tc = ((tab-->col)-->1) & TB_COLUMN_NUMBER;
            kov = KindAtomic(TC_KOV(tc));
            if (kov == UNKNOWN_TY) kov = NUMBER_TY;
            !print "tc = ", tc, " kov = ", kov, "^";
            sgn = 1;
            if (ch == '-') {
                ch = FileIO_GetC(auxf);
                if (ch == -1) jump NotTable;
                if (ch == '-') { ch = FileIO_GetC(auxf); jump EntryDone; }
                sgn = -1;
            }
            if (((tab-->col)-->1) & TB_COLUMN_ALLOCATED)
                ForceTableEntryNonBlank(tab, col, row);
            !print "A";
            v = BlkValueReadFromFile(0, 0, -1, kov);
            if (v) {
                if (((tab-->col)-->1) & TB_COLUMN_ALLOCATED)
                    v = BlkValueReadFromFile(TableLookUpEntry(tab, col, row),
                        auxf, ch, kov);
                else
                    v = BlkValueReadFromFile(0, auxf, ch, kov);
                ch = 32;
            } else {
                dg = ch - '0';
                if ((dg < 0) || (dg > 9)) jump NotTable;
                v = dg;
                for (::) {
                    ch = FileIO_GetC(auxf);
                    dg = ch - '0';
                    if ((dg < 0) || (dg > 9)) break;
                    v = 10*v + dg;
                }
                v = v*sgn;
            }
            !print "v=", v, " ";
            if (((tab-->col)-->1) & TB_COLUMN_ALLOCATED == 0)
                TableLookUpEntry(tab, col, row, true, v);
            .EntryDone;
            !print "First nd is ", ch, "^";
            while (ch == 9 or 32) ch = FileIO_GetC(auxf);
        }
        while (ch ~= -1 or 10 or 13) {
            if ((ch ~= '-') && (((ch-'0')<0) || ((ch-'0')>9))) jump NotTable;
            if (ch ~= 9 or 32) jump WontFit;
            ch = FileIO_GetC(auxf);
        }
    }
    .NoMore;
    while (ch == 9 or 32 or 10 or 13) ch = FileIO_GetC(auxf);
    if (ch == -1) return;
    .WontFit;
    return RunTimeProblem(RTP_TABLE_WONTFIT, tab);
    .NotTable;
    return RunTimeProblem(RTP_TABLE_BADFILE, tab);
];
#ENDIF; ! TARGET_GLULX
```

§**26. Print Rank.** The table of scoring ranks is a residue from the ancient times of early IF: it gets a tiny amount of special treatment here, even though I7 works tend not to use these now dated conventions.

```
[ PrintRank i j v;
#ifdef RANKING_TABLE;
    L__M(##Score, 3);
    j = TableRows(RANKING_TABLE);
    for (i=j:i>=1:i--)
        if (score >= TableLookUpEntry(RANKING_TABLE, 1, i)) {
            v = TableLookUpEntry(RANKING_TABLE, 2, i);
            if (v ofclass String) print (string) v;
            else v();
            ".";
        }
#endif;
    ".";
];
```

§27. Debugging. A routine to print the state of a table, for debugging purposes only.

```
[ TableColumnDebug tab col k i v;
    if (col >= 100) col=TableFindCol(tab, col, false);
    k = TableRows(tab);
    print "Table col ", col, ": ";
    for (i=1:i<=k:i++) {
        v = (tab-->col)-->(i+COL_HSIZE);
        if ((v == TABLE_NOVALUE) && (CheckTableEntryIsBlank(tab,col,i)))
            print "BLANK ";
        else
            print v, " ";
    }
    print "*^";
];
```

Sort Template B/sortt

Purpose

To sort arrays.

§**1. Storage.** We are required to use a stable sorting algorithm with very low, ideally zero, auxiliary storage requirement. Exchanges are generally slower than comparisons for the typical application (sorting tables, where entire rows must be exchanged whereas only entries in a single column need be compared).

In fact, we store some details in global variables for convenience and to avoid filling the stack with copies, but otherwise we will hardly need any auxiliary storage.

```
Global I7S_Tab; ! The array to be sorted, which can have almost any format
Global I7S_Col; ! The "column number" in the array, if any
Global I7S_Dir; ! The direction of sorting: ascending (1) or descending (-1)
Global I7S_Swap; ! The current routine for swapping two fields
Global I7S_Comp; ! The current routine for comparing two fields
#ifdef MEASURE_SORT_PERFORMANCE;
Global I7S_CCOUNT; Global I7S_CCOUNT2; Global I7S_XCOUNT; ! For testing only
#endif;
```

§**2. Front End.** To perform a sort, we first call `SetSortDomain` to declare the swap and compare functions to be used, and then call `SortArray` actually to sort. (It would be nice to combine these in a single call, but I6 allows a maximum of 7 call arguments for a routine, and that would make 8.) These are the only two routines which should ever be called from outside of this template segment.

The swap and compare functions are expected to take two arguments, which are the field numbers of the fields being swapped or compared, where fields are numbered from 1. Comparison is like `strcmp`: it returns 0 on equality, and then is positive or negative according to which of the fields is greater in value.

```
[ SetSortDomain swapf compf;
    I7S_Swap = swapf;
    I7S_Comp = compf;
];
[ SortArray tab col dir size test_flag algorithm;
    I7S_Tab = tab;
    I7S_Col = col;
    I7S_Dir = dir;
    #ifdef MEASURE_SORT_PERFORMANCE;
    I7S_CCOUNT = 0;
    I7S_CCOUNT2 = 0;
    I7S_XCOUNT = 0;
    #endif;
    SortRange(0, size, algorithm);
    #ifdef MEASURE_SORT_PERFORMANCE;
    if (test_flag)
        print "Sorted array of size ", size, " with ", I7S_CCOUNT2, "*10000 + ", I7S_CCOUNT,
            " comparisons and ", I7S_XCOUNT, " exchanges^";
    #endif;
];
```

§**3. Sort Range.** This routine sorts a range of fields $x \le i < y$ within the array. Fields are numbered from 0. The supplied `algorithm` is an I6 routine to implement a particular sorting algorithm; if it is not supplied, then in-place merge sort is used by default.

```
[ SortRange x y algorithm;
    if (y - x < 2) return;
    if (algorithm) {
        (algorithm)(x, y);
    } else {
        InPlaceMergeSortAlgorithm(x, y);
    }
];
```

§**4. Comparison and Exchange.** These are instrumented versions of how to swap and compare fields; note that the swap and compare functions are expected to number the fields from 1, not from 0. (This is convenient both for tables and lists, where rows and entries respectively are both numbered from 1.) The only access which the sorting algorithms have to the actual data being sorted is through these routines.

```
[ CompareFields x y;
    #ifdef MEASURE_SORT_PERFORMANCE;
    I7S_CCOUNT++;
    if (I7S_CCOUNT == 10000) { I7S_CCOUNT = 0; I7S_CCOUNT2++; }
    #endif;
    return I7S_Dir*I7S_Comp(I7S_Tab, I7S_Col, x+1, y+1, I7S_Dir);
];
[ ExchangeFields x y;
    #ifdef MEASURE_SORT_PERFORMANCE;
    I7S_XCOUNT++;
    if (I7S_XCOUNT < 0) { print "XO^"; I7S_XCOUNT = 0; }
    #endif;
    return I7S_Swap(I7S_Tab, x+1, y+1);
];
```

§**5. 4W37 Sort.** We now present three alternative sorting algorithms.

The first is the one used in builds up to and including 4W37: note that this is not quite bubble sort, and that it is unstable. It is now no longer used, but is so short that we might as well keep it in the code base in case anyone needs to resurrect a very early I7 project.

```
[ OldSortAlgorithm x y
    f i j;
    if (y - x < 2) return;
    f = true;
    while (f) {
        f = false;
        for (i=x:i<y:i++)
            for (j=i+1:j<y:j++)
                if (CompareFields(i, j) > 0) {
                    ExchangeFields(i, j); f = true; break;
                }
    }
];
```

§**6. Insertion Sort.** A stable algorithm which has $O(n^2)$ running time and therefore cannot be used with large arrays, but which has good performance on nearly sorted tables, and which has very low overhead.

```
[ InsertionSortAlgorithm from to
    i j;
    if (to > from+1) {
        for (i = from+1: i < to: i++) {
            for (j = i: j > from: j--) {
                if (CompareFields(j, j-1) < 0)
                    ExchangeFields(j, j-1);
                else break;
            }
        }
    }
];
```

§**7. In-Place Mergesort.** A stable algorithm with $O(n \log n)$ running time, at some stack cost, and which is generally good for nearly sorted tables, but which is also complex and has some overhead. The code here mostly follows Thomas Baudel's implementation, which in turn follows the C++ STL library.

```
[ InPlaceMergeSortAlgorithm from to
    middle;
    if (to - from < 12) {
        if (to - from < 2) return;
        InsertionSortAlgorithm(from, to);
        return;
    }
    middle = (from + to)/2;
    InPlaceMergeSortAlgorithm(from, middle);
    InPlaceMergeSortAlgorithm(middle, to);
    IPMS_Merge(from, middle, to, middle-from, to - middle);
];
[ IPMS_Lower from to val
    len half mid;
    len = to - from;
    while (len > 0) {
        half = len/2;
        mid = from + half;
        if (CompareFields(mid, val) < 0) {
            from = mid + 1;
            len = len - half -1;
        } else len = half;
    }
    return from;
];
[ IPMS_Upper from to val
    len half mid;
    len = to - from;
    while (len > 0) {
        half = len/2;
        mid = from + half;
        if (CompareFields(val, mid) < 0)
```

```
            len = half;
        else {
            from = mid + 1;
            len = len - half -1;
        }
    }
    return from;
];
[ IPMS_Reverse from to;
    while (from < to) {
        ExchangeFields(from++, to--);
    }
];
[ IPMS_Rotate from mid to
    n val shift p1 p2;
    if ((from==mid) || (mid==to)) return;
    IPMS_Reverse(from, mid-1);
    IPMS_Reverse(mid, to-1);
    IPMS_Reverse(from, to-1);
];
[ IPMS_Merge from pivot to len1 len2
    first_cut second_cut len11 len22 new_mid;
    if ((len1 == 0) || (len2 == 0)) return;
    if (len1+len2 == 2) {
        if (CompareFields(pivot, from) < 0)
        ExchangeFields(pivot, from);
        return;
    }
    if (len1 > len2) {
        len11 = len1/2;
        first_cut = from + len11;
        second_cut = IPMS_Lower(pivot, to, first_cut);
        len22 = second_cut - pivot;
    } else {
        len22 = len2/2;
        second_cut = pivot + len22;
        first_cut = IPMS_Upper(from, pivot, second_cut);
        len11 = first_cut - from;
    }
    IPMS_Rotate(first_cut, pivot, second_cut);
    new_mid = first_cut + len22;
    IPMS_Merge(from, first_cut, new_mid, len11, len22);
    IPMS_Merge(new_mid, second_cut, to, len1 - len11, len2 - len22);
];
```

Relations Template B/relt

Purpose

To manage run-time storage for relations between objects, and to find routes through relations and the map.

§1. Relation Records. See "RelationKind.i6t" for further explanation.

```
Constant RR_NAME      4;
Constant RR_PERMISSIONS 5;
Constant RR_STORAGE 6;
Constant RR_KIND 7;
Constant RR_HANDLER 8;
Constant RR_DESCRIPTION 9;
```

§2. Valency Adjectives. These are defined in the Standard Rules; the following routines must either test the state (if `set` is negative), or change the state to `set`.

```
Constant VALENCY_MASK = RELS_EQUIVALENCE+RELS_SYMMETRIC+RELS_X_UNIQUE+RELS_Y_UNIQUE;
[ RELATION_TY_EquivalenceAdjective rel set  perms state handler;
    perms = rel-->RR_PERMISSIONS;
    if (perms & RELS_EQUIVALENCE) state = true;
    if (set < 0) return state;
    if ((set) && (state == false)) {
        perms = perms + RELS_EQUIVALENCE;
        if (perms & RELS_SYMMETRIC == 0) perms = perms + RELS_SYMMETRIC;
    }
    if ((set == false) && (state)) {
        perms = perms - RELS_EQUIVALENCE;
        if (perms & RELS_SYMMETRIC) perms = perms - RELS_SYMMETRIC;
    }
    rel-->RR_PERMISSIONS = perms;
    handler = rel-->RR_HANDLER;
    if (handler(rel, RELS_SET_VALENCY, perms & VALENCY_MASK) == 0)
        "*** Can't change this to an equivalence relation ***";
];
[ RELATION_TY_SymmetricAdjective rel set  perms state handler;
    perms = rel-->RR_PERMISSIONS;
    if (perms & RELS_SYMMETRIC) state = true;
    if (set < 0) return state;
    if ((set) && (state == false)) perms = perms + RELS_SYMMETRIC;
    if ((set == false) && (state)) perms = perms - RELS_SYMMETRIC;
    rel-->RR_PERMISSIONS = perms;
    handler = rel-->RR_HANDLER;
    if (handler(rel, RELS_SET_VALENCY, perms & VALENCY_MASK) == 0)
        "*** Can't change this to a symmetric relation ***";
```

```
];
[ RELATION_TY_OToOAdjective rel set  perms state handler;
    perms = rel-->RR_PERMISSIONS;
    if (perms & (RELS_X_UNIQUE+RELS_Y_UNIQUE) == RELS_X_UNIQUE+RELS_Y_UNIQUE) state = true;
    if (set < 0) return state;
    if ((set) && (state == false)) {
        if (perms & RELS_X_UNIQUE == 0) perms = perms + RELS_X_UNIQUE;
        if (perms & RELS_Y_UNIQUE == 0) perms = perms + RELS_Y_UNIQUE;
        if (perms & RELS_EQUIVALENCE) perms = perms - RELS_EQUIVALENCE;
    }
    if ((set == false) && (state)) {
        if (perms & RELS_X_UNIQUE) perms = perms - RELS_X_UNIQUE;
        if (perms & RELS_Y_UNIQUE) perms = perms - RELS_Y_UNIQUE;
    }
    rel-->RR_PERMISSIONS = perms;
    handler = rel-->RR_HANDLER;
    if (handler(rel, RELS_SET_VALENCY, perms & VALENCY_MASK) == 0)
        "*** Can't change this to a one-to-one relation ***";
];
[ RELATION_TY_OToVAdjective rel set  perms state handler;
    perms = rel-->RR_PERMISSIONS;
    if (perms & (RELS_X_UNIQUE+RELS_Y_UNIQUE) == RELS_X_UNIQUE) state = true;
    if (set < 0) return state;
    if ((set) && (state == false)) {
        if (perms & RELS_X_UNIQUE == 0) perms = perms + RELS_X_UNIQUE;
        if (perms & RELS_Y_UNIQUE) perms = perms - RELS_Y_UNIQUE;
        if (perms & RELS_SYMMETRIC) perms = perms - RELS_SYMMETRIC;
        if (perms & RELS_EQUIVALENCE) perms = perms - RELS_EQUIVALENCE;
    }
    if ((set == false) && (state)) {
        if (perms & RELS_X_UNIQUE) perms = perms - RELS_X_UNIQUE;
        if (perms & RELS_Y_UNIQUE) perms = perms - RELS_Y_UNIQUE;
    }
    rel-->RR_PERMISSIONS = perms;
    handler = rel-->RR_HANDLER;
    if (handler(rel, RELS_SET_VALENCY, perms & VALENCY_MASK) == 0)
        "*** Can't change this to a one-to-various relation ***";
];
[ RELATION_TY_VToOAdjective rel set  perms state handler;
    perms = rel-->RR_PERMISSIONS;
    if (perms & (RELS_X_UNIQUE+RELS_Y_UNIQUE) == RELS_Y_UNIQUE) state = true;
    if (set < 0) return state;
    if ((set) && (state == false)) {
        if (perms & RELS_X_UNIQUE) perms = perms - RELS_X_UNIQUE;
        if (perms & RELS_Y_UNIQUE == 0) perms = perms + RELS_Y_UNIQUE;
        if (perms & RELS_SYMMETRIC) perms = perms - RELS_SYMMETRIC;
        if (perms & RELS_EQUIVALENCE) perms = perms - RELS_EQUIVALENCE;
    }
    if ((set == false) && (state)) {
        if (perms & RELS_X_UNIQUE) perms = perms - RELS_X_UNIQUE;
        if (perms & RELS_Y_UNIQUE) perms = perms - RELS_Y_UNIQUE;
    }
    rel-->RR_PERMISSIONS = perms;
    handler = rel-->RR_HANDLER;
    if (handler(rel, RELS_SET_VALENCY, perms & VALENCY_MASK) == 0)
        "*** Can't change this to a various-to-one relation ***";
];
[ RELATION_TY_VToVAdjective rel set  perms state handler;
    perms = rel-->RR_PERMISSIONS;
    if (perms & (RELS_X_UNIQUE+RELS_Y_UNIQUE) == 0) state = true;
    if (set < 0) return state;
    if ((set) && (state == false)) {
        if (perms & RELS_X_UNIQUE) perms = perms - RELS_X_UNIQUE;
        if (perms & RELS_Y_UNIQUE) perms = perms - RELS_Y_UNIQUE;
    }
    if ((set == false) && (state)) {
        if (perms & RELS_X_UNIQUE == 0) perms = perms + RELS_X_UNIQUE;
        if (perms & RELS_Y_UNIQUE == 0) perms = perms + RELS_Y_UNIQUE;
    }
    rel-->RR_PERMISSIONS = perms;
    handler = rel-->RR_HANDLER;
    if (handler(rel, RELS_SET_VALENCY, perms & VALENCY_MASK) == 0)
        "*** Can't change this to a various-to-various relation ***";
];
```

§3. One To One Relations. We provide routines to assert a 1-to-1 relation true, or to assert it false. The relation `rel` is represented by a property number, and the property in question is used to store the fact of a relationship: $O_1 \sim O_2$ if and only if `O1.rel == O2`.

There is no routine to test a 1-to-1 relation, since the predicate calculus code in NI simplifies propositions which test these into direct looking up of the property relation.

```
[ Relation_Now1to1 obj1 relation_property obj2 ol; ! Assert 1-1 true
    if (obj2) objectloop (ol provides relation_property)
        if (ol.relation_property == obj2) ol.relation_property = nothing;
    if (obj1) obj1.relation_property = obj2;
];
[ Relation_NowN1toV obj1 relation_property obj2; ! Assert 1-1 false
    if ((obj1) && (obj1.relation_property == obj2)) obj1.relation_property = nothing;
];
[ Relation_Now1to1V obj1 obj2 KOV relation_property ol N; ! Assert 1-1 true
    if (obj2) {
        N = KOVDomainSize(KOV);
        for (ol=1: ol<=N: ol++)
            if (GProperty(KOV, ol, relation_property) == obj2)
                WriteGProperty(KOV, ol, relation_property, 0);
    }
    if (obj1) WriteGProperty(KOV, obj1, relation_property, obj2);
];
[ Relation_NowN1toVV obj1 obj2 KOV relation_property; ! Assert 1-1 false
    if ((obj1) && (GProperty(KOV, obj1, relation_property) == obj2))
        WriteGProperty(KOV, obj1, relation_property, 0);
];
```

§4. Symmetric One To One Relations. Here the relation is used for both objects: $O_1 \sim O_2$ if and only if both `O1.relation_property == O2` and `O2.relation_property == O1`.

```
[ Relation_NowS1to1 obj1 relation_property obj2; ! Assert symmetric 1-1 true
    if ((obj1 ofclass Object) && (obj1 provides relation_property) &&
        (obj2 ofclass Object) && (obj2 provides relation_property)) {
        if (obj1.relation_property) { (obj1.relation_property).relation_property = 0; }
        if (obj2.relation_property) { (obj2.relation_property).relation_property = 0; }
        obj1.relation_property = obj2; obj2.relation_property = obj1;
    }
];
[ Relation_NowSN1to1 obj1 relation_property obj2; ! Assert symmetric 1-1 false
    if ((obj1 ofclass Object) && (obj1 provides relation_property) &&
        (obj2 ofclass Object) && (obj2 provides relation_property) &&
        (obj1.relation_property == obj2)) {
        obj1.relation_property = 0; obj2.relation_property = 0;
    }
];
[ Relation_NowS1to1V obj1 obj2 KOV relation_property; ! Assert symmetric 1-1 true
    if (GProperty(KOV, obj1, relation_property))
        WriteGProperty(KOV, GProperty(KOV, obj1, relation_property), relation_property, 0);
    if (GProperty(KOV, obj2, relation_property))
        WriteGProperty(KOV, GProperty(KOV, obj2, relation_property), relation_property, 0);
    WriteGProperty(KOV, obj1, relation_property, obj2);
    WriteGProperty(KOV, obj2, relation_property, obj1);
];
[ Relation_NowSN1to1V obj1 obj2 KOV relation_property; ! Assert symmetric 1-1 false
    if (GProperty(KOV, obj1, relation_property) == obj2) {
        WriteGProperty(KOV, obj1, relation_property, 0);
        WriteGProperty(KOV, obj2, relation_property, 0);
    }
];
```

§5. Various To Various Relations. Here the relation is represented by an array holding its metadata. Each object in the domain of the relation provides two properties, holding its left index and its right index. The index is its position in the left or right domain. For instance, suppose we relate things to doors, and there are five things in the world, two of which are doors; then the left indexes will range from 0 to 4, while the right indexes will range from 0 to 1. It's very likely that the doors will have different left and right indexes. (If the relation relates a given kind to itself, say doors to doors, then left and right indexes will always be equal.)

It is possible for either the left or right domain set to be an enumerated kind of value, where the I6 representation of values is 1, 2, 3, ..., N, where there are N possibilities. In that case we obtain the index simply by subtracting 1 in order to begin from 0. We mark the domain set as being a KOV rather than a kind of object by storing 0 instead of a property in the relevant part of the relation metadata: note that 0 is not a valid property number.

The structure for a relation consists of eight `-->` words, followed by a bitmap in which we store 16 bits in each `-->` word. (Yes, this is wasteful in Glulx, where `-->` words store 32 bits, but memory is not in short supply in Glulx and the total cost of relations is in practice small; we prefer to keep all the code involved simple.) The structure is precompiled by the Inform compiler: we do not create new ones on the fly.

In the case of a symmetric various to various relation, we could in theory save memory once again by storing only the lower triangle of the bitmap, but the time and complexity overhead are not worth it. When asserting that $O_1 \sim O_2$ for a symmetric V-to-V, we also automatically assert that $O_2 \sim O_1$, thus maintaining the bitmap as a symmetric matrix; but in reading the bitmap, we look only at the lower triangle. This costs a little time, but has the advantage of allowing the route-finding routine for V-to-V to use the same code for symmetric and asymmetric relations.

If this all seems rather suboptimally programmed in order to reduce code complexity, I can only say that careless drafts here were the source of some extremely difficult bugs to find.

```
Constant VTOVS_LEFT_INDEX_PROP = 0;
Constant VTOVS_RIGHT_INDEX_PROP = 1;
Constant VTOVS_LEFT_DOMAIN_SIZE = 2;
Constant VTOVS_RIGHT_DOMAIN_SIZE = 3;
Constant VTOVS_LEFT_PRINTING_ROUTINE = 4;
Constant VTOVS_RIGHT_PRINTING_ROUTINE = 5;
Constant VTOVS_CACHE_BROKEN = 6;
Constant VTOVS_CACHE = 7;
```

```
[ Relation_NowVtoV obj1 relation obj2 sym pr pr2 i1 i2 vtov_structure;
    if (sym && (obj2 ~= obj1)) { Relation_NowVtoV(obj2, relation, obj1, false); }
    vtov_structure = relation-->RR_STORAGE;
    pr = vtov_structure-->VTOVS_LEFT_INDEX_PROP;
    pr2 = vtov_structure-->VTOVS_RIGHT_INDEX_PROP;
    vtov_structure-->VTOVS_CACHE_BROKEN = true; ! Mark any cache as broken
    if (pr) {
        if ((obj1 ofclass Object) && (obj1 provides pr)) i1 = obj1.pr;
        else return RunTimeProblem(RTP_IMPREL, obj1, relation);
    } else i1 = obj1-1;
    if (pr2) {
        if ((obj2 ofclass Object) && (obj2 provides pr2)) i2 = obj2.pr2;
        else return RunTimeProblem(RTP_IMPREL, obj2, relation);
    } else i2 = obj2-1;
    pr = i1*(vtov_structure-->VTOVS_RIGHT_DOMAIN_SIZE) + i2;
    i1 = IncreasingPowersOfTwo_TB-->(pr%16);
    pr = pr/16 + 8;
    vtov_structure-->pr = (vtov_structure-->pr) | i1;
];
```

```
[ Relation_NowNVtoV obj1 relation obj2 sym pr pr2 i1 i2 vtov_structure;
    if (sym && (obj2 ~= obj1)) { Relation_NowNVtoV(obj2, relation, obj1, false); }
    vtov_structure = relation-->RR_STORAGE;
    pr = vtov_structure-->VTOVS_LEFT_INDEX_PROP;
    pr2 = vtov_structure-->VTOVS_RIGHT_INDEX_PROP;
    vtov_structure-->VTOVS_CACHE_BROKEN = true; ! Mark any cache as broken
    if (pr) {
        if ((obj1 ofclass Object) && (obj1 provides pr)) i1 = obj1.pr;
        else return RunTimeProblem(RTP_IMPREL, obj1, relation);
    } else i1 = obj1-1;
    if (pr2) {
        if ((obj2 ofclass Object) && (obj2 provides pr2)) i2 = obj2.pr2;
        else return RunTimeProblem(RTP_IMPREL, obj2, relation);
    } else i2 = obj2-1;
    pr = i1*(vtov_structure-->VTOVS_RIGHT_DOMAIN_SIZE) + i2;
    i1 = IncreasingPowersOfTwo_TB-->(pr%16);
    pr = pr/16 + 8;
    if ((vtov_structure-->pr) & i1) vtov_structure-->pr = vtov_structure-->pr - i1;
];
```

```
[ Relation_TestVtoV obj1 relation obj2 sym pr pr2 i1 i2 vtov_structure;
    vtov_structure = relation-->RR_STORAGE;
    pr = vtov_structure-->VTOVS_LEFT_INDEX_PROP;
    pr2 = vtov_structure-->VTOVS_RIGHT_INDEX_PROP;
    if (sym && (obj2 > obj1)) { sym = obj1; obj1 = obj2; obj2 = sym; }
    if (pr) {
        if ((obj1 ofclass Object) && (obj1 provides pr)) i1 = obj1.pr;
        else { RunTimeProblem(RTP_IMPREL, obj1, relation); rfalse; }
    } else i1 = obj1-1;
    if (pr2) {
        if ((obj2 ofclass Object) && (obj2 provides pr2)) i2 = obj2.pr2;
        else { RunTimeProblem(RTP_IMPREL, obj2, relation); rfalse; }
    } else i2 = obj2-1;
    pr = i1*(vtov_structure-->VTOVS_RIGHT_DOMAIN_SIZE) + i2;
    i1 = IncreasingPowersOfTwo_TB-->(pr%16);
    pr = pr/16 + 8;
    if ((vtov_structure-->pr) & i1) rtrue; rfalse;
];
```

§**6. Equivalence Relations.** For every equivalence relation there is a corresponding function f such that $x \sim y$ if and only if $f(x) = f(y)$, where $f(x)$ is a number identifying the equivalence class of x. Rather than inefficiently storing a large relation bitmap (and then having a very complicated time updating it to keep the relation transitive), we store f: that is, for every object in the domain set, there is a property `prop` such that `O.prop` is the value $f(O)$.

```
[ Relation_NowEquiv obj1 relation_property obj2 big little;
    big = obj1.relation_property; little = obj2.relation_property;
    if (big == little) return;
    if (big < little) { little = obj1.relation_property; big = obj2.relation_property; }
    objectloop (obj1 provides relation_property)
        if (obj1.relation_property == big) obj1.relation_property = little;
];
[ Relation_NowNEquiv obj1 relation_property obj2 old new;
    old = obj1.relation_property; new = obj2.relation_property;
    if (old ~= new) return;
    new = 0;
    objectloop (obj2 provides relation_property)
        if (obj2.relation_property > new) new = obj2.relation_property;
    new++;
    obj1.relation_property = new;
];
[ Relation_NowEquivV obj1 obj2 KOV relation_property n big little i;
    big = GProperty(KOV, obj1, relation_property);
    little = GProperty(KOV, obj2, relation_property);
    if (big == little) return;
    if (big < little) {
        little = GProperty(KOV, obj1, relation_property);
        big = GProperty(KOV, obj2, relation_property);
    }
    n = KOVDomainSize(KOV);
    for (i=1: i<=n: i++)
        if (GProperty(KOV, i, relation_property) == big)
```

```
            WriteGProperty(KOV, i, relation_property, little);
];
[ Relation_NowNEquivV obj1 obj2 KOV relation_property n old new i;
    old = GProperty(KOV, obj1, relation_property);
    new = GProperty(KOV, obj2, relation_property);
    if (old ~= new) return;
    new = 0;
    n = KOVDomainSize(KOV);
    for (i=1: i<=n: i++)
        if (GProperty(KOV, i, relation_property) > new)
            new = GProperty(KOV, i, relation_property);
    new++;
    WriteGProperty(KOV, obj1, relation_property, new);
];
```

§7. Show Various to Various. The rest of the code for relations has no use except for debugging: it implements the RELATIONS testing command. Speed is unimportant here.

```
[ Relation_ShowVtoV relation sym x obj1 obj2 pr pr2 proutine1 proutine2 vtov_structure;
    vtov_structure = relation-->RR_STORAGE;
    pr = vtov_structure-->VTOVS_LEFT_INDEX_PROP;
    pr2 = vtov_structure-->VTOVS_RIGHT_INDEX_PROP;
    proutine1 = vtov_structure-->VTOVS_LEFT_PRINTING_ROUTINE;
    proutine2 = vtov_structure-->VTOVS_RIGHT_PRINTING_ROUTINE;
    if (pr && pr2) {
        objectloop (obj1 provides pr)
        objectloop (obj2 provides pr2) {
                if (sym && obj2 > obj1) continue;
                if (Relation_TestVtoV(obj1, relation, obj2)) {
                    if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
                    print "  ", (The) obj1;
                    if (sym) print "  <=>  "; else print "  >=>  ";
                    print (the) obj2, "^";
                }
        }
        return;
    }
    if (pr && (pr2==0)) {
        objectloop (obj1 provides pr)
        for (obj2=1:obj2<=vtov_structure-->VTOVS_RIGHT_DOMAIN_SIZE:obj2++) {
                if (Relation_TestVtoV(obj1, relation, obj2)) {
                    if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
                    print "  ", (The) obj1, "  >=>  ";
                    (proutine2).call(obj2);
                    print "^";
                }
        }
        return;
    }
    if ((pr==0) && (pr2)) {
        for (obj1=1:obj1<=vtov_structure-->2:obj1++)
        objectloop (obj2 provides pr2) {
                if (Relation_TestVtoV(obj1, relation, obj2)) {
                    if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
                    print "  ";
                    (proutine1).call(obj1);
                    print "  >=>  ", (the) obj2, "^";
                }
        }
        return;
    }
    for (obj1=1:obj1<=vtov_structure-->2:obj1++)
        for (obj2=1:obj2<=vtov_structure-->VTOVS_RIGHT_DOMAIN_SIZE:obj2++)
            if (Relation_TestVtoV(obj1, relation, obj2)) {
                if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
                print "  ";
                (proutine1).call(obj1);
                print "  >=>  ";
                (proutine2).call(obj2);
                print "^";
        }
];
```

§8. Show One to One.

```
[ Relation_ShowOtoO relation sym x relation_property t N obj1 obj2;
    relation_property = relation-->RR_STORAGE;
    t = KindBaseTerm(relation-->RR_KIND, 0); ! Kind of left term
    N = KOVDomainSize(t);
    if (t == OBJECT_TY) {
        objectloop (obj1 provides relation_property) {
            obj2 = obj1.relation_property;
            if (sym && obj2 < obj1) continue;
            if (obj2 == 0) continue;
            if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
            print "  ", (The) obj1;
            if (sym) print "  ==  "; else print "  >=>  ";
            print (the) obj2, "^";
        }
    } else {
        for (obj1=1: obj1<=N: obj1++) {
            obj2 = GProperty(t, obj1, relation_property);
            if (sym && obj2 < obj1) continue;
            if (obj2 == 0) continue;
            if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
            print "  ";
            PrintKindValuePair(t, obj1);
            if (sym) print "  ==  "; else print "  >=>  ";
            PrintKindValuePair(t, obj2);
            print "^";
        }
    }
];
```

§**9. Show Reversed One to One.** There's no such kind of relation as this: but the same code used to show 1-to-1 relations is also used to show various-to-1 relations, since the storage is the same. To show 1-to-various relations, we need a transposed form of the same code in which left and right are exchanged: this is it.

```
[ Relation_RShowOtoO relation sym x relation_property obj1 obj2 t1 t2 N1 N2;
    relation_property = relation-->RR_STORAGE;
    t1 = KindBaseTerm(relation-->RR_KIND, 0); ! Kind of left term
    t2 = KindBaseTerm(relation-->RR_KIND, 1); ! Kind of right term
    if (t2 == OBJECT_TY) {
        if (t1 == OBJECT_TY) {
            objectloop (obj1) {
                objectloop (obj2 provides relation_property) {
                    if (obj2.relation_property ~= obj1) continue;
                    if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
                    print "  ", (The) obj1;
                    print "  >=>  ";
                    print (the) obj2, "^";
                }
            }
        } else {
            N1 = KOVDomainSize(t1);
            for (obj1=1: obj1<=N1: obj1++) {
                objectloop (obj2 provides relation_property) {
                    if (obj2.relation_property ~= obj1) continue;
                    if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
                    print "  "; PrintKindValuePair(t1, obj1);
                    print "  >=>  ";
                    print (the) obj2, "^";
                }
            }
        }
    } else {
        N2 = KOVDomainSize(t2);
        if (t1 == OBJECT_TY) {
            objectloop (obj1) {
                for (obj2=1: obj2<=N2: obj2++) {
                    if (GProperty(t2, obj2, relation_property) ~= obj1) continue;
                    if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
                    print "  ", (The) obj1;
                    print "  >=>  ";
                    PrintKindValuePair(t2, obj2);
                    print "^";
                }
            }
        } else {
            N1 = KOVDomainSize(t1);
            for (obj1=1: obj1<=N1: obj1++) {
                for (obj2=1: obj2<=N2: obj2++) {
                    if (GProperty(t2, obj2, relation_property) ~= obj1) continue;
                    if (x == 0) { print (string) relation-->RR_DESCRIPTION, ":^"; x=1; }
                    print "  ";
                    PrintKindValuePair(t1, obj1);
                    print "  >=>  ";
                    PrintKindValuePair(t2, obj2);
                    print "^";
                }
            }
        }
    }
];
```

§10. Show Equivalence.

```
[ RSE_Flip KOV v relation_property x;
    x = GProperty(KOV, v, relation_property); x = -x;
    WriteGProperty(KOV, v, relation_property, x);
];
[ RSE_Set KOV v relation_property;
    if (GProperty(KOV, v, relation_property) < 0) rtrue; rfalse;
];
[ Relation_ShowEquiv relation relation_property obj1 obj2 v c d somegroups t N x;
    print (string) relation-->RR_DESCRIPTION, ":^";
    relation_property = relation-->RR_STORAGE;
    t = KindBaseTerm(relation-->RR_KIND, 0); ! Kind of left term
    N = KOVDomainSize(t);
    if (t == OBJECT_TY) {
        objectloop (obj1 provides relation_property)
            obj1.relation_property = -(obj1.relation_property);
        objectloop (obj1 provides relation_property) {
            if (obj1.relation_property < 0) {
                v = obj1.relation_property; c = 0;
                objectloop (obj2 has workflag2) give obj2 ~workflag2;
                objectloop (obj2 provides relation_property) {
                    if (obj2.relation_property == v) {
                        give obj2 workflag2;
                        obj2.relation_property = -v;
                        c++;
                    }
                }
                if (c>1) {
                    somegroups = true;
                    print "  { ";
                    WriteListOfMarkedObjects(ENGLISH_BIT);
                    print " }^";
                } else obj1.relation_property = v;
            }
        }
        objectloop (obj2 has workflag2) give obj2 ~workflag2;
        c = 0; objectloop (obj1 provides relation_property)
            if (obj1.relation_property < 0) { c++; give obj1 workflag2; }
        if (c == 0) return;
        if (somegroups) print "  and "; else print "  ";
        if (c < 4) { WriteListOfMarkedObjects(ENGLISH_BIT); print " in"; }
        else print c;
        if (c == 1) print " a";
        print " single-member group";
        if (c > 1) print "s";
        print "^";
        objectloop (obj1 provides relation_property)
            if (obj1.relation_property < 0)
                obj1.relation_property = -(obj1.relation_property);
    } else {
        ! A slower method, since we have less efficient storage:
        for (obj1 = 1: obj1 <= N: obj1++)
            RSE_Flip(t, obj1, relation_property);
        for (obj1 = 1: obj1 <= N: obj1++) {
            if (RSE_Set(t, obj1, relation_property)) {
                v = GProperty(t, obj1, relation_property);
                c = 0;
                for (obj2 = 1: obj2 <= N: obj2++)
                    if (GProperty(t, obj2, relation_property) == v)
                        c++;
                if (c>1) {
                    somegroups = true;
                    print "  {";
                    d = 0;
                    for (obj2 = 1: obj2 <= N: obj2++) {
                        if (GProperty(t, obj2, relation_property) == v) {
                            print " "; PrintKindValuePair(t, obj2);
                            if (d < c-1) print ","; print " ";
                            RSE_Flip(t, obj2, relation_property);
                            d++;
                        }
                    }
                    print "}^";
                } else WriteGProperty(t, obj1, relation_property, v);
            }
        }
        objectloop (obj2 has workflag2) give obj2 ~workflag2;
        c = 0;
        for (obj1 = 1: obj1 <= N: obj1++)
            if (RSE_Set(t, obj1, relation_property)) c++;
        if (c == 0) return;
        if (somegroups) print "  and "; else print "  ";
        if (c == 1) print "a"; else print c;
        print " single-member group";
        if (c > 1) print "s";
        print "^";
        for (obj1 = 1: obj1 <= N: obj1++)
            if (RSE_Set(t, obj1, relation_property))
                RSE_Flip(t, obj1, relation_property);
    }
];
```

§11. Map Route-Finding. The general problem we have to solve here is: given $x, y \in R$, where R is the set of rooms and we write $x \sim y$ if there is a map connection from x to y,

(i) find the smallest m such that there exist $x = r_1 \sim r_2 \sim ... \sim r_m = y \in R$, or determine that no such m exists, and

(ii) find d, the first direction to take from x to lead to r_2, or set $d = 0$ if no such path exists or if $m = 1$ so that $x = y$.

Thus a typical outcome might be either "a shortest path from the Town Square to the Hilltop takes 11 moves, starting by going northeast from the Town Square", or alternatively "there's no path from the Town Square to the Hilltop at all". Note that the length of the shortest path is unambiguous, but that there might be many alternative paths of this minimum length: we deliberately do not specify which path is chosen if so, and the two algorithms used below do not necessarily choose the same one.

Route-finding is not an easy operation in computation terms: the various algorithms available have theoretical running times which are easy (if sobering) to compute, but which are not in practice typical of what will happen, because they are quite sensitive to the map in question. Are all the rooms laid out in a long line? Are there clusters of connected rooms like islands? Are there dense clumps of interconnecting rooms? Are there huge but possibly time-saving loops? And so on. Overhead is also important. We present a choice of two algorithms: the "fast" one has a theoretical running time of $O(n^3)$, where n is the number of rooms, whereas the "slow" one runs in $O(n^2)$, yet in practice the fast one easily outperforms the slow on typical heavy-use cases with large maps.

The other issue is memory usage: we essentially have to strike a bargain between speed and memory overhead. Our "slow" algorithm needs only $O(n)$ storage, whereas our "fast" algorithm needs $O(n^2)$, and this is very significant in the Z-machine where array space is in desperately short supply and where, if $n > 50$ or so, the user is already likely to be fighting for the last few bytes in readable memory.

The user is therefore offered the choice, by selecting the use options "Use fast route-finding" and "Use slow route-finding": and the defaults, if neither option is explicitly set, are fast on Glulx and slow on the Z-machine. If both use options are explicitly set – which might happen due to a disagreement between extensions – "fast" wins.

```
#ifndef FAST_ROUTE_FINDING;
#ifndef SLOW_ROUTE_FINDING;
#ifdef TARGET_GLULX;
Constant FAST_ROUTE_FINDING;
#ifnot;
Constant SLOW_ROUTE_FINDING;
#endif;
#endif;
#endif;
```

§12. Cache Control. We provide code to enable our route-finding algorithms to cache their partial results from one usage to the next (though at present only the "fast" algorithm does this). The difficulty here is that the result of a route search depends on three things, any of which may change:

(a) which subset of rooms we are route-finding through;

(b) which subset of doors we are allowing ourselves to use; and

(c) the current map connections between rooms.

We keep track of (c) by watching for calls to `SignalMapChange()` from the routines in "WorldModel.i6t" which alter the map. (a) and (b), however, require tracking from call to call what the current subset of rooms and doors is. (It is not sufficient to remember the criteria used last time and this time, because circumstances could have changed such that the criteria produce a different outcome. For instance, searching through lighted rooms and using unlocked doors will produce a different result if a door has been locked or unlocked since last time, or if a room has become lighted or not.) We store the set of applicable rooms and doors by enumerating them in the property `room_index` and by the flags in the `DoorRoutingViable` array respectively.

```
Constant NUM_DOORS = {-value:Data::Instances::count(K_door)};
Constant NUM_ROOMS = {-value:Data::Instances::count(K_room)};

Array DoorRoutingViable -> NUM_DOORS+1;

Global map_has_changed = true;
Global last_filter; Global last_use_doors;

[ SignalMapChange; map_has_changed = true; ];

[ MapRouteTo from to filter use_doors count  oy oyi ds;
    if (from == nothing) return nothing;
    if (to == nothing) return nothing;
    if (from == to) return nothing;
    if ((filter) && (filter(from) == 0)) return nothing;
    if ((filter) && (filter(to) == 0)) return nothing;
    if ((last_filter ~= filter) || (last_use_doors ~= use_doors)) map_has_changed = true;
    oyi = 0;
    objectloop (oy has mark_as_room) {
        if ((filter == 0) || (filter(oy))) {
            if (oy.room_index == -1) map_has_changed = true;
            oy.room_index = oyi++;
        } else {
            if (oy.room_index >= 0) map_has_changed = true;
            oy.room_index = -1;
        }
    }
    oyi = 0;
    objectloop (oy ofclass K4_door) {
        ds = false;
        if ((use_doors & 2) ||
            (oy has open) || ((oy has openable) && (oy hasnt locked))) ds = true;
        if (DoorRoutingViable->oyi ~= ds) map_has_changed = true;
        DoorRoutingViable->oyi = ds;
        oyi++;
    }
    if (map_has_changed) {
        #ifdef FAST_ROUTE_FINDING; ComputeFWMatrix(filter, use_doors); #endif;
        map_has_changed = false; last_filter = filter; last_use_doors = use_doors;
    }
    #ifdef FAST_ROUTE_FINDING;
    if (count) return FastCountRouteTo(from, to, filter, use_doors);
    return FastRouteTo(from, to, filter, use_doors);
    #ifnot;
    if (count) return SlowCountRouteTo(from, to, filter, use_doors);
    return SlowRouteTo(from, to, filter, use_doors);
    #endif;
];
```

§13. Fast Route-Finding. The following is a form of Floyd's adaptation of Warshall's algorithm for finding the transitive closure of a directed graph.

We need to store a matrix which for each pair of rooms R_i and R_j records a_{ij}, the shortest path length from R_i to R_j or 0 if no path exists, and also d_{ij}, the first direction to take on leaving R_i along a shortest path to R_j, or 0 if no path exists. For the sake of economy we represent the directions as their instance counts (numbered from 0 in order of creation), not as their direction object values, and then store a single word for each pair (i, j): we store $d_{ij} + Da_{ij}$. This restricts us on a signed 16-bit virtual machine, and with the conventional set of $D = 12$ directions, to the range $0 \leq a_{ij} \leq 5461$, that is, to path lengths of 5461 steps or fewer. A work of IF with 5461 rooms will not fit in the Z-machine anyway: such a work would be on Glulx, which is 32-bit, and where $0 \leq a_{ij} \leq 357,913,941$.

We begin with $a_{ij} = 0$ for all pairs except where there is a viable map connection between R_i and R_j: for those we set $a_{ij} = 1$ and d_{ij} equal to the direction of that map connection.

Following Floyd and Warshall we test if each known shortest path R_x to R_y can be used to shorten the best known path from R_x to anywhere else: that is, we look for cases where $a_{xy} + a_{yj} < a_{xj}$, since those show that going from R_x to R_j via R_y takes fewer steps than going directly. See for instance Robert Sedgewick, *Algorithms* (1988), chapter 32.

The trouble with the Floyd-Warshall algorithm is not so much that it takes in principle $O(n^3)$ time to construct the matrix: it does, but the coefficient is low, and in the early stages of the outer loop the fact that the vertex degree is at most D and usually much lower helps to reduce the work further. The trouble is that there is no way to compute only the part of the matrix we want: we have to have the entire thing, and that means storing n^2 words of data, by which point we have computed not only the fastest route from R_x to R_y but also the fastest route from anywhere to anywhere else. Even when the original map is sparse, the Floyd-Warshall matrix is not, and it is difficult to store in any very compressed way without greatly increasing the complexity of the code. This is why we cache the results: we might as well, since we had to build the entire memory structure anyway, and it means the time expense is only paid once (or once for every time the state of doors and map connections changes), and the cache is useful for all future routes whatever their endpoints.

```
#ifdef FAST_ROUTE_FINDING;
Array FWMatrix --> NUM_ROOMS*NUM_ROOMS;

[ FastRouteTo from to filter use_doors diri i dir oy;
    if (from == to) return nothing;
    i = (FWMatrix-->(from.room_index*NUM_ROOMS + to.room_index))/No_Directions;
    if (i == 0) return nothing;
    diri = (FWMatrix-->(from.room_index*NUM_ROOMS + to.room_index))%No_Directions;
    i=0; objectloop (dir ofclass K3_direction) {
        if (i == diri) return dir;
        i++;
    }
    return nothing;
];

[ FastCountRouteTo from to filter use_doors  k;
    if (from == to) return 0;
    k = (FWMatrix-->(from.room_index*NUM_ROOMS + to.room_index))/No_Directions;
    if (k == 0) return -1;
    return k;
];

[ ComputeFWMatrix filter use_doors  oy ox oj axy ayj axj dir diri nd row;
    objectloop (oy has mark_as_room) if (oy.room_index >= 0)
        objectloop (ox has mark_as_room) if (ox.room_index >= 0)
            FWMatrix-->(oy.room_index*NUM_ROOMS + ox.room_index) = 0;
    objectloop (oy has mark_as_room) if (oy.room_index >= 0) {
        row = (oy.IK1_Count)*No_Directions;
        for (diri=0: diri<No_Directions: diri++) {
            ox = Map_Storage-->(row+diri);
            if ((ox) && (ox has mark_as_room) && (ox.room_index >= 0)) {
                FWMatrix-->(oy.room_index*NUM_ROOMS + ox.room_index) = No_Directions + diri;
                continue;
            }
            if (use_doors && (ox ofclass K4_door) &&
                ((use_doors & 2) || (DoorRoutingViable->(ox.IK4_Count)))) {
                @push location; location = oy;
                ox = ox.door_to();
                @pull location;
                if ((ox) && (ox has mark_as_room) && (ox.room_index >= 0)) {
                    FWMatrix-->(oy.room_index*NUM_ROOMS + ox.room_index) = No_Directions + diri;
                    continue;
                }
            }
        }
    }
    objectloop (oy has mark_as_room) if (oy.room_index >= 0)
        objectloop (ox has mark_as_room) if (ox.room_index >= 0) {
            axy = (FWMatrix-->(ox.room_index*NUM_ROOMS + oy.room_index))/No_Directions;
            if (axy > 0)
                objectloop (oj has mark_as_room) if (oj.room_index >= 0) {
                    ayj = (FWMatrix-->(oy.room_index*NUM_ROOMS + oj.room_index))/No_Directions;
                    if (ayj > 0) {
                        !print "Is it faster to go from ", (name) ox, " to ",
                        !   (name) oj, " via ", (name) oy, "?^";
                        axj = (FWMatrix-->(ox.room_index*NUM_ROOMS + oj.room_index))/
                            No_Directions;
                        if ((axj == 0) || (axy + ayj < axj)) {
                            !print "Yes^";
                            FWMatrix-->(ox.room_index*NUM_ROOMS + oj.room_index) =
                                (axy + ayj)*No_Directions +
                                (FWMatrix-->(ox.room_index*NUM_ROOMS + oy.room_index))%
                                    No_Directions;
                        }
                    }
                }
        }
];
#ENDIF;
```

§**14. Slow Route-Finding.** The alternative algorithm, used when only $O(n)$ memory is available, computes only some of the shortest paths leading to R_y, and is not cached – both because the storage is likely to be reused often by other searches and because there is little gain from doing so, given that a subsequent search with different endpoints will not benefit from the results of this one. On the other hand, to call it "slow" is a little unfair. It is somewhat like Prim's algorithm for finding a minimum spanning tree, rooted at R_y, and grows the tree outward from R_y until either R_x is reached – in which case we stop immediately – or the (directed) component containing R_y has been exhausted – in which case R_x, which must lie outside this, can have no path to R_y. In principle, the running time is $O(dn^2)$, where $d \leq D$ is the maximum vertex degree and n is the number of rooms in the component containing R_y: in practice the degree is often much less than 12, while the algorithm finishes quickly in cases where R_y is relatively isolated and inaccessible or where a shortish route does exist, and those are very common cases in typical usage. There will be circumstances where, because few routes need to be found and because of the shape of the map, the "slow" algorithm will outperform the "fast" one: this is why the user is allowed to control which algorithm is used.

For each room R_z, the property `vector` stores the direction object of the way to go to its parent room in the tree rooted at R_y. Thus if the algorithm succeeds in finding a route from R_x to R_y then we generate the route by starting at R_x and repeatedly going in the `vector` direction from where we currently stand until we reach R_y. Since every room needs a `vector` value, this requires n words of storage. (The `vector` values store only enough of the minimal spanning tree to go upwards through the tree, but that's the only way we need to traverse it.)

The method can be summed up thus:

(a) Begin with every vector blank except that of R_y, the destination.
(b) Repeatedly: For every room in the domain set, try each direction: if this leads to a room whose vector was determined on the last round (*not* on this one, as that may be a suboptimal route), set the vector to point to that room.
(c) Stop as soon as the vector from the origin is set, or when a round happens in which no further vectors are found: in which case, we have completely explored the component of the map from which the destination can be reached, and the origin isn't in it, so we can return "no".

To prove the correctness of this, we show inductively that after round n we have set the `vector` for every room having a shortest path to R_y of length n, and that every `vector` points to a room having a `vector` in the direction of the shortest path from there to R_y.

```
#ifndef FAST_ROUTE_FINDING;
[ SlowRouteTo from to filter use_doors  obj dir in_direction progressed sl through_door;
    if (from == nothing) return nothing;
    if (to == nothing) return nothing;
    if (from == to) return nothing;
    objectloop (obj has mark_as_room) obj.vector = 0;
    to.vector = 1;
    !print "Routing from ", (the) from, " to ", (the) to, "^";
    while (true) {
        progressed = false;
        !print "Pass begins^";
        objectloop (obj has mark_as_room)
            if ((filter == 0) || (filter(obj)))
                if (obj.vector == 0)
                    objectloop (dir ofclass K3_direction) {
                        in_direction = Map_Storage-->((obj.IK1_Count)*No_Directions + dir.IK3_Count);
                        if (in_direction == nothing) continue;
                        !print (the) obj, " > ", (the) dir, " > ", (the) in_direction, "^";
                        if ((in_direction)
                            && (in_direction has mark_as_room)
                            && (in_direction.vector > 0)
```

```
                              && ((filter == 0) || (filter(in_direction)))) {
                              obj.vector = dir | WORD_HIGHBIT;
                              !print "* ", (the) obj, " vector is ", (the) dir, "^";
                              progressed = true;
                              continue;
                          }
                          if (use_doors && (in_direction ofclass K4_door) &&
                              ((use_doors & 2) ||
                              (in_direction has open) ||
                              ((in_direction has openable) && (in_direction hasnt locked)))) {
                              sl = location; location = obj;
                              through_door = in_direction.door_to();
                              location = sl;
                              !print "Through door is ", (the) through_door, "^";
                              if ((through_door)
                                  && (through_door has mark_as_room)
                                  && (through_door.vector > 0)
                                  && ((filter == 0) || (filter(through_door)))) {
                                  obj.vector = dir | WORD_HIGHBIT;
                                  !print "* ", (the) obj, " vector is ", (the) dir, "^";
                                  progressed = true;
                                  continue;
                              }
                          }
                      }
        objectloop (obj has mark_as_room) obj.vector = obj.vector &~ WORD_HIGHBIT;
        if (from.vector) return from.vector;
        if (progressed == false) return from.vector;
    }
];
[ SlowCountRouteTo from to filter use_doors obj i;
    if (from == nothing) return -1;
    if (to == nothing) return -1;
    if (from == to) return 0;
    if (from has mark_as_room && to has mark_as_room) {
        obj = MapRouteTo(from,to,filter,use_doors);
        if (obj == nothing) return -1;
        i = 0; obj = from;
        while ((obj ~= to) && (i<NUM_ROOMS)) { i++; obj = MapConnection(obj,obj.vector); }
        return i;
    }
    return -1;
];
#ENDIF;
```

§15. Relation Route-Finding. The general problem we have to solve here is: given $x, y \in D$, where $\sim$ is a relation on a domain set D of objects,

(i) find the smallest n such that there exist $x = r_1 \sim r_2 \sim ... \sim r_n = y \in D$ such that $r_i \sim r_{i+1}$, or determine that no such n exists, and if so

(ii) find a value of r_2 in such a "route" between x and y, or set $r_2 = 0$ if $x = y$ so that $n = 1$.

While in general a relation can have different left and right domains (a relation between doors and rooms, say), route-finding on those relations is unlikely to be very useful, so is discouraged. (In the case of doors and rooms, a route could never be longer than 1 step, since no object is both a door and a room, for instance.) The "fast" V-to-V algorithm requires D to have the same left and right domains; NI compiles the memory caches for V-to-V relations to force any cases with different domains into using the "slow" algorithm.

`MAX_ROUTE_LENGTH` is used simply as a sanity check to prevent hangs if something should go wrong, for instance if the property of a 1-to-V relation has been modified by some third-party code in such a way that it loses its defining invariant.

```
Constant MAX_ROUTE_LENGTH = {-value:Data::Instances::count(K_object)} + 32;
[ RelationRouteTo relation from to count  handler;
    if (count) {
        if (from == nothing) return -1;
        if (to == nothing) return -1;
        if (relation == 0) return -1;
    } else {
        if (from == nothing) return nothing;
        if (to == nothing) return nothing;
        if (relation == 0) return nothing;
    }
    if (from == to) return nothing;
    if (((relation-->RR_PERMISSIONS) & RELS_ROUTE_FIND) == 0) {
        RunTimeProblem(RTP_ROUTELESS);
        return nothing;
    }
    if (relation-->RR_STORAGE == 0) return nothing;
    handler = relation-->RR_HANDLER;
    if (count) return handler(relation, RELS_ROUTE_FIND_COUNT, from, to);
    return handler(relation, RELS_ROUTE_FIND, from, to);
];
[ RelFollowVector rv from to  obj i;
    if (rv == nothing) return -1;
    i = 0; obj = from;
    while ((obj ~= to) && (i<=MAX_ROUTE_LENGTH)) { i++; obj = obj.vector; }
    return i;
];
```

§16. One To Various Route-Finding. Here we can immediately determine, given y, the unique y' such that $y' \sim y$, so finding a path from x to y is a matter of following the only path leading to y and seeing if it ever passed through x; thus the running time is $O(n)$, where n is the size of the domain. It would be pointless to cache this.

Note that we can assume here that $x \neq y$, or rather, that `from ~= to`, because that case has already been taken care of.

```
[ OtoVRelRouteTo relation_property from to previous;
    while ((to) && (to provides relation_property) && (to.relation_property)) {
        previous = to.relation_property;
        previous.vector = to;
        if (previous == from) return to;
        to = previous;
    }
    return nothing;
];
```

§17. Various To One Route-Finding. This time the simplifying assumption is that, given x, we can immediately determine the unique x' such that $x \sim x'$, so it suffices to follow the only path forwards from x and see if it ever reaches y. The routine is not quite a mirror image of the one above, because both have the same return requirements: we have to ensure that the `vector` properties lay out the path, and also return the next step after x.

```
[ VtoORelRouteTo relation_property from to next  start;
    start = from;
    while ((from) && (from provides relation_property) && (from.relation_property)) {
        next = from.relation_property;
        from.vector = next;
        if (from == to) return start.vector;
        from = next;
    }
    return nothing;
];
```

§18. Slow Various To Various Route-Finding. Now there are no simplifying assumptions and the problem is essentially the same as the one solved for route-finding in the map, above. Once again we present two different algorithms: first, a form of Prim's algorithm for minimal spanning trees. Note that, whereas this algorithm was not always so "slow" for the map – because of the fairly low vertex degrees involved, i.e., because most rooms had few connections to other rooms – here the relation might well be almost complete, with almost all the objects related to each other, and then the algorithm will indeed be "slow". So it is likely that the "fast" algorithm will always be better, if the memory can be spared for it.

We use the fast algorithm for a given relation if and only if the NI compiler has allocated the necessary cache memory; the two use options above, for map route-finding, don't control this.

```
[ VtoVRelRouteTo relation from to count obj obj2 related progressed left_ix pr2 i vtov_structure;
    vtov_structure = relation-->RR_STORAGE;
    if (vtov_structure-->VTOVS_CACHE)
        return FastVtoVRelRouteTo(relation, from, to, count);
    left_ix = vtov_structure-->VTOVS_LEFT_INDEX_PROP;
    pr2 = vtov_structure-->VTOVS_RIGHT_INDEX_PROP;
    objectloop (obj ofclass Object && obj provides vector) obj.vector = 0;
    to.vector = 1;
    while (true) {
        progressed = false;
        objectloop (obj ofclass Object && obj provides left_ix)
            if (obj.vector == 0) {
                objectloop (obj2 ofclass Object && obj2 provides pr2 && obj2.vector > 0) {
                    if (Relation_TestVtoV(obj, relation, obj2)) {
                        obj.vector = obj2 | WORD_HIGHBIT;
                        progressed = true;
                        continue;
                    }
                }
            }
        objectloop (obj ofclass Object && obj provides left_ix)
            obj.vector = obj.vector &~ WORD_HIGHBIT;
        if (from.vector) break;
        if (progressed == false) break;
    }
    if (count) {
        if (from.vector == nothing) return -1;
        i = 0; obj = from;
        while ((obj ~= to) && (i<=MAX_ROUTE_LENGTH)) { i++; obj = obj.vector; }
        return i;
    }
    return from.vector;
];
```

§**19. Fast Various To Various Route-Finding.** Now, as above, a form of the Floyd-Warshall algorithm. The matrix is here stored in the cache of memory pointed to in the V-to-V relation structure. We are unable to combine a_{ij} and d_{ij} into a single cell of memory, so in fact we store two separate matrices: one for a_{ij} (this is `cache` below), the other for n_{ij}, where n_{ij} is the next object in the shortest path from O_i to O_j (this is `cache2` below).

Where $n < 256$ a shortest path must be such that $a_{ij} \leq 255$, so can be stored in a single byte, and we similarly store n_{ij} as the index of the object rather than the object value itself: the index ranges from 0 to $n-1$, so that $0 \leq n_{ij} < 255$ and we can use $n_{ij} = 255$ as a sentinel value meaning "no path". Although the reconversion of n_{ij} back into a valid object value takes a little time, it is only $O(n)$, and of course we know n is relatively small; and in this way we reduce the storage overhead to only n^2 bytes.

Where $n \geq 256$, we resign ourselves to storing two words for each pair (i, j), making $2n^2$ bytes of storage on the Z-machine and $4n^2$ bytes of storage on Glulx, but lookup of a cached result is slightly faster.

```
[ FastVtoVRelRouteTo relation from to count
    domainsize cache cache2 left_ix ox oy oj offset axy axj ayj;
    domainsize = relation-->RR_STORAGE-->2; ! Number of left instances
    left_ix = relation-->RR_STORAGE-->VTOVS_LEFT_INDEX_PROP;
    if ((from provides left_ix) && (to provides left_ix)) {
        if (domainsize < 256) {
            cache = relation-->RR_STORAGE-->VTOVS_CACHE;
            cache2 = cache + domainsize*domainsize;
            if (relation-->RR_STORAGE-->VTOVS_CACHE_BROKEN == true) {
                relation-->RR_STORAGE-->VTOVS_CACHE_BROKEN = false;
                objectloop (oy provides left_ix)
                    objectloop (ox provides left_ix)
                        if (Relation_TestVtoV(oy, relation, ox)) {
```

```
                    offset = ((oy.left_ix)*domainsize + (ox.left_ix));
                    cache->offset = 1;
                    cache2->offset = ox.left_ix;
                } else {
                    offset = ((oy.left_ix)*domainsize + (ox.left_ix));
                    cache->offset = 0;
                    cache2->offset = 255;
                }
        for (oy=0: oy<domainsize: oy++)
            for (ox=0: ox<domainsize: ox++) {
                axy = cache->(ox*domainsize + oy);
                if (axy > 0)
                    for (oj=0: oj<domainsize: oj++) {
                        ayj = cache->(oy*domainsize + oj);
                        if (ayj > 0) {
                            offset = ox*domainsize + oj;
                            axj = cache->offset;
                            if ((axj == 0) || (axy + ayj < axj)) {
                                cache->offset = (axy + ayj);
                                cache2->offset = cache2->(ox*domainsize + oy);
                            }
                        }
                    }
            }
    }
    if (count) {
        count = cache->((from.left_ix)*domainsize + (to.left_ix));
        if (count == 0) return -1;
        return count;
    }
    oy = cache2->((from.left_ix)*domainsize + (to.left_ix));
    if (oy < 255)
        objectloop (ox provides left_ix)
            if (ox.left_ix == oy) return oy;
    return nothing;
} else {
    cache = relation-->RR_STORAGE-->VTOVS_CACHE;
    cache2 = cache + WORDSIZE*domainsize*domainsize;
    if (relation-->RR_STORAGE-->VTOVS_CACHE_BROKEN == true) {
        relation-->RR_STORAGE-->VTOVS_CACHE_BROKEN = false;
        objectloop (oy provides left_ix)
            objectloop (ox provides left_ix)
                if (Relation_TestVtoV(oy, relation, ox)) {
                    offset = ((oy.left_ix)*domainsize + (ox.left_ix));
                    cache-->offset = 1;
                    cache2-->offset = ox;
                } else {
                    offset = ((oy.left_ix)*domainsize + (ox.left_ix));
                    cache-->offset = 0;
                    cache2-->offset = nothing;
                }
        for (oy=0: oy<domainsize: oy++)
            for (ox=0: ox<domainsize: ox++) {
```

```
                        axy = cache-->(ox*domainsize + oy);
                        if (axy > 0)
                            for (oj=0: oj<domainsize: oj++) {
                                ayj = cache-->(oy*domainsize + oj);
                                if (ayj > 0) {
                                    offset = ox*domainsize + oj;
                                    axj = cache-->offset;
                                    if ((axj == 0) || (axy + ayj < axj)) {
                                        cache-->offset = (axy + ayj);
                                        cache2-->offset = cache2-->(ox*domainsize + oy);
                                    }
                                }
                            }
                    }
            }
            if (count) {
                count = cache-->((from.left_ix)*domainsize + (to.left_ix));
                if (count == 0) return -1;
                return count;
            }
            return cache2-->((from.left_ix)*domainsize + (to.left_ix));
        }
    }
    if (count) return -1;
    return nothing;
];
```

§**20. Iterating Relations.** The following is provided to make it possible to run an I6 routine on each relation in turn. (Each right-way-round relation, at any rate.)

```
[ IterateRelations callback;
    {-call:Semantics::Relations::relations_command}
];
```

Figures Template B/figst

Purpose

To display figures and play sound effects.

§1. Resource Usage. We record whether pictures or sounds have been used before by storing single byte flags in the following array. (The extra 5 values allow for the fact that it can be legal to use low undeclared sound effect resource numbers in the Z-machine for short beeps, though this is deprecated in I7.)

Pictures and sounds are identified within blorb archives by resource ID numbers which count upwards from 1 in order of creation, but can mix pictures and sounds freely. (For instance, 1 might be a picture, 2 and 3 sound effects, then 4 a picture again, etc.) ID number 1 is in fact always a picture: it means the cover art, and is the I6 representation of the value "figure of cover".

```
Array ResourceUsageFlags ->
    ({-value:NUMBER_CREATED(blorb_figure)}+{-value:NUMBER_CREATED(blorb_sound)}+5);
```

§2. Figures.

```
[ DisplayFigure resource_ID one_time;
    if ((one_time) && (ResourceUsageFlags->resource_ID)) return;
    ResourceUsageFlags->resource_ID = true;
    print "^"; VM_Picture(resource_ID); print "^";
];
```

§3. Sound Effects.

```
[ PlaySound resource_ID one_time;
    if (resource_ID == 0) return; ! The "silence" non-sound effect
    if ((one_time) && (ResourceUsageFlags->resource_ID)) return;
    ResourceUsageFlags->resource_ID = true;
    VM_SoundEffect(resource_ID);
];
```

IndexedText Template — B/inxt

Purpose

Code to support the indexed text kind of value.

§**1. Head.** As ever: if there is no heap, there are no indexed texts.

```
#IFDEF MEMORY_HEAP_SIZE; ! Will exist if any use is made of indexed texts
```

§**2. Character Set.** On the Z-machine, we use the 8-bit ZSCII character set, stored in bytes; on Glulx, we use the opening 16-bit subset of Unicode (which though only a subset covers almost all letter forms used on Earth), stored in two-byte half-words.

The Z-machine does have very partial Unicode support, but not in a way that can help us here. It is capable of printing a wide range of Unicode characters, and on a good interpreter with a good font (such as Zoom for Mac OS X, using the Lucida Grande font) can produce many thousands of glyphs. But it is not capable of printing those characters into memory rather than the screen, an essential technique for indexed texts: it can only write each character to a single byte, and it does so in ZSCII. That forces our hand when it comes to choosing the indexed-text character set.

```
#IFDEF TARGET_ZCODE;
Constant IT_Storage_Flags = BLK_FLAG_MULTIPLE;
Constant ZSCII_Tables;
#IFNOT;
Constant IT_Storage_Flags = BLK_FLAG_MULTIPLE + BLK_FLAG_16_BIT;
Constant Large_Unicode_Tables;
#ENDIF;
{-segment:UnicodeData.i6t}
{-segment:Char.i6t}
```

§**3. KOV Support.** See the "BlockValues.i6t" segment for the specification of the following routines.

```
[ INDEXED_TEXT_TY_Support task arg1 arg2 arg3;
    switch(task) {
        CREATE_KOVS:     return INDEXED_TEXT_TY_Create(arg1);
        CAST_KOVS:       return INDEXED_TEXT_TY_Cast(arg1, arg2, arg3);
        DESTROY_KOVS:    rfalse;
        PRECOPY_KOVS:    rfalse;
        COPY_KOVS:       rfalse;
        COMPARE_KOVS:    return INDEXED_TEXT_TY_Compare(arg1, arg2);
        READ_FILE_KOVS:  if (arg3 == -1) rtrue;
                         return INDEXED_TEXT_TY_ReadFile(arg1, arg2, arg3);
        WRITE_FILE_KOVS: return INDEXED_TEXT_TY_WriteFile(arg1);
        HASH_KOVS:       return INDEXED_TEXT_TY_Hash(arg1);
    }
];
```

§**4. Creation.** Indexed texts are are simply "C strings", that is, the array entries in the block are a sequence of character codes terminated by the character code 0, which is free for this purpose in both ZSCII and Unicode. Since none of the data in an indexed-text is a pointer back onto the heap, it can all freely be bitwise copied or forgotten, which is why we need do nothing special to copy or destroy an indexed text.

Note that a freshly allocated block contains 0s in its data section, so its array entries already form a null-terminated empty text.

```
[ INDEXED_TEXT_TY_Create opcast x;
    x =  BlkAllocate(32, INDEXED_TEXT_TY, IT_Storage_Flags);
    if (opcast) INDEXED_TEXT_TY_Cast(opcast, TEXT_TY, x);
    return x;
];
```

§**5. Casting.** In general computing, "casting" is the process of translating data in one type into semantically equivalent data in another: for instance, translating the integer 1 into the floating point number 1.0, which will have an entirely different binary representation but has roughly the same meaning.

Here we are given a snippet – a word-selection of the player's command – or an ordinary text, and must cast it into an indexed text. In each case, what we do is simply to print out the value we have, but with the output stream set to memory rather than the screen. That gives us the character by character version, neatly laid out in an array, and all we have to do is to copy it into the indexed text and add a null termination byte.

What complicates things is that the two virtual machines handle printing to memory quite differently, and that the original text has unpredictable length. We are going to try printing it into the array `IT_MemoryBuffer`, but what if the text is too big? Disastrously, the Z-machine simply writes on in memory, corrupting all subsequent arrays and almost certainly causing the story file to crash soon after. There is nothing we can do to predict or avoid this, or to repair the damage: this is why the Inform documentation warns users to be wary of using indexed text with large strings in the Z-machine, and advises the use of Glulx instead. Glulx does handle overruns safely, and indeed allows us to dynamically allocate memory as necessary so that we can always avoid overruns entirely.

In either case, though, it's useful to have `IT_MemoryBufferSize`, the size of the temporary buffer, large enough that it will never be overrun in ordinary use. This is controllable with the use option "maximum indexed text length".

The following routine is not as messy as it looks: it is complicated by the fact that the Z-machine and Glulx (a) use different formats when printing text to memory, and (b) handle overruns differently, as explained above.

```
#ifndef IT_MemoryBufferSize;
Constant IT_MemoryBufferSize = 512;
#endif;

Constant IT_Memory_NoBuffers = 2;

#ifndef IT_Memory_NoBuffers;
Constant IT_Memory_NoBuffers = 1;
#endif;

#ifdef TARGET_ZCODE;
Array IT_MemoryBuffer -> IT_MemoryBufferSize*IT_Memory_NoBuffers; ! Where characters are bytes
#ifnot;
Array IT_MemoryBuffer --> (IT_MemoryBufferSize+2)*IT_Memory_NoBuffers; ! Where characters are words
#endif;

Global RawBufferAddress = IT_MemoryBuffer;
Global RawBufferSize = IT_MemoryBufferSize;
Global IT_cast_nesting;

[ INDEXED_TEXT_TY_Cast tx fromkov indt
    len i str oldstr offs realloc news buff buffx freebuff results;
    #ifdef TARGET_ZCODE;
    buffx = IT_MemoryBufferSize;
    #ifnot;
    buffx = (IT_MemoryBufferSize + 2)*WORDSIZE;
    #endif;

    buff = RawBufferAddress + IT_cast_nesting*buffx;
    IT_cast_nesting++;
    if (IT_cast_nesting > IT_Memory_NoBuffers) {
        buff = VM_AllocateMemory(buffx); freebuff = buff;
        if (buff == 0) {
            BlkAllocationError("ran out with too many simultaneous indexed text conversions");
            return;
        }
    }

    .RetryWithLargerBuffer;
    if (tx == 0) {
        #ifdef TARGET_ZCODE;
        buff-->0 = 1;
        buff->2 = 0;
        #ifnot;
        buff-->0 = 0;
        #endif;
        len = 1;
    } else {
        #ifdef TARGET_ZCODE;
        @output_stream 3 buff;
        #ifnot;
        if (unicode_gestalt_ok == false) { RunTimeProblem(RTP_NOGLULXUNICODE); jump Failed; }
        oldstr = glk_stream_get_current();
        str = glk_stream_open_memory_uni(buff, RawBufferSize, filemode_Write, 0);
        glk_stream_set_current(str);
        #endif;

        @push say__p; @push say__pc;
        ClearParagraphing();
        if (fromkov == SNIPPET_TY) print (PrintSnippet) tx;
        else {
            if (tx ofclass String) print (string) tx;
            if (tx ofclass Routine) (tx)();
        }
        @pull say__pc; @pull say__p;

        #ifdef TARGET_ZCODE;

        @output_stream -3;
        len = buff-->0;
        if (len > RawBufferSize-1) len = RawBufferSize-1;
        offs = 2;
        buff->(len+2) = 0;

        #ifnot; ! i.e. GLULX

        results = buff + buffx - 2*WORDSIZE;
        glk_stream_close(str, results);
        if (oldstr) glk_stream_set_current(oldstr);
        len = results-->1;
        if (len > RawBufferSize-1) {
            ! Glulx had to truncate text output because the buffer ran out:
            ! len is the number of characters which it tried to print
            news = RawBufferSize;
            while (news < len) news=news*2;
            news = news*4; ! Bytes rather than words
            i = VM_AllocateMemory(news);
            if (i ~= 0) {
                if (freebuff) VM_FreeMemory(freebuff);
                freebuff = i;
                buff = i;
                RawBufferSize = news/4;
                jump RetryWithLargerBuffer;
            }
            ! Memory allocation refused: all we can do is to truncate the text
            len = RawBufferSize-1;
        }
        offs = 0;
        buff-->(len) = 0;

        #endif;

        len++;
    }

    IT_cast_nesting--;

    if (indt == 0) {
        indt = BlkAllocate(len+1, INDEXED_TEXT_TY, IT_Storage_Flags);
        if (indt == 0) jump Failed;
    } else {
        if (BlkValueSetExtent(indt, len+1, 1) == false) { indt = 0; jump Failed; }
    }

    #ifdef TARGET_ZCODE;
    for (i=0:i<=len:i++) BlkValueWrite(indt, i, buff->(i+offs));
    #ifnot;
    for (i=0:i<=len:i++) BlkValueWrite(indt, i, buff-->(i+offs));
    #endif;

    .Failed;
    if (freebuff) VM_FreeMemory(freebuff);

    return indt;
];
```

§6. Comparison. This is more or less `strcmp`, the traditional C library routine for comparing strings.

```
[ INDEXED_TEXT_TY_Compare indtleft indtright pos ch1 ch2 dsizeleft dsizeright;
    dsizeleft = BlkValueExtent(indtleft);
    dsizeright = BlkValueExtent(indtright);
    for (pos=0:(pos<dsizeleft) && (pos<dsizeright):pos++) {
        ch1 = BlkValueRead(indtleft, pos);
        ch2 = BlkValueRead(indtright, pos);
        if (ch1 ~= ch2) return ch1-ch2;
        if (ch1 == 0) return 0;
    }
    if (pos == dsizeleft) return -1;
    return 1;
];
[ INDEXED_TEXT_TY_Distinguish indtleft indtright;
    if (INDEXED_TEXT_TY_Compare(indtleft, indtright) == 0) rfalse;
    rtrue;
];
```

§7. Hashing. This calculates a hash value for the string, using Bernstein's algorithm.

```
[ INDEXED_TEXT_TY_Hash indt  rv len i;
    rv = 0;
    len = BlkValueExtent(indt);
    for (i=0: i<len: i++)
        rv = rv * 33 + BlkValueRead(indt, i);
    return rv;
];
```

§8. Printing. Unicode is not the native character set on Glulx: it came along as a late addition to Glulx's specification. The deal is that we have to explicitly tell the Glk interface layer to perform certain operations in a Unicode way; if we simply perform `print (char) ch;` then the character `ch` will be printed in ZSCII rather than Unicode.

```
[ INDEXED_TEXT_TY_Say indt  ch i dsize;
    if ((indt==0) || (BlkType(indt) ~= INDEXED_TEXT_TY)) return;
    dsize = BlkValueExtent(indt);
    for (i=0:i<dsize:i++) {
        ch = BlkValueRead(indt, i);
        if (ch == 0) break;
        #ifdef TARGET_ZCODE;
        print (char) ch;
        #ifnot; ! TARGET_ZCODE
        glk_put_char_uni(ch);
        #endif;
    }
];
```

§**9. Serialisation.** Here we print a serialised form of an indexed text which can later be used to reconstruct the original text. The printing is apparently to the screen, but in fact always takes place when the output stream is a file.

The format chosen is a letter "S" for string, then a comma-separated list of decimal character codes, ending with the null terminator, and followed by a semicolon: thus `S65,66,67,0;` is the serialised form of the text "ABC".

```
[ INDEXED_TEXT_TY_WriteFile txb len pos ch;
    len = BlkValueExtent(txb);
    print "S";
    for (pos=0: pos<=len: pos++) {
        if (pos == len) ch = 0; else ch = BlkValueRead(txb, pos);
        if (ch == 0) {
            print "0;"; break;
        } else {
            print ch, ",";
        }
    }
];
```

§**10. Unserialisation.** If that's the word: the reverse process, in which we read a stream of characters from a file and reconstruct the indexed text which gave rise to them.

```
[ INDEXED_TEXT_TY_ReadFile indt auxf ch i v dg pos tsize;
    tsize = BlkValueExtent(indt);
    while (ch ~= 32 or 9 or 10 or 13 or 0 or -1) {
        ch = FileIO_GetC(auxf);
        if (ch == ',' or ';') {
            if (pos+1 >= tsize) {
                if (BlkValueSetExtent(indt, 2*pos, 20) == false) break;
                tsize = BlkValueExtent(indt);
            }
            BlkValueWrite(indt, pos++, v);
            v = 0;
            if (ch == ';') break;
        } else {
            dg = ch - '0';
            v = v*10 + dg;
        }
    }
    BlkValueWrite(indt, pos, 0);
    return indt;
];
```

§11. Recognition-only-GPR. An I6 general parsing routine to look at words from the position marker `wn` in the player's command to see if they match the contents of the indexed text `indt`, returning either `GPR_PREPOSITION` or `GPR_FAIL` according to whether a match could be made. This is used when the an object's name is set to include one of its properties, and the property in question is an indexed text: "A flowerpot is a kind of thing. A flowerpot has an indexed text called pattern. Understand the pattern property as describing a flowerpot." When the player types EXAMINE STRIPED FLOWERPOT, and there is a flowerpot in scope, the following routine is called to test whether its pattern property – an indexed text – matches any words at the position STRIPED FLOWERPOT. Assuming a pot does indeed have the pattern "striped", the routine advances `wn` by 1 and returns `GPR_PREPOSITION` to indicate a match.

This kind of GPR is called a "recognition-only-GPR", because it only recognises an existing value: it doesn't parse a new one.

```
[ INDEXED_TEXT_TY_ROGPR indt
    pos len wa wl wpos bdm ch own;
    if (indt == 0) return GPR_FAIL;
    bdm = true; own = wn;
    len = BlkValueExtent(indt);
    for (pos=0: pos<=len: pos++) {
        if (pos == len) ch = 0; else ch = BlkValueRead(indt, pos);
        if (ch == 32 or 9 or 10 or 0) {
            if (bdm) continue;
            bdm = true;
            if (wpos ~= wl) return GPR_FAIL;
            if (ch == 0) break;
        } else {
            if (bdm) {
                bdm = false;
                if (NextWordStopped() == -1) return GPR_FAIL;
                wa = WordAddress(wn-1);
                wl = WordLength(wn-1);
                wpos = 0;
            }
            if (wa->wpos ~= ch or IT_RevCase(ch)) return GPR_FAIL;
            wpos++;
        }
    }
    if (wn == own) return GPR_FAIL; ! Progress must be made to avoid looping
    return GPR_PREPOSITION;
];
```

§12. Blobs. That completes the compulsory services required for this KOV to function: from here on, the remaining routines provide definitions of text-related phrases in the Standard Rules.

What are the basic operations of text-handling? Clearly we want to be able to search, and replace, but that is left for the segment "RegExp.i6t" to handle. More basically we would like to be able to read and write characters from the text. But texts in I7 tend to be of natural language, rather than containing arbitrary material – that's indeed why we call them texts rather than strings. This means they are likely to be punctuated sequences of words, divided up perhaps into sentences and even paragraphs.

So we provide facilities which regard a text as being an array of "blobs", where a "blob" is a unit of text. The user can choose whether to see it as an array of characters, or words (of three different sorts: see the Inform documentation for details), or paragraphs, or lines.

```
Constant CHR_BLOB = 1; ! Construe as an array of characters
Constant WORD_BLOB = 2; ! Of words
Constant PWORD_BLOB = 3; ! Of punctuated words
Constant UWORD_BLOB = 4; ! Of unpunctuated words
Constant PARA_BLOB = 5; ! Of paragraphs
Constant LINE_BLOB = 6; ! Of lines
Constant REGEXP_BLOB = 7; ! Not a blob type as such, but needed as a distinct value
```

§13. Blob Access. The following routine runs a small finite-state-machine to count the number of blobs in an indexed text, using any of the above blob types (except `REGEXP_BLOB`, which is used for other purposes). If the optional arguments `cindt` and `wanted` are supplied, it also copies the text of blob number `wanted` (counting upwards from 1 at the start of the text) into the indexed text `cindt`. If the further optional argument `rindt` is supplied, then `cindt` is instead written with the original text `indt` as it would read if the blob in question were replaced with the indexed text in `rindt`.

```
Constant WS_BRM = 1;
Constant SKIPPED_BRM = 2;
Constant ACCEPTED_BRM = 3;
Constant ACCEPTEDP_BRM = 4;
Constant ACCEPTEDN_BRM = 5;
Constant ACCEPTEDPN_BRM = 6;
[ IT_BlobAccess indt blobtype cindt wanted rindt
    brm oldbrm ch i dsize csize blobcount gp cl j;
    if ((indt==0) || (BlkType(indt) ~= INDEXED_TEXT_TY)) return 0;
    if (blobtype == CHR_BLOB) return IT_CharacterLength(indt);
    dsize = BlkValueExtent(indt);
    if (cindt) csize = BlkValueExtent(cindt);
    else if (rindt) "*** rindt without cindt ***";
    brm = WS_BRM;
    for (i=0:i<dsize:i++) {
        ch = BlkValueRead(indt, i);
        if (ch == 0) break;
        oldbrm = brm;
        if (ch == 10 or 13 or 32 or 9) {
            if (oldbrm ~= WS_BRM) {
                gp = 0;
                for (j=i:j<dsize:j++) {
                    ch = BlkValueRead(indt, j);
                    if (ch == 0) { brm = WS_BRM; break; }
                    if (ch == 10 or 13) { gp++; continue; }
                    if (ch ~= 32 or 9) break;
                }
                ch = BlkValueRead(indt, i);
                if (j == dsize) brm = WS_BRM;
                switch (blobtype) {
                    PARA_BLOB: if (gp >= 2) brm = WS_BRM;
                    LINE_BLOB: if (gp >= 1) brm = WS_BRM;
                    default: brm = WS_BRM;
                }
            }
        } else {
            gp = false;
            if ((blobtype == WORD_BLOB or PWORD_BLOB or UWORD_BLOB) &&
                (ch == '.' or ',' or '!' or '?'
                        or '-' or '/' or '"' or ':' or ';'
                        or '(' or ')' or '[' or ']' or '{' or '}'))
                gp = true;
            switch (oldbrm) {
                WS_BRM:
                    brm = ACCEPTED_BRM;
                    if (blobtype == WORD_BLOB) {
                        if (gp) brm = SKIPPED_BRM;
                    }
                    if (blobtype == PWORD_BLOB) {
                        if (gp) brm = ACCEPTEDP_BRM;
                    }
                SKIPPED_BRM:
                    if (blobtype == WORD_BLOB) {
                        if (gp == false) brm = ACCEPTED_BRM;
                    }
                ACCEPTED_BRM:
                    if (blobtype == WORD_BLOB) {
                        if (gp) brm = SKIPPED_BRM;
                    }
                    if (blobtype == PWORD_BLOB) {
                        if (gp) brm = ACCEPTEDP_BRM;
                    }
                ACCEPTEDP_BRM:
                    if (blobtype == PWORD_BLOB) {
                        if (gp == false) brm = ACCEPTED_BRM;
                        else {
                            if ((ch == BlkValueRead(indt, i-1)) &&
                                (ch == '-' or '.')) blobcount--;
                            blobcount++;
                        }
                    }
                ACCEPTEDN_BRM:
                    if (blobtype == WORD_BLOB) {
                        if (gp) brm = SKIPPED_BRM;
                    }
                    if (blobtype == PWORD_BLOB) {
                        if (gp) brm = ACCEPTEDP_BRM;
                    }
                ACCEPTEDPN_BRM:
                    if (blobtype == PWORD_BLOB) {
                        if (gp == false) brm = ACCEPTED_BRM;
                        else {
                            if ((ch == BlkValueRead(indt, i-1)) &&
                                (ch == '-' or '.')) blobcount--;
                            blobcount++;
                        }
                    }
                }
            }
            if (brm == ACCEPTED_BRM or ACCEPTEDP_BRM) {
                if (oldbrm ~= brm) blobcount++;
                if ((cindt) && (blobcount == wanted)) {
                    if (rindt) {
                        BlkValueWrite(cindt, cl, 0);
                        IT_Concatenate(cindt, rindt, CHR_BLOB);
                        csize = BlkValueExtent(cindt);
                        cl = IT_CharacterLength(cindt);
                        if (brm == ACCEPTED_BRM) brm = ACCEPTEDN_BRM;
                        if (brm == ACCEPTEDP_BRM) brm = ACCEPTEDPN_BRM;
                    } else {
                        if (cl+1 >= csize) {
                            if (BlkValueSetExtent(cindt, 2*cl, 2) == false) break;
                            csize = BlkValueExtent(cindt);
                        }
                        BlkValueWrite(cindt, cl++, ch);
                    }
                } else {
                    if (rindt) {
                        if (cl+1 >= csize) {
                            if (BlkValueSetExtent(cindt, 2*cl, 3) == false) break;
                            csize = BlkValueExtent(cindt);
                        }
                        BlkValueWrite(cindt, cl++, ch);
                    }
                }
            } else {
                if ((rindt) && (brm ~= ACCEPTEDN_BRM or ACCEPTEDPN_BRM)) {
                    if (cl+1 >= csize) {
                        if (BlkValueSetExtent(cindt, 2*cl, 4) == false) break;
                        csize = BlkValueExtent(cindt);
                    }
                    BlkValueWrite(cindt, cl++, ch);
                }
            }
        }
        if (cindt) BlkValueWrite(cindt, cl++, 0);
        return blobcount;
    ];
```

§14. Get Blob. The front end which uses the above routine to read a blob. (Note that, for efficiency's sake, we read characters more directly.)

```
[ IT_GetBlob cindt indt wanted blobtype;
    if ((indt==0) || (BlkType(indt) ~= INDEXED_TEXT_TY)) return;
    if (blobtype == CHR_BLOB) return IT_GetCharacter(cindt, indt, wanted);
    IT_BlobAccess(indt, blobtype, cindt, wanted);
    return cindt;
];
```

§15. Replace Blob. The front end which uses the above routine to replace a blob. (Once again, characters are handled directly to avoid incurring all that overhead.)

```
[ IT_ReplaceBlob blobtype indt wanted rindt cindt ilen rlen i;
    if (blobtype == CHR_BLOB) {
        ilen = IT_CharacterLength(indt);
        rlen = IT_CharacterLength(rindt);
        wanted--;
        if ((wanted >= 0) && (wanted<ilen)) {
            if (rlen == 1) {
                BlkValueWrite(indt, wanted, BlkValueRead(rindt, 0));
            } else {
                cindt = BlkValueCreate(INDEXED_TEXT_TY);
                if (BlkValueSetExtent(cindt, ilen+rlen+1, 5)) {
                    for (i=0:i<wanted:i++)
                        BlkValueWrite(cindt, i, BlkValueRead(indt, i));
                    for (i=0:i<rlen:i++)
                        BlkValueWrite(cindt, wanted+i, BlkValueRead(rindt, i));
                    for (i=wanted+1:i<ilen:i++)
                        BlkValueWrite(cindt, rlen+i-1, BlkValueRead(indt, i));
                    BlkValueWrite(cindt, rlen+ilen, 0);
                    BlkValueCopy(indt, cindt);
                }
                BlkFree(cindt);
            }
        }
    } else {
        cindt = BlkValueCreate(INDEXED_TEXT_TY);
        IT_BlobAccess(indt, blobtype, cindt, wanted, rindt);
        BlkValueCopy(indt, cindt);
        BlkFree(cindt);
    }
];
```

§**16. Replace Text.** This is the general routine which searches for any instance of `findt`, as a blob, in `indt`, and replaces it with the text `rindt`. It works on any of the above blob-types, but two cases are special: first, if the blob-type is `CHR_BLOB`, then it can do more than search and replace for any instance of a single character: it can search and replace any instance of a substring, so that `findt` is not required to be only a single character. Second, if the blob-type is the special value `REGEXP_BLOB` then `findt` is interpreted as a regular expression rather than something literal to find: see "RegExp.i6t" for what happens next.

```
[ IT_ReplaceText blobtype indt findt rindt
    cindt csize ilen flen i cl mpos ch chm whitespace punctuation;
    if (blobtype == REGEXP_BLOB or CHR_BLOB)
        return IT_Replace_RE(blobtype, indt, findt, rindt);
    ilen = IT_CharacterLength(indt);
    flen = IT_CharacterLength(findt);
    cindt = BlkValueCreate(INDEXED_TEXT_TY);
    csize = BlkValueExtent(cindt);
    mpos = 0;
    whitespace = true; punctuation = false;
    for (i=0:i<=ilen:i++) {
        ch = BlkValueRead(indt, i);
        .MoreMatching;
        chm = BlkValueRead(findt, mpos++);
        if (mpos == 1) {
            switch (blobtype) {
                WORD_BLOB:
                    if ((whitespace == false) && (punctuation == false)) chm = -1;
            }
        }
        whitespace = false;
        if (ch == 10 or 13 or 32 or 9) whitespace = true;
        punctuation = false;
        if (ch == '.' or ',' or '!' or '?'
            or '-' or '/' or '"' or ':' or ';'
            or '(' or ')' or '[' or ']' or '{' or '}') {
            if (blobtype == WORD_BLOB) chm = -1;
            punctuation = true;
        }
        if (ch == chm) {
            if (mpos == flen) {
                if (i == ilen) chm = 0;
                else chm = BlkValueRead(indt, i+1);
                if ((blobtype == CHR_BLOB) ||
                    (chm == 0 or 10 or 13 or 32 or 9) ||
                    (chm == '.' or ',' or '!' or '?'
                        or '-' or '/' or '"' or ':' or ';'
                        or '(' or ')' or '[' or ']' or '{' or '}')) {
                    mpos = 0;
                    cl = cl - (flen-1);
                    BlkValueWrite(cindt, cl, 0);
                    IT_Concatenate(cindt, rindt, CHR_BLOB);
                    csize = BlkValueExtent(cindt);
                    cl = IT_CharacterLength(cindt);
                    continue;
                }
            }
        } else {
            mpos = 0;
        }
        if (cl+1 >= csize) {
            if (BlkValueSetExtent(cindt, 2*cl, 9) == false) break;
            csize = BlkValueExtent(cindt);
        }
        BlkValueWrite(cindt, cl++, ch);
    }
    BlkValueCopy(indt, cindt);
    BlkFree(cindt);
];
```

§17. Character Length. When accessing at the character-by-character level, things are much easier and we needn't go through any finite state machine palaver.

```
[ IT_CharacterLength indt ch i dsize;
    if ((indt==0) || (BlkType(indt) ~= INDEXED_TEXT_TY)) return 0;
    dsize = BlkValueExtent(indt);
    for (i=0:i<dsize:i++) {
        ch = BlkValueRead(indt, i);
        if (ch == 0) return i;
    }
    return dsize;
];
```

```
[ INDEXED_TEXT_TY_Empty indt;
    if ((indt==0) || (BlkType(indt) ~= INDEXED_TEXT_TY)) rfalse;
    if (IT_CharacterLength(indt) == 0) rtrue;
    rfalse;
];
```

§18. Get Character. Characters in a text are numbered upwards from 1 by the users of this routine: which is why we subtract 1 when reading the array in the block-value, which counts from 0.

```
[ IT_GetCharacter cindt indt i ch;
    if ((indt==0) || (BlkType(indt) ~= INDEXED_TEXT_TY)) return;
    if ((i<=0) || (i>IT_CharacterLength(indt))) ch = 0;
    else ch = BlkValueRead(indt, i-1);
    BlkValueWrite(cindt, 0, ch);
    BlkValueWrite(cindt, 1, 0);
    return cindt;
];
```

§19. Casing. In many programming languages, characters are a distinct data type from strings, but not in I7. To I7, a character is simply an indexed text which happens to have length 1 – this has its inefficiencies, but is conceptually easy for the user.

`IT_CharactersOfCase(indt, case)` determines whether all the characters in `indt` are letters of the given casing: 0 for lower case, 1 for upper case. In the case of ZSCII, this is done correctly handling all of the European accented letters; in the case of Unicode, it follows the Unicode standard.

Note that there is no requirement for `indt` to be only a single character long.

```
[ IT_CharactersOfCase indt case i ch len;
    if ((indt==0) || (BlkType(indt) ~= INDEXED_TEXT_TY)) rfalse;
    len = IT_CharacterLength(indt);
    for (i=0:i<len:i++) {
        ch = BlkValueRead(indt, i);
        if ((ch) && (CharIsOfCase(ch, case) == false)) rfalse;
    }
    rtrue;
];
```

§20. Change Case. We set `cindt` to the text in `indt`, except that all the letters are converted to the `case` given (0 for lower, 1 for upper). The definition of what is a "letter", what case it has and what the other-case form is are as specified in the ZSCII and Unicode standards.

```
[ IT_CharactersToCase cindt indt case i ch len bnd;
    if ((indt==0) || (BlkType(indt) ~= INDEXED_TEXT_TY)) return;
    len = IT_CharacterLength(indt);
    if (BlkValueSetExtent(cindt, len+1, 11) == false) return cindt;
    bnd = 1;
    for (i=0:i<len:i++) {
        ch = BlkValueRead(indt, i);
        if (case < 2) {
            BlkValueWrite(cindt, i, CharToCase(ch, case));
        } else {
            BlkValueWrite(cindt, i, CharToCase(ch, bnd));
            if (case == 2) {
                bnd = 0;
                if (ch == 0 or 10 or 13 or 32 or 9
                    or '.' or ',' or '!' or '?'
                    or '-' or '/' or '"' or ':' or ';'
                    or '(' or ')' or '[' or ']' or '{' or '}') bnd = 1;
            }
            if (case == 3) {
                if (ch ~= 0 or 10 or 13 or 32 or 9) {
                    if (bnd == 1) bnd = 0;
                    else {
                        if (ch == '.' or '!' or '?') bnd = 1;
                    }
                }
            }
        }
    }
    BlkValueWrite(cindt, len, 0);
    return cindt;
];
```

§**21. Concatenation.** To concatenate two indexed texts is to place one after the other: thus “green” concatenated with “horn” makes “greenhorn”. In this routine, `indt_from` would be “horn”, and is added at the end of `indt_to`, which is returned in its expanded state.

When the blob type is `REGEXP_BLOB`, the routine is used not for simple concatenation but to handle the concatenations occurring when a regular expression search-and-replace is going on: see “RegExp.i6t”.

```
[ IT_Concatenate indt_to indt_from blobtype indt_ref
    pos len ch i tosize x y case;
    if ((indt_to==0) || (BlkType(indt_to) ~= INDEXED_TEXT_TY)) rfalse;
    if ((indt_from==0) || (BlkType(indt_from) ~= INDEXED_TEXT_TY)) return indt_to;
    switch(blobtype) {
        CHR_BLOB, 0:
            pos = IT_CharacterLength(indt_to);
            len = IT_CharacterLength(indt_from);
            if (BlkValueSetExtent(indt_to, pos+len+1, 10) == false) return indt_to;
            for (i=0:i<len:i++) {
                ch = BlkValueRead(indt_from, i);
                BlkValueWrite(indt_to, i+pos, ch);
            }
            BlkValueWrite(indt_to, len+pos, 0);
            return indt_to;
        REGEXP_BLOB:
            return IT_RE_Concatenate(indt_to, indt_from, blobtype, indt_ref);
        default:
            print "*** IT_Concatenate used on impossible blob type ***^";
            rfalse;
    }
];
```

§**22. Setting the Player’s Command.** In effect, the text typed most recently by the player is a sort of indexed text already, though it isn’t in indexed text format, and doesn’t live on the heap. (We can’t simply make it an indexed text, as tidy as that would seem, because then no I7 work could ever compile without a heap to use, and that would severely affect many works which have to fit in the Z-machine and can’t afford the storage for a heap.)

```
[ SetPlayersCommand indt_from i len at;
    len = IT_CharacterLength(indt_from);
    if (len > 118) len = 118;
    #ifdef TARGET_ZCODE;
    buffer->1 = len; at = 2;
    #ifnot;
    buffer-->0 = len; at = 4;
    #endif;
    for (i=0:i<len:i++) buffer->(i+at) = CharToCase(BlkValueRead(indt_from, i), 0);
    for (:at+i<120:i++) buffer->(at+i) = ' ';
    VM_Tokenise(buffer, parse);
    players_command = 100 + WordCount(); ! The snippet variable ‘‘player’s command’’
];
```

§**23. Stubs.** And the usual meaningless versions to ensure that function-names exist if there is no heap, and there are no indexed texts anyway.

```
#IFNOT; ! IFDEF MEMORY_HEAP_SIZE
[ INDEXED_TEXT_TY_Support t a b c; rfalse; ];
[ INDEXED_TEXT_TY_Say indt; ];
[ SetPlayersCommand indt_from; ];
[ INDEXED_TEXT_TY_Create; ];
[ INDEXED_TEXT_TY_Cast a b c; ];
[ INDEXED_TEXT_TY_Empty t; rfalse; ];
#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```

RegExp Template B/regxt

Purpose

Code to match and replace on regular expressions against indexed text strings.

§**1. Head.** As ever: if there is no heap, there are no indexed texts, and if there are no indexed texts then there is no point in compiling any of this code.

```
#IFDEF MEMORY_HEAP_SIZE; ! Will exist if any use is made of indexed texts
Global IT_RE_Trace = false; ! Change to true for (a lot of) debugging data in use
[ IT_RE_SetTrace F; IT_RE_Trace = F; ];
```

§**2. Algorithm.** Once Inform 7 supported indexed text, regular-expression matching became an obvious goal: regexp-based features offer more or less the gold standard in text search and replace facilities, and I7 is so concerned with text that we shouldn't make do with less. But the best and most portable implementation of regular expression matching, PCRE by Philip Hazel, is about a hundred times larger than the code in this section, and also had unacceptable memory needs: there was no practicable way to make it small enough to do useful work on the Z-machine. Nor could an I6 regexp-matcher compile just-in-time code, or translate the expression into a suitable deterministic finite state machine. One day, though, I read one of the papers which Brian Kernighan writes every few years to the effect that regular-expression matching is much easier than you think. Kernighan is right: writing a regexp matcher is indeed easier than you think (one day's worth of cheerful hacking), but debugging one until it passes the trickiest hundred of Perl's 645 test cases is another matter (and it took a whole week more). Still, the result seems to be robust. The main compromise made is that backtracking is not always comprehensive with regexps like `^(a\1?){4}$`, because we do not allocate individual storage to backtrack individually through all possibilities of each of the four uses of the bracketed subexpression – which means we miss some cases, since the subexpression contains a reference to itself, so that its content can vary in the four uses. PCRE's approach here is to expand the expression as if it were a sequence of four bracketed expressions, thus removing the awkward quantifier `{4}`, but that costs memory: indeed this is why PCRE cannot solve all of Perl's test cases without its default memory allocation being raised. In other respects, the algorithm below appears to be accurate if not very fast.

§**3. Class Codes.** While in principle we could keep the match expression in textual form, in practice the syntax of regular expressions is complex enough that this would be tricky and rather slow: we would be parsing the same notations over and over again. So we begin by compiling it to a simple tree structure. The tree is made up of nodes, and each node has a "class code": these are identified by the `*_RE_CC` constants below. Note that the class codes below are all negative: this is so that they are distinct from all valid ZSCII or Unicode characters. (ZSCII is used only on the Z-machine, which has a 16-bit word but an 8-bit character set, so that all character values are positive; similarly, Unicode is (for our purposes) a 16-bit character set on a 32-bit virtual machine.)

```
! Character classes
Constant NEWLINE_RE_CC = -1;
Constant TAB_RE_CC = -2;
Constant DIGIT_RE_CC = -3;
Constant NONDIGIT_RE_CC = -4;
Constant WHITESPACE_RE_CC = -5;
Constant NONWHITESPACE_RE_CC = -6;
Constant PUNCTUATION_RE_CC = -7;
Constant NONPUNCTUATION_RE_CC = -8;
Constant WORD_RE_CC = -9;
Constant NONWORD_RE_CC = -10;
Constant ANYTHING_RE_CC = -11;
Constant NOTHING_RE_CC = -12;
Constant RANGE_RE_CC = -13;
Constant LCASE_RE_CC = -14;
Constant NONLCASE_RE_CC = -15;
Constant UCASE_RE_CC = -16;
Constant NONUCASE_RE_CC = -17;
! Control structures
Constant SUBEXP_RE_CC = -20;
Constant DISJUNCTION_RE_CC = -21;
Constant CHOICE_RE_CC = -22;
Constant QUANTIFIER_RE_CC = -23;
Constant IF_RE_CC = -24;
Constant CONDITION_RE_CC = -25;
Constant THEN_RE_CC = -26;
Constant ELSE_RE_CC = -27;
! Substring matchers
Constant VARIABLE_RE_CC = -30;
Constant LITERAL_RE_CC = -31;
! Positional matchers
Constant START_RE_CC = -40;
Constant END_RE_CC = -41;
Constant BOUNDARY_RE_CC = -42;
Constant NONBOUNDARY_RE_CC = -43;
Constant ALWAYS_RE_CC = -44;
Constant NEVER_RE_CC = -45;
! Mode switches
Constant SENSITIVITY_RE_CC = -50;
```

§4. Packets. The nodes of the compiled expression tree are stored in "packets", which are segments of a fixed array. A regexp complicated enough that it cannot be stored in `RE_MAX_PACKETS` packets will be rejected with an error: it looks like a rather low limit, but in fact suffices to handle all of Perl's test cases, some of which are works of diabolism.

A packet is then a record containing 14 fields, with offsets defined by the constants defined below. These fields combine the compilation of the corresponding fragment of the regexp with both the tree structure holding these packets together and also the current state of the temporary variables recording how far we have progressed in trying all of the possible ways to match the packet.

```
Constant RE_MAX_PACKETS = 32;

Constant RE_PACKET_SIZE = 14; ! Words of memory used per packet
Constant RE_PACKET_SIZE_IN_BYTES = WORDSIZE*RE_PACKET_SIZE; ! Bytes used per packet

Array RE_PACKET_space --> RE_MAX_PACKETS*RE_PACKET_SIZE;

Constant RE_CCLASS = 0;       ! One of the class codes defined above
Constant RE_PAR1 = 1;         ! Three parameters whose meaning depends on class code
Constant RE_PAR2 = 2;
Constant RE_PAR3 = 3;
Constant RE_NEXT = 4;         ! Younger sibling in the compiled tree
Constant RE_PREVIOUS = 5;     ! Elder sibling
Constant RE_DOWN = 6;         ! Child
Constant RE_UP = 7;           ! Parent
Constant RE_DATA1 = 8;        ! Backtracking data
Constant RE_DATA2 = 9;
Constant RE_CONSTRAINT = 10;
Constant RE_CACHE1 = 11;
Constant RE_CACHE2 = 12;
Constant RE_MODES = 13;
```

§5. Nodes. The routine to create a node, something which happens only during the compilation phase, and also the routine which returns the address of a given node. Nodes are numbered from 0 up to $M - 1$, where M is the constant `RE_MAX_PACKETS`.

```
[ IT_RE_Node n cc par1 par2 par3  offset;
    if ((n<0) || (n >= RE_MAX_PACKETS)) rfalse;
    offset = RE_PACKET_space + n*RE_PACKET_SIZE_IN_BYTES;
    offset-->RE_CCLASS = cc;
    offset-->RE_PAR1 = par1;
    offset-->RE_PAR2 = par2;
    offset-->RE_PAR3 = par3;
    offset-->RE_NEXT = NULL;
    offset-->RE_PREVIOUS = NULL;
    offset-->RE_DOWN = NULL;
    offset-->RE_UP = NULL;
    offset-->RE_DATA1 = -1; ! Match start
    offset-->RE_DATA2 = -1; ! Match end
    offset-->RE_CONSTRAINT = -1; ! Rewind edge
    return offset;
];

[ IT_RE_NodeAddress n;
    if ((n<0) || (n >= RE_MAX_PACKETS)) return -1;
    return RE_PACKET_space + n*RE_PACKET_SIZE_IN_BYTES;
];
```

§6. Match Variables. A bracketed subexpression can be used as a variable: we support `\1`, ..., `\9` to mean "the value of subexpression 1 to 9", and `\0` to mean "the whole text matched", as if the entire regexp were bracketed. (PCRE and Perl also allow `\10`, `\11`, ..., but we don't, because it complicates parsing and memory is too short.)

`RE_Subexpressions-->10` stores the number of subexpressions in use, not counting `\0`. During the compiling stage, `RE_Subexpressions-->N` is set to point to the node representating `\N`, where `N` varies from 1 to 9. When matching is complete, and assuming we care about the contents of these variables – which we might not, and if not we certainly don't want to waste time and memory – we call `IT_RE_CreateMatchVars` to allocate indexed text variables and fill them in as appropriate, memory permitting.

`IT_RE_EmptyMatchVars` empties any such variables which may survive from a previous match, setting them to the empty text.

```
Array RE_Subexpressions --> 11; ! Address of node for this subexpression
Array Allocated_Match_Vars --> 10; ! Indexed text to hold values of the variables
[ IT_RE_DebugMatchVars indt
    offset n i;
    print RE_Subexpressions-->10, " collecting subexps^";
    for (n=0:(n<RE_Subexpressions-->10) && (n<10): n++) {
        offset = RE_Subexpressions-->n;
        print "Subexp ", offset-->RE_PAR1,
            " = [", offset-->RE_DATA1, ",", offset-->RE_DATA2, "] = ";
        for (i=offset-->RE_DATA1:i<offset-->RE_DATA2:i++)
            print (char) BlkValueRead(indt, i);
        print "^";
    }
];
[ IT_RE_CreateMatchVars indt
    offset n i ch cindt cl csize;
    for (n=0:(n<RE_Subexpressions-->10) && (n<10): n++) {
        offset = RE_Subexpressions-->n;
        if (Allocated_Match_Vars-->n == 0)
            Allocated_Match_Vars-->n = INDEXED_TEXT_TY_Create(); ! Permanently
        cindt = Allocated_Match_Vars-->n;
        csize = BlkValueExtent(cindt);
        cl = 0;
        for (i=offset-->RE_DATA1:i<offset-->RE_DATA2:i++) {
            ch = BlkValueRead(indt, i);
            if (cl+1 >= csize) {
                if (BlkValueSetExtent(cindt, 2*cl, 6) == false) break;
                csize = BlkValueExtent(cindt);
            }
            BlkValueWrite(cindt, cl++, ch);
        }
        BlkValueWrite(cindt, cl, 0);
    }
];
[ IT_RE_EmptyMatchVars indt
    n;
    for (n=0:(n<RE_Subexpressions-->10) && (n<10): n++)
        if (Allocated_Match_Vars-->n ~= 0)
            BlkValueWrite(Allocated_Match_Vars-->n, 0, 0);
];
```

```
[ IT_RE_GetMatchVar indt vn
    offset;
    if ((vn<0) || (vn>=10) || (vn >= RE_Subexpressions-->10)) jump Nope;
    offset = RE_Subexpressions-->vn;
    if (offset == 0) jump Nope;
    if (offset-->RE_DATA1 < 0) jump Nope;
    if (Allocated_Match_Vars-->vn == 0) {
        print "*** ", vn, " unallocated! ***^";
        jump Nope;
    }
    BlkValueCopy(indt, Allocated_Match_Vars-->vn);
    return indt;
    .Nope;
    BlkValueWrite(indt, 0, 0);
    return indt;
];
```

§**7. Markers.** At each node, the `-->RE_DATA1` and `-->RE_DATA2` fields represent the character positions of the start and end of the text matched by the node and its subtree (if any). These are called markers.

Thus `IT_MV_End(N, 0)` returns the start of `\N` and `IT_MV_End(N, 1)` the end of `\N`, according to the current match of subexpression `N`.

```
[ IT_MV_End n end
    offset;
    offset = RE_Subexpressions-->n;
    if (end==0) return offset-->RE_DATA1;
    return offset-->RE_DATA2;
];

[ IT_RE_Clear_Markers token;
    for (: token ~= NULL: token = token-->RE_NEXT) {
        if (token-->RE_DOWN ~= NULL) IT_RE_Clear_Markers(token-->RE_DOWN);
        token-->RE_DATA1 = -1;
        token-->RE_DATA2 = -1;
        token-->RE_CONSTRAINT = -1;
    }
];
```

§**8. Debugging.** Code in this paragraph simply prints a convenient screen representation of the compiled regexp, together with the current values of its markers. It is invaluable for debugging purposes and, touch wood, may not be needed again, but it is relatively compact and we keep it just in case.

```
[ IT_RE_DebugTree findt detail;
    print "Pattern: ", (INDEXED_TEXT_TY_Say) findt, "^";
    IT_RE_DebugSubtree(findt, 1, RE_PACKET_space, detail);
];
[ IT_RE_DebugSubtree findt depth offset detail
    cup;
    if (offset ~= NULL) {
        cup = offset-->RE_UP;
        if (offset-->RE_PREVIOUS ~= NULL) print "*** broken initial previous ***^";
    }
    while (offset ~= NULL) {
        if (offset-->RE_UP ~= cup) print "*** broken up matching ***^";
        spaces(depth*2);
        IT_RE_DebugNode(offset, findt, detail);
        if (offset-->RE_DOWN ~= NULL) {
            if ((offset-->RE_DOWN)-->RE_UP ~= offset)
                print "*** broken down/up ***^";
            IT_RE_DebugSubtree(findt, depth+1, offset-->RE_DOWN, detail);
        }
        if (offset-->RE_NEXT ~= NULL) {
            if ((offset-->RE_NEXT)-->RE_PREVIOUS ~= offset)
                print "*** broken next/previous ***^";
        }
        offset = offset-->RE_NEXT;
    }
];
[ IT_RE_DebugNode offset findt detail
    i par1 par2 par3;
    if (offset == NULL) "[NULL]";
    print "[", (offset-RE_PACKET_space)/(RE_PACKET_SIZE_IN_BYTES), "] ";
    ! for (i=0:i<RE_PACKET_SIZE:i++) print offset-->i, " ";
    par1 = offset-->RE_PAR1;
    par2 = offset-->RE_PAR2;
    par3 = offset-->RE_PAR3;
    switch (offset-->RE_CCLASS) {
        DIGIT_RE_CC: print "DIGIT";
        NONDIGIT_RE_CC: print "NONDIGIT";
        UCASE_RE_CC: print "UCASE";
        NONUCASE_RE_CC: print "NONUCASE";
        LCASE_RE_CC: print "LCASE";
        NONLCASE_RE_CC: print "NONLCASE";
        WHITESPACE_RE_CC: print "WHITESPACE";
        NONWHITESPACE_RE_CC: print "NONWHITESPACE";
        PUNCTUATION_RE_CC: print "PUNCTUATION";
        NONPUNCTUATION_RE_CC: print "NONPUNCTUATION";
        WORD_RE_CC: print "WORD";
        NONWORD_RE_CC: print "NONWORD";
        ALWAYS_RE_CC: print "ALWAYS";
        NEVER_RE_CC: print "NEVER";
        START_RE_CC: print "START";
        END_RE_CC: print "END";
        BOUNDARY_RE_CC: print "BOUNDARY";
        NONBOUNDARY_RE_CC: print "NONBOUNDARY";
        ANYTHING_RE_CC: print "ANYTHING";
        NOTHING_RE_CC: print "NOTHING";
        RANGE_RE_CC: print "RANGE"; if (par3 == true) print " (negated)";
            print " ";
            for (i=par1:i<par2:i++) print (char) BlkValueRead(findt, i);
        VARIABLE_RE_CC: print "VARIABLE ", par1;
        SUBEXP_RE_CC:
            if (par1 == 0) print "EXP";
            else print "SUBEXP ";
            if (par1 >= 0) print "= V", par1;
            if (par2 == 1) {
                if (par3 == 0) print " (?=...) lookahead";
                else print " (?<=...) lookbehind of width ", par3;
            }
            if (par2 == 2) {
                if (par3 == 0) print " (?!...) negated lookahead";
                else print " (?<!...) negated lookbehind of width ", par3;
            }
            if (par2 == 3) print " uncollecting";
            if (par2 == 0 or 3) {
                if (par3 == 1) print " forcing case sensitivity";
                if (par3 == 2) print " forcing case insensitivity";
            }
            if (par2 == 4) print " (?>...) possessive";
        NEWLINE_RE_CC: print "NEWLINE";
        TAB_RE_CC: print "TAB";
        QUANTIFIER_RE_CC: print "QUANTIFIER min=", par1, " max=", par2;
            if (par3) print " (lazy)"; else print " (greedy)";
        LITERAL_RE_CC: print "LITERAL";
            print " ";
            for (i=par1:i<par2:i++) print (char) BlkValueRead(findt, i);
        DISJUNCTION_RE_CC: print "DISJUNCTION of ", par1, " choices";
        CHOICE_RE_CC: print "CHOICE no ", par1;
        SENSITIVITY_RE_CC: print "SENSITIVITY";
            if (par1) print " off"; else print " on";
        IF_RE_CC: print "IF"; if (par1 >= 1) print " = V", par1;
        CONDITION_RE_CC: print "CONDITION"; if (par1 >= 1) print " = V", par1;
        THEN_RE_CC: print "THEN";
        ELSE_RE_CC: print "ELSE";
    }
    if (detail)
        print ": ", offset-->RE_DATA1, ", ", offset-->RE_DATA2, ", ", offset-->RE_CONSTRAINT;
    print "^";
];
```

§9. Compiling Tree For Substring Search. When we search for a literal substring, say looking for "per" in "Supernumerary", we will in fact use the same apparatus as when searching for a regular expression: we compile a very simple node tree in which `\0` as the root contains just one child node, a `LITERAL_RE_CC` matching exactly the text "per". We return 2 since that's the number of nodes in the tree.

```
[ IT_CHR_CompileTree findt exactly
    root literal fto no_packets token attach_to;
    fto = IT_CharacterLength(findt);
    root = IT_RE_Node(0, SUBEXP_RE_CC, 0, 0, 0);
    literal = IT_RE_Node(1, LITERAL_RE_CC, 0, fto, 0);
    root-->RE_DOWN = literal;
    literal-->RE_UP = root;
    if (exactly) {
        no_packets = 2;
        if (no_packets+3 > RE_MAX_PACKETS) return "regexp too complex";
        exactly = RE_PACKET_space-->RE_DOWN;
        token = IT_RE_Node(no_packets++, START_RE_CC, 0, 0, 0);
        RE_PACKET_space-->RE_DOWN = token; token-->RE_UP = RE_PACKET_space;
        attach_to = IT_RE_Node(no_packets++, SUBEXP_RE_CC, -1, 3, 0);
        token-->RE_NEXT = attach_to; attach_to-->RE_PREVIOUS = token;
        attach_to-->RE_UP = RE_PACKET_space;
        attach_to-->RE_NEXT = IT_RE_Node(no_packets++, END_RE_CC, 0, 0, 0);
        (attach_to-->RE_NEXT)-->RE_PREVIOUS = attach_to;
        (attach_to-->RE_NEXT)-->RE_UP = RE_PACKET_space;
        attach_to-->RE_DOWN = exactly;
        while (exactly ~= NULL) {
            exactly-->RE_UP = attach_to; exactly = exactly-->RE_NEXT;
        }
    }
    no_packets = IT_RE_ExpandChoices(RE_PACKET_space, no_packets);
];
```

§10. Compiling Tree For Regexp Search. But in general we need to compile a regular expression string into a tree of the kind described above, and here is the routine which does that, returning the number of nodes used to build the tree. The syntax it accepts is very fully documented in *Writing with Inform*, so no details are given here.

```
Array Subexp_Posns --> 20;
[ IT_RE_CompileTree findt exactly
    no_packets ffrom fto cc par1 par2 par3
    quantifiable token attach_to no_subs blevel bits;
    fto = IT_CharacterLength(findt);
    if (fto == 0) {
        IT_RE_Node(no_packets++, NEVER_RE_CC, 0, 0, 0); ! Empty regexp never matches
        return 1;
    }
    attach_to = IT_RE_Node(no_packets++, SUBEXP_RE_CC, 0, 0, 0);
    RE_Subexpressions-->0 = attach_to; RE_Subexpressions-->10 = 1; no_subs = 1;
    quantifiable = false; blevel = 0;
    for (ffrom = 0: ffrom < fto: ) {
        cc = BlkValueRead(findt, ffrom++); par1 = 0; par2 = 0; par3 = 0;
```

```
if (cc == '\') {
    if (ffrom == fto) return "Search pattern not terminated";
    cc = BlkValueRead(findt, ffrom++);
    switch (cc) {
        'b': cc = BOUNDARY_RE_CC;
        'B': cc = NONBOUNDARY_RE_CC;
        'd': cc = DIGIT_RE_CC;
        'D': cc = NONDIGIT_RE_CC;
        'l': cc = LCASE_RE_CC;
        'L': cc = NONLCASE_RE_CC;
        'n': cc = NEWLINE_RE_CC;
        'p': cc = PUNCTUATION_RE_CC;
        'P': cc = NONPUNCTUATION_RE_CC;
        's': cc = WHITESPACE_RE_CC;
        'S': cc = NONWHITESPACE_RE_CC;
        't': cc = TAB_RE_CC;
        'u': cc = UCASE_RE_CC;
        'U': cc = NONUCASE_RE_CC;
        'w': cc = WORD_RE_CC;
        'W': cc = NONWORD_RE_CC;
        default:
            if ((cc >= '1') && (cc <= '9')) {
                par1 = cc-'0';
                cc = VARIABLE_RE_CC;
            } else {
                if (((cc >= 'a') && (cc <= 'z')) ||
                    ((cc >= 'A') && (cc <= 'Z'))) return "unknown escape";
                cc = LITERAL_RE_CC;
                par1 = ffrom-1; par2 = ffrom;
            }
    }
    quantifiable = true;
} else {
    switch (cc) {
        '(': par2 = 0;
            !if (BlkValueRead(findt, ffrom) == ')') return "empty subexpression";
            if (BlkValueRead(findt, ffrom) == '?') {
                ffrom++;
                bits = true;
                if (BlkValueRead(findt, ffrom) == '-') { ffrom++; bits = false; }
                else if (BlkValueRead(findt, ffrom) == '<') { ffrom++; bits = false; }
                switch (cc = BlkValueRead(findt, ffrom++)) {
                    '#': while (BlkValueRead(findt, ffrom++) ~= 0 or ')') ;
                        if (BlkValueRead(findt, ffrom-1) == 0)
                            return "comment never ends";
                        continue;
                    '(': cc = BlkValueRead(findt, ffrom);
                        if ((cc == '1' or '2' or '3' or '4' or
                            '5' or '6' or '7' or '8' or '9') &&
                            (BlkValueRead(findt, ffrom+1) ==')')) {
                            ffrom = ffrom + 2;
                            par1 = cc - '0';
                        } else ffrom--;
                        cc = IF_RE_CC; ! (?(...)...) conditional
                        quantifiable = false;
                        if (blevel == 20) return "subexpressions too deep";
                        Subexp_Posns-->(blevel++) = IT_RE_NodeAddress(no_packets);
                        jump CClassKnown;
                    '=': par2 = 1; ! (?=...) lookahead/behind
                        par3 = 0; if (bits == false) par3 = -1;
                    '!': par2 = 2; ! (?!...) negated lookahead/behind
                        par3 = 0; if (bits == false) par3 = -1;
                    ':': par2 = 3; ! (?:...) uncollecting subexpression
                    '>': par2 = 4; ! (?>...) possessive
                    default:
                        if (BlkValueRead(findt, ffrom) == ')') {
                            if (cc == 'i') {
                                cc = SENSITIVITY_RE_CC; par1 = bits; ffrom++;
                                jump CClassKnown;
                            }
                        }
                        if (BlkValueRead(findt, ffrom) == ':') {
                            if (cc == 'i') {
                                par1 = bits; par2 = 3; par3 = bits+1; ffrom++;
                                jump AllowForm;
                            }
                        }
                        return "unknown (?...) form";
                }
            }
            .AllowForm;
            if (par2 == 0) par1 = no_subs++; else par1 = -1;
            cc = SUBEXP_RE_CC;
            quantifiable = false;
            if (blevel == 20) return "subexpressions too deep";
            Subexp_Posns-->(blevel++) = IT_RE_NodeAddress(no_packets);
        ')': if (blevel == 0) return "subexpression bracket mismatch";
            blevel--;
            attach_to = Subexp_Posns-->blevel;
            if (attach_to-->RE_DOWN == NULL) {
                if (no_packets >= RE_MAX_PACKETS) return "regexp too complex";
                attach_to-->RE_DOWN =
                    IT_RE_Node(no_packets++, ALWAYS_RE_CC, 0, 0, 0);
                (attach_to-->RE_DOWN)-->RE_UP = attach_to;
            }
            quantifiable = true;
            continue;
        '.': cc = ANYTHING_RE_CC; quantifiable = true;
        '|': cc = CHOICE_RE_CC; quantifiable = false;
        '^': cc = START_RE_CC; quantifiable = false;
        '$': cc = END_RE_CC; quantifiable = false;
        '{': if (quantifiable == false) return "quantifier misplaced";
            par1 = 0; par2 = -1; bits = 1;
            while ((cc=BlkValueRead(findt, ffrom++)) ~= 0 or '}') {
                if (cc == ',') {
                    bits++;
                    if (bits >= 3) return "too many colons in ?{...}";
                    continue;
                }
                if ((cc >= '0') || (cc <= '9')) {
                    if (bits == 1) {
                        if (par1 < 0) par1 = 0;
                        par1 = par1*10 + (cc-'0');
                    } else {
                        if (par2 < 0) par2 = 0;
                        par2 = par2*10 + (cc-'0');
                    }
                } else return "non-digit in ?{...}";
            }
            if (cc ~= '}') return "{x,y} quantifier never ends";
            cc = QUANTIFIER_RE_CC;
            if (par2 == -1) {
                if (bits == 2) par2 = 30000;
                else par2 = par1;
            }
            if (par1 > par2) return "{x,y} with x greater than y";
            if (BlkValueRead(findt, ffrom) == '?') { ffrom++; par3 = true; }
            quantifiable = false;
        '<', '[': par3 = false; if (cc == '<') bits = '>'; else bits = ']';
            if (BlkValueRead(findt, ffrom) == '^') { ffrom++; par3 = true; }
            par1 = ffrom;
            if (BlkValueRead(findt, ffrom) == bits) { ffrom++; }
            while (cc ~= bits or 0) {
                cc = BlkValueRead(findt, ffrom++);
                if (cc == '\') {
                    cc = BlkValueRead(findt, ffrom++);
                    if (cc ~= 0) cc = BlkValueRead(findt, ffrom++);
                }
            }
            if (cc == 0) return "Character range never ends";
            par2 = ffrom-1;
            if ((par2 > par1 + 1) &&
                (BlkValueRead(findt, par1) == ':') &&
                (BlkValueRead(findt, par2-1) == ':') &&
                (BlkValueRead(findt, par2-2) ~= '\'))
                return "POSIX named character classes unsupported";
            bits = IT_RE_RangeSyntaxCorrect(findt, par1, par2);
            if (bits) return bits;
            if (par1 < par2) cc = RANGE_RE_CC;
            else cc = NOTHING_RE_CC;
            quantifiable = true;
        '*': if (quantifiable == false) return "quantifier misplaced";
            cc = QUANTIFIER_RE_CC;
            par1 = 0; par2 = 30000;
            if (BlkValueRead(findt, ffrom) == '?') { ffrom++; par3 = true; }
            quantifiable = false;
        '+': if (quantifiable == false) return "quantifier misplaced";
            cc = QUANTIFIER_RE_CC;
            par1 = 1; par2 = 30000;
```

```
                if (BlkValueRead(findt, ffrom) == '?') { ffrom++; par3 = true; }
                quantifiable = false;
            '?': if (quantifiable == false) return "quantifier misplaced";
                cc = QUANTIFIER_RE_CC;
                par1 = 0; par2 = 1;
                if (BlkValueRead(findt, ffrom) == '?') { ffrom++; par3 = true; }
                quantifiable = false;
        }
    }
    .CClassKnown;
    if (cc >= 0) {
        quantifiable = true;
        if ((attach_to-->RE_CCLASS == LITERAL_RE_CC) &&
            (BlkValueRead(findt, ffrom) ~= '*' or '+' or '?' or '{')) {
            (attach_to-->RE_PAR2)++;
            if (IT_RE_Trace == 2) {
                print "Extending literal by ", cc, "=", (char) cc, "^";
            }
            continue;
        }
        cc = LITERAL_RE_CC; par1 = ffrom-1; par2 = ffrom;
    }
    if (no_packets >= RE_MAX_PACKETS) return "regexp too complex";
    if (IT_RE_Trace == 2) {
        print "Attaching packet ", no_packets+1, " to ";
        IT_RE_DebugNode(attach_to, findt);
        IT_RE_DebugTree(findt);
    }
    token = IT_RE_Node(no_packets++, cc, par1, par2, par3);
    if ((token-->RE_CCLASS == SUBEXP_RE_CC) && (token-->RE_PAR2 == 0)) {
        RE_Subexpressions-->(token-->RE_PAR1) = token;
        (RE_Subexpressions-->10)++;
    }
    if ((attach_to-->RE_CCLASS == SUBEXP_RE_CC or CHOICE_RE_CC or IF_RE_CC) &&
        (attach_to-->RE_DOWN == NULL)) {
        attach_to-->RE_DOWN = token; token-->RE_UP = attach_to;
    } else {
        if ((token-->RE_CCLASS == CHOICE_RE_CC) &&
            ((attach_to-->RE_UP)-->RE_CCLASS == CHOICE_RE_CC)) {
            no_packets--; token = attach_to-->RE_UP;
        } else {
            if (token-->RE_CCLASS == CHOICE_RE_CC) {
                while (attach_to-->RE_PREVIOUS ~= NULL)
                    attach_to = attach_to-->RE_PREVIOUS;
            }
            if (token-->RE_CCLASS == QUANTIFIER_RE_CC or CHOICE_RE_CC) {
                token-->RE_PREVIOUS = attach_to-->RE_PREVIOUS;
                token-->RE_UP = attach_to-->RE_UP;
                if ((attach_to-->RE_UP ~= NULL) && (attach_to-->RE_PREVIOUS == NULL))
                    (attach_to-->RE_UP)-->RE_DOWN = token;
                token-->RE_DOWN = attach_to;
                bits = attach_to;
```

```
                    while (bits ~= NULL) {
                        bits-->RE_UP = token;
                        bits = bits-->RE_NEXT;
                    }
                    attach_to-->RE_PREVIOUS = NULL;
                    if (token-->RE_PREVIOUS ~= NULL)
                        (token-->RE_PREVIOUS)-->RE_NEXT = token;
                } else {
                    attach_to-->RE_NEXT = token; token-->RE_PREVIOUS = attach_to;
                    token-->RE_UP = attach_to-->RE_UP;
                }
            }
        }
        if (token-->RE_CCLASS == CHOICE_RE_CC) {
            if (no_packets >= RE_MAX_PACKETS) return "regexp too complex";
            token-->RE_NEXT = IT_RE_Node(no_packets++, CHOICE_RE_CC, 0, 0, 0);
            (token-->RE_NEXT)-->RE_PREVIOUS = token;
            (token-->RE_NEXT)-->RE_UP = token-->RE_UP;
            token = token-->RE_NEXT;
        }
        attach_to = token;
        if (IT_RE_Trace == 2) {
            print "Result:^";
            IT_RE_DebugTree(findt);
        }
    }
    if (blevel ~= 0) return "subexpression bracket mismatch";
    if (exactly) {
        if (no_packets+3 > RE_MAX_PACKETS) return "regexp too complex";
        exactly = RE_PACKET_space-->RE_DOWN;
        token = IT_RE_Node(no_packets++, START_RE_CC, 0, 0, 0);
        RE_PACKET_space-->RE_DOWN = token; token-->RE_UP = RE_PACKET_space;
        attach_to = IT_RE_Node(no_packets++, SUBEXP_RE_CC, -1, 3, 0);
        token-->RE_NEXT = attach_to; attach_to-->RE_PREVIOUS = token;
        attach_to-->RE_UP = RE_PACKET_space;
        attach_to-->RE_NEXT = IT_RE_Node(no_packets++, END_RE_CC, 0, 0, 0);
        (attach_to-->RE_NEXT)-->RE_PREVIOUS = attach_to;
        (attach_to-->RE_NEXT)-->RE_UP = RE_PACKET_space;
        attach_to-->RE_DOWN = exactly;
        while (exactly ~= NULL) {
            exactly-->RE_UP = attach_to; exactly = exactly-->RE_NEXT;
        }
    }
    no_packets = IT_RE_ExpandChoices(RE_PACKET_space, no_packets);
    if (IT_RE_Trace) {
        print "Compiled pattern:^";
        IT_RE_DebugTree(findt);
    }
    bits = IT_RE_CheckTree(RE_PACKET_space, no_subs); if (bits) return bits;
    return no_packets;
];
```

```
[ IT_RE_RangeSyntaxCorrect findt rf rt
    i chm;
    for (i=rf: i<rt: i++) {
        chm = BlkValueRead(findt, i);
        if ((chm == '\') && (i+1<rt)) {
            chm = BlkValueRead(findt, ++i);
            if (((chm >= 'a') && (chm <= 'z')) ||
                ((chm >= 'A') && (chm <= 'Z'))) {
                if (chm ~= 's' or 'S' or 'p' or 'P' or 'w' or 'W' or 'd'
                    or 'D' or 'n' or 't' or 'l' or 'L' or 'u' or 'U')
                    return "Invalid escape in {} range";
            }
        }
        if ((i+2<rt) && (BlkValueRead(findt, i+1) == '-')) {
            if (chm > BlkValueRead(findt, i+2)) return "Invalid {} range";
            i=i+2;
        }
    }
    rfalse;
];
[ IT_RE_ExpandChoices token no_packets
    rv prev nex holder new ct n cond_node then_node else_node;
    while (token ~= NULL) {
        if (token-->RE_CCLASS == IF_RE_CC) {
            if ((token-->RE_DOWN)-->RE_CCLASS == CHOICE_RE_CC) {
                for (nex=token-->RE_DOWN, n=0: nex~=NULL: nex=nex-->RE_NEXT) n++;
                if (n~=2) return "conditional has too many clauses";
                if (no_packets >= RE_MAX_PACKETS) return "regexp too complex";
                cond_node = IT_RE_Node(no_packets++, CONDITION_RE_CC, 0, 0, 0);
                if (token-->RE_PAR1 >= 1) {
                    cond_node-->RE_PAR1 = token-->RE_PAR1;
                }
                then_node = token-->RE_DOWN;
                then_node-->RE_CCLASS = THEN_RE_CC;
                else_node = then_node-->RE_NEXT;
                else_node-->RE_CCLASS = ELSE_RE_CC;
                if (cond_node-->RE_PAR1 < 1) {
                    cond_node-->RE_DOWN = then_node-->RE_DOWN;
                    then_node-->RE_DOWN = (then_node-->RE_DOWN)-->RE_NEXT;
                    if (then_node-->RE_DOWN ~= NULL)
                        (then_node-->RE_DOWN)-->RE_PREVIOUS = NULL;
                    (cond_node-->RE_DOWN)-->RE_NEXT = NULL;
                    (cond_node-->RE_DOWN)-->RE_UP = cond_node;
                }
                token-->RE_DOWN = cond_node; cond_node-->RE_UP = token;
                cond_node-->RE_NEXT = then_node; then_node-->RE_PREVIOUS = cond_node;
            } else {
                if (no_packets >= RE_MAX_PACKETS) return "regexp too complex";
                cond_node = IT_RE_Node(no_packets++, CONDITION_RE_CC, 0, 0, 0);
                if (no_packets >= RE_MAX_PACKETS) return "regexp too complex";
                then_node = IT_RE_Node(no_packets++, THEN_RE_CC, 0, 0, 0);
                if (token-->RE_PAR1 >= 1) {
                    cond_node-->RE_PAR1 = token-->RE_PAR1;
                    then_node-->RE_DOWN = token-->RE_DOWN;
                } else {
                    cond_node-->RE_DOWN = token-->RE_DOWN;
                    then_node-->RE_DOWN = (token-->RE_DOWN)-->RE_NEXT;
                    (cond_node-->RE_DOWN)-->RE_NEXT = NULL;
                    (cond_node-->RE_DOWN)-->RE_UP = cond_node;
                }
                token-->RE_DOWN = cond_node;
                cond_node-->RE_UP = token; cond_node-->RE_NEXT = then_node;
                then_node-->RE_PREVIOUS = cond_node; then_node-->RE_UP = token;
                then_node-->RE_NEXT = NULL;
                if (then_node-->RE_DOWN ~= NULL)
                    (then_node-->RE_DOWN)-->RE_PREVIOUS = NULL;
                for (nex = then_node-->RE_DOWN: nex ~= NULL: nex = nex-->RE_NEXT) {
                    nex-->RE_UP = then_node;
                }
            }
            if (cond_node-->RE_DOWN ~= NULL) {
                nex = cond_node-->RE_DOWN;
                if ((nex-->RE_CCLASS ~= SUBEXP_RE_CC) ||
                    (nex-->RE_NEXT ~= NULL) ||
                    (nex-->RE_PAR2 ~= 1 or 2)) {
                    !IT_RE_DebugSubtree(0, 0, nex, true);
                    return "condition not lookahead/behind";
                }
            }
        }
        if ((token-->RE_CCLASS == CHOICE_RE_CC) && (token-->RE_PAR1 < 1)) {
            prev = token-->RE_PREVIOUS;
            nex = token-->RE_NEXT;
            while ((nex ~= NULL) && (nex-->RE_CCLASS == CHOICE_RE_CC))
                nex = nex-->RE_NEXT;
            holder = token-->RE_UP; if (holder == NULL) return "bang";
            if (no_packets >= RE_MAX_PACKETS) return "regexp too complex";
            new = IT_RE_Node(no_packets++, DISJUNCTION_RE_CC, 0, 0, 0);
            holder-->RE_DOWN = new; new-->RE_UP = holder;
            if (prev ~= NULL) {
                prev-->RE_NEXT = new; new-->RE_PREVIOUS = prev;
            }
            if (nex ~= NULL) {
                nex-->RE_PREVIOUS = new; new-->RE_NEXT = nex;
            }
            new-->RE_DOWN = token;
            token-->RE_PREVIOUS = NULL;
            ct = 1;
            while (token ~= NULL) {
                token-->RE_PAR1 = ct++;
                token-->RE_UP = new;
                if ((token-->RE_NEXT ~= NULL) &&
                    ((token-->RE_NEXT)-->RE_CCLASS ~= CHOICE_RE_CC))
                    token-->RE_NEXT = NULL;
                token = token-->RE_NEXT;
            }
```

```
            new-->RE_PAR1 = ct-1;
            if (token ~= NULL) token-->RE_NEXT = NULL;
            token = new; continue;
        }
        if (token-->RE_DOWN ~= NULL) {
            no_packets = IT_RE_ExpandChoices(token-->RE_DOWN, no_packets);
            if ((no_packets<0) || (no_packets >= RE_MAX_PACKETS)) break;
        }
        token = token-->RE_NEXT;
    }
    return no_packets;
];
[ IT_RE_CheckTree token no_subs
    rv;
    while (token ~= NULL) {
        if (token-->RE_CCLASS == VARIABLE_RE_CC) {
            if (token-->RE_PAR1 >= no_subs) return "reference to nonexistent group";
        }
        if ((token-->RE_CCLASS == SUBEXP_RE_CC) &&
            (token-->RE_PAR2 == 1 or 2) &&
            (token-->RE_PAR3 == -1)) {
            token-->RE_PAR3 = IT_RE_Width(token-->RE_DOWN);
            if (token-->RE_PAR3 == -1) return "variable length lookbehind not implemented";
        }
        if (token-->RE_DOWN ~= NULL) {
            rv = IT_RE_CheckTree(token-->RE_DOWN, no_subs);
            if (rv) return rv;
        }
        token = token-->RE_NEXT;
    }
    rfalse;
];
[ IT_RE_Width token downwards
    w rv aw choice;
    while (token ~= NULL) {
        switch (token-->RE_CCLASS) {
            DIGIT_RE_CC, NONDIGIT_RE_CC, WHITESPACE_RE_CC, NONWHITESPACE_RE_CC,
            PUNCTUATION_RE_CC, NONPUNCTUATION_RE_CC, WORD_RE_CC, NONWORD_RE_CC,
            ANYTHING_RE_CC, NOTHING_RE_CC, RANGE_RE_CC, NEWLINE_RE_CC, TAB_RE_CC,
            UCASE_RE_CC, NONUCASE_RE_CC, LCASE_RE_CC, NONLCASE_RE_CC:
                w++;
            START_RE_CC, END_RE_CC, BOUNDARY_RE_CC, NONBOUNDARY_RE_CC, ALWAYS_RE_CC:
                ;
            LITERAL_RE_CC:
                w = w + token-->RE_PAR2 - token-->RE_PAR1;
            VARIABLE_RE_CC:
                return -1;
            IF_RE_CC:
                rv = IT_RE_Width((token-->RE_DOWN)-->RE_NEXT);
                if (rv == -1) return -1;
                if (rv ~= IT_RE_Width(((token-->RE_DOWN)-->RE_NEXT)-->RE_NEXT))
                    return -1;
                w = w + rv;
            SUBEXP_RE_CC:
                if (token-->RE_PAR2 == 1 or 2) rv = 0;
                else {
                    rv = IT_RE_Width(token-->RE_DOWN);
                    if (rv == -1) return -1;
                }
                w = w + rv;
            QUANTIFIER_RE_CC:
                if (token-->RE_PAR1 ~= token-->RE_PAR2) return -1;
                rv = IT_RE_Width(token-->RE_DOWN);
                if (rv == -1) return -1;
                w = w + rv*(token-->RE_PAR1);
            DISJUNCTION_RE_CC:
                aw = -1;
                for (choice = token-->RE_DOWN: choice ~= NULL: choice = choice-->RE_NEXT) {
                    rv = IT_RE_Width(choice-->RE_DOWN);
                    !print "Option found ", rv, "^";
                    if (rv == -1) return -1;
                    if ((aw >= 0) && (aw ~= rv)) return -1;
                    aw = rv;
                }
                w = w + aw;
            SENSITIVITY_RE_CC:
                ;
        }
        if (downwards) return w;
        if (token ~= NULL) token = token-->RE_NEXT;
    }
    return w;
];
```

§**11. Parser.** The virtue of all of that tree compilation is that the code which actually does the work – which parses the source text to see if the regular expression matches it – is much shorter and quicker: indeed, it takes up fewer lines than the compiler part, which goes to show that decoding regular expression syntax is a more complex task than acting upon it. This would have surprised the pioneers of regexp, but the syntax has become much more complicated over the decades because of a steady increase in the number of extended notations. The process shows no sign of stopping, with Python and PCRE continuing to push boundaries beyond Perl, which was once thought the superest, duperest regexp syntax there could be. However: to work.

The main matcher simply starts a recursive subroutine to perform the match. However, the subroutine tests for a match at a particular position in the source text; so the main routine tries the subroutine everywhere convenient in the source text, from left to right, until a match is made – unless the regexp is constrained by a ^ glyph to begin matching at the start of the source text, which will cause a `START_RE_CC` node to be the eldest child of the `\0` root.

```
Global IT_RE_RewindCount;
[ IT_RE_PrintNoRewinds; print IT_RE_RewindCount; ];

Constant CIS_MFLAG = 1;
Constant ACCUM_MFLAG = 2;

[ IT_RE_Parse findt indt ipos insens
    ilen rv root i initial_mode;

    ilen = IT_CharacterLength(indt);
    if ((ipos<0) || (ipos>ilen)) return -1;
    root = RE_PACKET_space;
    initial_mode = 0; if (insens) initial_mode = CIS_MFLAG;
    IT_RE_Clear_Markers(RE_PACKET_space);
    for (:ipos<=ilen:ipos++) {
        if ((RE_PACKET_space-->RE_DOWN ~= NULL) &&
            ((RE_PACKET_space-->RE_DOWN)-->RE_CCLASS == START_RE_CC) &&
            (ipos>0)) { rv = -1; break; }
        if (ipos > 0) IT_RE_EraseConstraints(RE_PACKET_space, initial_mode);
        IT_RE_RewindCount = 0;
        rv = IT_RE_ParseAtPosition(findt, indt, ipos, ilen, RE_PACKET_space, initial_mode);
        if (rv >= 0) break;
    }
    if (rv == -1) {
        root-->RE_DATA1 = -1;
        root-->RE_DATA2 = -1;
    } else {
        root-->RE_DATA1 = ipos;
        root-->RE_DATA2 = ipos+rv;
    }
    return rv;
];
```

§**12. Parse At Position.** `IT_RE_ParseAtPosition(findt, indt, ifrom, ito)` attempts to match text beginning at position `ifrom` in the indexed text `indt` and extending for any length up to position `ito`: it returns the number of characters which were matched (which can legitimately be 0), or -1 if no match could be made. `findt` is the original text of the regular expression in its precompiled form, which we need partly to print good debugging information, but mostly in order to match against a `LITERAL_RE_CC` node.

```
[ IT_RE_ParseAtPosition findt indt ifrom ito token mode_flags
    outcome ipos npos rv i ch edge rewind_this;
    if (ifrom > ito) return -1;
    ipos = ifrom;
    .Rewind;
    while (token ~= NULL) {
        outcome = false;
        if (IT_RE_Trace) {
            print "Matching at ", ipos, ": ";
            IT_RE_DebugNode(token, findt, true);
        }
        if (ipos<ito) ch = BlkValueRead(indt, ipos); else ch = 0;
        token-->RE_MODES = mode_flags; ! Save in case of backtrack
        switch (token-->RE_CCLASS) {
            ! Should never happen
            CHOICE_RE_CC: return "internal error";
            ! Mode switches
            SENSITIVITY_RE_CC:
                if (token-->RE_PAR1) mode_flags = mode_flags | CIS_MFLAG;
                else mode_flags = mode_flags & (~CIS_MFLAG);
                outcome = true;
            ! Zero-length positional markers
            ALWAYS_RE_CC:
                outcome = true;
            NEVER_RE_CC:
            START_RE_CC:
                if (ipos == 0) outcome = true;
            END_RE_CC:
                if (BlkValueRead(indt, ipos) == 0) outcome = true;
            BOUNDARY_RE_CC:
                rv = 0;
                if (BlkValueRead(indt, ipos) == 0 or 10 or 13 or 32 or 9
                    or '.' or ',' or '!' or '?'
                    or '-' or '/' or '"' or ':' or ';'
                    or '(' or ')' or '[' or ']' or '{' or '}') rv++;
                if (ipos == 0) ch = 0;
                else ch = BlkValueRead(indt, ipos-1);
                if (ch == 0 or 10 or 13 or 32 or 9
                    or '.' or ',' or '!' or '?'
                    or '-' or '/' or '"' or ':' or ';'
                    or '(' or ')' or '[' or ']' or '{' or '}') rv++;
                if (rv == 1) outcome = true;
            NONBOUNDARY_RE_CC:
                rv = 0;
                if (BlkValueRead(indt, ipos) == 0 or 10 or 13 or 32 or 9
                    or '.' or ',' or '!' or '?'
                    or '-' or '/' or '"' or ':' or ';'
                    or '(' or ')' or '[' or ']' or '{' or '}') rv++;
                if (ipos == 0) ch = 0;
                else ch = BlkValueRead(indt, ipos-1);
                if (ch == 0 or 10 or 13 or 32 or 9
                    or '.' or ',' or '!' or '?'
                    or '-' or '/' or '"' or ':' or ';'
                    or '(' or ')' or '[' or ']' or '{' or '}') rv++;
                if (rv ~= 1) outcome = true;
            ! Control constructs
            IF_RE_CC:
                i = token-->RE_PAR1; ch = false;
                if (IT_RE_Trace) {
                    print "Trying conditional from ", ipos, ": ";
                    IT_RE_DebugNode(token, findt, true);
                }
                if (i >= 1) {
                    if ((i<RE_Subexpressions-->10) &&
                         ((RE_Subexpressions-->i)-->RE_DATA1 >= 0)) ch = true;
                } else {
                    rv = IT_RE_ParseAtPosition(findt, indt, ipos, ito,
                         (token-->RE_DOWN)-->RE_DOWN, mode_flags);
                    if (rv >= 0) ch = true;
                }
                if (IT_RE_Trace) {
                    print "Condition found to be ", ch, "^";
                }
                if (ch) {
                    rv = IT_RE_ParseAtPosition(findt, indt, ipos, ito,
                        ((token-->RE_DOWN)-->RE_NEXT)-->RE_DOWN, mode_flags);
                    !print "Then clause returned ", rv, "^";
                } else {
                    if ((((token-->RE_DOWN)-->RE_NEXT)-->RE_NEXT) == NULL)
                        rv = 0; ! The empty else clause matches
                    else rv = IT_RE_ParseAtPosition(findt, indt, ipos, ito,
                        (((token-->RE_DOWN)-->RE_NEXT)-->RE_NEXT)-->RE_DOWN, mode_flags);
                    !print "Else clause returned ", rv, "^";
                }
                if (rv >= 0) {
                    outcome = true;
                    ipos = ipos + rv;
                }
            DISJUNCTION_RE_CC:
                if (IT_RE_Trace) {
                    print "Trying disjunction from ", ipos, ": ";
                    IT_RE_DebugNode(token, findt, true);
                }
                for (ch = token-->RE_DOWN: ch ~= NULL: ch = ch-->RE_NEXT) {
                    if (ch-->RE_PAR1 <= token-->RE_CONSTRAINT) continue;
                    if (IT_RE_Trace) {
                        print "Trying choice at ", ipos, ": ";
                        IT_RE_DebugNode(ch, findt, true);
                    }
                    rv = IT_RE_ParseAtPosition(findt, indt, ipos, ito,
                        ch-->RE_DOWN, mode_flags);
                    if (rv >= 0) {
                        token-->RE_DATA1 = ipos; ! Where match was made
                        token-->RE_DATA2 = ch-->RE_PAR1; ! Option taken
                        ipos = ipos + rv;
                        outcome = true;
                        if (IT_RE_Trace) {
                            print "Choice worked with width ", rv, ": ";
                            IT_RE_DebugNode(ch, findt, true);
                        }
                        break;
                    } else {
                        if (mode_flags & ACCUM_MFLAG == false)
                            IT_RE_FailSubexpressions(ch-->RE_DOWN);
                    }
                }
                if (outcome == false) {
                    if (IT_RE_Trace) {
                        print "Failed disjunction from ", ipos, ": ";
                        IT_RE_DebugNode(token, findt, true);
                    }
                    token-->RE_DATA1 = ipos; ! Where match was tried
                    token-->RE_DATA2 = -1; ! No option was taken
                }
            SUBEXP_RE_CC:
```

```
    if (token-->RE_PAR2 == 1 or 2) {
        npos = ipos - token-->RE_PAR3;
        if (npos<0) rv = -1; ! Lookbehind fails: nothing behind
        else rv = IT_RE_ParseAtPosition(findt, indt, npos, ito, token-->RE_DOWN,
            mode_flags);
    } else {
        switch (token-->RE_PAR3) {
            0: rv = IT_RE_ParseAtPosition(findt, indt, ipos, ito, token-->RE_DOWN,
                mode_flags);
            1: rv = IT_RE_ParseAtPosition(findt, indt, ipos, ito, token-->RE_DOWN,
                mode_flags & (~CIS_MFLAG));
            2: rv = IT_RE_ParseAtPosition(findt, indt, ipos, ito, token-->RE_DOWN,
                mode_flags | CIS_MFLAG);
        }
    }
    npos = ipos;
    if (rv >= 0) npos = ipos + rv;
    switch (token-->RE_PAR2) {
        1: if (rv >= 0) rv = 0;
        2: if (rv >= 0) rv = -1; else rv = 0;
    }
    if (rv >= 0) {
        token-->RE_DATA1 = ipos;
        ipos = ipos + rv;
        token-->RE_DATA2 = npos;
        outcome = true;
    } else {
        if (mode_flags & ACCUM_MFLAG == false) {
            token-->RE_DATA1 = -1;
            token-->RE_DATA2 = -1;
        }
    }
    if (token-->RE_PAR2 == 2) IT_RE_FailSubexpressions(token, true);
QUANTIFIER_RE_CC:
    token-->RE_DATA1 = ipos;
    if ((token-->RE_DOWN)-->RE_CCLASS == SUBEXP_RE_CC) {
        (token-->RE_DOWN)-->RE_CACHE1 = -1;
        (token-->RE_DOWN)-->RE_CACHE2 = -1;
    }
    if (IT_RE_Trace) {
        print "Trying quantifier from ", ipos, ": ";
        IT_RE_DebugNode(token, findt, true);
    }
    if (token-->RE_PAR3 == false) { ! Greedy quantifier
        !edge = ito; if (token-->RE_CONSTRAINT >= 0) edge = token-->RE_CONSTRAINT;
        edge = token-->RE_PAR2;
        if (token-->RE_CONSTRAINT >= 0) edge = token-->RE_CONSTRAINT;
        rv = -1;
        for (i=0, npos=ipos: i<edge: i++) {
            if (IT_RE_Trace) {
                print "Trying quant rep ", i+1, " at ", npos, ": ";
                IT_RE_DebugNode(token, findt, true);
            }
```

```
                rv = IT_RE_ParseAtPosition(findt, indt, npos, ito, token-->RE_DOWN,
                    mode_flags | ACCUM_MFLAG);
                if (rv < 0) break;
                if ((token-->RE_DOWN)-->RE_CCLASS == SUBEXP_RE_CC) {
                    (token-->RE_DOWN)-->RE_CACHE1 = (token-->RE_DOWN)-->RE_DATA1;
                    (token-->RE_DOWN)-->RE_CACHE2 = (token-->RE_DOWN)-->RE_DATA2;
                }
                if ((rv == 0) && (token-->RE_PAR2 == 30000) && (i>=1)) { i++; break; }
                npos = npos + rv;
            }
            if ((i >= token-->RE_PAR1) && (i <= token-->RE_PAR2))
                outcome = true;
        } else { ! Lazy quantifier
            edge = token-->RE_PAR1;
            if (token-->RE_CONSTRAINT > edge) edge = token-->RE_CONSTRAINT;
            for (i=0, npos=ipos: (npos<ito) && (i < token-->RE_PAR2): i++) {
                if (i >= edge) break;
                if (IT_RE_Trace) {
                    print "Trying quant rep ", i+1, " at ", npos, ": ";
                    IT_RE_DebugNode(token, findt, true);
                }
                rv = IT_RE_ParseAtPosition(findt, indt, npos, ito, token-->RE_DOWN,
                    mode_flags | ACCUM_MFLAG);
                if (rv < 0) break;
                if ((token-->RE_DOWN)-->RE_CCLASS == SUBEXP_RE_CC) {
                    (token-->RE_DOWN)-->RE_CACHE1 = (token-->RE_DOWN)-->RE_DATA1;
                    (token-->RE_DOWN)-->RE_CACHE2 = (token-->RE_DOWN)-->RE_DATA2;
                }
                if ((rv == 0) && (token-->RE_PAR2 == 30000) && (i>=1)) { i++; break; }
                npos = npos + rv;
            }
            if ((i >= edge) && (i <= token-->RE_PAR2))
                outcome = true;
        }
        if (outcome) {
            if (token-->RE_PAR3 == false) { ! Greedy quantifier
                if (i > token-->RE_PAR1) { ! I.e., if we have been greedy
                    token-->RE_DATA2 = i-1; ! And its edge limitation
                } else {
                    token-->RE_DATA2 = -1;
                }
            } else { ! Lazy quantifier
                if (i < token-->RE_PAR2) { ! I.e., if we have been lazy
                    token-->RE_DATA2 = i+1; ! And its edge limitation
                } else {
                    token-->RE_DATA2 = -1;
                }
            }
            ipos = npos;
            if ((i == 0) && (mode_flags & ACCUM_MFLAG == false))
                IT_RE_FailSubexpressions(token-->RE_DOWN);
            if ((token-->RE_DOWN)-->RE_CCLASS == SUBEXP_RE_CC) {
                (token-->RE_DOWN)-->RE_DATA1 = (token-->RE_DOWN)-->RE_CACHE1;
                (token-->RE_DOWN)-->RE_DATA2 = (token-->RE_DOWN)-->RE_CACHE2;
            }
            if (IT_RE_Trace) {
                print "Successful quant reps ", i, ": ";
                IT_RE_DebugNode(token, findt, true);
            }
        } else {
            !token-->RE_DATA2 = -1;
            if (mode_flags & ACCUM_MFLAG == false)
                IT_RE_FailSubexpressions(token-->RE_DOWN);
            if (IT_RE_Trace) {
                print "Failed quant reps ", i, ": ";
                IT_RE_DebugNode(token, findt, true);
            }
        }
    ! Character classes
    NOTHING_RE_CC: ;
    ANYTHING_RE_CC: if (ch) outcome = true; ipos++;
    WHITESPACE_RE_CC:
        if (ch == 10 or 13 or 32 or 9) { outcome = true; ipos++; }
    NONWHITESPACE_RE_CC:
        if ((ch) && (ch ~= 10 or 13 or 32 or 9)) { outcome = true; ipos++; }
    PUNCTUATION_RE_CC:
        if (ch == '.' or ',' or '!' or '?'
            or '-' or '/' or '"' or ':' or ';'
            or '(' or ')' or '[' or ']' or '{' or '}') { outcome = true; ipos++; }
    NONPUNCTUATION_RE_CC:
        if ((ch) && (ch ~= '.' or ',' or '!' or '?'
            or '-' or '/' or '"' or ':' or ';'
            or '(' or ')' or '[' or ']' or '{' or '}')) { outcome = true; ipos++; }
    WORD_RE_CC:
        if ((ch) && (ch ~= 10 or 13 or 32 or 9
            or '.' or ',' or '!' or '?'
            or '-' or '/' or '"' or ':' or ';'
            or '(' or ')' or '[' or ']' or '{' or '}')) { outcome = true; ipos++; }
    NONWORD_RE_CC:
        if (ch == 10 or 13 or 32 or 9
            or '.' or ',' or '!' or '?'
            or '-' or '/' or '"' or ':' or ';'
            or '(' or ')' or '[' or ']' or '{' or '}') { outcome = true; ipos++; }
    DIGIT_RE_CC:
        if (ch == '0' or '1' or '2' or '3' or '4'
            or '5' or '6' or '7' or '8' or '9') { outcome = true; ipos++; }
    NONDIGIT_RE_CC:
        if ((ch) && (ch ~= '0' or '1' or '2' or '3' or '4'
            or '5' or '6' or '7' or '8' or '9')) { outcome = true; ipos++; }
    LCASE_RE_CC:
        if (CharIsOfCase(ch, 0)) { outcome = true; ipos++; }
    NONLCASE_RE_CC:
        if ((ch) && (CharIsOfCase(ch, 0) == false)) { outcome = true; ipos++; }
    UCASE_RE_CC:
        if (CharIsOfCase(ch, 1)) { outcome = true; ipos++; }
    NONUCASE_RE_CC:
```

```
            if ((ch) && (CharIsOfCase(ch, 1) == false)) { outcome = true; ipos++; }
        NEWLINE_RE_CC: if (ch == 10) { outcome = true; ipos++; }
        TAB_RE_CC: if (ch == 9) { outcome = true; ipos++; }
        RANGE_RE_CC:
            if (IT_RE_Range(ch, findt,
                token-->RE_PAR1, token-->RE_PAR2, token-->RE_PAR3, mode_flags & CIS_MFLAG))
                { outcome = true; ipos++; }

        ! Substring matches

        LITERAL_RE_CC:
            rv = IT_RE_MatchSubstring(indt, ipos,
                findt, token-->RE_PAR1, token-->RE_PAR2, mode_flags & CIS_MFLAG);
            if (rv >= 0) { ipos = ipos + rv; outcome = true; }
        VARIABLE_RE_CC:
            i = token-->RE_PAR1;
            if ((RE_Subexpressions-->i)-->RE_DATA1 >= 0) {
                rv = IT_RE_MatchSubstring(indt, ipos,
                    indt, (RE_Subexpressions-->i)-->RE_DATA1,
                    (RE_Subexpressions-->i)-->RE_DATA2, mode_flags & CIS_MFLAG);
                if (rv >= 0) { ipos = ipos + rv; outcome = true; }
            }
            .NeverMatchIncompleteVar;
    }

    if (outcome == false) {
        if (IT_RE_RewindCount++ >= 10000) {
            if (IT_RE_RewindCount == 10001) {
                style bold; print "OVERFLOW^"; style roman;
            }
            return -1;
        }
        if (IT_RE_Trace) {
            print "Rewind sought from failure at pos ", ipos, " with: ";
                IT_RE_DebugNode(token, findt, true);
        }
        if ((token-->RE_CCLASS == QUANTIFIER_RE_CC) &&
            (IT_RE_SeekBacktrack(token-->RE_DOWN, findt, false, ito, false)))
            jump RewindFound;
        if (mode_flags & ACCUM_MFLAG == false) IT_RE_FailSubexpressions(token);
        token = token-->RE_PREVIOUS;
        while (token ~= NULL) {
            if (IT_RE_SeekBacktrack(token, findt, true, ito, false)) {
                .RewindFound;
                ipos = token-->RE_DATA1;
                mode_flags = token-->RE_MODES;
                if (mode_flags & ACCUM_MFLAG == false)
                    IT_RE_FailSubexpressions(token, true);
                if (ipos == -1)
                    IT_RE_DebugTree(findt, true);
                if (IT_RE_Trace) {
                    print "^[", ifrom, ",", ito, "] rewinding to ", ipos, " at ";
                    IT_RE_DebugNode(token, findt, true);
                }
                jump Rewind;
            }
```

```
                token = token-->RE_PREVIOUS;
            }
            if (IT_RE_Trace)
                print "^Rewind impossible^";
            return -1;
        }
        token = token-->RE_NEXT;
    }
    return ipos - ifrom;
];
```

§13. Backtracking. It would be very straightforward to match regular expressions with the above recursive code if, for every node, there were a fixed number of characters (depending on the node) such that there would either be a match eating that many characters, or else no match at all. If that were true, we could simply march through the text matching until we could match no more, and although some nodes might have ambiguous readings, we could always match the first possibility which worked. There would never be any need to retreat.

Well, in fact that happy state does apply to a surprising number of nodes, and some quite complicated regular expressions can be made which use only them: `<abc>{2}\d\d\1`, for instance, matches a sequence of exactly 6 characters or else fails to match altogether, and there is never any need to backtrack. One reason why backtracking is a fairly good algorithm in practice is that these "good" cases occur fairly often, in subexpressions if not in the entire expression, and the simple method above disposes of them efficiently.

But in an expression like `ab+bb`, there is no alternative to backtracking if we are going to try to match the nodes from left to right: we match the "a", then we match as many "b"s as we can – but then we find that we have to match "bb", and this is necessarily impossible because we have just eaten all of the "b"s available. We therefore backtrack one node to the `b+` and try again. We obviously can't literally try again because that would give the same result: instead we impose a constraint. Suppose it previously matched a row of 23 letter "b"s, so that the quantifier `+` resulted in a multiplicity of 23. We then constrain the node and in effect consider it to be `b{1,22}`, that is, to match at least 1 and at most 22 letter "b"s. That still won't work, as it happens, so we backtrack again with a constraint tightened to make it `b{1,21}`, and now the match occurs as we would hope. When the expression becomes more complex, backtracking becomes a longer-distance, recursive procedure – we have to exhaust all possibilities of a more recent node before tracking back to one from longer ago. (This is why the worst test cases are those which entice us into a long, long series of matches only to find that a guess made right back at the start was ill-fated.)

Rather than describing `IT_RE_SeekBacktrack` in detail here, it is probably more useful to suggest that the reader observe it in action by setting `IT_RE_Trace` and trying a few regular expressions.

```
[ IT_RE_SeekBacktrack token findt downwards ito report_only
    untried;
    for (: token ~= NULL: token = token-->RE_NEXT) {
        if ((IT_RE_Trace) && (report_only == false)) {
            print "Scan for rewind: ";
            IT_RE_DebugNode(token, findt, true);
        }
        if ((token-->RE_CCLASS == SUBEXP_RE_CC) &&
            (token-->RE_PAR2 == 1 or 2 or 4)) {
            if (downwards) rfalse;
            continue;
        }
        if (token-->RE_DOWN ~= NULL) {
            if ((IT_RE_Trace) && (report_only == false)) print "Descend^";
            if (IT_RE_SeekBacktrack(token-->RE_DOWN, findt, false, ito, report_only)) rtrue;
        }
        untried = false;
        switch (token-->RE_CCLASS) {
            DISJUNCTION_RE_CC:
                if ((token-->RE_DATA2 >= 1) &&
                    (token-->RE_DATA2 < token-->RE_PAR1) &&
                    (token-->RE_CONSTRAINT < token-->RE_PAR1)) { ! Matched but earlier than last
                    if (report_only) rtrue;
                    if (token-->RE_CONSTRAINT == -1)
                        token-->RE_CONSTRAINT = 1;
                    else
                        (token-->RE_CONSTRAINT)++;
                    untried = true;
                }
            QUANTIFIER_RE_CC:
                if (token-->RE_CONSTRAINT ~= -2) {
                    if ((IT_RE_Trace) && (report_only == false)) {
                        print "Quant with cons not -2: ";
                        IT_RE_DebugNode(token, findt, true);
                    }
                    if (token-->RE_DATA2 >= 0) {
                        if (report_only) rtrue;
                        token-->RE_CONSTRAINT = token-->RE_DATA2;
                        untried = true;
                    }
                }
        }
        if (untried) {
            if (IT_RE_Trace) {
                print "Grounds for rewind at: ";
                IT_RE_DebugNode(token, findt, true);
            }
            IT_RE_EraseConstraints(token-->RE_NEXT);
            IT_RE_EraseConstraints(token-->RE_DOWN);
            rtrue;
        }
        if (downwards) rfalse;
    }
    rfalse;
];
```

§**14. Fail Subexpressions.** Here, an attempt to make a complicated match against the node in `token` has failed: that means that any subexpressions which were matched in the course of the attempt must also in retrospect be considered unmatched. So we work down through the subtree at `token` and empty any markers for subexpressions, which in effect clears their backslash variables – this is important as, otherwise, the contents left over could cause the alternative reading of the `token` to be misparsed if it refers to the backslash variables in question. (If you think nobody would ever be crazy enough to write a regular expression like that, you haven't see Perl's test suite.)

If the `downwards` flag is clear, it not only invalidates subexpression matches below the node but also to the right of the node – this is useful for a backtrack which runs back quite some distance.

```
[ IT_RE_FailSubexpressions token downwards;
    for (: token ~= NULL: token = token-->RE_NEXT) {
        if (token-->RE_DOWN ~= NULL) IT_RE_FailSubexpressions(token-->RE_DOWN);
        if (token-->RE_CCLASS == SUBEXP_RE_CC) {
            token-->RE_DATA1 = -1;
            token-->RE_DATA2 = -1;
        }
        if (downwards) break;
    }
];
```

§**15. Erasing Constraints.** As explained above, temporary constraints are placed on some nodes when we are backtracking to test possible cases. When we do backtrack, though, it's important to lift any constraints left over from the previous attempt to parse material which is part of or subsequent to the token whose match attempt has been abandoned.

```
[ IT_RE_EraseConstraints token;
    while (token ~= NULL) {
        switch (token-->RE_CCLASS) {
            DISJUNCTION_RE_CC: token-->RE_CONSTRAINT = -1;
            QUANTIFIER_RE_CC: token-->RE_CONSTRAINT = -1;
        }
        if (token-->RE_DOWN) IT_RE_EraseConstraints(token-->RE_DOWN);
        token = token-->RE_NEXT;
    }
];
```

§16. Matching Literal Text. Here we attempt to make a match of the substring of the indexed text `mindt` which runs from character `mfrom` to character `mto`, looking for it at the given position `ipos` in the source text `indt`.

```
[ IT_RE_MatchSubstring indt ipos mindt mfrom mto insens
    i ch;
    if (mfrom < 0) return 0;
    if (insens)
        for (i=mfrom:i<mto:i++) {
            ch = BlkValueRead(mindt, i);
            if (BlkValueRead(indt, ipos++) ~= ch or IT_RevCase(ch))
                return -1;
        }
    else
        for (i=mfrom:i<mto:i++)
            if (BlkValueRead(indt, ipos++) ~= BlkValueRead(mindt, i))
                return -1;
    return mto-mfrom;
];
```

§17. Matching Character Range. Suppose that a character range is stored in `findt` between the character positions `rf` and `rt`. Then `IT_RE_Range(ch, findt, rf, rt, negate, insens)` tests whether a given character `ch` lies within that character range, negating the outcome if `negate` is set, and performing comparisons case insensitively if `insens` is set.

```
[ IT_RE_Range ch findt rf rt negate insens
    i chm upper crev;
    if (ch == 0) rfalse;
    if (negate == true) {
        if (IT_RE_Range(ch, findt, rf, rt, false, insens)) rfalse;
        rtrue;
    }
    for (i=rf: i<rt: i++) {
        chm = BlkValueRead(findt, i);
        if ((chm == '\') && (i+1<rt)) {
            chm = BlkValueRead(findt, ++i);
            switch (chm) {
                's':
                    if (ch == 10 or 13 or 32 or 9) rtrue;
                'S':
                    if ((ch) && (ch ~= 10 or 13 or 32 or 9)) rtrue;
                'p':
                    if (ch == '.' or ',' or '!' or '?'
                        or '-' or '/' or '"' or ':' or ';'
                        or '(' or ')' or '[' or ']' or '{' or '}') rtrue;
                'P':
                    if ((ch) && (ch ~= '.' or ',' or '!' or '?'
                        or '-' or '/' or '"' or ':' or ';'
                        or '(' or ')' or '[' or ']' or '{' or '}')) rtrue;
                'w':
                    if ((ch) && (ch ~= 10 or 13 or 32 or 9
                        or '.' or ',' or '!' or '?'
                        or '-' or '/' or '"' or ':' or ';'
                        or '(' or ')' or '[' or ']' or '{' or '}')) rtrue;
                'W':
                    if (ch == 10 or 13 or 32 or 9
                        or '.' or ',' or '!' or '?'
                        or '-' or '/' or '"' or ':' or ';'
                        or '(' or ')' or '[' or ']' or '{' or '}') rtrue;
                'd':
                    if (ch == '0' or '1' or '2' or '3' or '4'
                        or '5' or '6' or '7' or '8' or '9') rtrue;
                'D':
                    if ((ch) && (ch ~= '0' or '1' or '2' or '3' or '4'
                        or '5' or '6' or '7' or '8' or '9')) rtrue;
                'l': if (CharIsOfCase(ch, 0)) rtrue;
                'L': if (CharIsOfCase(ch, 0) == false) rtrue;
                'u': if (CharIsOfCase(ch, 1)) rtrue;
                'U': if (CharIsOfCase(ch, 1) == false) rtrue;
                'n': if (ch == 10) rtrue;
                't': if (ch == 9) rtrue;
            }
        }
        if ((i+2<rt) && (BlkValueRead(findt, i+1) == '-')) {
            upper = BlkValueRead(findt, i+2);
            if ((ch >= chm) && (ch <= upper)) rtrue;
            if (insens) {
                crev = IT_RevCase(ch);
                if ((crev >= chm) && (crev <= upper)) rtrue;
            }
            i=i+2;
        } else {
            if (chm == ch) rtrue;
            if ((insens) && (chm == IT_RevCase(ch))) rtrue;
        }
    }
    rfalse;
];
```

§18. Search And Replace. And finally, last but not least: the routine which searches an indexed text `indt` trying to match it against `findt`. If `findtype` is set to `REGEXP_BLOB` then `findt` is expected to be a regular expression such as `ab+(c*de)?`, whereas if `findtype` is `CHR_BLOB` then it is expected only to be a simple string of characters taken literally, such as `frog`.

Each match found is replaced with the indexed text in `rindt`, except that if the blob type is `REGEXP_BLOB` then we recognise a few syntaxes as special: for instance, `\2` expands to the value of subexpression 2 as it was matched – see *Writing with Inform* for details.

The optional argument `insens` is a flag which, if set, causes the matching to be done case insensitively; the optional argument `exactly`, if set, causes the matching to work only if the entire `indt` is matched. (This is not especially useful with regular expressions, because the effect can equally be achieved by turning `ab+c`, say, into `^ab+c$`, but it is indeed useful where the blob type is `CHR_BLOB`.)

For an explanation of the use of the word "blob", see "IndexedText.i6t".

```
[ IT_Replace_RE findtype indt findt rindt insens exactly
    cindt csize ilen i cl mpos cpos ch chm;
    ilen = IT_CharacterLength(indt);
```

```
IT_RE_Err = 0;
switch (findtype) {
    REGEXP_BLOB: i = IT_RE_CompileTree(findt, exactly);
    CHR_BLOB: i = IT_CHR_CompileTree(findt, exactly);
    default: "*** bad findtype ***";
}
if ((i<0) || (i>RE_MAX_PACKETS)) {
    IT_RE_Err = i;
    print "*** Regular expression error: ", (string) IT_RE_Err, " ***^";
    RunTimeProblem(RTP_REGEXPSYNTAXERROR);
    return 0;
}
if (IT_RE_Trace) {
    IT_RE_DebugTree(findt);
    print "(compiled to ", i, " packets)^";
}
if (findtype == REGEXP_BLOB) IT_RE_EmptyMatchVars();
mpos = 0; chm = 0; cpos = 0;
while (IT_RE_Parse(findt, indt, mpos, insens) >= 0) {
    chm++;
    if (IT_RE_Trace) {
        print "^*** Match ", chm, " found (", RE_PACKET_space-->RE_DATA1, ",",
            RE_PACKET_space-->RE_DATA2, "): ";
        if (RE_PACKET_space-->RE_DATA1 == RE_PACKET_space-->RE_DATA2) {
            print "<empty>";
        }
        for (i=RE_PACKET_space-->RE_DATA1:i<RE_PACKET_space-->RE_DATA2:i++) {
            print (char) BlkValueRead(indt, i);
        }
        print " ***^";
    }
    if (rindt == 0) break; ! Accept only one match, replace nothing
    if (rindt ~= 0 or 1) {
        if (chm == 1) {
            cindt = BlkValueCreate(INDEXED_TEXT_TY);
            csize = BlkValueExtent(cindt);
        }
        for (i=cpos:i<RE_PACKET_space-->RE_DATA1:i++) {
            ch = BlkValueRead(indt, i);
            if (cl+1 >= csize) {
                if (BlkValueSetExtent(cindt, 2*cl, 7) == false) break;
                csize = BlkValueExtent(cindt);
            }
            BlkValueWrite(cindt, cl++, ch);
        }
        BlkValueWrite(cindt, cl, 0);
        IT_Concatenate(cindt, rindt, findtype, indt);
        csize = BlkValueExtent(cindt);
        cl = IT_CharacterLength(cindt);
    }
    mpos = RE_PACKET_space-->RE_DATA2; cpos = mpos;
```

```
        if (RE_PACKET_space-->RE_DATA1 == RE_PACKET_space-->RE_DATA2)
            mpos++;
        if (IT_RE_Trace) {
            if (chm == 100) { ! Purely to keep the output from being excessive
                print "(Stopping after 100 matches.)^"; break;
            }
        }
    }
    if (chm > 0) {
        if (rindt ~= 0 or 1) {
            for (i=cpos:i<ilen:i++) {
                ch = BlkValueRead(indt, i);
                if (cl+1 >= csize) {
                    if (BlkValueSetExtent(cindt, 2*cl, 8) == false) break;
                    csize = BlkValueExtent(cindt);
                }
                BlkValueWrite(cindt, cl++, ch);
            }
        }
        if (findtype == REGEXP_BLOB) {
            IT_RE_CreateMatchVars(indt);
            if (IT_RE_Trace)
                IT_RE_DebugMatchVars(indt);
        }
        if (rindt ~= 0 or 1) {
            BlkValueWrite(cindt, cl, 0);
            BlkValueCopy(indt, cindt);
            BlkFree(cindt);
        }
    }
    return chm;
];
```

§**19. Concatenation.** See the corresponding routine in "IndexedText.i6t": this is a variation which handles the special syntaxes used in search-and-replace.

```
[ IT_RE_Concatenate indt_to indt_from blobtype indt_ref
    pos len ch i tosize x y case;
    if ((indt_to==0) || (BlkType(indt_to) ~= INDEXED_TEXT_TY)) rfalse;
    if ((indt_from==0) || (BlkType(indt_from) ~= INDEXED_TEXT_TY)) return indt_to;
    pos = IT_CharacterLength(indt_to);
    tosize = BlkValueExtent(indt_to);
    len = IT_CharacterLength(indt_from);
    for (i=0:i<len:i++) {
        ch = BlkValueRead(indt_from, i);
        if ((ch == '\') && (i < len-1)) {
            ch = BlkValueRead(indt_from, ++i);
            if (ch == 'n') ch = 10;
            if (ch == 't') ch = 9;
            case = -1;
            if (ch == 'l') case = 0;
            if (ch == 'u') case = 1;
            if (case >= 0) ch = BlkValueRead(indt_from, ++i);
            if ((ch >= '0') && (ch <= '9')) {
                ch = ch - '0';
                if (ch < RE_Subexpressions-->10) {
                    x = (RE_Subexpressions-->ch)-->RE_DATA1;
                    y = (RE_Subexpressions-->ch)-->RE_DATA2;
                    if (x >= 0) {
                         for (:x<y:x++) {
                             ch = BlkValueRead(indt_ref, x);
                             if (pos+1 >= tosize) {
                                 if (BlkValueSetExtent(indt_to, 2*tosize, 11) == false) break;
                                 tosize = BlkValueExtent(indt_to);
                             }
                             if (case >= 0)
                                 BlkValueWrite(indt_to, pos++, CharToCase(ch, case));
                             else
                                 BlkValueWrite(indt_to, pos++, ch);
                         }
                    }
                }
                continue;
            }
        }
        if (pos+1 >= tosize) {
            if (BlkValueSetExtent(indt_to, 2*tosize, 12) == false) break;
            tosize = BlkValueExtent(indt_to);
        }
        BlkValueWrite(indt_to, pos++, ch);
    }
    BlkValueWrite(indt_to, pos, 0);
    return indt_to;
];
```

§**20. Stubs.** This time, there are no stubs: if there are no indexed texts, none of these routines is ever referred to.

```
#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```

Char Template B/chart

Purpose

To decide whether letters are upper or lower case, and convert between the two.

§1. Char Is Of Case. The following decides whether a character `c` belongs to case `case`, where 0 means lower case and 1 means upper. `c` is interpreted according to the character casing chart in "UnicodeData.i6t", which means, it will be ZSCII for the Z-machine and Unicode for Glulx.

```
[ CharIsOfCase c case
    i tab min max len par;
    if (c<'A') rfalse;
    if (case == 0) {
        if ((c >= 'a') && (c <= 'z')) rtrue;
        tab = CharCasingChart0;
    } else {
        if ((c >= 'A') && (c <= 'Z')) rtrue;
        tab = CharCasingChart1;
    }
    if (c<128) rfalse;
    while (tab-->i) {
        min = tab-->i; i++;
        len = tab-->i; i++;
        i++;
        par = 0;
        if (len<0) { par = 1; len = -len; }
        if (c < min) rfalse;
        if (c < min+len) {
            if (par) { if ((c-min) % 2 == 0) rtrue; }
            else { rtrue; }
        }
    }
    rfalse;
];
```

§**2. Char To Case.** And the following converts character `c` to the desired case, or returns it unchanged if it is not a letter with variant casings.

```
[ CharToCase c case
    i tab min max len par del f;
    if (c<'A') return c;
    if (case == 1) {
        if ((c >= 'a') && (c <= 'z')) return c-32;
        tab = CharCasingChart0;
    } else {
        if ((c >= 'A') && (c <= 'Z')) return c+32;
        tab = CharCasingChart1;
    }
    if (c<128) return c;
    while (tab-->i) {
        min = tab-->i; i++;
        len = tab-->i; i++;
        del = tab-->i; i++;
        par = 0;
        if (len<0) { par = 1; len = -len; }
        if (c < min) return c;
        if (c < min+len) {
            f = false;
            if (par) { if ((c-min) % 2 == 0) f = true; }
            else { f = true; }
            if (f) {
                if (del == UNIC_NCT) return c;
                return c+del;
            }
        }
    }
    return c;
];
```

§**3. Reversing Case.** It's convenient to provide this relatively fast routine to reverse the case of a letter since this is an operation used frequently in regular expression matching (see "RegExp.i6t").

```
#IFDEF TARGET_ZCODE;
[ IT_RevCase ch;
    if (ch<'A') return ch;
    if ((ch >= 'a') && (ch <= 'z')) return ch-'a'+'A';
    if ((ch >= 'A') && (ch <= 'Z')) return ch-'A'+'a';
    if (ch<128) return ch;
    if ((ch >= 155) && (ch <= 157)) return ch+3; ! a, o, u umlaut in ZSCII
    if ((ch >= 158) && (ch <= 160)) return ch-3; ! A, O, U umlaut
    if ((ch >= 164) && (ch <= 165)) return ch+3; ! e, i umlaut
    if ((ch >= 167) && (ch <= 168)) return ch-3; ! E, I umlaut
    if ((ch >= 169) && (ch <= 174)) return ch+6; ! a, e, i, o, u, y acute
    if ((ch >= 175) && (ch <= 180)) return ch-6; ! A, E, I, O, U, Y acute
    if ((ch >= 181) && (ch <= 185)) return ch+5; ! a, e, i, o, u grave
    if ((ch >= 186) && (ch <= 190)) return ch-5; ! A, E, I, O, U grave
    if ((ch >= 191) && (ch <= 195)) return ch+5; ! a, e, i, o, u circumflex
    if ((ch >= 196) && (ch <= 200)) return ch-5; ! A, E, I, O, U circumflex
    if (ch == 201) return 202; ! a circle
    if (ch == 202) return 201; ! A circle
    if (ch == 203) return 204; ! o slash
    if (ch == 204) return 203; ! O slash
    if ((ch >= 205) && (ch <= 207)) return ch+3; ! a, n, o tilde
    if ((ch >= 208) && (ch <= 210)) return ch-3; ! A, N, O tilde
    if (ch == 211) return 212; ! ae ligature
    if (ch == 212) return 211; ! AE ligature
    if (ch == 213) return 214; ! c cedilla
    if (ch == 214) return 213; ! C cedilla
    if (ch == 215 or 216) return ch+2; ! thorn, eth
    if (ch == 217 or 218) return ch-2; ! Thorn, Eth
    if (ch == 220) return 221; ! oe ligature
    if (ch == 221) return 220; ! OE ligature
    return ch;
];
#IFNOT;
[ IT_RevCase ch;
    if (ch<'A') return ch;
    if ((ch >= 'a') && (ch <= 'z')) return ch-'a'+'A';
    if ((ch >= 'A') && (ch <= 'Z')) return ch-'A'+'a';
    if (ch<128) return ch;
    if (CharIsOfCase(ch, 0)) return CharToCase(ch, 1);
    if (CharIsOfCase(ch, 1)) return CharToCase(ch, 0);
    return ch;
];
#ENDIF;
```

§**4. Testing.** Not actually used: simply for testing the tables.

```
[ CharTestCases case i j;
    for (i=32: i<$E0; i++) {
        if ((i>=127) && (i<155)) continue;
        print i, " - ", (char) i, " -";
        if (CharIsOfCase(i, 0)) print "  lower";
        if (CharIsOfCase(i, 1)) print "  upper";
        j = CharToCase(i, 0); if (j ~= i) print "  tolower: ", (char) j;
        j = CharToCase(i, 1); if (j ~= i) print "  toupper: ", (char) j;
        print "^";
    }
];
```

UnicodeData Template B/unict

Purpose

To tabulate casings in the character set.

§**1. Source.** When this section is included, exactly one of the following constants is defined:

(a) `ZSCII_Tables`, meaning that we will use ZSCII as the character set for characters in indexed text strings.
(b) `Small_Unicode_Tables`, meaning that we will use Unicode but store only single-byte characters, so that only the codes 0 to 255 are valid: in effect ISO Latin-1.
(c) `Large_Unicode_Tables`, meaning that we will use Unicode and store two-byte characters, so that all of Unicode in the range 0 to 65535 are valid.

Whichever is defined, we must create two arrays:

(i) `CharCasingChart0`, a table indicating lower-case letters with transitions to convert them to upper case;
(ii) `CharCasingChart1`, vice versa.

Each array is a sequence of three-word records, consisting of the start of a character range, the size of the range (the number of characters in it), and the numerical offset to convert to the opposite case. For instance, the sequence $(97, 26, -32)$ means the 26 lower-case letters "a" to "z", and marks them as convertible to upper case by subtracting 32 from the character code (so "a", 97, becomes "A", 65). If the size of the range is negative, this indicates that only every alternate code is valid. (This makes for efficient storage since there are large parts of the Unicode number-space in which upper and lower case letters alternate.)

An offset of `UNIC_NCT` means no case change is possible; and any character not included in the ranges below is not a letter.

```
Constant UNIC_NCT = 10000; ! Safe as highest case-change delta is 8383
```

§**2. ZSCII Casing Tables.**

```
#IFDEF ZSCII_Tables;
Array CharCasingChart0 -->
    $0061 (  26) (     -32) $009b (   3) (       3) $00a1 (   1) (UNIC_NCT)
    $00a4 (   2) (       3) $00a6 (   1) (UNIC_NCT) $00a9 (   6) (       6)
    $00b5 (   5) (       5) $00bf (   5) (       5) $00c9 (  -3) (       1)
    $00cd (   3) (       3) $00d3 (  -3) (       1) $00d7 (   2) (       2)
    $00dc (   1) (       1) $0000
;
Array CharCasingChart1 -->
    $0041 (  26) (      32) $009e (   3) (      -3) $00a7 (   2) (      -3)
    $00af (   6) (      -6) $00ba (   5) (      -5) $00c4 (   5) (      -5)
    $00ca (  -3) (      -1) $00d0 (   3) (      -3) $00d4 (  -3) (      -1)
    $00d9 (   2) (      -2) $00dd (   1) (      -1) $0000
;
#ENDIF; ! ZSCII_Tables
```

§3. Small Unicode Casing Tables.

```
#IFDEF Small_Unicode_Tables;
Array CharCasingChart0 -->
    $0061 (  26) (     -32) $00aa (   1) (UNIC_NCT) $00b5 (   1) (UNIC_NCT) $00ba (   1) (UNIC_NCT)
    $00df (   1) (UNIC_NCT) $00e0 (  23) (     -32) $00f8 (   7) (     -32) $00ff (   1) (UNIC_NCT)
    $0000
;
Array CharCasingChart1 -->
    $0041 (  26) (      32) $00c0 (  23) (      32) $00d8 (   7) (      32) $0000
;
#ENDIF; ! Small_Unicode_Tables
```

§4. Large Unicode Casing Tables.

```
#IFDEF Large_Unicode_Tables;
Array CharCasingChart0 -->
    $0061 (  26) (     -32) $00aa (   1) (UNIC_NCT) $00b5 (   1) (     743) $00ba (   1) (UNIC_NCT)
    $00df (   1) (UNIC_NCT) $00e0 (  23) (     -32) $00f8 (   7) (     -32) $00ff (   1) (     121)
    $0101 ( -47) (      -1) $0131 (   1) (    -232) $0133 (  -5) (      -1) $0138 (   1) (UNIC_NCT)
    $013a ( -15) (      -1) $0149 (   1) (UNIC_NCT) $014b ( -45) (      -1) $017a (  -5) (      -1)
    $017f (   1) (    -300) $0180 (   1) (UNIC_NCT) $0183 (  -3) (      -1) $0188 (   1) (      -1)
    $018c (   1) (      -1) $018d (   1) (UNIC_NCT) $0192 (   1) (      -1) $0195 (   1) (      97)
    $0199 (   1) (      -1) $019a (   2) (UNIC_NCT) $019e (   1) (     130) $01a1 (  -5) (      -1)
    $01a8 (   1) (      -1) $01aa (   2) (UNIC_NCT) $01ad (   1) (      -1) $01b0 (   1) (      -1)
    $01b4 (  -3) (      -1) $01b9 (   1) (      -1) $01ba (   1) (UNIC_NCT) $01bd (   1) (      -1)
    $01be (   1) (UNIC_NCT) $01bf (   1) (      56) $01c6 (   1) (      -2) $01c9 (   1) (      -2)
    $01cc (   1) (      -2) $01ce ( -15) (      -1) $01dd (   1) (     -79) $01df ( -17) (      -1)
    $01f0 (   1) (UNIC_NCT) $01f3 (   1) (      -2) $01f5 (   1) (      -1) $01f9 ( -39) (      -1)
    $0221 (   1) (UNIC_NCT) $0223 ( -17) (      -1) $0234 (   3) (UNIC_NCT) $0250 (   3) (UNIC_NCT)
    $0253 (   1) (    -210) $0254 (   1) (    -206) $0255 (   1) (UNIC_NCT) $0256 (   2) (    -205)
    $0258 (   1) (UNIC_NCT) $0259 (   1) (    -202) $025a (   1) (UNIC_NCT) $025b (   1) (    -203)
    $025c (   4) (UNIC_NCT) $0260 (   1) (    -205) $0261 (   2) (UNIC_NCT) $0263 (   1) (    -207)
    $0264 (   4) (UNIC_NCT) $0268 (   1) (    -209) $0269 (   1) (    -211) $026a (   5) (UNIC_NCT)
    $026f (   1) (    -211) $0270 (   2) (UNIC_NCT) $0272 (   1) (    -213) $0273 (   2) (UNIC_NCT)
    $0275 (   1) (    -214) $0276 (  10) (UNIC_NCT) $0280 (   1) (    -218) $0281 (   2) (UNIC_NCT)
    $0283 (   1) (    -218) $0284 (   4) (UNIC_NCT) $0288 (   1) (    -218) $0289 (   1) (UNIC_NCT)
    $028a (   2) (    -217) $028c (   6) (UNIC_NCT) $0292 (   1) (    -219) $0293 (  29) (UNIC_NCT)
    $0390 (   1) (UNIC_NCT) $03ac (   1) (     -38) $03ad (   3) (     -37) $03b0 (   1) (UNIC_NCT)
    $03b1 (  17) (     -32) $03c2 (   1) (     -31) $03c3 (   9) (     -32) $03cc (   1) (     -64)
    $03cd (   2) (     -63) $03d0 (   1) (     -62) $03d1 (   1) (     -57) $03d5 (   1) (     -47)
    $03d6 (   1) (     -54) $03d7 (   1) (UNIC_NCT) $03d9 ( -23) (      -1) $03f0 (   1) (     -86)
    $03f1 (   1) (     -80) $03f2 (   1) (       7) $03f3 (   1) (UNIC_NCT) $03f5 (   1) (     -96)
    $03f8 (   1) (      -1) $03fb (   1) (      -1) $0430 (  32) (     -32) $0450 (  16) (     -80)
    $0461 ( -33) (      -1) $048b ( -53) (      -1) $04c2 ( -13) (      -1) $04d1 ( -37) (      -1)
    $04f9 (   1) (      -1) $0501 ( -15) (      -1) $0561 (  38) (     -48) $0587 (   1) (UNIC_NCT)
    $1d00 (  44) (UNIC_NCT) $1d62 (  10) (UNIC_NCT) $1e01 (-149) (      -1) $1e96 (   5) (UNIC_NCT)
    $1e9b (   1) (     -59) $1ea1 ( -89) (      -1) $1f00 (   8) (       8) $1f10 (   6) (       8)
    $1f20 (   8) (       8) $1f30 (   8) (       8) $1f40 (   6) (       8) $1f50 (   1) (UNIC_NCT)
    $1f51 (   1) (       8) $1f52 (   1) (UNIC_NCT) $1f53 (   1) (       8) $1f54 (   1) (UNIC_NCT)
    $1f55 (   1) (       8) $1f56 (   1) (UNIC_NCT) $1f57 (   1) (       8) $1f60 (   8) (       8)
    $1f70 (   2) (      74) $1f72 (   4) (      86) $1f76 (   2) (     100) $1f78 (   2) (     128)
    $1f7a (   2) (     112) $1f7c (   2) (     126) $1f80 (   8) (       8) $1f90 (   8) (       8)
    $1fa0 (   8) (       8) $1fb0 (   2) (       8) $1fb2 (   1) (UNIC_NCT) $1fb3 (   1) (       9)
    $1fb4 (  -3) (UNIC_NCT) $1fb7 (   1) (UNIC_NCT) $1fbe (   1) (   -7205) $1fc2 (   1) (UNIC_NCT)
    $1fc3 (   1) (       9) $1fc4 (  -3) (UNIC_NCT) $1fc7 (   1) (UNIC_NCT) $1fd0 (   2) (       8)
    $1fd2 (   2) (UNIC_NCT) $1fd6 (   2) (UNIC_NCT) $1fe0 (   2) (       8) $1fe2 (   3) (UNIC_NCT)
    $1fe5 (   1) (       7) $1fe6 (   2) (UNIC_NCT) $1ff2 (   1) (UNIC_NCT) $1ff3 (   1) (       9)
    $1ff4 (  -3) (UNIC_NCT) $1ff7 (   1) (UNIC_NCT) $2071 (   1) (UNIC_NCT) $207f (   1) (UNIC_NCT)
    $210a (   1) (UNIC_NCT) $210e (   2) (UNIC_NCT) $2113 (   1) (UNIC_NCT) $212f (   1) (UNIC_NCT)
    $2134 (   1) (UNIC_NCT) $2139 (   1) (UNIC_NCT) $213d (   1) (UNIC_NCT) $2146 (   4) (UNIC_NCT)
    $fb00 (   7) (UNIC_NCT) $fb13 (   5) (UNIC_NCT) $ff41 (  26) (     -32) $0000
;
Array CharCasingChart1 -->
    $0041 (  26) (      32) $00c0 (  23) (      32) $00d8 (   7) (      32) $0100 ( -47) (       1)
    $0130 (   1) (    -199) $0132 (  -5) (       1) $0139 ( -15) (       1) $014a ( -45) (       1)
    $0178 (   1) (    -121) $0179 (  -5) (       1) $0181 (   1) (     210) $0182 (  -3) (       1)
    $0186 (   1) (     206) $0187 (   1) (       1) $0189 (   2) (     205) $018b (   1) (       1)
    $018e (   1) (      79) $018f (   1) (     202) $0190 (   1) (     203) $0191 (   1) (       1)
    $0193 (   1) (     205) $0194 (   1) (     207) $0196 (   1) (     211) $0197 (   1) (     209)
    $0198 (   1) (       1) $019c (   1) (     211) $019d (   1) (     213) $019f (   1) (     214)
    $01a0 (  -5) (       1) $01a6 (   1) (     218) $01a7 (   1) (       1) $01a9 (   1) (     218)
    $01ac (   1) (       1) $01ae (   1) (     218) $01af (   1) (       1) $01b1 (   2) (     217)
    $01b3 (  -3) (       1) $01b7 (   1) (     219) $01b8 (   1) (       1) $01bc (   1) (       1)
    $01c4 (   1) (       2) $01c7 (   1) (       2) $01ca (   1) (       2) $01cd ( -15) (       1)
    $01de ( -17) (       1) $01f1 (   1) (       2) $01f4 (   1) (       1) $01f6 (   1) (     -97)
    $01f7 (   1) (     -56) $01f8 ( -39) (       1) $0220 (   1) (    -130) $0222 ( -17) (       1)
    $0386 (   1) (      38) $0388 (   3) (      37) $038c (   1) (      64) $038e (   2) (      63)
    $0391 (  17) (      32) $03a3 (   9) (      32) $03d2 (   3) (UNIC_NCT) $03d8 ( -23) (       1)
    $03f4 (   1) (     -60) $03f7 (   1) (       1) $03f9 (   1) (      -7) $03fa (   1) (       1)
    $0400 (  16) (      80) $0410 (  32) (      32) $0460 ( -33) (       1) $048a ( -53) (       1)
    $04c0 (   1) (UNIC_NCT) $04c1 ( -13) (       1) $04d0 ( -37) (       1) $04f8 (   1) (       1)
    $0500 ( -15) (       1) $0531 (  38) (      48) $10a0 (  38) (UNIC_NCT) $1e00 (-149) (       1)
    $1ea0 ( -89) (       1) $1f08 (   8) (      -8) $1f18 (   6) (      -8) $1f28 (   8) (      -8)
    $1f38 (   8) (      -8) $1f48 (   6) (      -8) $1f59 (  -7) (      -8) $1f68 (   8) (      -8)
    $1fb8 (   2) (      -8) $1fba (   2) (     -74) $1fc8 (   4) (     -86) $1fd8 (   2) (      -8)
    $1fda (   2) (    -100) $1fe8 (   2) (      -8) $1fea (   2) (    -112) $1fec (   1) (      -7)
    $1ff8 (   2) (    -128) $1ffa (   2) (    -126) $2102 (   1) (UNIC_NCT) $2107 (   1) (UNIC_NCT)
    $210b (   3) (UNIC_NCT) $2110 (   3) (UNIC_NCT) $2115 (   1) (UNIC_NCT) $2119 (   5) (UNIC_NCT)
    $2124 (   1) (UNIC_NCT) $2126 (   1) (   -7517) $2128 (   1) (UNIC_NCT) $212a (   1) (   -8383)
    $212b (   1) (   -8262) $212c (   2) (UNIC_NCT) $2130 (   2) (UNIC_NCT) $2133 (   1) (UNIC_NCT)
    $213e (   2) (UNIC_NCT) $2145 (   1) (UNIC_NCT) $ff21 (  26) (      32) $0000
;
#ENDIF; ! Large_Unicode_Tables
```

StoredAction Template B/stact

Purpose

Code to support the stored action kind of value.

§**1. Head.** As ever: if there is no heap, there are no stored actions.

```
#IFDEF MEMORY_HEAP_SIZE; ! Will exist if any use is made of indexed texts
```

§**2. KOV Support.** See the "BlockValues.i6t" segment for the specification of the following routines.

```
[ STORED_ACTION_TY_Support task arg1 arg2 arg3;
    switch(task) {
        CREATE_KOVS:     return STORED_ACTION_TY_Create();
        CAST_KOVS:       rfalse;
        DESTROY_KOVS:    return STORED_ACTION_TY_Destroy(arg1);
        PRECOPY_KOVS:    rfalse;
        COPY_KOVS:       return STORED_ACTION_TY_Copy(arg1, arg2);
        COMPARE_KOVS:    return STORED_ACTION_TY_Compare(arg1, arg2);
        READ_FILE_KOVS:  rfalse;
        WRITE_FILE_KOVS: rfalse;
        HASH_KOVS:       return STORED_ACTION_TY_Hash(arg1);
    }
];
```

§**3. Creation.** A stored action block has fixed size, so this is a single-block KOV: its data consists of six words, laid out as shown in the following routine. Note that it initialises to the default value for this KOV, an action in which the player waits.

An action which involves a topic – such as the one produced by the command LOOK UP JIM MCDIVITT IN ENCYCLOPAEDIA – cannot be tried without the text of that topic (JIM MCDIVITT) being available. That's no problem if the action is tried in the same turn in which it is generated, because the text will still be in the command buffer. But once we store actions up for future use it becomes an issue. So when we store an action involving a topic, we record the actual text typed at the time when it is stored, and this goes into array entry 5 of the block. Because that in turn is indexed text, and therefore a block value on the heap in its own right, we have to be a little more careful about destroying and copying stored actions than we otherwise would be.

Note that entries 1 and 2 are values whose kind depends on the action in entry 0: but they are never block values, because actions are not allowed to apply to block values. This simplifies matters considerably.

```
[ STORED_ACTION_TY_Create stora;
    stora = BlkAllocate(6*WORDSIZE, STORED_ACTION_TY, BLK_FLAG_WORD);
    BlkValueWrite(stora, 0, ##Wait); ! action
    BlkValueWrite(stora, 1, 0); ! noun
    BlkValueWrite(stora, 2, 0); ! second
    BlkValueWrite(stora, 3, player); ! actor
    BlkValueWrite(stora, 4, false); ! whether a request
    BlkValueWrite(stora, 5, 0); ! indexed text of command if necessary, 0 if not
    return stora;
];
```

§**4. Setting Up.** In practice it's convenient for NI to have a routine which creates a stored action with a given slate of action variables, rather than have to set them all one at a time, so the following is provided as a shorthand form.

```
[ STORED_ACTION_TY_New a n s ac req  stora;
    if (stora == 0) stora = STORED_ACTION_TY_Create();
    BlkValueWrite(stora, 0, a);
    BlkValueWrite(stora, 1, n);
    BlkValueWrite(stora, 2, s);
    BlkValueWrite(stora, 3, ac);
    BlkValueWrite(stora, 4, req);
    BlkValueWrite(stora, 5, 0);
    return stora;
];
```

§**5. Destruction.** Entries 0 to 4 are forgettable non-block values: only the optional indexed text requires destruction.

```
[ STORED_ACTION_TY_Destroy stora toc;
    toc = BlkValueRead(stora, 5);
    if (toc) BlkFree(toc);
];
```

§**6. Copying.** The only entry needing attention is, again, entry 5: if this is non-zero in the source, then we need to create a new indexed text block to hold a duplicate copy of the text.

```
[ STORED_ACTION_TY_Copy storato storafrom tocfrom tocto;
    tocfrom = BlkValueRead(storafrom, 5);
    if (tocfrom == 0) return;
    tocto = INDEXED_TEXT_TY_Support(CREATE_KOVS);
    BlkValueCopy(tocto, tocfrom);
    BlkValueWrite(storato, 5, tocto);
];
```

§7. Comparison. There is no very convincing ordering on stored actions, but we need to devise a comparison which will exhaustively determine whether two actions are or are not different.

```
[ STORED_ACTION_TY_Compare storaleft storaright delta itleft itright;
    delta = BlkValueRead(storaleft, 0) - BlkValueRead(storaright, 0);
    if (delta) return delta;
    delta = BlkValueRead(storaleft, 1) - BlkValueRead(storaright, 1);
    if (delta) return delta;
    delta = BlkValueRead(storaleft, 2) - BlkValueRead(storaright, 2);
    if (delta) return delta;
    delta = BlkValueRead(storaleft, 3) - BlkValueRead(storaright, 3);
    if (delta) return delta;
    delta = BlkValueRead(storaleft, 4) - BlkValueRead(storaright, 4);
    if (delta) return delta;
    itleft = BlkValueRead(storaleft, 5);
    itright = BlkValueRead(storaright, 5);
    if ((itleft ~= 0) && (itright ~= 0))
        return INDEXED_TEXT_TY_Support(COMPARE_KOVS, itleft, itright);
    return itleft - itright;
];
[ STORED_ACTION_TY_Distinguish stora1 stora2;
    if (STORED_ACTION_TY_Compare(stora1, stora2) == 0) rfalse;
    rtrue;
];
```

§8. Hashing.

```
[ STORED_ACTION_TY_Hash stora  rv it;
    rv = BlkValueRead(stora, 0);
    rv = rv * 33 + BlkValueRead(stora, 1);
    rv = rv * 33 + BlkValueRead(stora, 2);
    rv = rv * 33 + BlkValueRead(stora, 3);
    rv = rv * 33 + BlkValueRead(stora, 4);
    it = BlkValueRead(stora, 5);
    if (it ~= 0)
        rv = rv * 33 + INDEXED_TEXT_TY_Support(HASH_KOVS, it);
    return rv;
];
```

§**9. Printing.** We share some code here with the routines originally written for the ACTIONS testing command. (The `DB` in `DB_Action` stands for "debugging".) When printing a topic, it prints the relevant words from the player's command: so if our stored action is one which contains an entry 5, then we have to temporarily adopt this as the player's command, and restore the old player's command once printing is done. To do this, we need to save the old player's command, and we do that by creating an indexed text for the duration.

```
[ STORED_ACTION_TY_Say stora text_of_command saved_command saved_pn saved_action K1 K2 at;
    if ((stora==0) || (BlkType(stora) ~= STORED_ACTION_TY)) return;
    text_of_command = BlkValueRead(stora, 5);
    if (text_of_command) {
        saved_command = INDEXED_TEXT_TY_Support(CREATE_KOVS);
        INDEXED_TEXT_TY_Support(CAST_KOVS, players_command, SNIPPET_TY, saved_command);
        SetPlayersCommand(text_of_command);
    }
    saved_pn = parsed_number; saved_action = action;
    action = BlkValueRead(stora, 0);
    at = FindAction(-1);
    K1 = ActionData-->(at+AD_NOUN_KOV);
    K2 = ActionData-->(at+AD_SECOND_KOV);
    if (K1 ~= OBJECT_TY) {
        parsed_number = BlkValueRead(stora, 1);
        if ((K1 == UNDERSTANDING_TY) && (text_of_command == 0)) {
            if (saved_command == 0) saved_command = INDEXED_TEXT_TY_Create();
            INDEXED_TEXT_TY_Cast(players_command, SNIPPET_TY, saved_command);
            text_of_command = INDEXED_TEXT_TY_Create();
            INDEXED_TEXT_TY_Cast(parsed_number, TEXT_TY, text_of_command);
            SetPlayersCommand(text_of_command);
            parsed_number = players_command;
        }
    }
    if (K2 ~= OBJECT_TY) {
        parsed_number = BlkValueRead(stora, 2);
        if ((K2 == UNDERSTANDING_TY) && (text_of_command == 0)) {
            if (saved_command == 0) saved_command = INDEXED_TEXT_TY_Create();
            INDEXED_TEXT_TY_Cast(players_command, SNIPPET_TY, saved_command);
            text_of_command = INDEXED_TEXT_TY_Create();
            INDEXED_TEXT_TY_Cast(parsed_number, TEXT_TY, text_of_command);
            SetPlayersCommand(text_of_command);
            parsed_number = players_command;
        }
    }
    DB_Action(BlkValueRead(stora, 3), BlkValueRead(stora, 4), BlkValueRead(stora, 0),
        BlkValueRead(stora, 1), BlkValueRead(stora, 2), true);
    parsed_number = saved_pn; action = saved_action;
    if (text_of_command) {
        SetPlayersCommand(saved_command);
        BlkFree(saved_command);
    }
];
```

§10. Involvement. That completes the compulsory services required for this KOV to function: from here on, the remaining routines provide definitions of stored action-related phrases in the Standard Rules.

An action "involves" an object if it appears as either the actor or the first or second noun.

```
[ STORED_ACTION_TY_Involves stora item at;
    at = FindAction(BlkValueRead(stora, 0));
    if (at) {
        if ((ActionData-->(at+AD_NOUN_KOV) == OBJECT_TY) &&
            (BlkValueRead(stora, 1) == item)) rtrue;
        if ((ActionData-->(at+AD_SECOND_KOV) == OBJECT_TY) &&
            (BlkValueRead(stora, 2) == item)) rtrue;
    }
    if (BlkValueRead(stora, 3) == item) rtrue;
    rfalse;
];
```

§11. Nouns. Extracting the noun or second noun from an action is a delicate business because simply returning the values in entries 1 and 2 would not be type-safe; it would fail to be an object if the stored action did not apply to objects. So the following returns `nothing` if requested to produce noun or second noun for such an action.

```
[ STORED_ACTION_TY_Part stora ind at ado;
    if (ind == 1 or 2) {
        if (ind == 1) ado = AD_NOUN_KOV; else ado = AD_SECOND_KOV;
        at = FindAction(BlkValueRead(stora, 0));
        if ((at) && (ActionData-->(at+ado) == OBJECT_TY)) return BlkValueRead(stora, ind);
        return nothing;
    }
    return BlkValueRead(stora, ind);
];
```

§12. Pattern Matching. In order to apply an action pattern such as "doing something with the kazoo" to a stored action, it needs to be the current action, because the code which compiles conditions like this looks at the `action`, `noun`, ..., variables. We don't want to do anything as disruptive as temporarily starting the stored action and then halting it again, so instead we simply "adopt" it, saving the slate of action variables and setting them from the stored action: almost immediately after – the moment the condition has been tested – we "unadopt" it again, restoring the stored values. Since the action pattern cannot itself refer to a stored action, the following code won't be nested, and we don't need to worry about stacking up saved copies of the action variables.

`SAT_Tmp-->0` stores the outcome of the condition, and is set in code compiled by NI.

```
Array SAT_Tmp-->7;
[ STORED_ACTION_TY_Adopt stora at;
    SAT_Tmp-->1 = action;
    SAT_Tmp-->2 = noun;
    SAT_Tmp-->3 = second;
    SAT_Tmp-->4 = actor;
    SAT_Tmp-->5 = act_requester;
    SAT_Tmp-->6 = parsed_number;
    action = BlkValueRead(stora, 0);
    at = FindAction(-1);
    if (ActionData-->(at+AD_NOUN_KOV) == OBJECT_TY) noun = BlkValueRead(stora, 1);
    else { parsed_number = BlkValueRead(stora, 1); noun = nothing; }
    if (ActionData-->(at+AD_SECOND_KOV) == OBJECT_TY) second = BlkValueRead(stora, 2);
    else { parsed_number = BlkValueRead(stora, 2); second = nothing; }
    actor = BlkValueRead(stora, 3);
    if (BlkValueRead(stora, 4)) act_requester = player; else act_requester = nothing;
];
[ STORED_ACTION_TY_Unadopt;
    action = SAT_Tmp-->1;
    noun = SAT_Tmp-->2;
    second = SAT_Tmp-->3;
    actor = SAT_Tmp-->4;
    act_requester = SAT_Tmp-->5;
    parsed_number = SAT_Tmp-->6;
    return SAT_Tmp-->0;
];
```

§**13. Current Action.** Although we never cast other values to stored actions, because none of them really imply an action (not even an action name, since that gives no help as to what the nouns might be), there is of course one action almost always present within a story file at run-time, even if it is not a single value as such: the action which is currently running. The following routine translates that into a stored action – thus allowing us to store it.

This is the place where we look to see if the action applies to a topic as either its noun or second noun, and if it does, we copy the player's command into an indexed text block-value in entry 5.

```
[ STORED_ACTION_TY_Current stora at text_of_command;
    if ((stora==0) || (BlkType(stora) ~= STORED_ACTION_TY)) return 0;
    BlkValueWrite(stora, 0, action);
    at = FindAction(-1);
    if (ActionData-->(at+AD_NOUN_KOV) == OBJECT_TY) BlkValueWrite(stora, 1, noun);
    else BlkValueWrite(stora, 1, parsed_number);
    if (ActionData-->(at+AD_SECOND_KOV) == OBJECT_TY) BlkValueWrite(stora, 2, second);
    else BlkValueWrite(stora, 2, parsed_number);
    BlkValueWrite(stora, 3, actor);
    if (act_requester) BlkValueWrite(stora, 4, true); else BlkValueWrite(stora, 4, false);
    if ((at) && ((ActionData-->(at+AD_NOUN_KOV) == UNDERSTANDING_TY) ||
            (ActionData-->(at+AD_SECOND_KOV) == UNDERSTANDING_TY))) {
        text_of_command = BlkValueRead(stora, 5);
        if (text_of_command == 0) {
            text_of_command = INDEXED_TEXT_TY_Support(CREATE_KOVS);
            BlkValueWrite(stora, 5, text_of_command);
        }
        INDEXED_TEXT_TY_Support(CAST_KOVS, players_command, SNIPPET_TY, text_of_command);
    } else BlkValueWrite(stora, 5, 0);
    return stora;
];
```

§**14. Trying.** Finally: having stored an action for perhaps many turns, we now let it happen, either silently or not.

```
[ STORED_ACTION_TY_Try stora ks  text_of_command saved_command;
    if ((stora==0) || (BlkType(stora) ~= STORED_ACTION_TY)) return;
    if (ks) { @push keep_silent; keep_silent=1; }
    text_of_command = BlkValueRead(stora, 5);
    if (text_of_command) {
        saved_command = INDEXED_TEXT_TY_Support(CREATE_KOVS);
        INDEXED_TEXT_TY_Support(CAST_KOVS, players_command, SNIPPET_TY, saved_command);
        SetPlayersCommand(text_of_command);
    }
    TryAction(BlkValueRead(stora, 4), BlkValueRead(stora, 3),
        BlkValueRead(stora, 0), BlkValueRead(stora, 1), BlkValueRead(stora, 2));
    if (text_of_command) {
        SetPlayersCommand(saved_command);
        BlkFree(saved_command);
    }
    if (ks) { @pull keep_silent; }
];
```

§**15. Stubs.** And the usual meaningless versions to ensure that function-names exist if there is no heap, and there are no stored actions anyway.

```
#IFNOT; ! IFDEF MEMORY_HEAP_SIZE
[ STORED_ACTION_TY_Say stora; ];
[ STORED_ACTION_TY_New a n s ac req stora; return false; ];
[ STORED_ACTION_TY_Support t a b c; rfalse; ];
#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```

Lists Template B/listt

Purpose

Code to support the list of... kind of value constructor.

§**1. Head.** As ever: if there is no heap, there are no lists (in this sense).

```
#IFDEF MEMORY_HEAP_SIZE; ! Will exist if any use is made of heap
```

§**2. KOV Support.** See the "BlockValues.i6t" segment for the specification of the following routines.

```
[ LIST_OF_TY_Support task arg1 arg2 arg3;
    switch(task) {
        CREATE_KOVS:     arg3 = LIST_OF_TY_Create(arg2);
                         if (arg1) LIST_OF_TY_CopyRawArray(arg3, arg1, 2, 0);
                         return arg3;
        CAST_KOVS:       rfalse;
        DESTROY_KOVS:    return LIST_OF_TY_Destroy(arg1);
        PRECOPY_KOVS:    return LIST_OF_TY_PreCopy(arg1, arg2);
        COPY_KOVS:       return LIST_OF_TY_Copy(arg1, arg2);
        COMPARE_KOVS:    return LIST_OF_TY_Compare(arg1, arg2);
        READ_FILE_KOVS:  rfalse;
        WRITE_FILE_KOVS: rfalse;
        HASH_KOVS:       return LIST_OF_TY_Hash(arg1);
    }
];
```

§**3. Creation.** A list is a multiple-block value with word-sized entries: the first few entries of the block are used for details about the list; the items in the the list then follow. Thus, to convert an item index to an array entry index, add `LIST_ITEM_BASE`.

Lists are by default created empty but in a block-value with enough capacity to hold 26 items, this being what's left in a 32-word block once all overheads are taken care of: 4 words are consumed by the header, then 2 more by the list metadata entries below.

```
Constant LIST_ITEM_KOV_F = 0; ! Entry 0: the kind of the list
Constant LIST_LENGTH_F = 1; ! Entry 1: length, i.e., number of items
Constant LIST_ITEM_BASE = 2; ! List items begin at this entry
```

```
[ LIST_OF_TY_Create skov list;
    skov = KindBaseTerm(skov, 0);
    list = BlkAllocate(28*WORDSIZE, LIST_OF_TY, BLK_FLAG_MULTIPLE + BLK_FLAG_WORD);
    BlkValueWrite(list, LIST_ITEM_KOV_F, skov);
    BlkValueWrite(list, LIST_LENGTH_F, 0);
    return list;
];
```

§4. Setting Up. NI needs to compile code which will create constant lists such as `{1, 4, 9}` at run-time: the following routine is convenient for that. A "raw array" in this routine's sense is an array `raw` such that `raw-->2` contains the number of items, `raw-->1` the kind of value, and `raw-->3` onwards are the items themselves.

```
[ LIST_OF_TY_CopyRawArray list arr rea cast len i ex bk v w;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return false;
    ex = BlkValueExtent(list);
    len = arr-->2;
    if ((len+LIST_ITEM_BASE > ex) &&
        (BlkValueSetExtent(list, len+LIST_ITEM_BASE) == false)) return 0;
    BlkValueWrite(list, LIST_LENGTH_F, len);
    if (rea == 2) bk = BlkValueRead(list, LIST_ITEM_KOV_F);
    else {
        bk = arr-->1;
        BlkValueWrite(list, LIST_ITEM_KOV_F, bk);
    }
    for (i=0:i<len:i++) {
        v = arr-->(i+3);
        if (KindAtomic(bk) == LIST_OF_TY) {
            w = LIST_OF_TY_Create(v-->1);
            LIST_OF_TY_CopyRawArray(w, v, 0, KindBaseTerm(bk, 0));
            BlkValueWrite(list, i+LIST_ITEM_BASE, w);
        } else {
            if ((cast) && (cast ~= bk)) {
                if (KOVIsBlockValue(cast)) v = BlkValueCreate(cast, v, bk);
            } else {
                if (KOVIsBlockValue(bk)) v = BlkValueCreate(bk, v);
            }
            BlkValueWrite(list, i+LIST_ITEM_BASE, v);
        }
    }
    if ((cast) && (cast ~= bk)) BlkValueWrite(list, LIST_ITEM_KOV_F, cast);
    #ifdef SHOW_ALLOCATIONS;
    print "Copied raw array to list: "; LIST_OF_TY_Say(list, 1); print "^";
    #endif;
    return list;
];
```

§5. Destruction. If the list items are themselves block-values, they must all be freed before the list itself can be freed.

```
[ LIST_OF_TY_Destroy list no_items i;
    if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F))) {
        no_items = BlkValueRead(list, LIST_LENGTH_F);
        for (i=0; i<no_items; i++) BlkFree(BlkValueRead(list, i+LIST_ITEM_BASE));
    }
    return list;
];
```

§6. Copying. Again, if the list contains block-values then they must be duplicated rather than bitwise copied as pointers.

Note that we use the pre-copy stage to remember the kind of value stored in the list. Type-checking ordinarily means that one list can only be copied into another if they have the same kind of contents: but there is one exception, which is when the list being copied is the empty list.

```
Global precopied_list_kov;

[ LIST_OF_TY_PreCopy lto lfrom list;
    precopied_list_kov = BlkValueRead(lto, LIST_ITEM_KOV_F);
];

[ LIST_OF_TY_Copy lto lfrom list no_items i nv bk val splk;
    no_items = BlkValueRead(lfrom, LIST_LENGTH_F);
    bk = BlkValueRead(lfrom, LIST_ITEM_KOV_F);
    if (precopied_list_kov ~= 0 or 1) BlkValueWrite(lto, LIST_ITEM_KOV_F, precopied_list_kov);
    else BlkValueWrite(lto, LIST_ITEM_KOV_F, bk);
    if ((precopied_list_kov == INDEXED_TEXT_TY) && (bk == TEXT_TY)) {
        for (i=0; i<no_items; i++) {
            nv = BlkValueCreate(INDEXED_TEXT_TY);
            INDEXED_TEXT_TY_Cast(BlkValueRead(lfrom, i+LIST_ITEM_BASE), TEXT_TY, nv);
            BlkValueWrite(lto, i+LIST_ITEM_BASE, nv);
        }
    } else {
        if (KOVIsBlockValue(bk)) {
            for (i=0; i<no_items; i++) {
                val = BlkValueRead(lfrom, i+LIST_ITEM_BASE);
                nv = BlkValueCreate(BlkType(val));
                BlkValueCopy(nv, val);
                BlkValueWrite(lto, i+LIST_ITEM_BASE, nv);
            }
        }
    }
    precopied_list_kov = 0;
];
```

§7. Comparison. Lists of a given kind of value are always grouped together, in this comparison: but the effect of that is unlikely to be noticed since NI's type-checker will probably prevent comparisons of lists of differing items in any case. The next criterion is length: a short list precedes a long one. Beyond that, we use the list's own preferred comparison function to judge the items in turn, stopping as soon as a pair of corresponding items differs: thus we sort lists of equal size in lexicographic order.

Since the comparison function depends only on the KOV, it may seem wasteful of a word of memory to store it in the list, given that we are already storing the KOV in any case. But we do this because comparisons have to be fast: we don't want to incur the overhead of translating KOV to comparison function.

```
[ LIST_OF_TY_Compare listleft listright delta no_items i cf;
    delta = BlkValueRead(listleft, LIST_LENGTH_F) - BlkValueRead(listright, LIST_LENGTH_F);
    if (delta) return delta;
    no_items = BlkValueRead(listleft, LIST_LENGTH_F);
    if (no_items == 0) return 0;
    delta = BlkValueRead(listleft, LIST_ITEM_KOV_F) - BlkValueRead(listright, LIST_ITEM_KOV_F);
    if (delta) return delta;
    cf = LIST_OF_TY_ComparisonFn(listleft);
    if (cf == 0 or UnsignedCompare) {
        for (i=0; i<no_items; i++) {
            delta = BlkValueRead(listleft, i+LIST_ITEM_BASE) -
                BlkValueRead(listright, i+LIST_ITEM_BASE);
            if (delta) return delta;
        }
    } else {
        for (i=0; i<no_items; i++) {
            delta = cf(BlkValueRead(listleft, i+LIST_ITEM_BASE),
                BlkValueRead(listright, i+LIST_ITEM_BASE));
            if (delta) return delta;
        }
    }
    return 0;
];
[ LIST_OF_TY_ComparisonFn list;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return 0;
    return KOVComparisonFunction(BlkValueRead(list, LIST_ITEM_KOV_F));
];
[ LIST_OF_TY_Distinguish txb1 txb2;
    if (LIST_OF_TY_Compare(txb1, txb2) == 0) rfalse;
    rtrue;
];
```

§8. Hashing.

```
[ LIST_OF_TY_Hash list  len kov rv i;
    rv = 0;
    len = BlkValueRead(list, LIST_LENGTH_F);
    kov = BlkValueRead(list, LIST_ITEM_KOV_F);
    for (i=0: i<len: i++)
        rv = rv * 33 + KOVHashValue(kov, BlkValueRead(list, i+LIST_ITEM_BASE));
    return rv;
];
```

§9. Printing. Unusually, this function can print the value in one of several formats: 0 for a comma-separated list; 1 for a braced, set-notation list; 2 for a comma-separated list with definite articles, which only makes sense if the list contains objects; 3 for a comma-separated list with indefinite articles. Note that a list in this sense is *not* printed using the "ListWriter.i6t" code for elaborate lists of objects, and it doesn't use the "listing contents of..." activity in any circumstances.

```
[ LIST_OF_TY_Say list format no_items v i bk;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    bk = KindAtomic(BlkValueRead(list, LIST_ITEM_KOV_F));
    ! print "kov=", BlkValueRead(list, LIST_ITEM_KOV_F), ":";
    if (format == 1) print "{";
    for (i=0:i<no_items:i++) {
        v = BlkValueRead(list, i+LIST_ITEM_BASE);
        switch (format) {
            2: print (the) v;
            3: print (a) v;
            default:
                if (bk == LIST_OF_TY) LIST_OF_TY_Say(v, 1);
                else if ((bk == TEXT_TY or INDEXED_TEXT_TY) && (format == 1)) {
                    print "~"; PrintKindValuePair(bk, v); print "~";
                }
                else PrintKindValuePair(bk, v);
        }
        if (i<no_items-2) print ", ";
        if (i==no_items-2) {
            if (format == 1) print ", "; else {
                #ifdef SERIAL_COMMA; if (no_items ~= 2) print ","; #endif;
                print (string) LISTAND2__TX;
            }
        }
    }
    if (format == 1) print "}";
];
```

§**10. List From Description.** That completes the compulsory services required for this KOV to function: from here on, the remaining routines provide definitions of stored action-related phrases in the Standard Rules.

Given a description `D` which applies to some objects and not others – say, "lighted rooms adjacent to the Transport Hub" – we can cast this into a list of all objects satisfying `D` with the following routine. Slightly wastefully of time, we have to iterate through the objects twice in order first to work out the length of list we will need, and then to transcribe them.

```
[ LIST_OF_TY_Desc list desc kov obj no_items ex len i;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return false;
    ex = BlkValueExtent(list);
    len = desc(-3);
! for (len=0, obj=desc(-2, nothing, len): obj: len++, obj=desc(-2, obj, len)) ;
! len++;
    if (len+LIST_ITEM_BASE > ex) {
        if (BlkValueSetExtent(list, len+LIST_ITEM_BASE) == false)
            return 0;
    }
    if (kov) BlkValueWrite(list, LIST_ITEM_KOV_F, kov);
    else BlkValueWrite(list, LIST_ITEM_KOV_F, OBJECT_TY);
    BlkValueWrite(list, LIST_LENGTH_F, len);
    obj = 0;
    for (i=0: i<len: i++) {
        obj = desc(-2, obj, i);
        ! print "i = ", i, " and obj = ", obj, "^";
        BlkValueWrite(list, i+LIST_ITEM_BASE, obj);
    }
    return list;
];
```

§11. Find Item. We test whether a list `list` includes a value equal to `v` or not. Equality here is in the sense of the list's comparison function: thus for indexed texts or other lists, say, deep comparisons rather than simple pointer comparisons are performed. In other words, one copy of "Alert" is equal to another.

```
[ LIST_OF_TY_FindItem list v i no_items cf;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) rfalse;
    cf = LIST_OF_TY_ComparisonFn(list);
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if (cf == 0 or UnsignedCompare) {
        for (i=0; i<no_items; i++)
            if (v == BlkValueRead(list, i+LIST_ITEM_BASE)) rtrue;
    } else {
        for (i=0; i<no_items; i++)
            if (cf(v, BlkValueRead(list, i+LIST_ITEM_BASE)) == 0) rtrue;
    }
    rfalse;
];
```

§12. Insert Item. The following routine inserts an item into the list. If this would break the size of the current block-value, then we extend by at least enough room to hold at least another 16 entries.

In the call `LIST_OF_TY_InsertItem(list, v, posnflag, posn, nodups)`, only the first two arguments are compulsory.

(a) If `nodups` is set, and an item equal to `v` is already present in the list, we return and do nothing. (`nodups` means "no duplicates".)
(b) Otherwise, if `posnflag` is `false`, we append a new entry `v` at the back of the given `list`.
(c) Otherwise, when `posnflag` is `true`, `posn` indicates the insertion position, from 1 (before the current first item) to $N + 1$ (after the last), where N is the number of items in the list at present.

```
[ LIST_OF_TY_InsertItem list v posnflag posn nodups i no_items ex nv;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return false;
    if (nodups && (LIST_OF_TY_FindItem(list, v))) return list;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if ((posnflag) && ((posn<1) || (posn > no_items+1))) {
        print "*** Couldn't add at entry ", posn, " in the list ";
        LIST_OF_TY_Say(list, true);
        print ", which has entries in the range 1 to ", no_items, " ***^";
        RunTimeProblem(RTP_LISTRANGEERROR);
        rfalse;
    }
    ex = BlkValueExtent(list);
    if (no_items+LIST_ITEM_BASE+1 > ex) {
        if (BlkValueSetExtent(list, ex+16) == false) return 0;
    }
    if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F))) {
        nv = BlkValueCreate(BlkValueRead(list, LIST_ITEM_KOV_F));
        BlkValueCopy(nv, v);
        v = nv;
    }
    if (posnflag) {
        posn--;
        for (i=no_items:i>posn:i--) {
            BlkValueWrite(list, i+LIST_ITEM_BASE,
                BlkValueRead(list, i-1+LIST_ITEM_BASE));
        }
        BlkValueWrite(list, posn+LIST_ITEM_BASE, v);
    } else {
        BlkValueWrite(list, no_items+LIST_ITEM_BASE, v);
    }
    BlkValueWrite(list, LIST_LENGTH_F, no_items+1);
    return list;
];
```

§**13. Append List.** Instead of adjoining a single value, we adjoin an entire second list, which must be of a compatible kind of value (something which NI's type-checking machinery polices for us). Except that we have a list `more` rather than a value `v` to insert, the specification is the same as for `LIST_OF_TY_InsertItem`.

```
[ LIST_OF_TY_AppendList list more posnflag posn nodups v i j no_items msize ex nv;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return false;
    if ((more==0) || (BlkType(more) ~= LIST_OF_TY)) return list;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if ((posnflag) && ((posn<1) || (posn > no_items+1))) {
        print "*** Couldn't add at entry ", posn, " in the list ";
        LIST_OF_TY_Say(list, true);
        print ", which has entries in the range 1 to ", no_items, " ***^";
        RunTimeProblem(RTP_LISTRANGEERROR);
        rfalse;
    }
    msize = BlkValueRead(more, LIST_LENGTH_F);
    ex = BlkValueExtent(list);
    if (no_items+msize+LIST_ITEM_BASE > ex) {
        if (BlkValueSetExtent(list, no_items+msize+LIST_ITEM_BASE+8) == false)
            return 0;
    }
    if (posnflag) {
        posn--;
        for (i=no_items+msize:i>=posn+msize:i--) {
            BlkValueWrite(list, i+LIST_ITEM_BASE,
                BlkValueRead(list, i-msize+LIST_ITEM_BASE));
        }
        ! BlkValueWrite(list, posn, v);
        for (j=0: j<msize: j++) {
            v = BlkValueRead(more, j+LIST_ITEM_BASE);
            if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F))) {
                nv = BlkValueCreate(BlkValueRead(list, LIST_ITEM_KOV_F));
                BlkValueCopy(nv, v);
                v = nv;
            }
            BlkValueWrite(list, posn+j+LIST_ITEM_BASE, v);
        }
    } else {
        for (i=0, j=0: i<msize: i++) {
            v = BlkValueRead(more, i+LIST_ITEM_BASE);
            if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F))) {
                nv = BlkValueCreate(BlkValueRead(list, LIST_ITEM_KOV_F));
                BlkValueCopy(nv, v);
                v = nv;
            }
            if ((nodups == 0) || (LIST_OF_TY_FindItem(list, v) == false)) {
                BlkValueWrite(list, no_items+j+LIST_ITEM_BASE, v);
                j++;
            }
        }
    }
    BlkValueWrite(list, LIST_LENGTH_F, no_items+j);
    return list;
];
```

§14. Remove Value. We remove every instance of the value `v` from the given `list`. If the optional flag `forgive` is set, then we make no complaint if no value of `v` was present in the first place: otherwise, we issue a run-time problem.

Note that if the list contains block-values then the value must be properly destroyed with `BlkFree` before being overwritten as the items shuffle down.

```
[ LIST_OF_TY_RemoveValue list v forgive i j no_items odsize f cf delendum;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) rfalse;
    cf = LIST_OF_TY_ComparisonFn(list);
    no_items = BlkValueRead(list, LIST_LENGTH_F); odsize = no_items;
    for (i=0; i<no_items; i++) {
        delendum = BlkValueRead(list, i+LIST_ITEM_BASE);
        if (cf == 0 or UnsignedCompare)
            f = (v == delendum);
        else
            f = (cf(v, delendum) == 0);
        if (f) {
            if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F)))
                BlkFree(delendum);
            for (j=i+1; j<no_items; j++)
                BlkValueWrite(list, j-1+LIST_ITEM_BASE,
                    BlkValueRead(list, j+LIST_ITEM_BASE));
            no_items--; i--;
            BlkValueWrite(list, LIST_LENGTH_F, no_items);
        }
    }
    if (odsize ~= no_items) rfalse;
    if (forgive) rfalse;
    print "*** Couldn't remove: the value ";
    PrintKindValuePair(BlkValueRead(list, LIST_ITEM_KOV_F), v);
    print " was not present in the list ";
    LIST_OF_TY_Say(list, true);
    print " ***^";
    RunTimeProblem(RTP_LISTRANGEERROR);
];
```

§**15. Remove Item Range.** We excise items `from` to `to` from the given `list`, which numbers its items upwards from 1. If the optional flag `forgive` is set, then we truncate a range overspilling the actual list, and we make no complaint if it turns out that there is then nothing to be done: otherwise, in either event, we issue a run-time problem.

Once again, we destroy any block-values whose pointers will be overwritten as the list shuffles down to fill the void.

```
[ LIST_OF_TY_RemoveItemRange list from to forgive i d no_items;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) rfalse;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if ((from > to) || (from <= 0) || (to > no_items)) {
        if (forgive) {
            if (from <= 0) from = 1;
            if (to >= no_items) to = no_items;
            if (from > to) return list;
        } else {
            print "*** Couldn't remove entries ", from, " to ", to, " from the list ";
            LIST_OF_TY_Say(list, true);
            print ", which has entries in the range 1 to ", no_items, " ***^";
            RunTimeProblem(RTP_LISTRANGEERROR);
            rfalse;
        }
    }
    to--; from--;
    d = to-from+1;
    if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F)))
        for (i=0; i<d; i++)
            BlkFree(BlkValueRead(list, from+i+LIST_ITEM_BASE));
    for (i=from: i<no_items-d: i++)
        BlkValueWrite(list, i+LIST_ITEM_BASE,
            BlkValueRead(list, i+d+LIST_ITEM_BASE));
    BlkValueWrite(list, LIST_LENGTH_F, no_items-d);
    return list;
];
```

§**16. Remove List.** We excise all values from the removal list `rlist`, wherever they occur in `list`. Inevitably, given that we haven't sorted these lists and can spare neither time nor storage to do so, this is an expensive process with a running time proportional to the product of the two list sizes: we accept that as an overhead because in practice the `rlist` is almost always small in real-world use.

If the initial lists were disjoint, so that no removals occur, we always forgive the user: the request was not necessarily a foolish one, it only happened in this situation to be unhelpful.

```
[ LIST_OF_TY_Remove_List list rlist i j k v w no_items odsize rsize cf f;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) rfalse;
    no_items = BlkValueRead(list, LIST_LENGTH_F); odsize = no_items;
    rsize = BlkValueRead(rlist, LIST_LENGTH_F);
    cf = LIST_OF_TY_ComparisonFn(list);
    for (i=0: i<no_items: i++) {
        v = BlkValueRead(list, i+LIST_ITEM_BASE);
        for (k=0: k<rsize: k++) {
            w = BlkValueRead(rlist, k+LIST_ITEM_BASE);
            if (cf == 0 or UnsignedCompare)
```

```
                f = (v == w);
            else
                f = (cf(v, w) == 0);
            if (f) {
                if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F)))
                    BlkFree(v);
                for (j=i+1: j<no_items: j++)
                    BlkValueWrite(list, j+LIST_ITEM_BASE-1,
                        BlkValueRead(list, j+LIST_ITEM_BASE));
                no_items--; i--;
                BlkValueWrite(list, LIST_LENGTH_F, no_items);
                break;
            }
        }
    }
    rfalse;
];
```

§17. Get Length.

```
[ LIST_OF_TY_GetLength list;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return 0;
    return BlkValueRead(list, LIST_LENGTH_F);
];
[ LIST_OF_TY_Empty list;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) rfalse;
    if (BlkValueRead(list, LIST_LENGTH_F) == 0) rtrue;
    rfalse;
];
```

§18. Set Length. This is rather harder: it might lengthen the list, in which case we have to pad out with the default value for the kind of value stored – padding a list of numbers with 0s, a list of texts with copies of the empty text, and so on – creating such block-values as might be needed; or else it might shorten the list, in which case we must cut items, destroying them properly if they were block-values.

`LIST_OF_TY_SetLength(list, newsize, this_way_only, truncation_end)` alters the length of the given `list` to `newsize`. If `this_way_only` is 1, the list is only allowed to grow, and nothing happens if we have asked to shrink it; if it is −1, the list is only allowed to shrink; if it is 0, the list is allowed either to grow or shrink. In the event that the list does have to shrink, entries must be removed, and we remove from the end if `truncation_end` is 1, or from the start if it is −1.

```
[ LIST_OF_TY_SetLength list newsize this_way_only truncation_end no_items ex i dv;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return 0;
    if (newsize < 0) "*** Cannot resize a list to negative length ***";
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if (no_items < newsize) {
        if (this_way_only == -1) return list;
        ex = BlkValueExtent(list);
        if (newsize+LIST_ITEM_BASE > ex) {
            if (BlkValueSetExtent(list, newsize+LIST_ITEM_BASE) == false)
                return 0;
        }
        dv = DefaultValueOfKOV(BlkValueRead(list, LIST_ITEM_KOV_F));
        for (i=no_items: i<newsize: i++)
            BlkValueWrite(list, LIST_ITEM_BASE+i, dv);
        BlkValueWrite(list, LIST_LENGTH_F, newsize);
    }
    if (no_items > newsize) {
        if (this_way_only == 1) return list;
        if (truncation_end == -1) {
            if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F)))
                for (i=0: i<no_items-newsize: i++)
                    BlkFree(BlkValueRead(list, LIST_ITEM_BASE+i));
            for (i=0: i<newsize: i++)
                BlkValueWrite(list, LIST_ITEM_BASE+i,
                    BlkValueRead(list, LIST_ITEM_BASE+no_items-newsize+i));
        } else {
            if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F)))
                for (i=newsize: i<no_items: i++)
                    BlkFree(BlkValueRead(list, LIST_ITEM_BASE+i));
        }
        BlkValueWrite(list, LIST_LENGTH_F, newsize);
    }
    return list;
];
```

§19. Get Item.

```
[ LIST_OF_TY_GetItem list i forgive no_items;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return false;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if ((i<=0) || (i>no_items)) {
        if (forgive) return false;
        print "*** Couldn't read from entry ", i, " of a list which";
        switch (no_items) {
            0: print " is empty ***^";
            1: print " has only one entry, numbered 1 ***^";
            default: print " has entries numbered from 1 to ", no_items, " ***^";
        }
        RunTimeProblem(RTP_LISTRANGEERROR);
        if (no_items >= 1) i = 1;
        else return false;
    }
    return BlkValueRead(list, LIST_ITEM_BASE+i-1);
];
```

§**20. Write Item.** The slightly odd name for this function comes about because our usual way to convert an rvalue such as `LIST_OF_TY_GetItem(L, 4)` is to prefix `Write`, so that it becomes `WriteLIST_OF_TY_GetItem(L, 4)`.

```
[ WriteLIST_OF_TY_GetItem list i val no_items;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return false;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if ((i<=0) || (i>no_items)) {
        print "*** Couldn't write to list entry ", i, " of a list which";
        switch (no_items) {
            0: print " is empty ***^";
            1: print " has only one entry, numbered 1 ***^";
            default: print " has entries numbered from 1 to ", no_items, " ***^";
        }
        return RunTimeProblem(RTP_LISTRANGEERROR);
    }
    BlkValueWrite(list, LIST_ITEM_BASE+i-1, val);
];
```

§**21. Put Item.** Higher-level code should not use `Write_LIST_OF_TY_GetItem`, because it does not properly keep track of block-value copying: the following should be used instead.

```
[ LIST_OF_TY_PutItem list i v  no_items nv;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return false;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if (KOVIsBlockValue(BlkValueRead(list, LIST_ITEM_KOV_F))) {
        nv = BlkValueCreate(BlkValueRead(list, LIST_ITEM_KOV_F));
        BlkValueCopy(nv, v);
        v = nv;
    }
    if ((i<=0) || (i>no_items)) return false;
    BlkValueWrite(list, LIST_ITEM_BASE+i-1, v);
];
```

§**22. Multiple Object List.** The parser uses one data structure which is really a list: but which can't be represented as such because the heap might not exist. This is the multiple object list, which is used to handle commands like TAKE ALL by firing off a sequence of actions with one of the objects taken from entries in turn of the list. The following converts it to a list structure.

```
[ LIST_OF_TY_Mol list len i;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return 0;
    len = multiple_object-->0;
    LIST_OF_TY_SetLength(list, len);
    for (i=1: i<=len: i++)
        LIST_OF_TY_PutItem(list, i, multiple_object-->i);
    return list;
];
[ LIST_OF_TY_Set_Mol list len i;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return 0;
    len = BlkValueRead(list, LIST_LENGTH_F);
    if (len > 63) len = 63;
    multiple_object-->0 = len;
    for (i=1: i<=len: i++)
        multiple_object-->i = BlkValueRead(list, LIST_ITEM_BASE+i-1);
];
```

§23. Reversing. Reversing a list is, happily, a very efficient operation when the list contains block-values: because the pointers are rearranged but none is duplicated or destroyed, we can for once ignore the fact that they are pointers to block-values and simply move them around like any other data.

```
[ LIST_OF_TY_Reverse list no_items i v;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return 0;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if (no_items < 2) return list;
    for (i=0;i*2<no_items;i++) {
        v = BlkValueRead(list, LIST_ITEM_BASE+i);
        BlkValueWrite(list, LIST_ITEM_BASE+i,
            BlkValueRead(list, LIST_ITEM_BASE+no_items-1-i));
        BlkValueWrite(list, LIST_ITEM_BASE+no_items-1-i, v);
    }
    return list;
];
```

§24. Rotation. The same is true of rotation. Here, “forwards” rotation means towards the end of the list, “backwards” means towards the start.

```
[ LIST_OF_TY_Rotate list backwards  no_items i v;
    if ((list==0) || (BlkType(list) ~= LIST_OF_TY)) return 0;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if (no_items < 2) return list;
    if (backwards) {
        v = BlkValueRead(list, LIST_ITEM_BASE);
        for (i=0:i<no_items-1:i++)
            BlkValueWrite(list, LIST_ITEM_BASE+i,
                BlkValueRead(list, LIST_ITEM_BASE+i+1));
        BlkValueWrite(list, no_items-1+LIST_ITEM_BASE, v);
    } else {
        v = BlkValueRead(list, no_items-1+LIST_ITEM_BASE);
        for (i=no_items-1:i>0:i--)
            BlkValueWrite(list, LIST_ITEM_BASE+i,
                BlkValueRead(list, LIST_ITEM_BASE+i-1));
        BlkValueWrite(list, LIST_ITEM_BASE, v);
    }
    return list;
];
```

§**25. Sorting.** And the same, again, is true of sorting: but we do have to take note of block values when it comes to performing comparisons, because we can only compare items in the list by looking at their contents, not the pointers to their contents.

`LIST_OF_TY_Sort(list, dir, prop)` sorts the given `list` in ascending order if `dir` is 1, in descending order if `dir` is −1, or in random order if `dir` is 2. The comparison used is the one for the kind of value stored in the list, unless the optional argument `prop` is supplied, in which case we sort based not on the item values but on their values for the property `prop`. (This only makes sense if the list contains objects.)

```
Global LIST_OF_TY_Sort_cf;

[ LIST_OF_TY_Sort list dir prop cf  i j no_items v;
    no_items = BlkValueRead(list, LIST_LENGTH_F);
    if (dir == 2) {
        if (no_items < 2) return;
        for (i=1:i<no_items:i++) {
            j = random(i+1) - 1;
            v = BlkValueRead(list, LIST_ITEM_BASE+i);
            BlkValueWrite(list, LIST_ITEM_BASE+i, BlkValueRead(list, LIST_ITEM_BASE+j));
            BlkValueWrite(list, LIST_ITEM_BASE+j, v);
        }
        return;
    }
    SetSortDomain(ListSwapEntries, ListCompareEntries);
    if (cf) { LIST_OF_TY_Sort_cf = BlkValueCompare; ! dir = -dir;
    }
    else LIST_OF_TY_Sort_cf = 0;
    SortArray(list, prop, dir, no_items, false, 0);
];

[ ListSwapEntries list i j v;
    if (i==j) return;
    v = BlkValueRead(list, LIST_ITEM_BASE+i-1);
    BlkValueWrite(list, LIST_ITEM_BASE+i-1, BlkValueRead(list, LIST_ITEM_BASE+j-1));
    BlkValueWrite(list, LIST_ITEM_BASE+j-1, v);
];

[ ListCompareEntries list col i j d cf;
    if (i==j) return 0;
    i = BlkValueRead(list, LIST_ITEM_BASE+i-1);
    j = BlkValueRead(list, LIST_ITEM_BASE+j-1);
    if (I7S_Col) {
        if (i provides I7S_Col) i=i.I7S_Col; else i=0;
        if (j provides I7S_Col) j=j.I7S_Col; else j=0;
        cf = LIST_OF_TY_Sort_cf;
    } else {
        cf = LIST_OF_TY_ComparisonFn(list);
    }
    if (cf == 0)
        return i - j;
    else
        return cf(i, j);
];

#IFNOT; ! IFDEF MEMORY_HEAP_SIZE

[ LIST_OF_TY_Support t a b c; rfalse; ];
[ LIST_OF_TY_Say list; ];
[ LIST_OF_TY_FindItem list v; rfalse; ];
[ LIST_OF_TY_Empty list; rfalse; ];
[ LIST_OF_TY_SetLength l n; rfalse; ];
[ LIST_OF_TY_InsertItem a b c d e; rfalse; ];
#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```

Combinations Template B/combt

Purpose

Code to support the combination kind of value constructor.

§**1. Head.** As ever: if there is no heap, there are no combinations (in this sense).

```
#IFDEF MEMORY_HEAP_SIZE; ! Will exist if any use is made of heap
```

§**2. KOV Support.** See the "BlockValues.i6t" segment for the specification of the following routines.

```
[ COMBINATION_TY_Support task arg1 arg2 arg3;
    switch(task) {
        CREATE_KOVS:      arg3 = COMBINATION_TY_Create(arg2);
                          if (arg1) COMBINATION_TY_CopyRawArray(arg3, arg1, 2);
                          return arg3;
        CAST_KOVS:        rfalse;
        DESTROY_KOVS:     return COMBINATION_TY_Destroy(arg1);
        PRECOPY_KOVS:     return COMBINATION_TY_PreCopy(arg1, arg2);
        COPY_KOVS:        return COMBINATION_TY_Copy(arg1, arg2);
        COMPARE_KOVS:     return COMBINATION_TY_Compare(arg1, arg2);
        READ_FILE_KOVS:   rfalse;
        WRITE_FILE_KOVS: rfalse;
        HASH_KOVS:        return COMBINATION_TY_Hash(arg1);
    }
];
```

§**3. Creation.** A combination is like a list, but simpler; it has a fixed, usually short, size. On the other hand, its entries are not all of the same kind as each other.

Combinations are stored as a fixed-sized block of word entries. The first block is the only header information: a pointer to a further structure in memory, describing the kind. The subsequent blocks are the actual records. Thus, a triple (x, y, z) uses 4 words.

```
Constant COMBINATION_KIND_F = 0; ! A pointer to a block indicating the kind
Constant COMBINATION_ITEM_BASE = 1; ! List items begin at this entry
[ COMBINATION_TY_Create kind comb N i bk v;
    N = KindBaseArity(kind);
    comb = BlkAllocate((COMBINATION_ITEM_BASE+N)*WORDSIZE, COMBINATION_TY, BLK_FLAG_WORD);
    BlkValueWrite(comb, COMBINATION_KIND_F, kind);
    for (i=0; i<N; i++) {
        bk = KindBaseTerm(kind, i);
        if (KOVIsBlockValue(bk))
            v = BlkValueCreate(bk);
        else
            v = DefaultValueOfKOV(bk);
        BlkValueWrite(comb, COMBINATION_ITEM_BASE+i, v);
    }
    return comb;
];
```

§**4. Setting Up.** NI needs to compile code which will create constant combinations at run-time: the following routine is convenient for that. A "raw array" in this routine's sense is an array `raw` such that `raw-->2` contains the number of items, `raw-->1` the kind of value, and `raw-->3` onwards are the items themselves; note that this is the same format used for constant lists.

```
[ COMBINATION_TY_CopyRawArray comb raw rea len i ex bk v w;
    if ((comb==0) || (BlkType(comb) ~= COMBINATION_TY)) return false;
    ex = BlkValueExtent(comb);
    len = raw-->2;
    if ((len+COMBINATION_ITEM_BASE > ex) &&
        (BlkValueSetExtent(comb, len+COMBINATION_ITEM_BASE) == false)) return 0;
    BlkValueWrite(comb, LIST_LENGTH_F, len);
    if (rea == 2) bk = BlkValueRead(comb, COMBINATION_KIND_F);
    else {
        bk = raw-->1;
        BlkValueWrite(comb, COMBINATION_KIND_F, bk);
    }
    for (i=0:i<len:i++) {
        v = raw-->(i+3);
        if (KOVIsBlockValue(bk)) v = BlkValueCreate(bk, v);
        BlkValueWrite(comb, i+COMBINATION_ITEM_BASE, v);
    }
    #ifdef SHOW_ALLOCATIONS;
    print "Copied raw array to comb: "; COMBINATION_TY_Say(comb, 1); print "^";
    #endif;
    return comb;
];
```

§**5. Destruction.** If the comb items are themselves block-values, they must all be freed before the comb itself can be freed.

```
[ COMBINATION_TY_Destroy comb kind no_items i bk;
    kind = BlkValueRead(comb, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    for (i=0; i<no_items; i++) {
        bk = KindBaseTerm(kind, i);
        if (KOVIsBlockValue(bk))
            BlkFree(BlkValueRead(comb, i+COMBINATION_ITEM_BASE));
    }
    return comb;
];
```

§**6. Copying.** Again, if the comb contains block-values then they must be duplicated rather than bitwise copied as pointers.

```
Global precopied_comb_kov;

[ COMBINATION_TY_PreCopy lto lfrom comb no_items i nv bk;
    precopied_comb_kov = BlkValueRead(lto, COMBINATION_KIND_F);
];

[ COMBINATION_TY_Copy lto lfrom no_items i nv kind bk;
    kind = BlkValueRead(lto, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    BlkValueWrite(lto, COMBINATION_KIND_F, precopied_comb_kov);
    for (i=0; i<no_items; i++) {
        bk = KindBaseTerm(kind, i);
        if (KOVIsBlockValue(bk)) {
            nv = BlkValueCreate(bk);
            BlkValueCopy(nv, BlkValueRead(lfrom, i+COMBINATION_ITEM_BASE));
            BlkValueWrite(lto, i+COMBINATION_ITEM_BASE, nv);
        }
    }
];
```

§**7. Comparison.** This is a lexicographic comparison and assumes both combinations have the same kind.

```
[ COMBINATION_TY_Compare listleft listright delta no_items i cf kind bk;
    kind = BlkValueRead(listleft, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    for (i=0; i<no_items; i++) {
        bk = KindBaseTerm(kind, i);
        cf = KOVComparisonFunction(bk);
        if (cf == 0 or UnsignedCompare) {
            delta = BlkValueRead(listleft, i+COMBINATION_ITEM_BASE) -
                BlkValueRead(listright, i+COMBINATION_ITEM_BASE);
            if (delta) return delta;
        } else {
            delta = cf(BlkValueRead(listleft, i+COMBINATION_ITEM_BASE),
                BlkValueRead(listright, i+COMBINATION_ITEM_BASE));
            if (delta) return delta;
        }
    }
    return 0;
];

[ COMBINATION_TY_Distinguish txb1 txb2;
    if (COMBINATION_TY_Compare(txb1, txb2) == 0) rfalse;
    rtrue;
];
```

§8. Hashing.

```
[ COMBINATION_TY_Hash comb  kind rv no_items i bk;
    rv = 0;
    kind = BlkValueRead(comb, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    for (i=0: i<no_items: i++) {
        bk = KindBaseTerm(kind, i);
        rv = rv * 33 + KOVHashValue(bk, BlkValueRead(comb, i+COMBINATION_ITEM_BASE));
    }
    return rv;
];
```

§9. Printing.

```
[ COMBINATION_TY_Say comb format no_items v i kind bk;
    if ((comb==0) || (BlkType(comb) ~= COMBINATION_TY)) return;
    kind = BlkValueRead(comb, COMBINATION_KIND_F);
    no_items = KindBaseArity(kind);
    print "(";
    for (i=0; i<no_items; i++) {
        if (i>0) print ", ";
        bk = KindBaseTerm(kind, i);
        v = BlkValueRead(comb, i+COMBINATION_ITEM_BASE);
        if (bk == LIST_OF_TY) LIST_OF_TY_Say(v, 1);
        else PrintKindValuePair(bk, v);
    }
    print ")";
];
#IFNOT; ! IFDEF MEMORY_HEAP_SIZE
[ COMBINATION_TY_Support t a b c; rfalse; ];
[ COMBINATION_TY_Say comb; ];
#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```

Relation Kind Template B/relkt

Purpose

Code to support the relation kind.

§**1. Run-Time Storage.** Inform uses a rich variety of relations, with many different data representations, but we aim to hide that complexity from the user. At run-time, a relation is represented by a block value – that is, by a pointer to a block of data. For a relation conjured up like so:

let R be a relation of numbers to texts;

this will be a block on the heap, much like a list or an indexed text. For a built-in relation like "containment" or one constructed by a line like

Proximity relates various things to various rooms.

the block of data will not actually live on the heap but will be elsewhere in memory; it will have the `BLK_FLAG_RESIDENT` bit set, to mark that the heap code isn't allowed to destroy it, but in other respects will look identical.

The data structure contains six words of metadata; for relations on the heap, actual data then sometimes follows on. The low-level routines in "Relations.i6t" access these six words by direct use of `-->`, for speed, and they use the offset constants `RR_NAME` and so on; but we will use the `BlkValueRead` and `BlkValueWrite` routines in this section, which need offsets in the form `RRV_NAME`. (The discrepancy of 4 is to allow for the four-word block header.)

The six words in the data structure hold:

(1) The name of the relation, as a packed string.
(2) A bitmap of its permissions (see below).
(3) A storage word whose meaning depends on its private choice of data representation.
(4) Its strong kind ID, which will always be a pointer to a block reading `RELATION_TY 2 X Y` where `X` and `Y` are in turn strong kinds IDs for the left and right domains of the relation. For instance, the strong kind ID for a relation of numbers to indexed texts is a pointer to the sequence of words `RELATION_TY 2 NUMBER_TY INDEXED_TY`.
(5) A handler routine which tests or asserts the relation (see below).
(6) Descriptive text about the relation, used in SHOWME or run-time problems.

```
Constant RRV_NAME      RR_NAME-4;
Constant RRV_PERMISSIONS RR_PERMISSIONS-4;
Constant RRV_STORAGE RR_STORAGE-4;
Constant RRV_KIND RR_KIND-4;
Constant RRV_HANDLER RR_HANDLER-4;
Constant RRV_DESCRIPTION RR_DESCRIPTION-4;
Constant RRV_USED 6;
Constant RRV_FILLED 7;
Constant RRV_DATA_BASE 8;

! valencies
Constant RRVAL_V_TO_V 0;
Constant RRVAL_V_TO_O RELS_Y_UNIQUE;
Constant RRVAL_O_TO_V RELS_X_UNIQUE;
Constant RRVAL_O_TO_O RELS_X_UNIQUE+RELS_Y_UNIQUE;
Constant RRVAL_EQUIV RELS_EQUIVALENCE+RELS_SYMMETRIC;
Constant RRVAL_SYM_V_TO_V RELS_SYMMETRIC;
Constant RRVAL_SYM_O_TO_O RELS_SYMMETRIC+RELS_X_UNIQUE+RELS_Y_UNIQUE;

! dictionary entry flags
Constant RRF_USED $0001; ! entry contains a value
Constant RRF_DELETED $0002; ! entry used to contain a value
Constant RRF_SINGLE $0004; ! entry's Y is a value, not a list
Constant RRF_HASX $0010; ! 2-in-1 entry contains a corresponding key
Constant RRF_HASY $0020; ! 2-in-1 entry contains a corresponding value
Constant RRF_ENTKEYX $0040; ! 2-in-1 entry key is left side KOV
Constant RRF_ENTKEYY $0080; ! 2-in-1 entry key is right side KOV

! permission/task constants (those commented out here are generated by I7)
!Constant RELS_SYMMETRIC $8000;
!Constant RELS_EQUIVALENCE $4000;
!Constant RELS_X_UNIQUE $2000;
!Constant RELS_Y_UNIQUE $1000;
!Constant RELS_TEST $0800;
!Constant RELS_ASSERT_TRUE $0400;
!Constant RELS_ASSERT_FALSE $0200;
!Constant RELS_SHOW $0100;
!Constant RELS_ROUTE_FIND $0080;
!Constant RELS_ROUTE_FIND_COUNT $0040;
Constant RELS_COPY $0020;
Constant RELS_DESTROY $0010;
!Constant RELS_LOOKUP_ANY $0008;
!Constant RELS_LOOKUP_ALL_X $0004;
!Constant RELS_LOOKUP_ALL_Y $0002;
!Constant RELS_LIST $0001;

Constant RELS_EMPTY $0003;
Constant RELS_SET_VALENCY $0005;

! RELS_LOOKUP_ANY mode selection constants
Constant RLANY_GET_X 1;
Constant RLANY_GET_Y 2;
Constant RLANY_CAN_GET_X 3;
Constant RLANY_CAN_GET_Y 4;

! RELS_LIST mode selection constant
Constant RLIST_ALL_X 1;
Constant RLIST_ALL_Y 2;
Constant RLIST_ALL_PAIRS 3;
```

§2. Tunable Parameters. These constants affect the performance characteristics of the dictionary structures used for relations on the heap. Changing their values may alter the balance between memory consumption and running time.

`RRP_MIN_SIZE`, `RRP_RESIZE_SMALL`, and `RRP_RESIZE_LARGE` must all be powers of two.

```
Constant RRP_MIN_SIZE      8;   ! minimum number of entries (DO NOT CHANGE)
Constant RRP_PERTURB_SHIFT 5;   ! affects the probe sequence
Constant RRP_RESIZE_SMALL  4;   ! resize factor for small tables
Constant RRP_RESIZE_LARGE  2;   ! resize factor for large tables
Constant RRP_LARGE_IS      256; ! how many entries make a table "large"?
Constant RRP_CROWDED_IS    2;   ! when filled entries outnumber unfilled by _ to 1
```

§3. Abstract Relations. As the following shows, we can abstractly use a relation – that is, we can use a relation whose identity we know little about – by calling its handler routine `R` in the form `R(rel, task, X, Y)`.

The task should be one of: `RELS_TEST`, `RELS_ASSERT_TRUE`, `RELS_ASSERT_FALSE`, `RELS_SHOW`, `RELS_ROUTE_FIND`, `RELS_ROUTE_FIND_COUNT`, `RELS_COPY`, `RELS_DESTROY`, `RELS_LOOKUP_ANY`, `RELS_LOOKUP_ALL_X`, `RELS_LOOKUP_ALL_Y`, `RELS_LIST`, or `RELS_EMPTY`.

`RELS_SHOW` produces output for the SHOWME testing command. `RELS_ROUTE_FIND` finds the next step in a route from `X` to `Y`, and `RELS_ROUTE_FIND_COUNT` counts the shortest number of steps or returns -1 if no route exists. `RELS_COPY` makes a deep copy of the relation by replacing all block values with duplicates, and `RELS_DESTROY` frees all block values. `RELS_LOOKUP_ANY` finds any one of the `X` values related to a given `Y`, or vice versa, or checks whether such an `X` or `Y` value exists. `RELS_LOOKUP_ALL_X` and `RELS_LOOKUP_ALL_Y` produce a list of all the `X` values related to a given `Y`, or vice versa. `RELS_LIST` produces a list of all `X` values for which a corresponding `Y` exists, or vice versa, or a list of all `(X,Y)` pairs for which `X` is related to `Y`. `RELS_EMPTY` either makes the relation empty (if `X` is 1) or non-empty (if `X` is 0) or makes no change (if `X` is negative), and in any case returns true or false indicating whether the relation is now empty.

Because not every relation supports all of these operations, the "permissions" word in the block is always a bitmap which is a sum of those operations it does offer.

At present, these permissions are not checked as rigorously as they should be (they're correctly set, but not much monitored).

```
[ RelationTest relation task X Y  handler;
    handler = relation-->RR_HANDLER;
    return handler(relation, task, X, Y);
];
```

§4. Empty Relations. The absolute minimum relation is one which can only be tested, and which is always empty, that is, where no two values are ever related to each other. The necessary handler routine is `EmptyRelationHandler`.

```
[ EmptyRelationHandler relation task X Y;
    if (task == RELS_EMPTY) rtrue;
    rfalse;
];
```

§5. Head. As ever: if there is no heap, there are no relations stored as values.

```
#IFDEF MEMORY_HEAP_SIZE; ! Will exist if any use is made of heap
```

§**6. KOV Support.** See the "BlockValues.i6t" segment for the specification of the following routines.

```
[ RELATION_TY_Support task arg1 arg2 arg3;
    switch(task) {
        CREATE_KOVS:     arg3 = RELATION_TY_Create(arg2, arg1);
                         return arg3;
        CAST_KOVS:       rfalse;
        DESTROY_KOVS:    return RELATION_TY_Destroy(arg1);
        PRECOPY_KOVS:    rfalse;
        COPY_KOVS:       return RELATION_TY_Copy(arg1, arg2);
        COMPARE_KOVS:    return RELATION_TY_Compare(arg1, arg2);
        READ_FILE_KOVS:  rfalse;
        WRITE_FILE_KOVS: rfalse;
        HASH_KOVS:       return arg1;
    }
];
```

§**7. Creation.** Something we have to be careful about is what we mean by copying, or indeed creating, a relation. For example, if we write

let Q be a relation of objects to objects;

let Q be the containment relation;

...we aren't literally asking for Q to be a duplicate copy of containment, which can then independently evolve – we mean in some sense that Q is a pointer to the one and only containment relation. On the other hand, if we write

let Q be a relation of numbers to numbers;

make Q relate 3 to 7;

then the second line clearly expects Q to be its own relation, newly created.

We cope with this at creation time. If we're invited to create a copy of an existing relation, we look to see if it is empty – which we detect by its use of the `EmptyRelationHandler` handler. The empty relations are exactly those used as default values for the relation kinds; thus that's what will happen when Q is created. If we find this handler, we intercept and replace it with one of the heap relation handlers, which thus makes the relation a newly constructed data structure which can grow freely from here.

```
[ RELATION_TY_Create kov from rel i skov handler;
    rel = BlkAllocate((RRV_DATA_BASE + 3*RRP_MIN_SIZE)*WORDSIZE,
        RELATION_TY, BLK_FLAG_WORD+BLK_FLAG_MULTIPLE);
    if ((from == 0) && (kov ~= 0)) from = DefaultValueFinder(kov);
    if (from) {
        for (i=0: i<RRV_DATA_BASE: i++) BlkValueWrite(rel, i, BlkValueRead(from, i));
        if (BlkValueRead(from, RRV_HANDLER) == EmptyRelationHandler) {
            handler = ChooseRelationHandler(BlkValueRead(rel, RRV_KIND));
            BlkValueWrite(rel, RRV_NAME, "anonymous relation");
            BlkValueWrite(rel, RRV_PERMISSIONS,
                RELS_TEST+RELS_ASSERT_TRUE+RELS_ASSERT_FALSE+RELS_SHOW);
            BlkValueWrite(rel, RRV_HANDLER, handler);
            BlkValueWrite(rel, RRV_STORAGE, RRP_MIN_SIZE-1);
            BlkValueWrite(rel, RRV_DESCRIPTION, "an anonymous relation");
            BlkValueWrite(rel, RRV_USED, 0);
            BlkValueWrite(rel, RRV_FILLED, 0);
        }
    } else {
        handler = ChooseRelationHandler(kov);
        BlkValueWrite(rel, RRV_NAME, "anonymous relation");
        BlkValueWrite(rel, RRV_PERMISSIONS,
            RELS_TEST+RELS_ASSERT_TRUE+RELS_ASSERT_FALSE+RELS_SHOW);
        BlkValueWrite(rel, RRV_STORAGE, RRP_MIN_SIZE-1);
        BlkValueWrite(rel, RRV_KIND, kov);
        BlkValueWrite(rel, RRV_HANDLER, handler);
        BlkValueWrite(rel, RRV_DESCRIPTION, "an anonymous relation");
        BlkValueWrite(rel, RRV_USED, 0);
        BlkValueWrite(rel, RRV_FILLED, 0);
    }
    return rel;
];
```

§**8. Destruction.** If the relation stores block values on either side, invoke the handler using a special task value to free the memory associated with them.

```
[ RELATION_TY_Destroy rel  handler;
    handler = BlkValueRead(rel, RRV_HANDLER);
    handler(rel, RELS_DESTROY);
    return rel;
];
```

§**9. Copying.** Same as destruction: invoke the handler using a special value to tell it to perform deep copying.

```
[ RELATION_TY_Copy lto lfrom  handler;
    handler = BlkValueRead(lto, RRV_HANDLER);
    handler(lto, RELS_COPY);
];
```

§**10. Comparison.** It really isn't clear how to define equality for relations, but we follow the doctrine above. What we don't do is to test its actual state – that would be very slow and might be impossible.

```
[ RELATION_TY_Compare rleft rright ind1 ind2;
    ind1 = BlkValueRead(rleft, RRV_HANDLER);
    ind2 = BlkValueRead(rright, RRV_HANDLER);
    if (ind1 ~= ind2)
        return ind1 - ind2;
    return rleft - rright;
];
[ RELATION_TY_Distinguish rleft rright;
    if (RELATION_TY_Compare(rleft, rright) == 0) rfalse;
    rtrue;
];
```

§11. Printing.

```
[ RELATION_TY_Say rel;
    if (rel == 0) print "(null relation)"; ! shouldn't happen
    else print (string) rel-->RR_NAME;
];
```

§12. Naming.

```
[ RELATION_TY_Name rel txt;
    if (rel) {
        BlkValueWrite(rel, RRV_NAME, txt);
        BlkValueWrite(rel, RRV_DESCRIPTION, txt);
    }
];
```

§13. Choose Relation Handler. We implement two different various-to-various handler routines for the sake of efficiency. The choice of handler routines is made based on the kinds of value being related. Each handler also has a corresponding wrapper for symmetric relations.

```
[ ChooseRelationHandler kov sym;
    if (KOVIsBlockValue(KindBaseTerm(kov, 0))) {
        if (sym) return SymHashListRelationHandler;
        return HashListRelationHandler;
    }
    if (sym) return SymDoubleHashSetRelationHandler;
    return DoubleHashSetRelationHandler;
];
```

§14. Valency. "Valency" refers to the number of participants allowed on either side of the relation: various-to-various, one-to-various, various-to-one, or one-to-one. A newly created relation is always various-to-various. We allow the author to change the valency, but only if no entries have been added yet.

```
[ RELATION_TY_SetValency rel val  kov filled cur handler ext;
    filled = BlkValueRead(rel, RRV_FILLED);
    if (filled) { print "*** Illegal valency change ***^"; rfalse; }
    kov = BlkValueRead(rel, RRV_KIND);
    if (val == RRVAL_EQUIV or RRVAL_SYM_V_TO_V or RRVAL_SYM_O_TO_O) {
        if (KindBaseTerm(kov, 0) ~= KindBaseTerm(kov, 1)) {
            print "*** Relation cannot be made symmetric ***^";
            rfalse;
        }
    }
    cur = BlkValueRead(rel, RRV_HANDLER);
    switch (val) {
        RRVAL_V_TO_V: handler = ChooseRelationHandler(kov, false);
        RRVAL_V_TO_O: handler = HashTableRelationHandler;
        RRVAL_O_TO_V: handler = ReversedHashTableRelationHandler;
        RRVAL_O_TO_O: handler = TwoInOneHashTableRelationHandler;
        RRVAL_EQUIV: handler = EquivHashTableRelationHandler;
        RRVAL_SYM_V_TO_V: handler = ChooseRelationHandler(kov, true);
        RRVAL_SYM_O_TO_O: handler = Sym2in1HashTableRelationHandler;
        default: print "*** Illegal valency value ***^"; rfalse;
    }
    if (cur == handler) rtrue;
    ! adjust size when going to or from 2-in-1
    if (cur == TwoInOneHashTableRelationHandler) {
        ext = BlkValueRead(rel, RRV_STORAGE) + 1;
        BlkValueSetExtent(rel, RRV_DATA_BASE + 3*ext);
    } else if (handler == TwoInOneHashTableRelationHandler) {
        ext = BlkValueRead(rel, RRV_STORAGE) + 1;
        BlkValueSetExtent(rel, RRV_DATA_BASE + 4*ext);
    }
    BlkValueWrite(rel, RRV_HANDLER, handler);
];

[ RELATION_TY_GetValency rel  handler;
    return BlkValueRead(rel, RRV_PERMISSIONS) & VALENCY_MASK;
];
```

§**15. Double Hash Set Relation Handler.** This implements relations which are stored as a double-hashed set. The storage comprises a list of three-word entries (F, X, Y), where F is a flags word. The ordering of the list is determined by a probe sequence which depends on the combined hash values of X and Y.

The "storage" word in the header stores one less than the number of entries in the list; the number of entries in the list is always a power of two, so this will always be a bit mask. The "used" and "filled" words store the number of entries which currently hold a value, and the number of entries which have ever held a value (even if it was since deleted), respectively.

The utility routine `DoubleHashSetLookUp` locates the hash entry for a key/value pair. It returns either the (non-negative) number of the entry where the pair was found, or the (negative) bitwise NOT of the number of the first unused entry where the pair could be inserted. It uses the utility routine `DoubleHashSetEntryMatches` to compare entries to the sought pair.

The utility routine `DoubleHashSetCheckResize` checks whether the dictionary has become too full after inserting a pair, and expands it if so.

```
[ DoubleHashSetRelationHandler rel task X Y sym  kov kx ky at tmp v;
    kov = BlkValueRead(rel, RRV_KIND);
    kx = KindBaseTerm(kov, 0); ky = KindBaseTerm(kov, 1);
    if (task == RELS_SET_VALENCY) {
        return RELATION_TY_SetValency(rel, X);
    } else if (task == RELS_DESTROY) {
        ! clear
        kx = KOVIsBlockValue(kx); ky = KOVIsBlockValue(ky);
        if (~~(kx || ky)) return;
        for (at = BlkValueRead(rel, RRV_STORAGE); at >= 0; at--) {
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
            if (tmp & RRF_USED) {
                if (kx) BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1));
                if (ky) BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2));
            }
            at--;
        }
        return;
    } else if (task == RELS_COPY) {
        X = KOVIsBlockValue(kx); Y = KOVIsBlockValue(ky);
        if (~~(X || Y)) return;
        at = BlkValueRead(rel, RRV_STORAGE);
        while (at >= 0) {
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
            if (tmp & RRF_USED) {
                if (X) {
                    tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1);
                    tmp = BlkValueCopy(BlkValueCreate(kx), tmp);
                    BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, tmp);
                }
                if (Y) {
                    tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
                    tmp = BlkValueCopy(BlkValueCreate(ky), tmp);
                    BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, tmp);
                }
            }
            at--;
        }
        return;
    } else if (task == RELS_SHOW) {
        print (string) BlkValueRead(rel, RRV_DESCRIPTION), ":^";
        if (sym) {
            kov = KOVComparisonFunction(kx);
            if (~~kov) kov = UnsignedCompare;
        }
        for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
            if (tmp & RRF_USED) {
                X = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1);
                Y = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
                if (sym && (kov(X, Y) > 0)) continue;
                print "  ";
                PrintKindValuePair(kx, X);
                if (sym) print " <=> "; else print " >=> ";
                PrintKindValuePair(ky, Y);
                print "^";
            }
        }
        return;
    } else if (task == RELS_EMPTY) {
        if (BlkValueRead(rel, RRV_USED) == 0) rtrue;
        if (X == 1) {
            DoubleHashSetRelationHandler(rel, RELS_DESTROY);
            for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                tmp = RRV_DATA_BASE + 3*at;
                BlkValueWrite(rel, tmp, 0);
                BlkValueWrite(rel, tmp + 1, 0);
                BlkValueWrite(rel, tmp + 2, 0);
            }
            BlkValueWrite(rel, RRV_USED, 0);
            BlkValueWrite(rel, RRV_FILLED, 0);
            rtrue;
        }
        rfalse;
    } else if (task == RELS_LOOKUP_ANY) {
        for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
            tmp = RRV_DATA_BASE + 3*at;
            if (BlkValueRead(rel, tmp) & RRF_USED) {
                if (Y == RLANY_GET_X or RLANY_CAN_GET_X) {
                    v = BlkValueRead(rel, tmp + 2);
                    if (KOVIsBlockValue(ky)) {
                        if (BlkValueCompare(v, X) ~= 0) continue;
                    } else {
                        if (v ~= X) continue;
                    }
                    if (Y == RLANY_CAN_GET_X) rtrue;
                    return BlkValueRead(rel, tmp + 1);
                } else {
                    v = BlkValueRead(rel, tmp + 1);
                    if (KOVIsBlockValue(kx)) {
                        if (BlkValueCompare(v, X) ~= 0) continue;
                    } else {
                        if (v ~= X) continue;
                    }
                    if (Y == RLANY_CAN_GET_Y) rtrue;
                    return BlkValueRead(rel, tmp + 2);
                }
            }
        }
        if (Y == RLANY_GET_X or RLANY_GET_Y)
            print "*** Lookup failed: value not found ***^";
        rfalse;
    } else if (task == RELS_LOOKUP_ALL_X) {
        if (BlkType(Y) ~= LIST_OF_TY) rfalse;
        LIST_OF_TY_SetLength(Y, 0);
        for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
            tmp = RRV_DATA_BASE + 3*at;
            if (BlkValueRead(rel, tmp) & RRF_USED) {
                v = BlkValueRead(rel, tmp + 2);
                if (KOVIsBlockValue(ky)) {
                    if (BlkValueCompare(v, X) ~= 0) continue;
                } else {
                    if (v ~= X) continue;
                }
                LIST_OF_TY_InsertItem(Y, BlkValueRead(rel, tmp + 1));
            }
        }
        return Y;
    } else if (task == RELS_LOOKUP_ALL_Y) {
        if (BlkType(Y) ~= LIST_OF_TY) rfalse;
        LIST_OF_TY_SetLength(Y, 0);
        for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
            tmp = RRV_DATA_BASE + 3*at;
            if (BlkValueRead(rel, tmp) & RRF_USED) {
                v = BlkValueRead(rel, tmp + 1);
                if (KOVIsBlockValue(kx)) {
```

```
                if (BlkValueCompare(v, X) ~= 0) continue;
            } else {
                if (v ~= X) continue;
            }
            LIST_OF_TY_InsertItem(Y, BlkValueRead(rel, tmp + 2));
        }
    }
    return Y;
} else if (task == RELS_LIST) {
    if (X == 0 || BlkType(X) ~= LIST_OF_TY) rfalse;
    LIST_OF_TY_SetLength(X, 0);
    switch (Y) {
        RLIST_ALL_X, RLIST_ALL_Y:
            for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                tmp = RRV_DATA_BASE + 3*at;
                if (BlkValueRead(rel, tmp) & RRF_USED) {
                    tmp++;
                    if (Y == RLIST_ALL_Y) tmp++;
                    v = BlkValueRead(rel, tmp);
                    LIST_OF_TY_InsertItem(X, v, false, 0, true);
                }
            }
            return X;
        RLIST_ALL_PAIRS:
            ! LIST_OF_TY_InsertItem will make a deep copy of the item,
            ! so we can reuse a single combination value here
            Y = BlkValueCreate(COMBINATION_TY, 0, kov);
            for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                tmp = RRV_DATA_BASE + 3*at;
                if (BlkValueRead(rel, tmp) & RRF_USED) {
                    v = BlkValueRead(rel, tmp + 1);
                    BlkValueWrite(Y, COMBINATION_ITEM_BASE, v);
                    v = BlkValueRead(rel, tmp + 2);
                    BlkValueWrite(Y, COMBINATION_ITEM_BASE + 1, v);
                    LIST_OF_TY_InsertItem(X, Y);
                }
            }
            BlkValueWrite(Y, COMBINATION_ITEM_BASE, 0);
            BlkValueWrite(Y, COMBINATION_ITEM_BASE + 1, 0);
            BlkFree(Y);
            return X;
    }
    rfalse;
}
at = DoubleHashSetLookUp(rel, kx, ky, X, Y);
switch(task) {
    RELS_TEST:
        if (at >= 0) rtrue;
        rfalse;
    RELS_ASSERT_TRUE:
        if (at >= 0) rtrue;
        at = ~at;
        BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) + 1);
```

```
            if (BlkValueRead(rel, RRV_DATA_BASE + 3*at) == 0)
                BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_FILLED) + 1);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at, RRF_USED+RRF_SINGLE);
            if (KOVIsBlockValue(kx)) { X = BlkValueCopy(BlkValueCreate(kx), X); }
            if (KOVIsBlockValue(ky)) { Y = BlkValueCopy(BlkValueCreate(ky), Y); }
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, X);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, Y);
            DoubleHashSetCheckResize(rel);
            rtrue;
        RELS_ASSERT_FALSE:
            if (at < 0) rtrue;
            BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) - 1);
            if (KOVIsBlockValue(kx))
                BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1));
            if (KOVIsBlockValue(ky))
                BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2));
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at, RRF_DELETED);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, 0);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, 0);
            rtrue;
    }
];
[ DoubleHashSetLookUp rel kx ky X Y  hashv i free mask perturb flags;
    ! calculate a hash value for the pair
    hashv = KOVHashValue(kx, x) + KOVHashValue(ky, y);
    ! look in the first expected slot
    mask = BlkValueRead(rel, RRV_STORAGE);
    i = hashv & mask;
    flags = BlkValueRead(rel, RRV_DATA_BASE + 3*i);
    if (flags == 0) return ~i;
    if (DoubleHashSetEntryMatches(rel, i, kx, ky, X, Y)) return i;
    ! not here, keep looking in sequence
    free = -1;
    if (flags & RRF_DELETED) free = i;
    perturb = hashv;
    hashv = i;
    for (::) {
        hashv = hashv*5 + perturb + 1;
        i = hashv & mask;
        flags = BlkValueRead(rel, RRV_DATA_BASE + 3*i);
        if (flags == 0) {
            if (free >= 0) return ~free;
            return ~i;
        }
        if (DoubleHashSetEntryMatches(rel, i, kx, ky, X, Y))
            return i;
        if ((free < 0) && (flags & RRF_DELETED)) free = i;
        #ifdef TARGET_ZCODE;
        @log_shift perturb (-RRP_PERTURB_SHIFT) -> perturb;
        #ifnot;
        @ushiftr perturb RRP_PERTURB_SHIFT perturb;
        #endif;
    }
```

```
];
[ DoubleHashSetCheckResize rel  filled ext newext temp i at kov kx ky F X Y;
    filled = BlkValueRead(rel, RRV_FILLED);
    ext = BlkValueRead(rel, RRV_STORAGE) + 1;
    if (filled >= (ext - filled) * RRP_CROWDED_IS) {
        ! copy entries to temporary space
        temp = BlkAllocate(ext * (3*WORDSIZE), INDEXED_TEXT_TY, BLK_FLAG_WORD+BLK_FLAG_MULTIPLE);
        for (i=0: i<ext*3: i++)
            BlkValueWrite(temp, i, BlkValueRead(rel, RRV_DATA_BASE+i));
        ! resize and clear our data
        if (ext >= RRP_LARGE_IS) newext = ext * RRP_RESIZE_LARGE;
        else newext = ext * RRP_RESIZE_SMALL;
        BlkValueSetExtent(rel, RRV_DATA_BASE + newext*3);
        BlkValueWrite(rel, RRV_STORAGE, newext - 1);
        BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_USED));
        for (i=0: i<newext*3: i++)
            BlkValueWrite(rel, RRV_DATA_BASE+i, 0);
        ! copy entries back from temporary space
        kov = BlkValueRead(rel, RRV_KIND);
        kx = KindBaseTerm(kov, 0); ky = KindBaseTerm(kov, 1);
        for (i=0: i<ext: i++) {
            F = BlkValueRead(temp, 3*i);
            if (F == 0 || (F & RRF_DELETED)) continue;
            X = BlkValueRead(temp, 3*i + 1);
            Y = BlkValueRead(temp, 3*i + 2);
            at = DoubleHashSetLookUp(rel, kx, ky, X, Y);
            if (at >= 0) { print "*** Duplicate entry while resizing ***^"; rfalse; }
            at = ~at;
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at, F);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, X);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, Y);
        }
        ! done with temporary space
        BlkFree(temp);
    }
];
[ DoubleHashSetEntryMatches rel at kx ky X Y  cx cy;
    cx = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1);
    if (KOVIsBlockValue(kx)) {
        if (BlkValueCompare(cx, X) ~= 0) rfalse;
    } else {
        if (cx ~= X) rfalse;
    }
    cy = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
    if (KOVIsBlockValue(ky)) {
        if (BlkValueCompare(cy, Y) ~= 0) rfalse;
    } else {
        if (cy ~= Y) rfalse;
    }
    rtrue;
];
```

§16. Hash List Relation Handler. This implements relations which are stored as a hash table mapping keys to either single values or lists of values. The storage comprises a list of three-word entries, either (F, X, Y) or (F, X, L), where F is a flags word distinguishing between the two cases (among other things). In the latter case, L is a pointer to a list (`LIST_OF_TY`) containing the values.

The "storage", "used", and "filled" words have the same meanings as above.

`HashListRelationHandler` is a thin wrapper around `HashCoreRelationHandler`, which is shared with two other handlers below.

```
[ HashListRelationHandler rel task X Y  sym kov kx ky;
    kov = BlkValueRead(rel, RRV_KIND);
    kx = KindBaseTerm(kov, 0); ky = KindBaseTerm(kov, 1);
    return HashCoreRelationHandler(rel, task, kx, ky, X, Y, 1);
];
```

§17. Hash Table Relation Handler. This is the same as the Hash List Relation Handler above, except that only one value may be stored for each key. This implements various-to-one relations.

```
[ HashTableRelationHandler rel task X Y  kov kx ky;
    kov = BlkValueRead(rel, RRV_KIND);
    kx = KindBaseTerm(kov, 0); ky = KindBaseTerm(kov, 1);
    return HashCoreRelationHandler(rel, task, kx, ky, X, Y, 0);
];
```

§18. Reversed Hash Table Relation Handler. This is the same as the Hash Table Relation Handler except that the sides are reversed. This implements one-to-various relations.

```
[ ReversedHashTableRelationHandler rel task X Y  kov kx ky;
    kov = BlkValueRead(rel, RRV_KIND);
    kx = KindBaseTerm(kov, 0); ky = KindBaseTerm(kov, 1);
    switch (task) {
        RELS_SET_VALENCY:
            return RELATION_TY_SetValency(rel, X);
        RELS_TEST, RELS_ASSERT_TRUE, RELS_ASSERT_FALSE:
            return HashCoreRelationHandler(rel, task, ky, kx, Y, X, 0);
        RELS_LOOKUP_ANY:
            switch (Y) {
                RLANY_GET_X: Y = RLANY_GET_Y;
                RLANY_GET_Y: Y = RLANY_GET_X;
                RLANY_CAN_GET_X: Y = RLANY_CAN_GET_Y;
                RLANY_CAN_GET_Y: Y = RLANY_CAN_GET_X;
            }
        RELS_LOOKUP_ALL_X:
            task = RELS_LOOKUP_ALL_Y;
        RELS_LOOKUP_ALL_Y:
            task = RELS_LOOKUP_ALL_X;
        RELS_SHOW:
        RELS_LIST:
            switch (Y) {
                RLIST_ALL_X: Y = RLIST_ALL_Y;
                RLIST_ALL_Y: Y = RLIST_ALL_X;
            }
    }
    return HashCoreRelationHandler(rel, task, kx, ky, X, Y, 0);
];
```

§**19. Symmetric Relation Handlers.** These are simple wrappers around the asymmetric handlers defined above. When a pair is inserted or removed, the wrappers insert or remove the reversed pair as well.

`SymDoubleHashSetRelationHandler` and `SymHashListRelationHandler` implement symmetric V-to-V relations. `Sym2in1HashTableRelationHandler` implements symmetric 1-to-1. ("`SymTwoInOneHashTableRelationHandler`" would have exceeded Inform 6's 32-character name limit.)

```
[ SymDoubleHashSetRelationHandler rel task X Y;
    if (task == RELS_ASSERT_TRUE or RELS_ASSERT_FALSE)
        DoubleHashSetRelationHandler(rel, task, Y, X);
    return DoubleHashSetRelationHandler(rel, task, X, Y, 1);
];
[ SymHashListRelationHandler rel task X Y;
    if (task == RELS_ASSERT_TRUE or RELS_ASSERT_FALSE)
        HashListRelationHandler(rel, task, Y, X);
    return HashListRelationHandler(rel, task, X, Y);
];
[ Sym2in1HashTableRelationHandler rel task X Y;
    if (task == RELS_ASSERT_TRUE or RELS_ASSERT_FALSE)
        TwoInOneHashTableRelationHandler(rel, task, Y, X);
    return TwoInOneHashTableRelationHandler(rel, task, X, Y, 1);
];
```

§**20. Hash Core Relation Handler.** This implements the core functionality that is shared between `HashListRelationHandler`, `HashTableRelationHandler`, and `ReversedHashTableRelationHandler`. All three handlers are the same except for whether the left or right side is the "key" and whether or not multiple values may be stored for a single key.

As noted above, the table contains three-word entries, (F, X, Y), where F is a flags word. Only the hash code of X is used. If F includes `RRF_SINGLE`, Y is a single value; otherwise, Y is a list (`LIST_OF_TY`) of values. If `mult` is zero, `RRF_SINGLE` must always be set, allowing only one value per key: a new pair (X, Y') will replace the existing pair (X, Y).

```
[ HashCoreRelationHandler rel task kx ky X Y mult  sym rev at tmp fl;
    if (task == RELS_SET_VALENCY) {
        return RELATION_TY_SetValency(rel, X);
    } else if (task == RELS_DESTROY) {
        ! clear
        kx = KOVIsBlockValue(kx); ky = KOVIsBlockValue(ky);
        if (~~(kx || ky)) return;
        at = BlkValueRead(rel, RRV_STORAGE);
        while (at >= 0) {
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
            if (fl & RRF_USED) {
                if (kx) BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1));
                if (ky || ~~(fl & RRF_SINGLE))
                    BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2));
            }
            at--;
        }
        return;
    } else if (task == RELS_COPY) {
        X = KOVIsBlockValue(kx); Y = KOVIsBlockValue(ky);
        if (~~(X || Y)) return;
        at = BlkValueRead(rel, RRV_STORAGE);
        while (at >= 0) {
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
            if (fl & RRF_USED) {
                if (X) {
                    tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1);
                    tmp = BlkValueCopy(BlkValueCreate(kx), tmp);
                    BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, tmp);
                }
                if (Y || ~~(fl & RRF_SINGLE)) {
                    tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
                    tmp = BlkValueCopy(BlkValueCreate(BlkType(tmp)), tmp);
                    BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, tmp);
                }
            }
            at--;
        }
        return;
    } else if (task == RELS_SHOW) {
        print (string) BlkValueRead(rel, RRV_DESCRIPTION), ":^";
        ! Z-machine doesn't have the room to let us pass sym/rev as parameters
        switch (RELATION_TY_GetValency(rel)) {
            RRVAL_SYM_V_TO_V:
                sym = 1;
                tmp = KOVComparisonFunction(kx);
                if (~~tmp) tmp = UnsignedCompare;
            RRVAL_O_TO_V:
                rev = 1;
        }
        for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
            if (fl & RRF_USED) {
                X = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1);
                Y = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
                if (fl & RRF_SINGLE) {
                    if (sym && tmp(X, Y) > 0) continue;
                    print "  ";
                    if (rev) PrintKindValuePair(ky, Y);
                    else PrintKindValuePair(kx, X);
                    if (sym) print " <=> "; else print " >=> ";
                    if (rev) PrintKindValuePair(kx, X);
                    else PrintKindValuePair(ky, Y);
                    print "^";
                } else {
                    for (mult=1: mult<=LIST_OF_TY_GetLength(Y): mult++) {
                        fl = LIST_OF_TY_GetItem(Y, mult);
                        if (sym && tmp(X, fl) > 0) continue;
                        print "  ";
                        if (rev) PrintKindValuePair(ky, fl);
                        else PrintKindValuePair(kx, X);
                        if (sym) print " <=> "; else print " >=> ";
                        if (rev) PrintKindValuePair(kx, X);
                        else PrintKindValuePair(ky, fl);
                        print "^";
                    }
                }
            }
        }
        return;
   } else if (task == RELS_EMPTY) {
        if (BlkValueRead(rel, RRV_USED) == 0) rtrue;
        if (X == 1) {
            HashCoreRelationHandler(rel, RELS_DESTROY);
            for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                tmp = RRV_DATA_BASE + 3*at;
                BlkValueWrite(rel, tmp, 0);
                BlkValueWrite(rel, tmp + 1, 0);
                BlkValueWrite(rel, tmp + 2, 0);
            }
            BlkValueWrite(rel, RRV_USED, 0);
            BlkValueWrite(rel, RRV_FILLED, 0);
            rtrue;
        }
        rfalse;
   } else if (task == RELS_LOOKUP_ANY) {
        if (Y == RLANY_GET_Y or RLANY_CAN_GET_Y) {
            at = HashCoreLookUp(rel, kx, X);
            if (at >= 0) {
                if (Y == RLANY_CAN_GET_Y) rtrue;
                tmp = RRV_DATA_BASE + 3*at;
                fl = BlkValueRead(rel, tmp);
                tmp = BlkValueRead(rel, tmp + 2);
                if (fl & RRF_SINGLE) return tmp;
                return LIST_OF_TY_GetItem(tmp, 1);
            }
        } else {
            for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                tmp = RRV_DATA_BASE + 3*at;
                fl = BlkValueRead(rel, tmp);
                if (fl & RRF_USED) {
                    sym = BlkValueRead(rel, tmp + 2);
                    if (fl & RRF_SINGLE) {
                        if (KOVIsBlockValue(ky)) {
                            if (BlkValueCompare(X, sym) ~= 0) continue;
                        } else {
                            if (X ~= sym) continue;
                        }
                    } else {
                        if (LIST_OF_TY_FindItem(sym, X) == 0) continue;
                    }
                    if (Y == RLANY_CAN_GET_X) rtrue;
                    return BlkValueRead(rel, tmp + 1);
                }
            }
        }
        if (Y == RLANY_GET_X or RLANY_GET_Y)
            print "*** Lookup failed: value not found ***^";
        rfalse;
    } else if (task == RELS_LOOKUP_ALL_X) {
        if (BlkType(Y) ~= LIST_OF_TY) rfalse;
        LIST_OF_TY_SetLength(Y, 0);
        for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
            tmp = RRV_DATA_BASE + 3*at;
            fl = BlkValueRead(rel, tmp);
            if (fl & RRF_USED) {
                sym = BlkValueRead(rel, tmp + 2);
                if (fl & RRF_SINGLE) {
                    if (KOVIsBlockValue(ky)) {
                        if (BlkValueCompare(X, sym) ~= 0) continue;
                    } else {
                        if (X ~= sym) continue;
                    }
                } else {
                    if (LIST_OF_TY_FindItem(sym, X) == 0) continue;
                }
                LIST_OF_TY_InsertItem(Y, BlkValueRead(rel, tmp + 1));
            }
        }
        return Y;
    } else if (task == RELS_LOOKUP_ALL_Y) {
        if (BlkType(Y) ~= LIST_OF_TY) rfalse;
        LIST_OF_TY_SetLength(Y, 0);
        at = HashCoreLookUp(rel, kx, X);
        if (at >= 0) {
            tmp = RRV_DATA_BASE + 3*at;
            fl = BlkValueRead(rel, tmp);
            tmp = BlkValueRead(rel, tmp + 2);
            if (fl & RRF_SINGLE)
                LIST_OF_TY_InsertItem(Y, tmp);
            else
                LIST_OF_TY_AppendList(Y, tmp);
        }
        return Y;
    } else if (task == RELS_LIST) {
        if (BlkType(X) ~= LIST_OF_TY) rfalse;
        LIST_OF_TY_SetLength(X, 0);
        switch (Y) {
            RLIST_ALL_X:
                for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                    tmp = RRV_DATA_BASE + 3*at;
                    fl = BlkValueRead(rel, tmp);
                    if (fl & RRF_USED)
                        LIST_OF_TY_InsertItem(X, BlkValueRead(rel, tmp + 1));
                }
                return X;
            RLIST_ALL_Y:
                for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                    tmp = RRV_DATA_BASE + 3*at;
                    fl = BlkValueRead(rel, tmp);
                    if (fl & RRF_USED) {
                        tmp = BlkValueRead(rel, tmp + 2);
                        if (fl & RRF_SINGLE)
                            LIST_OF_TY_InsertItem(X, tmp, false, 0, true);
                        else
                            LIST_OF_TY_AppendList(X, tmp, false, 0, true);
                    }
                }
                return X;
            RLIST_ALL_PAIRS:
                if (RELATION_TY_GetValency(rel) == RRVAL_O_TO_V) rev = 1;
                ! LIST_OF_TY_InsertItem will make a deep copy of the item,
                ! so we can reuse a single combination value here
                Y = BlkValueCreate(COMBINATION_TY, 0, tmp);
                for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                    tmp = RRV_DATA_BASE + 3*at;
                    fl = BlkValueRead(rel, tmp);
                    if (fl & RRF_USED) {
                        BlkValueWrite(Y, COMBINATION_ITEM_BASE + rev, BlkValueRead(rel, tmp + 1));
                        tmp = BlkValueRead(rel, tmp + 2);
                        if (fl & RRF_SINGLE) {
                            BlkValueWrite(Y, COMBINATION_ITEM_BASE + 1 - rev, tmp);
                            LIST_OF_TY_InsertItem(X, Y);
                        } else {
                            for (mult = LIST_OF_TY_GetLength(tmp): mult > 0: mult--) {
                                BlkValueWrite(Y, COMBINATION_ITEM_BASE + 1 - rev,
                                    LIST_OF_TY_GetItem(tmp, mult));
                                LIST_OF_TY_InsertItem(X, Y);
                            }
                        }
                    }
                }
                BlkValueWrite(Y, COMBINATION_ITEM_BASE, 0);
                BlkValueWrite(Y, COMBINATION_ITEM_BASE + 1, 0);
                BlkFree(Y);
                return X;
        }
        rfalse;
    }
    at = HashCoreLookUp(rel, kx, X);
    switch(task) {
        RELS_TEST:
            if (at < 0) rfalse;
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
            if (fl & RRF_SINGLE) {
                if (KOVIsBlockValue(ky)) {
                    if (BlkValueCompare(tmp, Y) == 0) rtrue;
                } else {
                    if (tmp == Y) rtrue;
                }
                rfalse;
            } else {
                return LIST_OF_TY_FindItem(tmp, Y);
            }
        RELS_ASSERT_TRUE:
            if (at < 0) {
                ! no entry exists for this key, just add one
                at = ~at;
                BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) + 1);
                if (BlkValueRead(rel, RRV_DATA_BASE + 3*at) == 0)
                    BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_FILLED) + 1);
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at, RRF_USED+RRF_SINGLE);
                if (KOVIsBlockValue(kx)) { X = BlkValueCopy(BlkValueCreate(kx), X); }
                if (KOVIsBlockValue(ky)) { Y = BlkValueCopy(BlkValueCreate(ky), Y); }
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, X);
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, Y);
                HashCoreCheckResize(rel);
                break;
            }
            ! an entry exists: could be a list or a single value
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at); ! flags
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2); ! value or list
            if (fl & RRF_SINGLE) {
                ! if Y is the same as the stored key, we have nothing to do
                if (KOVIsBlockValue(ky)) {
                    if (BlkValueCompare(tmp, Y) == 0) rtrue;
                } else {
                    if (tmp == Y) rtrue;
                }
                ! it's different: either replace it or expand into a list,
                ! depending on the value of mult
                if (mult) {
                    fl = LIST_OF_TY_Create(UNKNOWN_TY); ! new list
                    BlkValueWrite(fl, LIST_ITEM_KOV_F, ky);
                    LIST_OF_TY_SetLength(fl, 2);
                    BlkValueWrite(fl, LIST_ITEM_BASE, tmp); ! do not copy
                    LIST_OF_TY_PutItem(fl, 2, Y); ! copy if needed
                    BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, fl);
                    BlkValueWrite(rel, RRV_DATA_BASE + 3*at, RRF_USED);
                } else {
                    if (KOVIsBlockValue(ky)) {
                        BlkFree(tmp);
                        Y = BlkValueCopy(BlkValueCreate(ky), Y);
                    }
                    BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, Y);
                }
            } else {
                ! if Y is present already, do nothing. otherwise add it.
                LIST_OF_TY_InsertItem(tmp, Y, 0, 0, 1);
            }
            rtrue;
        RELS_ASSERT_FALSE:
            if (at < 0) rtrue;
            ! an entry exists: could be a list or a single value
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at); ! flags
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2); ! value or list
            if (fl & RRF_SINGLE) {
                ! if the stored key isn't Y, we have nothing to do
                if (KOVIsBlockValue(ky)) {
                    if (BlkValueCompare(tmp, Y) ~= 0) rtrue;
                } else {
                    if (tmp ~= Y) rtrue;
                }
                ! delete the entry
                if (KOVIsBlockValue(ky))
                    BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2));
                .DeleteEntryIgnoringY;
                BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) - 1);
                if (KOVIsBlockValue(kx))
                    BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1));
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at, RRF_DELETED);
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, 0);
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, 0);
            } else {
                ! remove Y from the list if present
                LIST_OF_TY_RemoveValue(tmp, Y, 1);
                ! if the list is now empty, delete the whole entry
                if (LIST_OF_TY_GetLength(tmp) == 0) {
                    BlkFree(tmp);
                    jump DeleteEntryIgnoringY;
                }
            }
            rtrue;
    }
    rtrue;
];
[ HashCoreLookUp rel kx X  hashv i free mask perturb flags;
!print "[HCLU rel=", rel, " kx=", kx, " X=", X, ": ";
    ! calculate a hash value for the key
    hashv = KOVHashValue(kx, x);
    ! look in the first expected slot
    mask = BlkValueRead(rel, RRV_STORAGE);
    i = hashv & mask;
!print "hv=", hashv, ", trying ", i;
    if (HashCoreEntryMatches(rel, i, kx, X)) {
!print " - found]^";
        return i;
    }
    flags = BlkValueRead(rel, RRV_DATA_BASE + 3*i);
    if (flags == 0) {
!print " - not found]^";
        return ~i;
    }
    ! not here, keep looking in sequence
    free = -1;
    if (flags & RRF_DELETED) free = i;
    perturb = hashv;
    hashv = i;
    for (::) {
        hashv = hashv*5 + perturb + 1;
        i = hashv & mask;
!print ", ", i;
        flags = BlkValueRead(rel, RRV_DATA_BASE + 3*i);
        if (flags == 0) {
!print " - not found]^";
            if (free >= 0) return ~free;
            return ~i;
        }
        if (HashCoreEntryMatches(rel, i, kx, X)) {
!print " - found]^";
            return i;
        }
        if ((free < 0) && (flags & RRF_DELETED)) free = i;
        #ifdef TARGET_ZCODE;
        @log_shift perturb (-RRP_PERTURB_SHIFT) -> perturb;
        #ifnot;
        @ushiftr perturb RRP_PERTURB_SHIFT perturb;
        #endif;
    }
];
[ HashCoreCheckResize rel  filled ext newext temp i at kov kx F X Y;
    filled = BlkValueRead(rel, RRV_FILLED);
    ext = BlkValueRead(rel, RRV_STORAGE) + 1;
    if (filled >= (ext - filled) * RRP_CROWDED_IS) {
        ! copy entries to temporary space
        temp = BlkAllocate(ext * (3*WORDSIZE), INDEXED_TEXT_TY, BLK_FLAG_WORD+BLK_FLAG_MULTIPLE);
        for (i=0: i<ext*3: i++)
            BlkValueWrite(temp, i, BlkValueRead(rel, RRV_DATA_BASE+i));
        ! resize and clear our data
        if (ext >= RRP_LARGE_IS) newext = ext * RRP_RESIZE_LARGE;
        else newext = ext * RRP_RESIZE_SMALL;
        BlkValueSetExtent(rel, RRV_DATA_BASE + newext*3);
        BlkValueWrite(rel, RRV_STORAGE, newext - 1);
        BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_USED));
        for (i=0: i<newext*3: i++)
            BlkValueWrite(rel, RRV_DATA_BASE+i, 0);
        ! copy entries back from temporary space
        kov = BlkValueRead(rel, RRV_KIND);
        kx = KindBaseTerm(kov, 0);
        for (i=0: i<ext: i++) {
            F = BlkValueRead(temp, 3*i);
            if (F == 0 || (F & RRF_DELETED)) continue;
            X = BlkValueRead(temp, 3*i + 1);
            Y = BlkValueRead(temp, 3*i + 2);
            at = HashCoreLookUp(rel, kx, X);
            if (at >= 0) { print "*** Duplicate entry while resizing ***^"; rfalse; }
            at = ~at;
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at, F);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, X);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, Y);
        }
        ! done with temporary space
        BlkFree(temp);
    }
];

[ HashCoreEntryMatches rel at kx X  cx cy;
    cx = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1);
    if (KOVIsBlockValue(kx)) {
        if (BlkValueCompare(cx, X) ~= 0) rfalse;
    } else {
        if (cx ~= X) rfalse;
    }
    rtrue;
];
```

§**21. Equivalence Hash Table Relation Handler.** This implements group relations. The table format is identical to that used by `HashCoreRelationHandler`, but we use it differently. Although the relation appears to relate Xs to Xs as far as the game is concerned, the table actually relates Xs to numbers, where each number identifies a group of related items. Any X not listed in the table is implicitly in a single-member group.

When a pair (X, Y) is inserted, one of four cases occurs:

1. Neither X nor Y has a table entry. We search the table to find the next unused group number, then add both X and Y to that group.

2. Both X and Y have existing table entries. If the group numbers differ, we walk through the table and change all occurrences of the higher number to the lower one.

3. X has an existing table entry but Y does not. We add a Y entry using the group number of X.

4. Y has an existing table entry but X does not. We add an X entry using the group number of Y.

When a pair (X, Y) is removed, we first verify that X and Y are in the same group, then delete the table entry for X. This may leave Y in a single-member group, which could be deleted, but detecting that situation would be inefficient, so we keep the Y entry regardless.

This code uses the Hash Core utility functions defined above.

```
[ EquivHashTableRelationHandler rel task X Y  kx at at2 tmp fl i ext;
    kx = KindBaseTerm(BlkValueRead(rel, RRV_KIND), 0);
    if (task == RELS_SET_VALENCY) {
        return RELATION_TY_SetValency(rel, X);
    } else if (task == RELS_DESTROY) {
        ! clear
        if (KOVIsBlockValue(kx)) {
            at = BlkValueRead(rel, RRV_STORAGE);
            while (at >= 0) {
                fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
                if (fl & RRF_USED) {
                    BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1));
                }
                at--;
            }
        }
        return;
    } else if (task == RELS_COPY) {
        if (KOVIsBlockValue(kx)) {
            at = BlkValueRead(rel, RRV_STORAGE);
            while (at >= 0) {
                fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
                if (fl & RRF_USED) {
                    tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1);
                    tmp = BlkValueCopy(BlkValueCreate(kx), tmp);
                    BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1);
                }
                at--;
            }
        }
        return;
    } else if (task == RELS_SHOW) {
        print (string) BlkValueRead(rel, RRV_DESCRIPTION), ":^";
        ext = BlkValueRead(rel, RRV_STORAGE);
        ! flag all items by negating their group numbers
        for (at=0, X=RRV_DATA_BASE: at<=ext: at++, X=X+3)
            if (BlkValueRead(rel, X) & RRF_USED)
                BlkValueWrite(rel, X + 2, -(BlkValueRead(rel, X + 2)));
        ! display groups, unflagging them as we go
        for (at=0, X=RRV_DATA_BASE, fl=0: at<=ext: at++, X=X+3, fl=0) {
            if (BlkValueRead(rel, X) & RRF_USED) {
                fl = BlkValueRead(rel, X + 2);
                if (fl > 0) continue; ! already visited
                BlkValueWrite(rel, X + 2, -fl); ! unflag it
                ! display the group starting with this member, but only
                ! if there are more members in the group
                tmp = BlkValueRead(rel, X + 1);
                i = 0;
                for (at2=at+1, Y=RRV_DATA_BASE+3*at2: at2<=ext: at2++, Y=Y+3) {
                    if (BlkValueRead(rel, Y) & RRF_USED) {
                        if (BlkValueRead(rel, Y + 2) ~= fl) continue;
                        BlkValueWrite(rel, Y + 2, -fl);
                        if (~~i) {
                            ! print the saved first member
                            print "  { ";
                            PrintKindValuePair(kx, tmp);
                            i = 1;
                        }
                        print ", ";
                        PrintKindValuePair(kx, BlkValueRead(rel, Y + 1));
                    }
                }
                if (i) print " }^";
            }
        }
        return;
    } else if (task == RELS_EMPTY) {
        ! never empty since R(x,x) is always true
        rfalse;
```

```
} else if (task == RELS_LOOKUP_ANY) {
    ! kind of a cheat, but it's faster than searching for a better value to return
    if (Y == RLANY_CAN_GET_X or RLANY_CAN_GET_Y) rtrue;
    return X;
} else if (task == RELS_LOOKUP_ALL_X or RELS_LOOKUP_ALL_Y) {
    if (BlkType(Y) ~= LIST_OF_TY) rfalse;
    LIST_OF_TY_SetLength(Y, 0);
    BlkValueWrite(Y, LIST_ITEM_KOV_F, kx);
    at = HashCoreLookUp(rel, kx, X);
    if (at < 0) {
        LIST_OF_TY_InsertItem(Y, X);
    } else {
        X = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
        for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
            tmp = RRV_DATA_BASE + 3*at;
            fl = BlkValueRead(rel, tmp);
            if (fl & RRF_USED) {
                if (BlkValueRead(rel, tmp + 2) ~= X) continue;
                LIST_OF_TY_InsertItem(Y, BlkValueRead(rel, tmp + 1));
            }
        }
    }
    return Y;
} else if (task == RELS_LIST) {
    print "*** Domains of equivalence relations cannot be listed ***^";
    return X;
}
at = HashCoreLookUp(rel, kx, X);
at2 = HashCoreLookUp(rel, kx, Y);
switch(task) {
    RELS_TEST:
        if (at < 0) {
            ! X is a loner, but could still be true if X == Y
            if (KOVIsBlockValue(kx)) {
                if (BlkValueCompare(X, Y) == 0) rtrue;
            } else {
                if (X == Y) rtrue;
            }
            rfalse;
        }
        if (at2 < 0) rfalse;
        if (at == at2) rtrue;
        tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
        if (BlkValueRead(rel, RRV_DATA_BASE + 3*at2 + 2) == tmp) rtrue;
        rfalse;
    RELS_ASSERT_TRUE:
        ! if X and Y are the same, we have nothing to do
        if (KOVIsBlockValue(kx)) {
            if (BlkValueCompare(X, Y) == 0) rtrue;
        } else {
            if (X == Y) rtrue;
        }
        if (at < 0) {
```

```
        if (at2 < 0) {
            ! X and Y both missing: find a new group number and add both entries
            tmp = 0; ! candidate group number
            ext = BlkValueRead(rel, RRV_STORAGE);
            for (i=0: i<=ext: i++) {
                fl = BlkValueRead(rel, RRV_DATA_BASE + 3*i);
                if (fl & RRF_USED) {
                    fl = BlkValueRead(rel, RRV_DATA_BASE + 3*i + 2);
                    if (fl > tmp) tmp = fl;
                }
            }
            tmp++; ! new group number
            BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) + 2);
            ! add X entry
            at = ~at;
            if (KOVIsBlockValue(kx)) { X = BlkValueCopy(BlkValueCreate(kx), X); }
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
            if (fl == 0)
                BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_FILLED) + 1);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at, RRF_USED+RRF_SINGLE);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, X);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, tmp);
            ! add Y entry. at2 might change if X and Y have the same hash code.
            at2 = ~(HashCoreLookUp(rel, kx, Y));
            if (KOVIsBlockValue(kx)) { Y = BlkValueCopy(BlkValueCreate(kx), Y); }
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at2);
            if (fl == 0)
                BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_FILLED) + 1);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at2, RRF_USED+RRF_SINGLE);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at2 + 1, Y);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at2 + 2, tmp);
            jump CheckResize;
        }
        ! X missing, Y present: add a new X entry
        at = ~at;
        if (KOVIsBlockValue(kx)) { X = BlkValueCopy(BlkValueCreate(kx), X); }
        BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) + 1);
        fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at);
        if (fl == 0)
            BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_FILLED) + 1);
        BlkValueWrite(rel, RRV_DATA_BASE + 3*at, RRF_USED+RRF_SINGLE);
        BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, X);
        tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at2 + 2);
        BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, tmp);
        jump CheckResize;
    }
    if (at2 < 0) {
        ! X present, Y missing: add a new Y entry
        at2 = ~at2;
        if (KOVIsBlockValue(kx)) { Y = BlkValueCopy(BlkValueCreate(kx), Y); }
        BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) + 1);
        fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at2);
        if (fl == 0)
```

```
                    BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_FILLED) + 1);
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at2, RRF_USED+RRF_SINGLE);
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at2 + 1, Y);
                tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
                BlkValueWrite(rel, RRV_DATA_BASE + 3*at2 + 2, tmp);
                jump CheckResize;
            }
            ! X and Y both present: merge higher group into lower group
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2); ! higher group
            fl = BlkValueRead(rel, RRV_DATA_BASE + 3*at2 + 2); ! lower group
            if (tmp < fl) { i = tmp; tmp = fl; fl = i; }
            ext = BlkValueRead(rel, RRV_STORAGE);
            for (at=0: at<=ext: at++) {
                i = RRV_DATA_BASE + 3*at + 2;
                if (BlkValueRead(rel, i) == tmp)
                    BlkValueWrite(rel, i, fl);
            }
            .CheckResize;
            HashCoreCheckResize(rel);
            rtrue;
        RELS_ASSERT_FALSE:
            ! if X and Y are already in different groups, we have nothing to do
            if (at < 0 || at2 < 0) rtrue;
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 3*at + 2);
            if (BlkValueRead(rel, RRV_DATA_BASE + 3*at2 + 2) ~= tmp) rtrue;
            ! delete the entry for X
            BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) - 1);
            if (KOVIsBlockValue(kx))
                BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 3*at + 1));
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at, RRF_DELETED);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 1, 0);
            BlkValueWrite(rel, RRV_DATA_BASE + 3*at + 2, 0);
            rtrue;
    }
];
```

§**22. Two-In-One Hash Table Relation Handler.** This implements one-to-one relations, which are stored as a hash table mapping keys to single values and vice versa. To enforce the one-to-one constraint, we need the ability to quickly check whether a value is present. This could be done with two separate hash tables, one mapping X to Y and one the opposite, but in the interest of conserving memory, we use a single table for both.

Each four-word entry (F, E, K, V) consists of a flags word F, an entry key E (which may be a "key" or "value" in the hash table sense), a corresponding key K (when E is used as a value), and a corresponding value V (when E is used as a key). The pair of related values (X, Y) is represented as two table entries: $(F, X, , Y)$ and $(F, Y, X,)$.

To conserve memory when block values are used, we only create one copy of X and/or Y to share between both entries. When adding a key or value which already exists on either side of the relation, the previous copy will be used. Copies are freed when they are no longer used as entry keys.

Each entry's flags word F indicates, in addition to the standard flags `RRF_USED` and `RRF_DELETED`, also whether the entry contains a corresponding key K and/or value V (`RRF_HASX`, `RRF_HASY`) and whether the entry's key is the same kind of value as X or Y (`RRF_ENTKEYX`, `RRF_ENTKEYY`). If both sides of the relation use the same kind of value, or if both sides are word values, both `RRF_ENTKEYX` and `RRF_ENTKEYY` will be set on every used entry.

Of particular note for this handler is the utility function `TwoInOneDelete`, which clears one half of an entry (given its entry key), and optionally clears the corresponding other half stored in a different entry. That is, given the entries $(F, X, , Y)$ at index `i` and $(F, Y, X,)$ elsewhere, `TwoInOneDelete(rel, i, kx, ky, RRF_ENTKEYX, 1)` will clear both entries and mark them as deleted. If, however, those entries overlap with other pairs – say they're (F, X, A, Y) and (F, Y, X, B) – then the same call to `TwoInOneDelete` will leave us with $(F, X, A,)$ and $(F, Y, , B)$, having cleared the parts corresponding to the pair (X, Y) but not the parts corresponding to the pairs (A, X) and (Y, B), and will not mark either as deleted. (Such overlap is only possible when the domains of X and Y are the same kind of value.)

```
[ TwoInOneHashTableRelationHandler rel task X Y sym  kov kx ky at at2 tmp fl;
    kov = BlkValueRead(rel, RRV_KIND);
    kx = KindBaseTerm(kov, 0); ky = KindBaseTerm(kov, 1);
    if (task == RELS_SET_VALENCY) {
        return RELATION_TY_SetValency(rel, X);
    } else if (task == RELS_DESTROY) {
        ! clear
        kx = KOVIsBlockValue(kx); ky = KOVIsBlockValue(ky);
        if (~~(kx || ky)) return;
        at = BlkValueRead(rel, RRV_STORAGE);
        while (at >= 0) {
            fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
            if (fl & RRF_USED)
                if ((kx && (fl & RRF_ENTKEYX)) || (ky && (fl & RRF_ENTKEYY))) {
                    BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 4*at + 1));
                }
            at--;
        }
        return;
    } else if (task == RELS_COPY) {
        X = KOVIsBlockValue(kx); Y = KOVIsBlockValue(ky);
        if (~~(X || Y)) return;
        at = BlkValueRead(rel, RRV_STORAGE);
        while (at >= 0) {
            fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
            if (fl & RRF_USED) {
                if ((X && (fl & RRF_ENTKEYX)) || (Y && (fl & RRF_ENTKEYY))) {
                    ! copy the entry key
                    tmp = BlkValueRead(rel, RRV_DATA_BASE + 4*at + 1);
                    if (fl & RRF_ENTKEYX)
                        tmp = BlkValueCopy(BlkValueCreate(kx), tmp);
                    else
                        tmp = BlkValueCopy(BlkValueCreate(ky), tmp);
                    BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 1, tmp);
                    ! update references in X/Y fields pointing here
                    if (fl & RRF_HASX) {
                        at2 = TwoInOneLookUp(rel, kx,
                            BlkValueRead(rel, RRV_DATA_BASE + 4*at + 2),
                            RRF_ENTKEYX);
                        if (at2 >= 0)
                            BlkValueWrite(rel, RRV_DATA_BASE + 4*at2 + 3, tmp);
                    }
                    if (fl & RRF_HASY) {
                        at2 = TwoInOneLookUp(rel, ky,
                            BlkValueRead(rel, RRV_DATA_BASE + 4*at + 3),
                            RRF_ENTKEYY);
                        if (at2 >= 0)
                            BlkValueWrite(rel, RRV_DATA_BASE + 4*at2 + 2, tmp);
                    }
                }
            }
            at--;
        }
        return;
    } else if (task == RELS_SHOW) {
        print (string) BlkValueRead(rel, RRV_DESCRIPTION), ":^";
        if (sym) {
            kov = KOVComparisonFunction(kx);
            if (~~kov) kov = UnsignedCompare;
        }
        for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
            fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
            if ((fl & (RRF_USED+RRF_ENTKEYX+RRF_HASY)) ==
                (RRF_USED+RRF_ENTKEYX+RRF_HASY)) {
                X = BlkValueRead(rel, RRV_DATA_BASE + 4*at + 1);
                Y = BlkValueRead(rel, RRV_DATA_BASE + 4*at + 3);
                if (sym && kov(X, Y) > 0) continue;
                print "  ";
                PrintKindValuePair(kx, X);
                if (sym) print " <=> "; else print " >=> ";
                PrintKindValuePair(ky, Y);
                print "^";
            }
        }
        return;
    } else if (task == RELS_EMPTY) {
        if (BlkValueRead(rel, RRV_USED) == 0) rtrue;
        if (X == 1) {
            TwoInOneHashTableRelationHandler(rel, RELS_DESTROY);
            for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                tmp = RRV_DATA_BASE + 4*at;
                BlkValueWrite(rel, tmp, 0);
                BlkValueWrite(rel, tmp + 1, 0);
                BlkValueWrite(rel, tmp + 2, 0);
                BlkValueWrite(rel, tmp + 3, 0);
            }
            BlkValueWrite(rel, RRV_USED, 0);
            BlkValueWrite(rel, RRV_FILLED, 0);
            rtrue;
        }
        rfalse;
    } else if (task == RELS_LOOKUP_ANY) {
        switch (Y) {
            RLANY_GET_X, RLANY_CAN_GET_X:
                at = TwoInOneLookUp(rel, ky, X, RRF_ENTKEYY);
```

```
                if (at >= 0) {
                    tmp = RRV_DATA_BASE + 4*at;
                    if (BlkValueRead(rel, tmp) & RRF_HASX) {
                        if (Y == RLANY_CAN_GET_X) rtrue;
                        return BlkValueRead(rel, tmp + 2);
                    }
                }
            RLANY_GET_Y, RLANY_CAN_GET_Y:
                at = TwoInOneLookUp(rel, kx, X, RRF_ENTKEYX);
                if (at >= 0) {
                    tmp = RRV_DATA_BASE + 4*at;
                    if (BlkValueRead(rel, tmp) & RRF_HASY) {
                        if (Y == RLANY_CAN_GET_Y) rtrue;
                        return BlkValueRead(rel, tmp + 3);
                    }
                }
        }
        if (Y == RLANY_GET_X or RLANY_GET_Y)
            print "*** Lookup failed: value not found ***^";
        rfalse;
    } else if (task == RELS_LOOKUP_ALL_X) {
        at = TwoInOneLookUp(rel, ky, X, RRF_ENTKEYY);
        if (at >= 0) {
            tmp = RRV_DATA_BASE + 4*at;
            if (BlkValueRead(rel, tmp) & RRF_HASX)
                LIST_OF_TY_InsertItem(Y, BlkValueRead(rel, tmp + 2));
        }
        return Y;
    } else if (task == RELS_LOOKUP_ALL_Y) {
        at = TwoInOneLookUp(rel, kx, X, RRF_ENTKEYX);
        if (at >= 0) {
            tmp = RRV_DATA_BASE + 4*at;
            if (BlkValueRead(rel, tmp) & RRF_HASY)
                LIST_OF_TY_InsertItem(Y, BlkValueRead(rel, tmp + 3));
        }
        return Y;
    } else if (task == RELS_LIST) {
        switch (Y) {
            RLIST_ALL_X:
                fl = RRF_USED+RRF_ENTKEYX+RRF_HASY;
                jump ListEntryKeys;
            RLIST_ALL_Y:
                fl = RRF_USED+RRF_ENTKEYY+RRF_HASX;
                .ListEntryKeys;
                for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                    tmp = RRV_DATA_BASE + 4*at;
                    if ((BlkValueRead(rel, tmp) & fl) == fl)
                        LIST_OF_TY_InsertItem(X, BlkValueRead(rel, tmp + 1), false, 0, true);
                }
            RLIST_ALL_PAIRS:
                tmp = BlkValueRead(X, LIST_ITEM_KOV_F);
                if (KindAtomic(tmp) ~= COMBINATION_TY) rfalse;
                ! LIST_OF_TY_InsertItem will make a deep copy of the item,
                ! so we can reuse a single combination value here
                Y = BlkValueCreate(COMBINATION_TY, 0, tmp);
                for (at = BlkValueRead(rel, RRV_STORAGE): at >= 0: at--) {
                    tmp = RRV_DATA_BASE + 4*at;
                    fl = BlkValueRead(rel, tmp);
                    if ((fl & (RRF_USED+RRF_ENTKEYX+RRF_HASY)) ==
                        (RRF_USED+RRF_ENTKEYX+RRF_HASY)) {
                        BlkValueWrite(Y, COMBINATION_ITEM_BASE, BlkValueRead(rel, tmp + 1));
                        BlkValueWrite(Y, COMBINATION_ITEM_BASE + 1, BlkValueRead(rel, tmp + 3));
                        LIST_OF_TY_InsertItem(X, Y);
                    }
                }
                BlkValueWrite(Y, COMBINATION_ITEM_BASE, 0);
                BlkValueWrite(Y, COMBINATION_ITEM_BASE + 1, 0);
                BlkFree(Y);
                return X;
        }
        return X;
    }
    at = TwoInOneLookUp(rel, kx, X, RRF_ENTKEYX);
    switch(task) {
        RELS_TEST:
            if (at < 0) rfalse;
            fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
            if (~~(fl & RRF_HASY)) rfalse;
            tmp = BlkValueRead(rel, RRV_DATA_BASE + 4*at + 3);
            if (KOVIsBlockValue(ky)) {
                if (BlkValueCompare(tmp, Y) == 0) rtrue;
            } else {
                if (tmp == Y) rtrue;
            }
            rfalse;
        RELS_ASSERT_TRUE:
            if (at < 0) {
                ! create a new forward entry
                at = ~at;
                BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) + 1);
                fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
                if (fl == 0)
                    BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_FILLED) + 1);
                fl = RRF_USED+RRF_HASY+RRF_ENTKEYX;
                if (kx == ky || ~~(KOVIsBlockValue(kx) || KOVIsBlockValue(ky)))
                    fl = fl + RRF_ENTKEYY;
                BlkValueWrite(rel, RRV_DATA_BASE + 4*at, fl);
                if (KOVIsBlockValue(kx)) { X = BlkValueCopy(BlkValueCreate(kx), X); }
                BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 1, X);
                BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 2, 0);
            } else {
                fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
                if (fl & RRF_HASY) {
                    ! if the Y we're inserting is already there, we're done
                    tmp = BlkValueRead(rel, RRV_DATA_BASE + 4*at + 3);
                    if (KOVIsBlockValue(ky)) {
```

```
                if (BlkValueCompare(tmp, Y) == 0) rtrue;
            } else {
                if (tmp == Y) rtrue;
            }
            ! it's different, so delete the reverse entry
            at2 = TwoInOneLookUp(rel, ky, tmp, RRF_ENTKEYY);
            if (at2 >= 0) TwoInOneDelete(rel, at2, kx, ky, RRF_ENTKEYY);
        } else {
            BlkValueWrite(rel, RRV_DATA_BASE + 4*at, fl + RRF_HASY);
        }
        ! use the existing copy of X
        X = BlkValueRead(rel, RRV_DATA_BASE + 4*at + 1);
    }
    ! use the existing copy of Y if there is one
    at2 = TwoInOneLookUp(rel, ky, Y, RRF_ENTKEYY);
    if (KOVIsBlockValue(ky)) {
        if (at2 >= 0)
            Y = BlkValueRead(rel, RRV_DATA_BASE + 4*at2 + 1);
        else
            Y = BlkValueCopy(BlkValueCreate(ky), Y);
    }
    BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 3, Y);
    if (at2 >= 0) {
        ! delete existing reverse entry (and its own forward entry)
        TwoInOneDelete(rel, at2, kx, ky, RRF_ENTKEYY, 1);
    } else {
        at2 = ~at2;
    }
    ! create reverse entry
    BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) + 1);
    fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at2);
    if (fl == 0)
        BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_FILLED) + 1);
    fl = fl | (RRF_USED+RRF_HASX+RRF_ENTKEYY);
    if (kx == ky || ~~(KOVIsBlockValue(kx) || KOVIsBlockValue(ky)))
        fl = fl | RRF_ENTKEYX;
    BlkValueWrite(rel, RRV_DATA_BASE + 4*at2, fl);
    BlkValueWrite(rel, RRV_DATA_BASE + 4*at2 + 1, Y);
    BlkValueWrite(rel, RRV_DATA_BASE + 4*at2 + 2, X);
    TwoInOneCheckResize(rel);
    rtrue;
RELS_ASSERT_FALSE:
    ! we only have work to do if the entry exists and has a Y which
    ! matches the Y we're deleting
    if (at < 0) rtrue;
    fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
    if ((fl & RRF_HASY) == 0) rtrue;
    tmp = BlkValueRead(rel, RRV_DATA_BASE + 4*at + 3);
    if (KOVIsBlockValue(ky)) {
        if (BlkValueCompare(tmp, Y) ~= 0) rtrue;
    } else {
        if (tmp ~= Y) rtrue;
    }
```

```
            TwoInOneDelete(rel, at, kx, ky, RRF_ENTKEYX, 1);
            rtrue;
    }
];
[ TwoInOneDelete rel at kx ky ekflag both  fl at2 E i;
!print "[2in1DEL at=", at, " (E=", BlkValueRead(rel, RRV_DATA_BASE + 4*at + 1), ") ekflag=", ekflag
  ...  , " both=", both, "]^";
    fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
    if (ekflag == RRF_ENTKEYX) {
        if (fl & RRF_HASY) {
            i = RRV_DATA_BASE + 4*at + 3;
            if (both) E = BlkValueRead(rel, i);
            BlkValueWrite(rel, i, 0);
            ! delete matching Y<-X entry if needed
            if (both) {
                at2 = TwoInOneLookUp(rel, ky, E, RRF_ENTKEYY);
                if (at2 >= 0) TwoInOneDelete(rel, at2, kx, ky, RRF_ENTKEYY);
                if (at2 == at) fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
            }
            fl = fl & ~RRF_HASY;
        }
    } else {
        if (fl & RRF_HASX) {
            i = RRV_DATA_BASE + 4*at + 2;
            if (both) E = BlkValueRead(rel, i);
            BlkValueWrite(rel, i, 0);
            ! delete matching X->Y entry if needed
            if (both) {
                at2 = TwoInOneLookUp(rel, kx, E, RRF_ENTKEYX);
                if (at2 >= 0) {
                    TwoInOneDelete(rel, at2, kx, ky, RRF_ENTKEYX);
                    if (at2 == at) fl = BlkValueRead(rel, RRV_DATA_BASE + 4*at);
                }
            }
            fl = fl & ~RRF_HASX;
        }
    }
    if ((fl & (RRF_HASX+RRF_HASY)) == 0) {
        ! entry is now empty, mark it deleted
        if (((fl & RRF_ENTKEYX) && KOVIsBlockValue(kx)) ||
            ((ky ~= kx) && (fl & RRF_ENTKEYY) && KOVIsBlockValue(ky))) {
            BlkFree(BlkValueRead(rel, RRV_DATA_BASE + 4*at + 1));
        }
        BlkValueWrite(rel, RRV_DATA_BASE + 4*at, RRF_DELETED);
        BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 1, 0);
        BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 2, 0);
        BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 3, 0);
        BlkValueWrite(rel, RRV_USED, BlkValueRead(rel, RRV_USED) - 1);
    } else {
        BlkValueWrite(rel, RRV_DATA_BASE + 4*at, fl);
    }
];
[ TwoInOneLookUp rel ke E ekflag  hashv i free mask perturb flags;
```

```
!print "[2in1LU rel=", rel, " ke=", ke, " E=", E, " ekf=", ekflag, ": ";
    ! calculate a hash value for the key
    hashv = KOVHashValue(ke, E);
    ! look in the first expected slot
    mask = BlkValueRead(rel, RRV_STORAGE);
    i = hashv & mask;
!print "hv=", hashv, ", trying ", i;
    flags = BlkValueRead(rel, RRV_DATA_BASE + 4*i);
    if (flags == 0) {
!print " - not found]^";
        return ~i;
    }
    if ((flags & ekflag) && TwoInOneEntryMatches(rel, i, ke, E)) {
!print " - found]^";
        return i;
    }
    ! not here, keep looking in sequence
    free = -1;
    if (flags & RRF_DELETED) free = i;
    perturb = hashv;
    hashv = i;
    for (::) {
        hashv = hashv*5 + perturb + 1;
        i = hashv & mask;
!print ", ", i;
        flags = BlkValueRead(rel, RRV_DATA_BASE + 4*i);
        if (flags == 0) {
!print " - not found]^";
            if (free >= 0) return ~free;
            return ~i;
        }
        if ((flags & ekflag) && TwoInOneEntryMatches(rel, i, ke, E)) {
!print " - found]^";
            return i;
        }
        if ((free < 0) && (flags & RRF_DELETED)) free = i;
        #ifdef TARGET_ZCODE;
        @log_shift perturb (-RRP_PERTURB_SHIFT) -> perturb;
        #ifnot;
        @ushiftr perturb RRP_PERTURB_SHIFT perturb;
        #endif;
    }
];
[ TwoInOneCheckResize rel  filled ext newext temp i at kov kx ky F E X Y;
    filled = BlkValueRead(rel, RRV_FILLED);
    ext = BlkValueRead(rel, RRV_STORAGE) + 1;
    if (filled >= (ext - filled) * RRP_CROWDED_IS) {
        ! copy entries to temporary space
        temp = BlkAllocate(ext * (4*WORDSIZE), INDEXED_TEXT_TY, BLK_FLAG_WORD+BLK_FLAG_MULTIPLE);
        for (i=0: i<ext*4: i++)
            BlkValueWrite(temp, i, BlkValueRead(rel, RRV_DATA_BASE+i));
        ! resize and clear our data
        if (ext >= RRP_LARGE_IS) newext = ext * RRP_RESIZE_LARGE;
        else newext = ext * RRP_RESIZE_SMALL;
        BlkValueSetExtent(rel, RRV_DATA_BASE + newext*4);
        BlkValueWrite(rel, RRV_STORAGE, newext - 1);
        BlkValueWrite(rel, RRV_FILLED, BlkValueRead(rel, RRV_USED));
        for (i=0: i<newext*4: i++)
            BlkValueWrite(rel, RRV_DATA_BASE+i, 0);
        ! copy entries back from temporary space
        kov = BlkValueRead(rel, RRV_KIND);
        kx = KindBaseTerm(kov, 0); ky = KindBaseTerm(kov, 1);
        for (i=0: i<ext: i++) {
            F = BlkValueRead(temp, 4*i);
            if (F == 0 || (F & RRF_DELETED)) continue;
            E = BlkValueRead(temp, 4*i + 1);
            X = BlkValueRead(temp, 4*i + 2);
            Y = BlkValueRead(temp, 4*i + 3);
            if (F & RRF_ENTKEYX) at = TwoInOneLookUp(rel, kx, E, RRF_ENTKEYX);
            else at = TwoInOneLookUp(rel, ky, E, RRF_ENTKEYY);
            if (at >= 0) { print "*** Duplicate entry while resizing ***^"; rfalse; }
            at = ~at;
            BlkValueWrite(rel, RRV_DATA_BASE + 4*at, F);
            BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 1, E);
            BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 2, X);
            BlkValueWrite(rel, RRV_DATA_BASE + 4*at + 3, Y);
        }
        ! done with temporary space
        BlkFree(temp);
    }
];
[ TwoInOneEntryMatches rel at ke E  ce;
    ce = BlkValueRead(rel, RRV_DATA_BASE + 4*at + 1);
    if (KOVIsBlockValue(ke)) {
        if (BlkValueCompare(ce, E) ~= 0) rfalse;
    } else {
        if (ce ~= E) rfalse;
    }
    rtrue;
];
```

§23. Foot.

```
#IFNOT; ! IFDEF MEMORY_HEAP_SIZE
[ RELATION_TY_Support t a b c; rfalse; ];
[ RELATION_TY_Say comb; ];
[ RELATION_TY_Name rel txt; ];
#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```

§**24. Empty.** This implements the “empty” adjective. We can always check whether a relation is empty. For most relation types, we can cause the relation to become empty by removing all pairs: but this is impossible for equivalence relations, which are never empty, since any X is equivalent to itself. And we can never force a relation to become non-empty, since that would require making up data.

In any case, the implementation is delegated to the relation handler.

```
[ RELATION_TY_Empty rel set  handler;
    handler = rel-->RR_HANDLER;
    return handler(rel, RELS_EMPTY, set);
];
```

BlockValues Template B/blkvt

Purpose

Routines for copying, comparing, creating and destroying block values, and for reading and writing them as if they were arrays.

§**1. Head.** As with Flex.i6t, none of this material is worth compiling unless there is a heap on which the block values can be stored.

```
#IFDEF MEMORY_HEAP_SIZE;
```

§**2. Data Section.** In this segment, we provide a layer which comes between "Flex.i6t" and its eventual users. These are routines which handle the data section for the allocated blocks, and which isolate the end user from having to know how the blocks are divided. As far as the user is concerned, each block value has an "extent" E, and consists of an array of values indexed from 0 to $E-1$. These are bytes unless `BLK_FLAG_WORD` is set in the header, in which case they are words; unless `BLK_FLAG_16_BIT` is set, in which case they are two-byte quantities. (For instance, indexed text is stored using bytes on the Z-machine but 16-bit quantities on Glulx, since the ZSCII character set can fit into the range 0 to 255 but Unicode cannot.)

`BlkValueExtent(B)` returns the value E for the block `B`.

`BlkValueSetExtent(B, N)` resizes the block `B` so that value E is at least `N`. If this is a reduction, entries are lost from the end, i.e., from the highest-indexed entries. If it is an expansion, entries are added at the end and the existing entries are preserved.

`BlkValueRead(B, I)` returns the value of array entry `I` in `B`.

`BlkValueWrite(B, I, V)` puts the value `V` into array entry `I` in `B`.

```
[ BlkValueExtent block  tsize flags;
    if (block == 0) return 0;
    flags = block->BLK_HEADER_FLAGS;
    if (flags & BLK_FLAG_MULTIPLE == 0)
        tsize = BlkSize(block) - BLK_DATA_OFFSET;
    else
        for (:block~=NULL:block=block-->BLK_NEXT)
            tsize = tsize + BlkSize(block) - BLK_DATA_MULTI_OFFSET;
    if (flags & BLK_FLAG_16_BIT) return tsize/2;
    if (flags & BLK_FLAG_WORD) return tsize/WORDSIZE;
    return tsize;
];
[ BlkValueSetExtent block tsize flags wsize;
    if (block == 0) return 0;
    flags = block->BLK_HEADER_FLAGS; wsize = 1;
    if (flags & BLK_FLAG_WORD) wsize = WORDSIZE;
    if (flags & BLK_FLAG_16_BIT) wsize = 2;
    return BlkResize(block, (tsize)*wsize);
];
[ BlkValueRead block pos dsize hsize flags wsize ot op;
    if (block==0) rfalse;
    flags = block->BLK_HEADER_FLAGS; wsize = 1;
    if (flags & BLK_FLAG_WORD) wsize = WORDSIZE;
    if (flags & BLK_FLAG_16_BIT) wsize = 2;
    ot = block; op = pos;
    pos = pos*wsize;
    if (flags & BLK_FLAG_MULTIPLE) hsize = BLK_DATA_MULTI_OFFSET;
    else hsize = BLK_DATA_OFFSET;
    for (:block~=NULL:block=block-->BLK_NEXT) {
        dsize = BlkSize(block) - hsize;
        if ((pos >= 0) && (pos<dsize)) {
            block = block + hsize + pos;
            switch(wsize) {
                1: return block->0;
                2: #Iftrue (WORDSIZE == 2); return block-->0;
                    #ifnot; return (block->0)*256 + (block->1);
                    #endif;
                4: return block-->0;
            }
        }
        pos = pos - dsize;
    }
    "*** BlkValueRead: reading from index out of range: ", op, " in ", ot, " ***";
];
[ BlkValueWrite block pos val dsize hsize flags wsize ot op;
    if (block==0) rfalse;
    flags = block->BLK_HEADER_FLAGS; wsize = 1;
    if (flags & BLK_FLAG_WORD) wsize = WORDSIZE;
    if (flags & BLK_FLAG_16_BIT) wsize = 2;
    ot = block; op = pos;
    pos = pos*wsize;
    if (flags & BLK_FLAG_MULTIPLE) hsize = BLK_DATA_MULTI_OFFSET;
    else hsize = BLK_DATA_OFFSET;
    for (:block~=NULL:block=block-->BLK_NEXT) {
        dsize = BlkSize(block) - hsize;
        if ((pos >= 0) && (pos<dsize)) {
            block = block + hsize + pos;
            switch(wsize) {
                1: block->0 = val;
                2: #Iftrue (WORDSIZE == 2); block-->0 = val;
                    #ifnot; block->0 = (val/256)%256; block->1 = val%256;
                    #endif;
                4: block-->0 = val;
            }
            return;
        }
        pos = pos - dsize;
    }
    "*** BlkValueWrite: writing to index out of range: ", op, " in ", ot, " ***";
];
```

§**3. KOV Support.** To carry out the four fundamental operations on block values – creating, destroying, copying and comparing – we need to take account of what data they hold. Letting an indexed text go matters little: it contains only character codes. But letting a list of lists of numbers disappear is risky, because we might leave allocated blocks all over the heap for those individual lists of numbers which are now no longer in their parent list. Every KOV has different needs, because every KOV uses its data differently.

Each block KOV, then, provides a single "KOV support" function. These are named systematically by suffixing `_Support`: that is, the support function for `INDEXED_TEXT_TY` is called `INDEXED_TEXT_TY_Support` and so on. NI compiles a function called `KOVSupportFunction` which takes a KOV number as its argument and returns the relevant function.

The support function can be called with any of the following task constants as its first argument: it then has a further one to three arguments depending on the task in hand.

```
Constant CREATE_KOVS     = 1;
Constant CAST_KOVS       = 2;
Constant DESTROY_KOVS    = 3;
Constant PRECOPY_KOVS    = 4;
Constant COPY_KOVS       = 5;
Constant COMPARE_KOVS    = 6;
Constant READ_FILE_KOVS  = 7;
Constant WRITE_FILE_KOVS = 8;
Constant HASH_KOVS       = 9;
```

§**4. Creation.** To create a new value, we can't simply allocate a block and then fill it in some way appropriate to the kind of value needed: because we don't know what size of block would be a good idea, and even that much depends on the KOV. So the following routine delegates the whole process to the relevant KOV's support routines.

Depending on the KOV, we can also create it with a value cast from some existing value: for instance, an indexed text can be created with a value cast from the ordinary text "Marshmellow".

`K_Support(CREATE_KOVS, cast_from, skov)` – where `cast_from` and `skov` are optional arguments – is called to allocate a block on the heap and, if `cast_from` is non-zero, cast that data into the new block value. Of course `cast_from` itself is a value, and we don't specify its KOV: the support function is expected to know what it will get (if anything). `skov` stands for "strong kind of value" and is only applicable if `K` is a KOV constructor rather than a base KOV: for instance, if `K` is "list of...", then to create a "list of numbers" we would have `skov` equal to `NUMBER_TY`, whereas to create a "list of lists of texts" `skov` would be `LIST_OF_TY`.

```
Global block_value_tally;
[ BlkValueCreate kov cast_from skov  block sf;
    if (skov == 0 && (kov < 0 || kov >= BASE_KIND_HWM)) skov = kov;
    sf = KOVSupportFunction(kov);
    if (sf) block = sf(CREATE_KOVS, cast_from, skov);
    else { print "*** Impossible runtime creation ***^"; rfalse; }
#ifdef SHOW_ALLOCATIONS;
    print "[created ", kov, " at ", block, ": ", block_value_tally++, "]^";
#endif;
    return block;
];
```

§**5. Casting.** Just as we can create a value with a cast, we can also perform an assignment to an already-created block value in the form of a cast: well, for some kinds of value we can.

`K_Support(CAST_KOVS, fromval, fromkov, block)` casts from a value `fromval` with a specified KOV `fromkov` into the existing block value `block`.

```
[ BlkValueCast block tokov fromkov fromval  sf;
    sf = KOVSupportFunction(tokov);
    if (sf) return sf(CAST_KOVS, fromval, fromkov, block);
    else { print "*** Impossible runtime cast ***^"; rfalse; }
];
```

§**6. Destruction.** This must be called whenever the current contents of a block value is about to be lost, either because of overwriting or because the block value is in a variable which is going out of scope. What must be done depends on the kind of value: for instance, for indexed text nothing need be done, because the array entries only contains character values. But for a list of lists of numbers, for instance, array entries are pointers to lists of numbers. Simply overwriting or forgetting them means that the lists to which they point will be stuck on the heap forever, and will never be deallocated. So value destruction for a list involves destroying each individual entry and then deallocating it: and this may of course recurse.

`K_Support(DESTROY_KOVS, block)` takes whatever action is necessary to ensure that the values remaining in `block` can be discarded without leaving unreferenced data left on the heap.

```
[ BlkValueDestroy block  k rv sf;
    if (block == 0) return;
    k = block-->BLK_HEADER_KOV;
    sf = KOVSupportFunction(k);
    if (sf) return sf(DESTROY_KOVS, block);
    else { print "*** Impossible runtime deallocation ***^"; rfalse; }
];
```

§**7. Deep Copy.** A deep copy transfers the data contents – that is, the arrays described above – from one block to another; the old contents must be destroyed first. Note that we first perform a simplistic copy of the bits over, but that for some block values that won't be good enough. If the two blocks contain lists of lists, then each entry is itself a list, and a deep copy of each entry must also be performed.

`K_Support(COPY_KOVS, blockto, blockfrom)` takes whatever additional action might be necessary to ensure that pointers to heap data are not duplicated; it runs after the raw data has been copied bitwise and may only have to correct a few entries, or even none at all.

```
[ BlkValueCopy blockto blockfrom dsize i sf;
    if (blockto == 0) { print "*** Deep copy failed: destination empty ***^"; rfalse; }
    if (blockfrom == 0) { print "*** Deep copy failed: source empty ***^"; rfalse; }
    if (blockfrom->BLK_HEADER_N == 0) {
        ! A hack to handle precompiled array constants: N=0 blocks otherwise don't exist
        LIST_OF_TY_CopyRawArray(blockto, blockfrom, 1, 0);
        return blockto;
    }
    if (blockfrom-->BLK_HEADER_KOV ~= blockto-->BLK_HEADER_KOV) {
        print "*** Deep copy failed: types mismatch ***^"; rfalse;
    }
    BlkValueDestroy(blockto);
    dsize = BlkValueExtent(blockfrom);
    if (((blockfrom->BLK_HEADER_FLAGS) & BLK_FLAG_MULTIPLE) &&
        (BlkValueSetExtent(blockto, dsize, -1) == false)) {
        print "*** Deep copy failed: resizing failed ***^"; rfalse;
    }
    sf = KOVSupportFunction(blockfrom-->BLK_HEADER_KOV);
    if (sf) sf(PRECOPY_KOVS, blockto, blockfrom);
    for (i=0:i<dsize:i++) BlkValueWrite(blockto, i, BlkValueRead(blockfrom, i));
    if (sf) sf(COPY_KOVS, blockto, blockfrom);
    else { print "*** Impossible runtime copy ***^"; rfalse; }
    return blockto;
];
```

§**8. Comparison.** And it's a similar story with comparison.

`K_Support(COMPARE_KOVS, blockleft, blockright)` looks at the data in the two blocks and returns 0 if they are equal, a positive number if `blockright` is "greater than" `blockleft`, and a negative number if not. The interpretation of "greater than" depends on the KOV: it should be something which the user would find natural.

```
[ BlkValueCompare blockleft blockright  kov sf;
    if ((blockleft == 0) && (blockright == 0)) return 0;
    if (blockleft == 0) return 1;
    if (blockright == 0) return -1;
    if (blockleft-->BLK_HEADER_KOV ~= blockright-->BLK_HEADER_KOV)
        return blockleft-->BLK_HEADER_KOV - blockright-->BLK_HEADER_KOV;
    kov = blockleft-->BLK_HEADER_KOV;
    sf = KOVSupportFunction(kov);
    if (sf) return sf(COMPARE_KOVS, blockleft, blockright);
    else { print "*** Impossible runtime comparison ***^"; rfalse; }
];
```

§**9. Creating Constants.** If the NI compiler reads, say, the string literal "Sea Devils" in the source text, and it is expecting to find indexed text – for instance if it's in a column of a table marked as containing indexed text – then how is this to find its way into memory as a block value? The heaps starts entirely empty, and NI cannot precompile parts of it: nor would we really want that, since it would hardwire assumptions about the heap's format into NI itself.

The answer is that at start-up time we perform creations of all of the constants we will need. Because they are constants, they will never be deallocated and never change in their contents – so they need not live on the heap at all, and in fact we store them elsewhere in memory. NI simply creates a blank array of the right 2^n size. Suppose that `X` is the address of this array. Then during the start-up rules, we at some point do this:

```
    BlkValueInitialCopy(X, BlkValueCreate(INDEXED_TEXT_TY, "Sea Devils"))
```

The effect is to create a suitable structure on the heap to hold this as indexed text, and then to copy it bitwise into `X`. A bitwise copy is ordinarily against the rules because it duplicates pointers, but of course the created block has been created for this purpose only: nothing points to it, and it is about to vanish – indeed the address of the block in question exists only briefly on the VM stack. (It is because of this that we don't perform a deep copy using the routine above.)

```
[ BlkValueInitialCopy blockto blockfrom dsize i;
    if (blockto == 0) { print "*** Initial copy failed: destination empty ***^"; rfalse; }
    if (blockfrom == 0) { print "*** Initial copy failed: source empty ***^"; rfalse; }
    dsize = 1; for (i=1: i<=blockfrom->BLK_HEADER_N: i++) dsize=dsize*2;
    for (i=0:i<dsize:i++) blockto->i = blockfrom->i;
    return blockto;
];
```

§10. Hashing. `K_Support(HASH_KOVS, block)` returns a hash value for the block. The algorithm used for hashing depends on the KOV, but it must have the property that two equivalent blocks (for which `COMPARE_KOVS` returns 0) produce the same hash value.

```
[ BlkValueHash block  kov sf;
    if (block == 0) return 0;
    kov = block-->BLK_HEADER_KOV;
    sf = KOVSupportFunction(kov);
    if (sf) return sf(HASH_KOVS, block);
    else { print "*** Impossible runtime hashing ***^"; rfalse; }
];
[ KOVHashValue kov value;
    if (KOVIsBlockValue(kov)) return BlkValueHash(value);
    return value;
];
```

§11. Serialisation. Some block values can be written to external files (on Glulx): others cannot. The following routines abstract that.

If `ch` is -1, then `K_Support(READ_FILE_KOVS, block, auxf, ch)` returns `true` or `false` according to whether it is possible to read data from an auxiliary file `auxf` into the block value `block`. If `ch` is any other value, then the routine should do exactly that, taking `ch` to be the first character of the text read from the file which makes up the serialised form of the data.

`K_Support(WRITE_FILE_KOVS, block)` is simpler because, strictly speaking, it doesn't write to a file at all: it simply prints a serialised form of the data in `block` to the output stream. Since it is called only when that output stream has been redirected to an auxiliary file, and since the serialised form would often be illegible on screen, it seems reasonable to call it a file input-output function just the same.

```
[ BlkValueReadFromFile block auxf ch kov  sf;
    sf = KOVSupportFunction(kov);
    if (sf) return sf(READ_FILE_KOVS, block, auxf, ch);
    rfalse;
];
[ BlkValueWriteToFile block kov  sf;
    sf = KOVSupportFunction(kov);
    if (sf) return sf(WRITE_FILE_KOVS, block);
    rfalse;
];
```

§**12. Stubs.** To ensure that the template code will still compile if `MEMORY_HEAP_SIZE` is undefined and there's no heap: none of these routines do anything in such a situation, but nor are they ever called – it's just that I6 source code may refer to them anyway, so they need to exist as routine names.

```
#IFNOT; ! IFDEF MEMORY_HEAP_SIZE
[ BlkValueReadFromFile; rfalse; ];
[ BlkValueWriteToFile; rfalse; ];
[ BlkValueCreate x y z; ];
[ BlkValueDestroy x; ];
[ BlkValueCopy x y; ];
[ BlkValueCompare x y; ];
#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```

Flex Template B/flext

Purpose

To allocate flexible-sized blocks of memory as needed to hold arbitrary-length strings of text, stored actions or other block values.

§**1. Overview.** Each I7 value is represented at run-time by an I6 word: on the Z-machine, a 16-bit number, and on Glulx, a 32-bit number. The correspondence between these numbers and the original values depends on the kind of value: "number" comes out as a signed twos-complement number, but "time" as an integer number of minutes since midnight, "rulebook" as the index of the rulebook in order of creation, and so on.

Even if a 32-bit number is available, this is not enough to represent the full range of values we might want: consider all the possible hundred-word essays of text, for instance. In some cases, then, the value is (either directly or indirectly) a pointer, telling the run-time code that the data is not in the value itself but can be found at a given location in memory. For instance, a "text" value is a (packed) address of either some encoded text or a routine to print text. When NI compiles references to a text value, it also creates the data to which this address belongs.

This works well if the data need not change in size, and if we never need to create or throw away values. But if we want a variable which can hold an arbitrary string of text which we will compose for ourselves, say, then we need at run-time to find space somewhere in memory to hold that data, and we need to be able to cope if that space runs out, and lastly we need a way to reclaim the memory again when the text's usefulness has finished. For instance, if a rule uses a temporary "let" variable which will hold indexed text then the block of indexed text data must be allocated; the variable must then be set to a pointer to this data; the rule must then run, and lastly the block of data deallocated again.

Values of this kind are called "block values" since they are pointers to blocks of memory. We have to be careful when making assignments to variables having these kinds of value. In I7, text like "change the motto text to the player's command" must not be compiled to I6 code such as `MT = PC;`, because that simply sets `MT` to have the same address as `PC`. There are now two independent pointers to the same piece of text, so that changing either one would change both, while the previous contents of `MT` are un-pointed to. The latter problem is almost as bad as the former, because it means that memory has been wasted forever – it would never be reclaimed. Repeat the process often enough and all the memory will be lost in this way: this is called a "leak".

I7 treats block values exactly like other values, as far as the writer is concerned. There is no concept of "a pointer to..." in I7 source text. Instead:

(1) For every allocated block of data, there is exactly one I7 value – stored in an I6 local or global variable, in a property, in a table entry, or even as part of another allocated block of data – which points to it.
(2) When assigning `Y` to `X`, we always perform a "deep copy" of the contents of the block pointed to by `Y` into that pointed to by `X`: we never copy pointers, which would be a "shallow copy".
(3) If `X` is the I6 representation of a block kind of value, then `X` is either 0 – meaning "not allocated yet" – or a valid pointer to a block of data. This is purely for efficiency's sake, making table columns of indexed text (for instance) a sparse representation when, as often happens, not many are filled in. When assigning to 0, we must first allocate a block, then change the pointer from 0 to point to it, and assign to the newly-allocated block. The user is oblivious to all of this.
(4) Any value which will be lost – for instance, a value in a variable which goes out of scope – must have its block deallocated first.
(5) Any value which is passed as an argument to a function must be copied first, as otherwise there are two pointers to the same block of data, one in the calling stack frame and the other in the one called;

in other words, we call by value, never by reference. (As we shall see, I7 phrases can indeed by defined which work by reference rather than by value, but those are defined using inline I6, not as function calls.)

As simple as this scheme looks – there is no need for garbage collection or reference counting – it is not entirely easy to get right. There are many complications, but the basic slogan is one pointer, one block.

§2. Blocks. A "block" is a continuous range of 2^n bytes of memory, where $n \geq 3$ for a 16-bit VM (i.e., for the Z-machine) and $n \geq 4$ for a 32-bit VM (i.e., on Glulx). Internally, a block is divided into a header followed by a data section.

The header consists of 4, 8 or 16 bytes, depending on the word size and the kind of block (see below). It always begins with a byte specifying n, the binary logarithm of its size: thus the largest block representable is 2^{255} bytes long, but somehow I think we can live with that. The second byte contains a bitmap of (at present) four flags, whose meanings will be explained below. The second *word* of the block, which might be at byte offset 2 or 4 from the start of the block depending on the word-size of the VM, is a number specifying the kind of value (KOV) which the block contains data of.

It might be objected that KOVs are not reducible to simple numbers. For instance, for any KOV K there is another KOV "list of K", so there is an infinite range of possibilities. "List of..." is what, in other languages, would be called a type constructor; whereas a KOV like "indexed text" is what would be called a base type, since it is not the result of any type constructor. In I7, there is a finite range of base types and type constructors, and these have distinct ID numbers: that is what is stored in the `BLK_HEADER_KOV` field. A block which has KOV "list of indexed texts" will have the same value here as a block which has KOV "list of numbers": it will store the I6 constant `LIST_OF_TY`. (To find out whether such a list does indeed contain numbers or texts – and it is essential to be able to do this – one must look at the data section of the block. See "Lists.i6t".)

The data section of a block begins at the byte offset `BLK_DATA_OFFSET` from the address of the block: but see below for how multiple-blocks behave differently.

These definitions must not be altered without making matching changes to the compiler.

```
Constant BLK_HEADER_N = 0;
Constant BLK_HEADER_FLAGS = 1;
Constant BLK_FLAG_MULTIPLE = $$00000001;
Constant BLK_FLAG_16_BIT   = $$00000010;
Constant BLK_FLAG_WORD     = $$00000100;
Constant BLK_FLAG_RESIDENT = $$00001000;
Constant BLK_HEADER_KOV = 1;
Constant BLK_DATA_OFFSET = 2*WORDSIZE;
```

§**3. Multiple Blocks.** There are two kinds of block values: those which can always be stored in a single block (for instance, a floating-point number stored in exactly 8 bytes of data would be suitable for this), and those which can change unpredictably in size and might at any point overflow their current storage, so that they may need to occupy multiple blocks (for instance, an indexed text). In such a multiple-block KOV, the data is stored in a doubly linked list of blocks, and the I6 value for the result is the pointer to the block which heads the linked list. For instance, the indexed text

```
"But now I worship a celestiall Sunne"
```

might be represented by an I6 value `BN` which points to a list of blocks like so:

```
NULL <-- BN: "But now I wor" <--> BN2: "ship a celestiall Sunne" --> NULL
```

Note that the unique pointer to `BN2` is the one in the header of the `BN` block. When we need to grow such a text, we add additional blocks; if the text should shrink, blocks at the end can at our discretion be deallocated. If the entire text should be deallocated, then all of the blocks used for it are deallocated, starting at the back and working towards the front.

A multiple-block is one whose flags byte contains the `BLK_FLAG_MULTIPLE`. This information is redundant since it could in principle be deduced from the kind of value stored in the block, which is recorded in the `-->BLK_HEADER_KOV` word, but that would be too slow. `BLK_FLAG_MULTIPLE` can never change for a currently allocated block, just as it can never change its KOV.

A multiple-block header is longer than that of an ordinary block, because it contains two extra words: `-->BLK_NEXT` is the next block in the doubly-linked list of blocks representing the current value, or `NULL` if this is the end; `-->BLK_PREV` is the previous block, or `NULL` if this is the beginning. The need to fit these two extra words in means that the data section is deferred, and so for a multiple-block data begins at the byte offset `BLK_DATA_MULTI_OFFSET` rather than `BLK_DATA_OFFSET`.

```
Constant BLK_DATA_MULTI_OFFSET = 4*WORDSIZE;
Constant BLK_NEXT 2;
Constant BLK_PREV 3;
```

§**4. Head.** On the Z-machine, though not always on Glulx, the entire heap has to be allocated at compile-time, and we tend to want to make it reasonably large to cover most eventualities (though the heap size is controllable with use options, so the user does have control over this).

Many small works of IF never have need of the heap at all, and at the same time can't spare the memory for it. NI only creates the constant `MEMORY_HEAP_SIZE`, the number of bytes initially given over to the heap, if need arises. So the code and arrays in this segment will never be compiled unless needed (but see also the "Stubs" paragraph below).

```
#IFDEF MEMORY_HEAP_SIZE;

! Constant SHOW_ALLOCATIONS = 1; ! Uncomment this for debugging purposes
```

§5. Block Routines.

```
[ BlkType txb;
    return txb-->BLK_HEADER_KOV;
];
[ BlkSize txb bsize n; ! Size of an individual block, including header
    if (txb == 0) return 0;
    for (bsize=1: n<txb->BLK_HEADER_N: bsize=bsize*2) n++;
    return bsize;
];
[ BlkTotalSize txb tsize; ! Combined size of multiple-blocks for a value
    if (txb == 0) return 0;
    if ((txb->BLK_HEADER_FLAGS) & BLK_FLAG_MULTIPLE == 0)
        return BlkSize(txb);
    for (:txb~=NULL:txb=txb-->BLK_NEXT) {
        tsize = tsize + BlkSize(txb);
    }
    return tsize;
];
```

§6. Debugging Routines.

These two routines are purely for testing the code.

```
[ BlkDebug txb n k i bsize tot dtot kov;
    if (txb == 0) "Block never created.";
    kov = txb-->BLK_HEADER_KOV;
    print "Block ", txb, " (kov ", kov, "): ";
    for (:txb~=NULL:txb = txb-->BLK_NEXT) {
        if (k++ == 100) " ... and so on.";
        if (txb-->BLK_HEADER_KOV ~= kov)
            print "*Wrong kov=", txb-->BLK_HEADER_KOV, "* ";
        n = txb->BLK_HEADER_N;
        for (bsize=1:n>0:n--) bsize=bsize*2;
        i = bsize - BLK_DATA_OFFSET;
        dtot = dtot+i;
        tot = tot+bsize;
        print txb, "(", bsize, ") > ";
    }
    print dtot, " data in ", tot, " bytes^";
];
[ BlkDebugDecomposition from to txb pf;
    if (to==0) to = NULL;
    for (txb=from:(txb~=to) && (txb~=NULL):txb=txb-->BLK_NEXT) {
        if (pf) print "+";
        print BlkSize(txb);
        pf = true;
    }
    print "^";
];
```

§**7. The Heap.** Properly speaking, a "heap" is a specific kind of structure often used for managing uneven-sized or unpredictably changing data. We use "heap" here in the looser sense of being an amorphous-sized collection of blocks of memory, some free, others allocated; our actual representation of free space on the heap is not a heap structure in computer science terms. (Though this segment could easily be rewritten to make it so, or to adopt any other scheme which might be faster, without modifying the rest of the template or NI itself.) The heap begins as a contiguous region of memory, but it need not remain so: on Glulx we use dynamic memory allocation to extend it.

For I7 purposes we don't need a way to represent allocated memory, only the free memory. A block is free if and only if it has `-->BLK_HEADER_KOV` equal to 0, which is never a valid kind of value, and also has the multiple flag set. We do that because we construct the whole collection of free blocks, at any given time, as a single, multiple-block "value": a doubly linked list joined by the `-->BLK_NEXT` and `<--BLK_PREV`.

A single block, at the bottom of memory and never moving, never allocated to anyone, is preserved in order to be the head of this linked list of free blocks. This is a 16-byte (i.e., $n = 4$) block, which we format when the heap is initialised in `HeapInitialise()`. Thus the heap is full if and only if the `-->BLK_NEXT` of the head-free-block is `NULL`.

So far we have described a somewhat lax regime. After many allocations and deallocations one could imagine the list of free blocks becoming a very long list of individually small blocks, which would both make it difficult to allocate large blocks, and also slow to look through the list. To ameliorate matters, we maintain the following invariants:

(a) In the free blocks list, `B-->BLK_NEXT` is always an address after `B`;
(b) For any contiguous run of free space blocks in memory (excluding the head-free-block), taking up a total of T bytes, the last block in the run has size 2^n where n is the largest integer such that $2^n \leq T$.

For instance, there can never be two consecutive free blocks of size 128: they would form a "run" in the sense of rule (b) of size $T = 256$, and when T is a power of two the run must contain a single block. In general, it's easy to prove that the number of blocks in the run is exactly the number of 1s when T is written out as a binary number, and that the blocks are ordered in memory from small to large (the reverse of the direction of reading, i.e., rightmost 1 digit first). Maintaining (b) is a matter of being careful to fragment blocks only from the front when smaller blocks are needed, and to rejoin from the back when blocks are freed and added to the free space object.

```
Array Blk_Heap -> MEMORY_HEAP_SIZE + 16; ! Plus 16 to allow room for head-free-block
```

§**8. Initialisation.** To recap: the constant `MEMORY_HEAP_SIZE` has been predefined by the NI compiler, and is always itself a power of 2, say 2^n. We therefore have $2^n + 2^4$ bytes available to us, and we format these as a free space list of two blocks: the 2^4-sized "head-free-block" described above followed by a 2^n-sized block exactly containing the whole of the rest of the heap.

```
[ HeapInitialise n bsize blk2;
    blk2 = Blk_Heap + 16;
    Blk_Heap->BLK_HEADER_N = 4;
    Blk_Heap-->BLK_HEADER_KOV = 0;
    Blk_Heap->BLK_HEADER_FLAGS = BLK_FLAG_MULTIPLE;
    Blk_Heap-->BLK_NEXT = blk2;
    Blk_Heap-->BLK_PREV = NULL;
    for (bsize=1: bsize < MEMORY_HEAP_SIZE: bsize=bsize*2) n++;
    blk2->BLK_HEADER_N = n;
    blk2-->BLK_HEADER_KOV = 0;
    blk2->BLK_HEADER_FLAGS = BLK_FLAG_MULTIPLE;
    blk2-->BLK_NEXT = NULL;
    blk2-->BLK_PREV = Blk_Heap;
];
```

§**9. Net Free Space.** "Net" in the sense of "after deductions for the headers": this is the actual number of free bytes left on the heap which could be used for data. Note that it is used to predict whether it is possible to fit something further in: so there are two answers, depending on whether the something is multiple-block data (with a larger header and therefore less room for data) or single-block data (smaller header, more room).

```
[ HeapNetFreeSpace multiple txb asize;
    for (txb=Blk_Heap-->BLK_NEXT: txb~=NULL: txb=txb-->BLK_NEXT) {
        asize = asize + BlkSize(txb);
        if (multiple) asize = asize - BLK_DATA_MULTI_OFFSET;
        else asize = asize - BLK_DATA_OFFSET;
    }
    return asize;
];
```

§**10. Make Space.** The following routine determines if there is enough free space to accommodate another `size` bytes of data, given that it has to be multiple-block data if the `multiple` flag is set. If the answer turns out to be "no", we see if the host virtual machine is able to allocate more for us: if it is, then we ask for 2^m further bytes, where 2^m is at least `size` plus the worst-case header storage requirement (16 bytes), and in addition is large enough to make it worth while allocating. We don't want to bother the VM by asking for trivial amounts of memory.

This looks to be more memory than is needed, since after all we've asked for enough that the new data can fit entirely into the new block allocated, and we might have been able to squeeze some of it into the existing free space. But it ensures that heap invariant (b) above is preserved, and besides, running out of memory tends to be something you don't do only once.

(The code below is a refinement on the original, suggested by Jesse McGrew, which handles non-multiple blocks better.)

```
Constant SMALLEST_BLK_WORTH_ALLOCATING = 12; ! i.e. 2^12 = 4096 bytes

[ HeapMakeSpace size multiple  newblocksize newblock B n;
    for (::) {
        if (multiple) {
            if (HeapNetFreeSpace(multiple) >= size) rtrue;
        } else {
            if (HeapLargestFreeBlock(0) >= size) rtrue;
        }
        newblocksize = 1;
        for (n=0: (n<SMALLEST_BLK_WORTH_ALLOCATING) || (newblocksize<size): n++)
            newblocksize = newblocksize*2;
        while (newblocksize < size+16) newblocksize = newblocksize*2;
        newblock = VM_AllocateMemory(newblocksize);
        if (newblock == 0) rfalse;
        newblock->BLK_HEADER_N = n;
        newblock-->BLK_HEADER_KOV = 0;
        newblock->BLK_HEADER_FLAGS = BLK_FLAG_MULTIPLE;
        newblock-->BLK_NEXT = NULL;
        newblock-->BLK_PREV = NULL;
        for (B = Blk_Heap-->BLK_NEXT:B ~= NULL:B = B-->BLK_NEXT)
            if (B-->BLK_NEXT == NULL) {
                B-->BLK_NEXT = newblock;
                newblock-->BLK_PREV = B;
                jump Linked;
            }
        Blk_Heap-->BLK_NEXT = newblock;
        newblock-->BLK_PREV = Blk_Heap;
        .Linked; ;
        #ifdef SHOW_ALLOCATIONS;
        print "Increasing heap to free space map: "; BlkDebugDecomposition(Blk_Heap, 0);
        #endif;
    }
    rtrue;
];

[ HeapLargestFreeBlock multiple txb asize best;
    best = 0;
    for (txb=Blk_Heap-->BLK_NEXT: txb~=NULL: txb=txb-->BLK_NEXT) {
        asize = BlkSize(txb);
        if (multiple) asize = asize - BLK_DATA_MULTI_OFFSET;
        else asize = asize - BLK_DATA_OFFSET;
        if (asize > best) best = asize;
    }
    return best;
];
```

§**11. Block Allocation.** The routine `BlkAllocate(N, K, F)` allocates a block with room for `size` net bytes of data, which will have kind of value `K` and with flags `F`. If the flags include `BLK_FLAG_MULTIPLE`, this may be either a list of blocks or a single block. It returns either the address of the block or else throws run-time problem message and returns 0.

In allocation, we try to find a block which is as close as possible to the right size, and we may have to subdivide blocks: see case II below. For instance, if a block of size 2^n is available and we only need a block of size 2^k where $k < n$ then we break it up in memory as a sequence of blocks of size $2^k, 2^k, 2^{k+1}, 2^{k+2}, ..., 2^{n-1}$: note that the sum of these sizes is the 2^n we started with. We then use the first block of size 2^k. To continue the comparison with binary arithmetic, this is like a subtraction with repeated carries:

$$10000000_2 - 00001000_2 = 01111000_2$$

```
[ BlkAllocate size kov flags
    dsize n m free_block min_m max_m smallest_oversized_block secondhalf i hsize head tail;

    if (HeapMakeSpace(size, flags & BLK_FLAG_MULTIPLE) == false)
        return BlkAllocationError("ran out");

    ! Calculate the header size for a block of this KOV
    if (flags & BLK_FLAG_MULTIPLE) hsize = BLK_DATA_MULTI_OFFSET;
    else hsize = BLK_DATA_OFFSET;

    ! Calculate the data size
    n=0; for (dsize=1: dsize < hsize+size: dsize=dsize*2) n++;

    ! Seek a free block closest to the correct size, but starting from the
    ! block after the fixed head-free-block, which we can't touch
    min_m = 10000; max_m = 0;
    for (free_block = Blk_Heap-->BLK_NEXT:
        free_block ~= NULL:
        free_block = free_block-->BLK_NEXT) {
        m = free_block->BLK_HEADER_N;
        ! Current block the ideal size
        if (m == n) jump CorrectSizeFound;
        ! Current block too large: find the smallest which is larger than needed
        if (m > n) {
            if (min_m > m) {
                min_m = m;
                smallest_oversized_block = free_block;
            }
        }
        ! Current block too small: find the largest which is smaller than needed
        if (m < n) {
            if (max_m < m) {
                max_m = m;
            }
        }
    }
    if (min_m == 10000) {
        ! Case I: No block is large enough to hold the entire size
        if (flags & BLK_FLAG_MULTIPLE == 0) return BlkAllocationError("too fragmented");
        ! Set dsize to the size in bytes if the largest block available
        for (dsize=1: max_m > 0: dsize=dsize*2) max_m--;
        ! Split as a head (dsize-hsize), which we can be sure fits into one block,
        ! plus a tail (size-(dsize-hsize), which might be a list of blocks
        head = BlkAllocate(dsize-hsize, kov, flags);
        if (head == 0) return BlkAllocationError("head block not available");
        tail = BlkAllocate(size-(dsize-hsize), kov, flags);
        if (tail == 0) return BlkAllocationError("tail block not available");
        head-->BLK_NEXT = tail;
        tail-->BLK_PREV = head;
        return head;
    }
    ! Case II: No block is the right size, but some exist which are too big
    ! Set dsize to the size in bytes of the smallest oversized block
    for (dsize=1,m=1: m<=min_m: dsize=dsize*2) m++;
    free_block = smallest_oversized_block;
    while (min_m > n) {
        ! Repeatedly halve free_block at the front until the two smallest
        ! fragments left are the correct size: then take the frontmost
        dsize = dsize/2;
        secondhalf = free_block + dsize;
        secondhalf-->BLK_NEXT = free_block-->BLK_NEXT;
        if (secondhalf-->BLK_NEXT ~= NULL)
            (secondhalf-->BLK_NEXT)-->BLK_PREV = secondhalf;
        secondhalf-->BLK_PREV = free_block;
        free_block-->BLK_NEXT = secondhalf;
        free_block->BLK_HEADER_N = (free_block->BLK_HEADER_N) - 1;
        secondhalf->BLK_HEADER_N = free_block->BLK_HEADER_N;
        secondhalf-->BLK_HEADER_KOV = free_block-->BLK_HEADER_KOV;
        secondhalf->BLK_HEADER_FLAGS = free_block->BLK_HEADER_FLAGS;
        min_m--;
    }
    ! Once that is done, free_block points to a block which is exactly the
    ! right size, so we can fall into...
    ! Case III: There is a free block which has the correct size.
    .CorrectSizeFound;
    ! Delete the free block from the double linked list of free blocks: note
    ! that it cannot be the head of this list, which is fixed
    if (free_block-->BLK_NEXT == NULL) {
        ! We remove final block, so previous is now final
        (free_block-->BLK_PREV)-->BLK_NEXT = NULL;
    } else {
        ! We remove a middle block, so join previous to next
        (free_block-->BLK_PREV)-->BLK_NEXT = free_block-->BLK_NEXT;
        (free_block-->BLK_NEXT)-->BLK_PREV = free_block-->BLK_PREV;
    }
    free_block-->BLK_HEADER_KOV = KindAtomic(kov);
    free_block->BLK_HEADER_FLAGS = flags;
    if (flags & BLK_FLAG_MULTIPLE) {
        free_block-->BLK_NEXT = NULL;
        free_block-->BLK_PREV = NULL;
    }
    ! Zero out the data bytes in the memory allocated
    for (i=hsize:i<dsize:i++) free_block->i=0;
    return free_block;
];
[ BlkAllocationError reason;
    print "*** Memory ", (string) reason, " ***^";
    RunTimeProblem(RTP_HEAPERROR);
    rfalse;
];
```

§**12. Merging.** Given a free block `block`, find the maximal contiguous run of free blocks which contains it, and then call `BlkRecut` to recut it to conform to invariant (b) above.

```
[ BlkMerge block first last pv nx;
    first = block; last = block;
    while (last-->BLK_NEXT == last+BlkSize(last))
        last = last-->BLK_NEXT;
    while ((first-->BLK_PREV + BlkSize(first-->BLK_PREV) == first) &&
        (first-->BLK_PREV ~= Blk_Heap))
        first = first-->BLK_PREV;
    pv = first-->BLK_PREV;
    nx = last-->BLK_NEXT;
    #ifdef SHOW_ALLOCATIONS;
    print "Merging: "; BlkDebugDecomposition(pv-->BLK_NEXT, nx); print "^";
    #endif;
    if (BlkRecut(first, last)) {
        #ifdef SHOW_ALLOCATIONS;
        print " --> "; BlkDebugDecomposition(pv-->BLK_NEXT, nx); print "^";
        #endif;
    }
];
```

§13. Recutting. Given a segment of the free block list, containing blocks known to be contiguous in memory, we recut into a sequence of blocks satisfying invariant (b): we repeatedly cut the largest 2^m-sized chunk off the back end until it is all used up.

```
[ BlkRecut first last tsize backsize mfrom mto bnext backend n dsize fine_so_far;
    if (first == last) rfalse;
    mfrom = first; mto = last + BlkSize(last);
    bnext = last-->BLK_NEXT;
    fine_so_far = true;
    for (:mto>mfrom: mto = mto - backsize) {
        for (n=0, backsize=1: backsize*2 <= mto-mfrom: n++) backsize=backsize*2;
        if ((fine_so_far) && (backsize == BlkSize(last))) {
            bnext = last; last = last-->BLK_PREV;
            bnext-->BLK_PREV = last;
            last-->BLK_NEXT = bnext;
            continue;
        }
        fine_so_far = false; ! From this point, "last" is meaningless
        backend = mto - backsize;
        backend->BLK_HEADER_N = n;
        backend-->BLK_HEADER_KOV = 0;
        backend->BLK_HEADER_FLAGS = BLK_FLAG_MULTIPLE;
        backend-->BLK_NEXT = bnext;
        if (bnext ~= NULL) {
            bnext-->BLK_PREV = backend;
            bnext = backend;
        }
    }
    if (fine_so_far) rfalse;
    rtrue;
];
```

§14. Deallocation. There are two complications: first, when we free a multiple block we need to free all of the blocks in the list, starting from the back end and working forwards to the front – this is the job of `BlkFree`. Second, when any given block is freed it has to be put into the free block list at the correct position to preserve invariant (a): it might either come after all of the currently free blocks in memory, and have to be added to the end of the list, or in between two, and have to be inserted mid-list, but it can't be before all of them because the head-free-block is kept lowest in memory of all possible blocks. (Note that Glulx can't allocate memory dynamically which undercuts the ordinary array space created by I6: I6 arrays fill up memory from the bottom.)

Certain blocks *outside* the heap are marked as "resident" in memory, that is, are indestructible. This enables Inform to compile constant values.

```
[ BlkFree block fromtxb ptxb;
    if (block == 0) return;
    if ((block->BLK_HEADER_FLAGS) & BLK_FLAG_RESIDENT) return;
    BlkValueDestroy(block);
    if ((block->BLK_HEADER_FLAGS) & BLK_FLAG_MULTIPLE) {
        if (block-->BLK_PREV ~= NULL) (block-->BLK_PREV)-->BLK_NEXT = NULL;
        fromtxb = block;
        for (:(block-->BLK_NEXT)~=NULL:block = block-->BLK_NEXT) ;
        while (block ~= fromtxb) {
            ptxb = block-->BLK_PREV; BlkFreeSingleBlock(block); block = ptxb;
        }
    }
    BlkFreeSingleBlock(block);
];
[ BlkFreeSingleBlock block free nx;
    block-->BLK_HEADER_KOV = 0;
    block->BLK_HEADER_FLAGS = BLK_FLAG_MULTIPLE;
    for (free = Blk_Heap:free ~= NULL:free = free-->BLK_NEXT) {
        nx = free-->BLK_NEXT;
        if (nx == NULL) {
            free-->BLK_NEXT = block;
            block-->BLK_PREV = free;
            block-->BLK_NEXT = NULL;
            BlkMerge(block);
            return;
        }
        if (UnsignedCompare(nx, block) == 1) {
            free-->BLK_NEXT = block;
            block-->BLK_PREV = free;
            block-->BLK_NEXT = nx;
            nx-->BLK_PREV = block;
            BlkMerge(block);
            return;
        }
    }
];
```

§**15. Resizing.** When the content of a value stretches or shrinks, we will sometimes need to change the size of the block(s) containing the data – though not always: we might sometimes need to resize a 1052-byte text to a 1204-byte text and find that we are sitting in a 2048-byte block in any case. We either shed blocks from the end of the chain, or add new blocks at the end, that being the simplest thing to do. Sometimes it might mean preserving a not very efficient block division, but it minimises the churn of blocks being allocated and freed, which is probably good.

```
[ BlkResize block req newsize dsize newblk kov n i otxb flags;
    if (block == 0) "*** Cannot resize null block ***";
    kov = block-->BLK_HEADER_KOV;
    flags = block->BLK_HEADER_FLAGS;
    if (flags & BLK_FLAG_MULTIPLE == 0) "*** Cannot resize inextensible block ***";
    otxb = block;
    newsize = req;
    for (:: block = block-->BLK_NEXT) {
        n = block->BLK_HEADER_N;
        for (dsize=1: n>0: n--) dsize = dsize*2;
        i = dsize - BLK_DATA_MULTI_OFFSET;
        newsize = newsize - i;
        if (newsize > 0) {
            if (block-->BLK_NEXT ~= NULL) continue;
            newblk = BlkAllocate(newsize, kov, flags);
            if (newblk == 0) rfalse;
            block-->BLK_NEXT = newblk;
            newblk-->BLK_PREV = block;
            rtrue;
        }
        if (block-->BLK_NEXT ~= NULL) {
            BlkFree(block-->BLK_NEXT);
            block-->BLK_NEXT = NULL;
        }
        rtrue;
    }
];

[ DebugHeap;
    print "Managing a heap of initially ", MEMORY_HEAP_SIZE+16, " bytes.^";
    print HeapNetFreeSpace(false), " bytes currently free.^";
    print "Free space decomposition: "; BlkDebugDecomposition(Blk_Heap);
    print "Free space map: "; BlkDebug(Blk_Heap);
];
```

§**16. Stubs.** To ensure that the template code will still compile if `MEMORY_HEAP_SIZE` is undefined and there's no heap: none of these routines do anything in such a situation, but nor are they ever called – it's just that I6 source code may refer to them anyway, so they need to exist as routine names.

```
#IFNOT; ! IFDEF MEMORY_HEAP_SIZE

[ HeapInitialise; ];
[ BlkFree; ];
[ DebugHeap;
    "This story file does not use a heap of managed memory.";
];

#ENDIF; ! IFDEF MEMORY_HEAP_SIZE
```

ZMachine Template B/zmt

Purpose

To provide routines handling low-level Z-machine facilities.

§1. Summary. This segment closely parallels "Glulx.i6t", which provides exactly equivalent functionality (indeed, usually the same-named functions and in the same order) for the Glulx VM. This is intended to make the rest of the template code independent of the choice of VM, although that is more of an ideal than a reality, because there are so many fiddly differences in some of the grammar and dictionary tables that it is not really practical for the parser (for instance) to call VM-neutral routines to get the data it wants out of these arrays.

§2. Variables and Arrays.

```
Global top_object; ! largest valid number of any tree object
Global xcommsdir; ! true if command recording is on
Global transcript_mode; ! true if game scripting is on

Constant INPUT_BUFFER_LEN = 120; ! Length of buffer array

Array  buffer    -> 123;            ! Buffer for parsing main line of input
Array  buffer2   -> 123;            ! Buffers for supplementary questions
Array  buffer3   -> 123;            ! Buffer retaining input for "again"
Array  parse     buffer 63;         ! Parse table mirroring it
Array  parse2    buffer 63;         !

Global dict_start;
Global dict_entry_size;
Global dict_end;
```

§3. Starting Up. `VM_Initialise()` is almost the first routine called, except that the "starting the virtual machine" activity is allowed to go first.

```
[ VM_Initialise i;
    standard_interpreter = HDR_TERPSTANDARD-->0;
    transcript_mode = ((HDR_GAMEFLAGS-->0) & 1);

    dict_start = HDR_DICTIONARY-->0;
    dict_entry_size = dict_start->(dict_start->0 + 1);
    dict_start = dict_start + dict_start->0 + 4;
    dict_end = dict_start + ((dict_start - 2)-->0) * dict_entry_size;

    buffer->0  = INPUT_BUFFER_LEN;
    buffer2->0 = INPUT_BUFFER_LEN;
    buffer3->0 = INPUT_BUFFER_LEN;
    parse->0   = 15;
    parse2->0  = 15;

    top_object = #largest_object-255;

    #ifdef FIX_RNG;
    @random 10000 -> i;
    i = -i-2000;
    print "[Random number generator seed is ", i, "]^";
    @random i -> i;
    #endif; ! FIX_RNG
];
```

§4. Enable Acceleration. This rule enables use of March 2009 extension to Glulx which optimises the speed of Inform-compiled story files, so for the Z-machine it has no effect.

```
[ ENABLE_GLULX_ACCEL_R;
    rfalse;
];
```

§5. Release Number. Like all software, IF story files have release numbers to mark revised versions being circulated: unlike most software, and partly for traditional reasons, the version number is recorded not in some print statement or variable but is branded on, so to speak, in a specific memory location of the story file header.

`VM_Describe_Release()` describes the release and is used as part of the "banner", IF's equivalent to a title page.

```
[ VM_Describe_Release i;
    print "Release ", (HDR_GAMERELEASE-->0) & $03ff, " / Serial number ";
    for (i=0 : i<6 : i++) print (char) HDR_GAMESERIAL->i;
];
```

§6. Keyboard Input. The VM must provide three routines for keyboard input:

(a) `VM_KeyChar()` waits for a key to be pressed and then returns the character chosen as a ZSCII character.
(b) `VM_KeyDelay(N)` waits up to $N/10$ seconds for a key to be pressed, returning the ZSCII character if so, or 0 if not.
(c) `VM_ReadKeyboard(b, t)` reads a whole newline-terminated command into the buffer `b`, then parses it into a word stream in the table `t`.

There are elaborations to due with mouse clicks, but this isn't the place to document all of that.

```
[ VM_KeyChar win  key;
    if (win) @set_window win;
    @read_char 1 -> key;
    return key;
];
[ VM_KeyDelay tenths  key;
    @read_char 1 tenths VM_KeyDelay_Interrupt -> key;
    return key;
];
[ VM_KeyDelay_Interrupt; rtrue; ];
[ VM_ReadKeyboard  a_buffer a_table i;
    read a_buffer a_table;
    #ifdef ECHO_COMMANDS;
    print "** ";
    for (i=2: i<=(a_buffer->1)+1: i++) print (char) a_buffer->i;
    print "^";
    #ifnot;
    i=0;  ! suppress compiler warning
    #endif;
    #Iftrue (#version_number == 6);
    @output_stream -1;
    @loadb a_buffer 1 -> sp;
    @add a_buffer 2 -> sp;
    @print_table sp sp;
    new_line;
    @output_stream 1;
    #Endif;
];
```

§7. Buffer Functions. A "buffer", in this sense, is an array containing a stream of characters typed from the keyboard; a "parse buffer" is an array which resolves this into individual words, pointing to the relevant entries in the dictionary structure. Because each VM has its own format for each of these arrays (not to mention the dictionary), we have to provide some standard operations needed by the rest of the template as routines for each VM.

The Z-machine buffer and parse buffer formats are documented in the DM4.

`VM_CopyBuffer(to, from)` copies one buffer into another.

`VM_Tokenise(buff, parse_buff)` takes the text in the buffer `buff` and produces the corresponding data in the parse buffer `parse_buff` – this is called tokenisation since the characters are divided into words: in traditional computing jargon, such clumps of characters treated syntactically as units are called tokens.

`LTI_Insert` is documented in the DM4 and the `LTI` prefix stands for "Language To Informese": it's used only by translations into non-English languages of play, and is not called in the template.

```
[ VM_CopyBuffer bto bfrom i;
    for (i=0: i<INPUT_BUFFER_LEN: i++) bto->i = bfrom->i;
];

[ VM_PrintToBuffer buf len a b c;
    @output_stream 3 buf;
    switch (metaclass(a)) {
        String: print (string) a;
        Routine: a(b, c);
        Object, Class: if (b) PrintOrRun(a, b, true); else print (name) a;
    }
    @output_stream -3;
    if (buf-->0 > len) print "Error: Overflow in VM_PrintToBuffer.^";
    return buf-->0;
];

[ VM_Tokenise b p; b->(2 + b->1) = 0; @tokenise b p; ];

[ LTI_Insert i ch  b y;
    ! Protect us from strict mode, as this isn't an array in quite the
    ! sense it expects
    b = buffer;

    ! Insert character ch into buffer at point i.
    ! Being careful not to let the buffer possibly overflow:
    y = b->1;
    if (y > b->0) y = b->0;

    ! Move the subsequent text along one character:
    for (y=y+2 : y>i : y--) b->y = b->(y-1);
    b->i = ch;

    ! And the text is now one character longer:
    if (b->1 < b->0) (b->1)++;
];
```

§8. Dictionary Functions. Again, the dictionary structure is differently arranged on the different VMs. This is a data structure containing, in compressed form, the text of all the words to be recognised by tokenisation (above). In I6 for Z, a dictionary word value is represented at run-time by its record number in the dictionary, 0, 1, 2, ..., in alphabetical order.

`VM_InvalidDictionaryAddress(A)` tests whether `A` is a valid record address in the dictionary data structure.

`VM_DictionaryAddressToNumber(A)` and `VM_NumberToDictionaryAddress(N)` convert between record numbers and dictionary addresses.

```
[ VM_InvalidDictionaryAddress addr;
    if ((UnsignedCompare(addr, dict_start) < 0) ||
        (UnsignedCompare(addr, dict_end) >= 0) ||
        ((addr - dict_start) % dict_entry_size ~= 0)) rtrue;
    rfalse;
];
[ VM_DictionaryAddressToNumber w; return (w-(HDR_DICTIONARY-->0 + 7))/9; ];
[ VM_NumberToDictionaryAddress n; return HDR_DICTIONARY-->0 + 7 + 9*n; ];
```

§9. Command Tables. The VM is also generated containing a data structure for the grammar produced by I6's `Verb` and `Extend` directives: this is essentially a list of command verbs such as DROP or PUSH, together with a list of synonyms, and then the grammar for the subsequent commands to be recognised by the parser.

```
[ VM_CommandTableAddress i;
    return (HDR_STATICMEMORY-->0)-->i;
];
[ VM_PrintCommandWords i da j;
    da = HDR_DICTIONARY-->0;
    for (j=0 : j<(da+5)-->0 : j++)
        if (da->(j*9 + 14) == $ff-i)
            print "'", (address) VM_NumberToDictionaryAddress(j), "' ";
];
```

§10. SHOWVERB support. Further VM-specific tables cover actions and attributes, and these are used by the SHOWVERB testing command.

```
#Ifdef DEBUG;
[ DebugAction a anames;
    if (a >= 4096) { print "<fake action ", a-4096, ">"; return; }
    anames = #identifiers_table;
    anames = anames + 2*(anames-->0) + 2*48;
    print (string) anames-->a;
];
[ DebugAttribute a anames;
    if (a < 0 || a >= 48) print "<invalid attribute ", a, ">";
    else {
        anames = #identifiers_table; anames = anames + 2*(anames-->0);
        print (string) anames-->a;
    }
];
#Endif;
```

§**11. RNG.** No routine is needed for extracting a random number, since I6's built-in `random` function does that, but it's useful to abstract the process of seeding the RNG so that it produces a repeatable sequence of "random" numbers from here on: the necessary opcodes are different for the two VMs.

```
[ VM_Seed_RNG n;
    if (n > 0) n = -n;
    @random n -> n;
];
```

§**12. Memory Allocation.** This is dynamic memory allocation: something which is never practicable in the Z-machine, because the whole address range is already claimed, but which is viable on recent revisions of Glulx.

```
[ VM_AllocateMemory amount;
    return 0;
];
[ VM_FreeMemory address;
];
```

§**13. Audiovisual Resources.** The Z-machine only barely supports figures and sound effects. I7 only allows us to use them for Version 6 of the Z-machine, even though sound effects have a longer pedigree and Infocom used them on some version 5 and even some version 3 works: really, though, from an I7 point of view we would prefer that anyone needing figures and sounds use Glulx instead.

```
[ VM_Picture resource_ID;
    #IFTRUE #version_number == 6; ! Z-machine version 6
    @draw_picture resource_ID;
    #ENDIF;
];
[ VM_SoundEffect resource_ID;
    #IFTRUE #version_number == 6; ! Z-machine version 6
    @sound_effect resource_ID;
    #ENDIF;
];
```

§**14. Typography.** Relatively few typographic effects are available on the Z-machine, so that many of the semantic markups for text which would be distinguishable on Glulx are indistinguishable here.

```
[ VM_Style sty;
    switch (sty) {
        NORMAL_VMSTY, NOTE_VMSTY: style roman;
        HEADER_VMSTY, SUBHEADER_VMSTY, ALERT_VMSTY: style bold;
    }
];
```

§15. Character Casing. The following are the equivalent of `tolower` and `toupper`, the traditional C library functions for forcing letters into lower and upper case form, for the ZSCII character set.

```
[ VM_UpperToLowerCase c;
switch (c) {
    'A' to 'Z': c = c + 32;
    202, 204, 212, 214, 221: c--;
    217, 218: c = c - 2;
    158 to 160, 167 to 169, 208 to 210: c = c - 3;
    186 to 190, 196 to 200: c = c - 5 ;
    175 to 180: c = c - 6;
}
return c;
];
[ VM_LowerToUpperCase c;
switch (c) {
    'a' to 'z': c = c - 32;
    201, 203, 211, 213, 220: c++;
    215, 216: c = c + 2;
    155 to 157, 164 to 166, 205 to 207: c = c + 3;
    181 to 185, 191 to 195: c = c + 5 ;
    169 to 174: c = c + 6;
}
return c;
];
```

§16. The Screen. Our generic screen model is that the screen is made up of windows: we tend to refer only to two of these, the main window and the status line, but others may also exist from time to time. Windows have unique ID numbers: the special window ID −1 means "all windows" or "the entire screen", which usually amounts to the same thing.

Screen height and width are measured in characters, with respect to the fixed-pitch font used for the status line. The main window normally contains variable-pitch text which may even have been kerned, and character dimensions make little sense there.

Clearing all windows (`WIN_ALL` here) has the side-effect of collapsing the status line, so we need to ensure that `statuswin_cursize` is reduced to 0, in order to keep it accurate.

```
[ VM_ClearScreen window;
    switch (window) {
        WIN_ALL:    @erase_window -1; statuswin_cursize = 0;
        WIN_STATUS: @erase_window 1;
        WIN_MAIN:   @erase_window 0;
    }
];
#Iftrue (#version_number == 6);
[ VM_ScreenWidth  width charw;
    @get_wind_prop 1 3 -> width;
    @get_wind_prop 1 13 -> charw;
    charw = charw & $FF;
    return (width+charw-1) / charw;
];
#Ifnot;
[ VM_ScreenWidth; return (HDR_SCREENWCHARS->0); ];
```

```
#Endif;
[ VM_ScreenHeight; return (HDR_SCREENHLINES->0); ];
```

§17. Window Colours. Each window can have its own foreground and background colours.

The colour of individual letters or words of type is not controllable in Glulx, to the frustration of many, and so the template layer of I7 has no framework for handling this (even though it is controllable on the Z-machine, which is greatly superior in this respect).

```
[ VM_SetWindowColours f b window;
    if (clr_on && f && b) {
        if (window == 0) {  ! if setting both together, set reverse
            clr_fgstatus = b;
            clr_bgstatus = f;
            }
        if (window == 1) {
            clr_fgstatus = f;
            clr_bgstatus = b;
        }
        if (window == 0 or 2) {
            clr_fg = f;
            clr_bg = b;
        }
        if (statuswin_current)
            @set_colour clr_fgstatus clr_bgstatus;
        else
            @set_colour clr_fg clr_bg;
    }
];
[ VM_RestoreWindowColours; ! compare I6 library patch L61007
    if (clr_on) { ! check colour has been used
        VM_SetWindowColours(clr_fg, clr_bg, 2); ! make sure both sets of variables are restored
        VM_SetWindowColours(clr_fgstatus, clr_bgstatus, 1, true);
        VM_ClearScreen();
    }
    #Iftrue (#version_number == 6); ! request screen update
    (0-->8) = (0-->8) | $$00000100;
    #Endif;
];
```

§18. Main Window. The part of the screen on which commands and responses are printed, which ordinarily occupies almost all of the screen area.

`VM_MainWindow()` switches printing back from another window, usually the status line, to the main window. Note that the Z-machine implementation emulates the Glulx model of window rather than text colours.

```
[ VM_MainWindow;
    if (statuswin_current) {
        if (clr_on && clr_bgstatus > 1) @set_colour clr_fg clr_bg;
        else style roman;
        @set_window 0;
    }
    statuswin_current = false;
];
```

§19. Status Line. Despite the name, the status line need not be a single line at the top of the screen: that's only the conventional default arrangement. It can expand to become the equivalent of an old-fashioned VT220 terminal, with menus and grids and mazes displayed lovingly in character graphics, or it can close up to invisibility.

`VM_StatusLineHeight(n)` sets the status line to have a height of `n` lines of type. (The width of the status line is always the width of the whole screen, and the position is always at the top, so the height is the only controllable aspect.) The $n = 0$ case makes the status line disappear.

`VM_MoveCursorInStatusLine(x, y)` switches printing to the status line, positioning the "cursor" – the position at which printing will begin – at the given character grid position (x, y). Line 1 represents the top line; line 2 is underneath, and so on; columns are similarly numbered from 1 at the left.

```
[ VM_MoveCursorInStatusLine line column; ! 1-based position on text grid
    if (~~statuswin_current) {
        @set_window 1;
        if (clr_on && clr_bgstatus > 1) @set_colour clr_fgstatus clr_bgstatus;
        else                            style reverse;
    }
    if (line == 0) {
        line = 1;
        column = 1;
    }
    #Iftrue (#version_number == 6);
    Z6_MoveCursor(line, column);
    #Ifnot;
    @set_cursor line column;
    #Endif;
    statuswin_current = true;
];

#Iftrue (#version_number == 6);
[ Z6_MoveCursor line column  charw charh; ! 1-based position on text grid
    @get_wind_prop 1 13 -> charw; ! font size
    @log_shift charw $FFF8 -> charh;
    charw = charw / $100;
    line = 1 + charh*(line-1);
    column = 1 + charw*(column-1);
    @set_cursor line column;
];
#Endif;
```

```
#Iftrue (#version_number == 6);
[ VM_StatusLineHeight height  wx wy x y charh;
    ! Split the window. Standard 1.0 interpreters should keep the window 0
    ! cursor in the same absolute position, but older interpreters,
    ! including Infocom's don't - they keep the window 0 cursor in the
    ! same position relative to its origin. We therefore compensate
    ! manually.
    @get_wind_prop 0 0 -> wy; @get_wind_prop 0 1 -> wx;
    @get_wind_prop 0 13 -> charh; @log_shift charh $FFF8 -> charh;
    @get_wind_prop 0 4 -> y; @get_wind_prop 0 5 -> x;
    height = height * charh;
    @split_window height;
    y = y - height + wy - 1;
    if (y < 1) y = 1;
    x = x + wx - 1;
    @set_cursor y x 0;
    statuswin_cursize = height;
];
#Ifnot;
[ VM_StatusLineHeight height;
    if (statuswin_cursize ~= height)
        @split_window height;
    statuswin_cursize = height;
];
#Endif;
```

```
#Iftrue (#version_number == 6);
[ Z6_DrawStatusLine width x charw scw;
    (0-->8) = (0-->8) &~ $$00000100;
    @push say__p; @push say__pc;
    BeginActivity(CONSTRUCTING_STATUS_LINE_ACT);
    VM_StatusLineHeight(statuswin_size);
    ! Now clear the window. This isn't totally trivial. Our approach is to select the
    ! fixed space font, measure its width, and print an appropriate
    ! number of spaces. We round up if the screen isn't a whole number
    ! of characters wide, and rely on window 1 being set to clip by default.
    VM_MoveCursorInStatusLine(1, 1);
    @set_font 4 -> x;
    width = VM_ScreenWidth();
    spaces width;
    ClearParagraphing();
    if (ForActivity(CONSTRUCTING_STATUS_LINE_ACT) == false) {
        ! Back to standard font for the display. We use output_stream 3 to
        ! measure the space required, the aim being to get 50 characters
        ! worth of space for the location name.
        VM_MoveCursorInStatusLine(1, 2);
        @set_font 1 -> x;
        switch (metaclass(left_hand_status_line)) {
            String: print (string) left_hand_status_line;
            Routine: left_hand_status_line();
        }
        @get_wind_prop 1 3 -> width;
        @get_wind_prop 1 13 -> charw;
        charw = charw & $FF;
        @output_stream 3 StorageForShortName;
        print (PrintText) right_hand_status_line;
        @output_stream -3; scw = HDR_PIXELSTO3-->0 + charw;
        x = 1+width-scw;
        @set_cursor 1 x; print (PrintText) right_hand_status_line;
    }
    ! Reselect roman, as Infocom's interpreters go funny if reverse is selected twice.
    VM_MainWindow();
    ClearParagraphing();
    EndActivity(CONSTRUCTING_STATUS_LINE_ACT);
    @pull say__pc; @pull say__p;
];
#Endif;
```

§20. Quotation Boxes. No routine is needed to produce quotation boxes: the I6 `box` statement generates the necessary Z-machine opcodes all by itself.

§21. Undo. These simply wrap the relevant opcodes.

```
[ VM_Undo result_code;
    @restore_undo result_code;
    return result_code;
];
[ VM_Save_Undo result_code;
    @save_undo result_code;
    return result_code;
];
```

§22. Quit The Game Rule.

```
[ QUIT_THE_GAME_R; if (actor ~= player) rfalse;
    GL__M(##Quit,2); if (YesOrNo()~=0) quit; ];
```

§23. Restart The Game Rule.

```
[ RESTART_THE_GAME_R;
    if (actor ~= player) rfalse;
    GL__M(##Restart,1);
    if (YesOrNo()~=0) { @restart; GL__M(##Restart,2); }
];
```

§24. Restore The Game Rule.

```
[ RESTORE_THE_GAME_R;
    if (actor ~= player) rfalse;
    restore Rmaybe;
    return GL__M(##Restore,1);
    .RMaybe; GL__M(##Restore,2);
];
```

§25. Save The Game Rule.

```
[ SAVE_THE_GAME_R flag;
    if (actor ~= player) rfalse;
    #IFV5;
    @save -> flag;
    switch (flag) {
        0: GL__M(##Save,1);
        1: GL__M(##Save,2);
        2: GL__M(##Restore,2);
    }
    #IFNOT;
    save Smaybe;
    return GL__M(##Save,1);
    .SMaybe; GL__M(##Save,2);
    #ENDIF;
];
```

§26. Verify The Story File Rule. This is a fossil now, really, but in the days of Infocom, the 110K story file occupying an entire disc was a huge data set: floppy discs were by no means a reliable medium, and cheap hardware often used hit-and-miss components, as on the notorious Commodore 64 disc controller. If somebody experienced an apparent bug in play, it could easily be that he had a corrupt disc or was unable to read data of that density. So the VERIFY command, which took up to ten minutes on some early computers, would chug through the entire story file and compute a checksum, compare it against a known result in the header, and determine that the story file could or could not properly be read. The Z-machine provided this service as an opcode, and so Glulx followed suit.

```
[ VERIFY_THE_STORY_FILE_R;
    if (actor ~= player) rfalse;
    @verify ?Vmaybe;
    jump Vwrong;
    .Vmaybe; return GL__M(##Verify,1);
    .Vwrong;
    GL__M(##Verify,2);
];
```

§27. Switch Transcript On Rule.

```
[ SWITCH_TRANSCRIPT_ON_R;
    if (actor ~= player) rfalse;
    transcript_mode = ((0-->8) & 1);
    if (transcript_mode) return GL__M(##ScriptOn,1);
    @output_stream 2;
    if (((0-->8) & 1) == 0) return GL__M(##ScriptOn,3);
    GL__M(##ScriptOn,2); VersionSub();
    transcript_mode = true;
];
```

§28. Switch Transcript Off Rule.

```
[ SWITCH_TRANSCRIPT_OFF_R;
    if (actor ~= player) rfalse;
    transcript_mode = ((0-->8) & 1);
    if (transcript_mode == false) return GL__M(##ScriptOff,1);
    GL__M(##ScriptOff,2);
    @output_stream -2;
    if ((0-->8) & 1) return GL__M(##ScriptOff,3);
    transcript_mode = false;
];
```

§29. Announce Story File Version Rule.

```
[ ANNOUNCE_STORY_FILE_VERSION_R ix;
    if (actor ~= player) rfalse;
    Banner();
    print "Identification number: ";
    for (ix=6: ix <= UUID_ARRAY->0: ix++) print (char) UUID_ARRAY->ix;
    print "^";
    ix = 0; ! shut up compiler warning
    if (standard_interpreter > 0) {
        print "Standard interpreter ",
            standard_interpreter/256, ".", standard_interpreter%256,
            " (", HDR_TERPNUMBER->0;
        #Iftrue (#version_number == 6);
        print (char) '.', HDR_TERPVERSION->0;
        #Ifnot;
        print (char) HDR_TERPVERSION->0;
        #Endif;
        print ") / ";
    } else {
        print "Interpreter ", HDR_TERPNUMBER->0, " Version ";
        #Iftrue (#version_number == 6);
        print HDR_TERPVERSION->0;
        #Ifnot;
        print (char) HDR_TERPVERSION->0;
        #Endif;
        print " / ";
    }
    print "Library serial number ", (string) LibSerial, "^";
    #Ifdef LanguageVersion;
    print (string) LanguageVersion, "^";
    #Endif; ! LanguageVersion
    #ifdef ShowExtensionVersions;
    ShowExtensionVersions();
    #endif;
    say__p = 1;
];
```

§30. Descend To Specific Action Rule. There are 100 or so actions, typically, and this rule is for efficiency's sake: rather than perform 100 or so comparisons to see which routine to call, we indirect through a jump table. The routines called are the `-Sub` routines: thus, for instance, if `action` is `##Wait` then `WaitSub` is called. It is essential that this routine not be called for fake actions: in I7 use this is guaranteed, since fake actions are not allowed into the action machinery at all.

Strangely, Glulx's action routines table is numbered in an off-by-one way compared to the Z-machine's: hence the `+1`.

```
[ DESCEND_TO_SPECIFIC_ACTION_R;
    indirect(#actions_table-->action);
    rtrue;
];
```

§31. Veneer.

```
[ OhLookItsReal; ];
[ OhLookItsRoom; ];
[ OhLookItsThing; ];
[ OC__Cl obj cla j a n objflag;
! if (cla > 4) OhLookItsReal();
! if (cla == K1_room) OhLookItsRoom();
! if (cla == K2_thing) OhLookItsThing();
    @jl obj 1 ?NotObj;
    @jg obj max_z_object ?NotObj;
    @inc objflag;
    @je cla K1_room ?~NotRoom;
    @test_attr obj mark_as_room ?rtrue;
    @rfalse;
    .NotRoom;
    @je cla K2_thing ?~NotObj;
    @test_attr obj mark_as_thing ?rtrue;
    @rfalse;
    .NotObj;
    @je cla Object Class ?ObjOrClass;
    @je cla Routine String ?RoutOrStr;
    @jin cla 1 ?~Mistake;
    @jz objflag ?rfalse;
    @get_prop_addr obj 2 -> a;
    @jz a ?rfalse;
    @get_prop_len a -> n;
    @div n 2 -> n;
    .Loop;
    @loadw a j -> sp;
    @je sp cla ?rtrue;
    @inc j;
    @jl j n ?Loop;
    @rfalse;
    .ObjOrClass;
    @jz objflag ?rfalse;
    @je cla Object ?JustObj;
    ! So now cla is Class
    @jg obj String ?~rtrue;
    @jin obj Class ?rtrue;
    @rfalse;
    .JustObj;
    ! So now cla is Object
    @jg obj String ?~rfalse;
    @jin obj Class ?rfalse;
    @rtrue;
    .RoutOrStr;
    @jz objflag ?~rfalse;
    @call_2s Z__Region obj -> sp;
    @inc sp;
    @je sp cla ?rtrue;
    @rfalse;
    .Mistake;
    RT__Err("apply 'ofclass' for", cla, -1);
    rfalse;
];
[ Unsigned__Compare x y u v;
    @je x y ?rfalse; ! i.e., return 0
    @jl x 0 ?XNegative;
    ! So here x >= 0 and x ~= y
    @jl y 0 ?XPosYNeg;
    ! Here x >=0, y >= 0, x ~= y
    @jg x y ?rtrue; ! i.e., return 1
    @ret -1;
    .XPosYNeg;
    ! Here x >= 0, y < 0, x ~= y
    @ret -1;
    .XNegative;
    @jl y 0 ?~rtrue; ! if x < 0, y >= 0, return 1
    ! Here x < 0, y < 0, x ~= y
    @jg x y ?rtrue;
    @ret -1;
];
[ RT__ChLDW base offset;
    @loadw base offset -> sp;
    @ret sp;
];
```

Glulx Template B/glut

Purpose

To start up the Glk interface for the Glulx virtual machine, and provide Glulx-specific printing functions.

§**1. Summary.** This segment closely parallels “ZMachine.i6t”, which provides exactly equivalent functionality (indeed, usually the same-named functions and in the same order) for the Z-machine VM. This is intended to make the rest of the template code independent of the choice of VM, although that is more of an ideal than a reality, because there are so many fiddly differences in some of the grammar and dictionary tables that it is not really practical for the parser (for instance) to call VM-neutral routines to get the data it wants out of these arrays.

§2. Variables and Arrays.

```
Array gg_event --> 4;
Array gg_arguments buffer 28;
Global gg_mainwin = 0;
Global gg_statuswin = 0;
Global gg_quotewin = 0;
Global gg_scriptfref = 0;
Global gg_scriptstr = 0;
Global gg_savestr = 0;
Global gg_commandstr = 0;
Global gg_command_reading = 0;      ! true if gg_commandstr is being replayed
Global gg_foregroundchan = 0;
Global gg_backgroundchan = 0;

Constant INPUT_BUFFER_LEN = 260;    ! No extra byte necessary
Constant MAX_BUFFER_WORDS = 20;
Constant PARSE_BUFFER_LEN = 61;

Array  buffer    buffer INPUT_BUFFER_LEN;
Array  buffer2   buffer INPUT_BUFFER_LEN;
Array  buffer3   buffer INPUT_BUFFER_LEN;
Array  parse     --> PARSE_BUFFER_LEN;
Array  parse2    --> PARSE_BUFFER_LEN;
```

§**3. Infglk.** This section is a verbatim copy of John Cater's invaluable I6 header file `infglk.h`, kindly donated by the author. The routines are convenient to have on hand, and also provide a canonical set of I6 names for the many gestalt and other codes.

```
#IFNDEF infglk_h; ! Standard Glulx definitions contributed by John Cater
Constant infglk_h;
!------------------------------------------------------------------------------
!  infglk.h - an Inform library to allow easy access to glk functions
!    under glulx
!  Dynamically created by glk2inf.pl on 08/31/2006 at 19:20:21.
!  Send comments or suggestions to: katre@ruf.rice.edu
!------------------------------------------------------------------------------
#Ifdef infglk_h;  ! remove "Constant declared but not used" warnings
#Endif;
Constant GLK_NULL 0;
! Constant definitions from glk.h
Constant gestalt_Version 0;
Constant gestalt_CharInput 1;
Constant gestalt_LineInput 2;
Constant gestalt_CharOutput 3;
Constant gestalt_CharOutput_CannotPrint 0;
Constant gestalt_CharOutput_ApproxPrint 1;
Constant gestalt_CharOutput_ExactPrint 2;
Constant gestalt_MouseInput 4;
Constant gestalt_Timer 5;
Constant gestalt_Graphics 6;
Constant gestalt_DrawImage 7;
Constant gestalt_Sound 8;
Constant gestalt_SoundVolume 9;
Constant gestalt_SoundNotify 10;
Constant gestalt_Hyperlinks 11;
Constant gestalt_HyperlinkInput 12;
Constant gestalt_SoundMusic 13;
Constant gestalt_GraphicsTransparency 14;
Constant gestalt_Unicode 15;
Constant evtype_None 0;
Constant evtype_Timer 1;
Constant evtype_CharInput 2;
Constant evtype_LineInput 3;
Constant evtype_MouseInput 4;
Constant evtype_Arrange 5;
Constant evtype_Redraw 6;
Constant evtype_SoundNotify 7;
Constant evtype_Hyperlink 8;
Constant keycode_Unknown $ffffffff;
Constant keycode_Left $fffffffe;
Constant keycode_Right $fffffffd;
Constant keycode_Up $fffffffc;
Constant keycode_Down $fffffffb;
Constant keycode_Return $fffffffa;
Constant keycode_Delete $fffffff9;
Constant keycode_Escape $fffffff8;
Constant keycode_Tab $fffffff7;
Constant keycode_PageUp $fffffff6;
Constant keycode_PageDown $fffffff5;
Constant keycode_Home $fffffff4;
Constant keycode_End $fffffff3;
Constant keycode_Func1 $ffffffef;
Constant keycode_Func2 $ffffffee;
Constant keycode_Func3 $ffffffed;
Constant keycode_Func4 $ffffffec;
Constant keycode_Func5 $ffffffeb;
Constant keycode_Func6 $ffffffea;
Constant keycode_Func7 $ffffffe9;
Constant keycode_Func8 $ffffffe8;
Constant keycode_Func9 $ffffffe7;
Constant keycode_Func10 $ffffffe6;
Constant keycode_Func11 $ffffffe5;
Constant keycode_Func12 $ffffffe4;
Constant keycode_MAXVAL 28;
Constant style_Normal 0;
Constant style_Emphasized 1;
Constant style_Preformatted 2;
Constant style_Header 3;
Constant style_Subheader 4;
Constant style_Alert 5;
Constant style_Note 6;
Constant style_BlockQuote 7;
Constant style_Input 8;
Constant style_User1 9;
Constant style_User2 10;
Constant style_NUMSTYLES 11;
Constant wintype_AllTypes 0;
Constant wintype_Pair 1;
Constant wintype_Blank 2;
Constant wintype_TextBuffer 3;
Constant wintype_TextGrid 4;
Constant wintype_Graphics 5;
Constant winmethod_Left $00;
Constant winmethod_Right $01;
Constant winmethod_Above $02;
Constant winmethod_Below $03;
Constant winmethod_DirMask $0f;
Constant winmethod_Fixed $10;
Constant winmethod_Proportional $20;
Constant winmethod_DivisionMask $f0;
Constant fileusage_Data $00;
Constant fileusage_SavedGame $01;
Constant fileusage_Transcript $02;
Constant fileusage_InputRecord $03;
Constant fileusage_TypeMask $0f;
Constant fileusage_TextMode $100;
Constant fileusage_BinaryMode $000;
Constant filemode_Write $01;
Constant filemode_Read $02;
Constant filemode_ReadWrite $03;
Constant filemode_WriteAppend $05;
Constant seekmode_Start 0;
Constant seekmode_Current 1;
Constant seekmode_End 2;
Constant stylehint_Indentation 0;
Constant stylehint_ParaIndentation 1;
Constant stylehint_Justification 2;
Constant stylehint_Size 3;
Constant stylehint_Weight 4;
Constant stylehint_Oblique 5;
Constant stylehint_Proportional 6;
Constant stylehint_TextColor 7;
Constant stylehint_BackColor 8;
Constant stylehint_ReverseColor 9;
Constant stylehint_NUMHINTS 10;
Constant stylehint_just_LeftFlush 0;
Constant stylehint_just_LeftRight 1;
Constant stylehint_just_Centered 2;
Constant stylehint_just_RightFlush 3;
Constant imagealign_InlineUp $01;
Constant imagealign_InlineDown $02;
Constant imagealign_InlineCenter $03;
Constant imagealign_MarginLeft $04;
Constant imagealign_MarginRight $05;

! The actual glk functions.
[ glk_exit _vararg_count ret;
! glk_exit ()
! And now the @glk call
@glk 1 _vararg_count ret;
return ret;
];

[ glk_set_interrupt_handler _vararg_count ret;
! glk_set_interrupt_handler (func)
! And now the @glk call
@glk 2 _vararg_count ret;
return ret;
];

[ glk_tick _vararg_count ret;
! glk_tick ()
! And now the @glk call
@glk 3 _vararg_count ret;
return ret;
];

[ glk_gestalt _vararg_count ret;
! glk_gestalt (sel val)
! And now the @glk call
@glk 4 _vararg_count ret;
return ret;
];

[ glk_gestalt_ext _vararg_count ret;
! glk_gestalt_ext (sel val arr arrlen)
! And now the @glk call
@glk 5 _vararg_count ret;
return ret;
];
[ glk_char_to_lower _vararg_count ret;
! glk_char_to_lower (ch)
! And now the @glk call
@glk 160 _vararg_count ret;
return ret;
];
[ glk_char_to_upper _vararg_count ret;
! glk_char_to_upper (ch)
! And now the @glk call
@glk 161 _vararg_count ret;
return ret;
];
[ glk_window_get_root _vararg_count ret;
! glk_window_get_root ()
! And now the @glk call
@glk 34 _vararg_count ret;
return ret;
];
[ glk_window_open _vararg_count ret;
! glk_window_open (split method size wintype rock)
! And now the @glk call
@glk 35 _vararg_count ret;
return ret;
];
[ glk_window_close _vararg_count ret;
! glk_window_close (win result)
! And now the @glk call
@glk 36 _vararg_count ret;
return ret;
];
[ glk_window_get_size _vararg_count ret;
! glk_window_get_size (win widthptr heightptr)
! And now the @glk call
@glk 37 _vararg_count ret;
return ret;
];
[ glk_window_set_arrangement _vararg_count ret;
! glk_window_set_arrangement (win method size keywin)
! And now the @glk call
@glk 38 _vararg_count ret;
return ret;
];
[ glk_window_get_arrangement _vararg_count ret;
! glk_window_get_arrangement (win methodptr sizeptr keywinptr)
! And now the @glk call
@glk 39 _vararg_count ret;
return ret;
];
```

```
[ glk_window_iterate _vararg_count ret;
! glk_window_iterate (win rockptr)
! And now the @glk call
@glk 32 _vararg_count ret;
return ret;
];
[ glk_window_get_rock _vararg_count ret;
! glk_window_get_rock (win)
! And now the @glk call
@glk 33 _vararg_count ret;
return ret;
];
[ glk_window_get_type _vararg_count ret;
! glk_window_get_type (win)
! And now the @glk call
@glk 40 _vararg_count ret;
return ret;
];
[ glk_window_get_parent _vararg_count ret;
! glk_window_get_parent (win)
! And now the @glk call
@glk 41 _vararg_count ret;
return ret;
];
[ glk_window_get_sibling _vararg_count ret;
! glk_window_get_sibling (win)
! And now the @glk call
@glk 48 _vararg_count ret;
return ret;
];
[ glk_window_clear _vararg_count ret;
! glk_window_clear (win)
! And now the @glk call
@glk 42 _vararg_count ret;
return ret;
];
[ glk_window_move_cursor _vararg_count ret;
! glk_window_move_cursor (win xpos ypos)
! And now the @glk call
@glk 43 _vararg_count ret;
return ret;
];
[ glk_window_get_stream _vararg_count ret;
! glk_window_get_stream (win)
! And now the @glk call
@glk 44 _vararg_count ret;
return ret;
];
[ glk_window_set_echo_stream _vararg_count ret;
! glk_window_set_echo_stream (win str)
! And now the @glk call
@glk 45 _vararg_count ret;
return ret;
];
[ glk_window_get_echo_stream _vararg_count ret;
! glk_window_get_echo_stream (win)
! And now the @glk call
@glk 46 _vararg_count ret;
return ret;
];
[ glk_set_window _vararg_count ret;
! glk_set_window (win)
! And now the @glk call
@glk 47 _vararg_count ret;
return ret;
];
[ glk_stream_open_file _vararg_count ret;
! glk_stream_open_file (fileref fmode rock)
! And now the @glk call
@glk 66 _vararg_count ret;
return ret;
];
[ glk_stream_open_memory _vararg_count ret;
! glk_stream_open_memory (buf buflen fmode rock)
! And now the @glk call
@glk 67 _vararg_count ret;
return ret;
];
[ glk_stream_close _vararg_count ret;
! glk_stream_close (str result)
! And now the @glk call
@glk 68 _vararg_count ret;
return ret;
];
[ glk_stream_iterate _vararg_count ret;
! glk_stream_iterate (str rockptr)
! And now the @glk call
@glk 64 _vararg_count ret;
return ret;
];
[ glk_stream_get_rock _vararg_count ret;
! glk_stream_get_rock (str)
! And now the @glk call
@glk 65 _vararg_count ret;
return ret;
];
[ glk_stream_set_position _vararg_count ret;
! glk_stream_set_position (str pos seekmode)
! And now the @glk call
@glk 69 _vararg_count ret;
return ret;
];
```

```
[ glk_stream_get_position _vararg_count ret;
! glk_stream_get_position (str)
! And now the @glk call
@glk 70 _vararg_count ret;
return ret;
];
[ glk_stream_set_current _vararg_count ret;
! glk_stream_set_current (str)
! And now the @glk call
@glk 71 _vararg_count ret;
return ret;
];
[ glk_stream_get_current _vararg_count ret;
! glk_stream_get_current ()
! And now the @glk call
@glk 72 _vararg_count ret;
return ret;
];
[ glk_put_char _vararg_count ret;
! glk_put_char (ch)
! And now the @glk call
@glk 128 _vararg_count ret;
return ret;
];
[ glk_put_char_stream _vararg_count ret;
! glk_put_char_stream (str ch)
! And now the @glk call
@glk 129 _vararg_count ret;
return ret;
];
[ glk_put_string _vararg_count ret;
! glk_put_string (s)
! And now the @glk call
@glk 130 _vararg_count ret;
return ret;
];
[ glk_put_string_stream _vararg_count ret;
! glk_put_string_stream (str s)
! And now the @glk call
@glk 131 _vararg_count ret;
return ret;
];
[ glk_put_buffer _vararg_count ret;
! glk_put_buffer (buf len)
! And now the @glk call
@glk 132 _vararg_count ret;
return ret;
];
[ glk_put_buffer_stream _vararg_count ret;
! glk_put_buffer_stream (str buf len)
! And now the @glk call
@glk 133 _vararg_count ret;
return ret;
];
[ glk_set_style _vararg_count ret;
! glk_set_style (styl)
! And now the @glk call
@glk 134 _vararg_count ret;
return ret;
];
[ glk_set_style_stream _vararg_count ret;
! glk_set_style_stream (str styl)
! And now the @glk call
@glk 135 _vararg_count ret;
return ret;
];
[ glk_get_char_stream _vararg_count ret;
! glk_get_char_stream (str)
! And now the @glk call
@glk 144 _vararg_count ret;
return ret;
];
[ glk_get_line_stream _vararg_count ret;
! glk_get_line_stream (str buf len)
! And now the @glk call
@glk 145 _vararg_count ret;
return ret;
];
[ glk_get_buffer_stream _vararg_count ret;
! glk_get_buffer_stream (str buf len)
! And now the @glk call
@glk 146 _vararg_count ret;
return ret;
];
[ glk_stylehint_set _vararg_count ret;
! glk_stylehint_set (wintype styl hint val)
! And now the @glk call
@glk 176 _vararg_count ret;
return ret;
];
[ glk_stylehint_clear _vararg_count ret;
! glk_stylehint_clear (wintype styl hint)
! And now the @glk call
@glk 177 _vararg_count ret;
return ret;
];
[ glk_style_distinguish _vararg_count ret;
! glk_style_distinguish (win styl1 styl2)
! And now the @glk call
@glk 178 _vararg_count ret;
return ret;
];
```

```
[ glk_style_measure _vararg_count ret;
! glk_style_measure (win styl hint result)
! And now the @glk call
@glk 179 _vararg_count ret;
return ret;
];
[ glk_fileref_create_temp _vararg_count ret;
! glk_fileref_create_temp (usage rock)
! And now the @glk call
@glk 96 _vararg_count ret;
return ret;
];
[ glk_fileref_create_by_name _vararg_count ret;
! glk_fileref_create_by_name (usage name rock)
! And now the @glk call
@glk 97 _vararg_count ret;
return ret;
];
[ glk_fileref_create_by_prompt _vararg_count ret;
! glk_fileref_create_by_prompt (usage fmode rock)
! And now the @glk call
@glk 98 _vararg_count ret;
return ret;
];
[ glk_fileref_create_from_fileref _vararg_count ret;
! glk_fileref_create_from_fileref (usage fref rock)
! And now the @glk call
@glk 104 _vararg_count ret;
return ret;
];
[ glk_fileref_destroy _vararg_count ret;
! glk_fileref_destroy (fref)
! And now the @glk call
@glk 99 _vararg_count ret;
return ret;
];
[ glk_fileref_iterate _vararg_count ret;
! glk_fileref_iterate (fref rockptr)
! And now the @glk call
@glk 100 _vararg_count ret;
return ret;
];
[ glk_fileref_get_rock _vararg_count ret;
! glk_fileref_get_rock (fref)
! And now the @glk call
@glk 101 _vararg_count ret;
return ret;
];
[ glk_fileref_delete_file _vararg_count ret;
! glk_fileref_delete_file (fref)
! And now the @glk call
@glk 102 _vararg_count ret;
return ret;
];
[ glk_fileref_does_file_exist _vararg_count ret;
! glk_fileref_does_file_exist (fref)
! And now the @glk call
@glk 103 _vararg_count ret;
return ret;
];
[ glk_select _vararg_count ret;
! glk_select (event)
! And now the @glk call
@glk 192 _vararg_count ret;
return ret;
];
[ glk_select_poll _vararg_count ret;
! glk_select_poll (event)
! And now the @glk call
@glk 193 _vararg_count ret;
return ret;
];
[ glk_request_timer_events _vararg_count ret;
! glk_request_timer_events (millisecs)
! And now the @glk call
@glk 214 _vararg_count ret;
return ret;
];
[ glk_request_line_event _vararg_count ret;
! glk_request_line_event (win buf maxlen initlen)
! And now the @glk call
@glk 208 _vararg_count ret;
return ret;
];
[ glk_request_char_event _vararg_count ret;
! glk_request_char_event (win)
! And now the @glk call
@glk 210 _vararg_count ret;
return ret;
];
[ glk_request_mouse_event _vararg_count ret;
! glk_request_mouse_event (win)
! And now the @glk call
@glk 212 _vararg_count ret;
return ret;
];
[ glk_cancel_line_event _vararg_count ret;
! glk_cancel_line_event (win event)
! And now the @glk call
@glk 209 _vararg_count ret;
return ret;
];
```

```
[ glk_cancel_char_event _vararg_count ret;
! glk_cancel_char_event (win)
! And now the @glk call
@glk 211 _vararg_count ret;
return ret;
];
[ glk_cancel_mouse_event _vararg_count ret;
! glk_cancel_mouse_event (win)
! And now the @glk call
@glk 213 _vararg_count ret;
return ret;
];
[ glk_buffer_to_lower_case_uni _vararg_count ret;
! glk_buffer_to_lower_case_uni (buf len numchars)
! And now the @glk call
@glk 288 _vararg_count ret;
return ret;
];
[ glk_buffer_to_upper_case_uni _vararg_count ret;
! glk_buffer_to_upper_case_uni (buf len numchars)
! And now the @glk call
@glk 289 _vararg_count ret;
return ret;
];
[ glk_buffer_to_title_case_uni _vararg_count ret;
! glk_buffer_to_title_case_uni (buf len numchars lowerrest)
! And now the @glk call
@glk 290 _vararg_count ret;
return ret;
];
[ glk_put_char_uni _vararg_count ret;
! glk_put_char_uni (ch)
! And now the @glk call
@glk 296 _vararg_count ret;
return ret;
];
[ glk_put_string_uni _vararg_count ret;
! glk_put_string_uni (s)
! And now the @glk call
@glk 297 _vararg_count ret;
return ret;
];
[ glk_put_buffer_uni _vararg_count ret;
! glk_put_buffer_uni (buf len)
! And now the @glk call
@glk 298 _vararg_count ret;
return ret;
];
[ glk_put_char_stream_uni _vararg_count ret;
! glk_put_char_stream_uni (str ch)
! And now the @glk call
@glk 299 _vararg_count ret;
return ret;
];
[ glk_put_string_stream_uni _vararg_count ret;
! glk_put_string_stream_uni (str s)
! And now the @glk call
@glk 300 _vararg_count ret;
return ret;
];
[ glk_put_buffer_stream_uni _vararg_count ret;
! glk_put_buffer_stream_uni (str buf len)
! And now the @glk call
@glk 301 _vararg_count ret;
return ret;
];
[ glk_get_char_stream_uni _vararg_count ret;
! glk_get_char_stream_uni (str)
! And now the @glk call
@glk 304 _vararg_count ret;
return ret;
];
[ glk_get_buffer_stream_uni _vararg_count ret;
! glk_get_buffer_stream_uni (str buf len)
! And now the @glk call
@glk 305 _vararg_count ret;
return ret;
];
[ glk_get_line_stream_uni _vararg_count ret;
! glk_get_line_stream_uni (str buf len)
! And now the @glk call
@glk 306 _vararg_count ret;
return ret;
];
[ glk_stream_open_file_uni _vararg_count ret;
! glk_stream_open_file_uni (fileref fmode rock)
! And now the @glk call
@glk 312 _vararg_count ret;
return ret;
];
[ glk_stream_open_memory_uni _vararg_count ret;
! glk_stream_open_memory_uni (buf buflen fmode rock)
! And now the @glk call
@glk 313 _vararg_count ret;
return ret;
];
[ glk_request_char_event_uni _vararg_count ret;
! glk_request_char_event_uni (win)
! And now the @glk call
@glk 320 _vararg_count ret;
return ret;
];
```

```
[ glk_request_line_event_uni _vararg_count ret;
! glk_request_line_event_uni (win buf maxlen initlen)
! And now the @glk call
@glk 321 _vararg_count ret;
return ret;
];
[ glk_image_draw _vararg_count ret;
! glk_image_draw (win image val1 val2)
! And now the @glk call
@glk 225 _vararg_count ret;
return ret;
];
[ glk_image_draw_scaled _vararg_count ret;
! glk_image_draw_scaled (win image val1 val2 width height)
! And now the @glk call
@glk 226 _vararg_count ret;
return ret;
];
[ glk_image_get_info _vararg_count ret;
! glk_image_get_info (image width height)
! And now the @glk call
@glk 224 _vararg_count ret;
return ret;
];
[ glk_window_flow_break _vararg_count ret;
! glk_window_flow_break (win)
! And now the @glk call
@glk 232 _vararg_count ret;
return ret;
];
[ glk_window_erase_rect _vararg_count ret;
! glk_window_erase_rect (win left top width height)
! And now the @glk call
@glk 233 _vararg_count ret;
return ret;
];
[ glk_window_fill_rect _vararg_count ret;
! glk_window_fill_rect (win color left top width height)
! And now the @glk call
@glk 234 _vararg_count ret;
return ret;
];
[ glk_window_set_background_color _vararg_count ret;
! glk_window_set_background_color (win color)
! And now the @glk call
@glk 235 _vararg_count ret;
return ret;
];
[ glk_schannel_create _vararg_count ret;
! glk_schannel_create (rock)
! And now the @glk call
@glk 242 _vararg_count ret;
return ret;
];
[ glk_schannel_destroy _vararg_count ret;
! glk_schannel_destroy (chan)
! And now the @glk call
@glk 243 _vararg_count ret;
return ret;
];
[ glk_schannel_iterate _vararg_count ret;
! glk_schannel_iterate (chan rockptr)
! And now the @glk call
@glk 240 _vararg_count ret;
return ret;
];
[ glk_schannel_get_rock _vararg_count ret;
! glk_schannel_get_rock (chan)
! And now the @glk call
@glk 241 _vararg_count ret;
return ret;
];
[ glk_schannel_play _vararg_count ret;
! glk_schannel_play (chan snd)
! And now the @glk call
@glk 248 _vararg_count ret;
return ret;
];
[ glk_schannel_play_ext _vararg_count ret;
! glk_schannel_play_ext (chan snd repeats notify)
! And now the @glk call
@glk 249 _vararg_count ret;
return ret;
];
[ glk_schannel_stop _vararg_count ret;
! glk_schannel_stop (chan)
! And now the @glk call
@glk 250 _vararg_count ret;
return ret;
];
[ glk_schannel_set_volume _vararg_count ret;
! glk_schannel_set_volume (chan vol)
! And now the @glk call
@glk 251 _vararg_count ret;
return ret;
];
[ glk_sound_load_hint _vararg_count ret;
! glk_sound_load_hint (snd flag)
! And now the @glk call
@glk 252 _vararg_count ret;
return ret;
];
```

```
[ glk_set_hyperlink _vararg_count ret;
! glk_set_hyperlink (linkval)
! And now the @glk call
@glk 256 _vararg_count ret;
return ret;
];
[ glk_set_hyperlink_stream _vararg_count ret;
! glk_set_hyperlink_stream (str linkval)
! And now the @glk call
@glk 257 _vararg_count ret;
return ret;
];
[ glk_request_hyperlink_event _vararg_count ret;
! glk_request_hyperlink_event (win)
! And now the @glk call
@glk 258 _vararg_count ret;
return ret;
];
[ glk_cancel_hyperlink_event _vararg_count ret;
! glk_cancel_hyperlink_event (win)
! And now the @glk call
@glk 259 _vararg_count ret;
return ret;
];
#ENDIF;
```

§**4. Rocks.** These are unique ID codes used to mark resources; think of them as inedible cookies.

```
Constant GG_MAINWIN_ROCK        201;
Constant GG_STATUSWIN_ROCK      202;
Constant GG_QUOTEWIN_ROCK       203;
Constant GG_SAVESTR_ROCK        301;
Constant GG_SCRIPTSTR_ROCK      302;
Constant GG_COMMANDWSTR_ROCK    303;
Constant GG_COMMANDRSTR_ROCK    304;
Constant GG_SCRIPTFREF_ROCK     401;
Constant GG_FOREGROUNDCHAN_ROCK 410;
Constant GG_BACKGROUNDCHAN_ROCK 411;
```

§**5. Stubs.** These are I6 library-style entry point routines, not used by I7, but retained in case I7 extensions want to do interesting things with Glulx.

```
#Stub HandleGlkEvent    2;
#Stub IdentifyGlkObject 4;
#Stub InitGlkWindow     1;
```

§**6. Starting Up.** `VM_Initialise()` is almost the first routine called, except that the "starting the virtual machine" activity is allowed to go first; and, come to think of it, memory allocation has to be set up before even that, and that in turn calls `VM_PreInitialise()` to do the absolute minimum.

Arrangements are a little different here from on the Z-machine, because some data is retained in the case of a restart.

(Many thanks are due to Eliuk Blau, who found several tricky timing errors here and elsewhere in the Glulx-specific code. Frankly, I feel like hanging a sign on the following routines which reads "Congratulations on bringing light to the Dark Room.")

```
[ VM_PreInitialise res;
    @gestalt 4 2 res; ! Test if this interpreter has Glk...
    if (res == 0) quit; ! ...without which there would be nothing we could do
    unicode_gestalt_ok = false;
    if (glk_gestalt(gestalt_Unicode, 0))
        unicode_gestalt_ok = true;
    ! Set the VM's I/O system to be Glk.
    @setiosys 2 0;
];
[ VM_Initialise res sty i;
    @gestalt 4 2 res; ! Test if this interpreter has Glk...
    if (res == 0) quit; ! ...without which there would be nothing we could do
    ! First, we must go through all the Glk objects that exist, and see
    ! if we created any of them. One might think this strange, since the
    ! program has just started running, but remember that the player might
    ! have just typed "restart".
    GGRecoverObjects();
    ! Sound channel initialisation, and RNG fixing, must be done now rather
    ! than later in case InitGlkWindow() returns a non-zero value.
    if (glk_gestalt(gestalt_Sound, 0)) {
        if (gg_foregroundchan == 0)
            gg_foregroundchan = glk_schannel_create(GG_FOREGROUNDCHAN_ROCK);
        if (gg_backgroundchan == 0)
            gg_backgroundchan = glk_schannel_create(GG_BACKGROUNDCHAN_ROCK);
    }
    #ifdef FIX_RNG;
    @random 10000 i;
    i = -i-2000;
    print "[Random number generator seed is ", i, "]^";
    @setrandom i;
    #endif; ! FIX_RNG
    res = InitGlkWindow(0);
    if (res ~= 0) return;
    ! Now, gg_mainwin and gg_storywin might already be set. If not, set them.
    if (gg_mainwin == 0) {
        ! Open the story window.
        res = InitGlkWindow(GG_MAINWIN_ROCK);
        if (res == 0) {
            ! Left-justify the header style
            glk_stylehint_set(wintype_TextBuffer, style_Header, stylehint_Justification, 0);
            ! Try to make emphasized type in italics and not boldface
            glk_stylehint_set(wintype_TextBuffer, style_Emphasized, stylehint_Weight, 0);
            glk_stylehint_set(wintype_TextBuffer, style_Emphasized, stylehint_Oblique, 1);
            gg_mainwin = glk_window_open(0, 0, 0, wintype_TextBuffer, GG_MAINWIN_ROCK);
        }
        if (gg_mainwin == 0) quit; ! If we can't even open one window, give in
    } else {
        ! There was already a story window. We should erase it.
        glk_window_clear(gg_mainwin);
    }
    if (gg_statuswin == 0) {
        res = InitGlkWindow(GG_STATUSWIN_ROCK);
        if (res == 0) {
            statuswin_cursize = statuswin_size;
            for (sty=0: sty<style_NUMSTYLES: sty++)
                glk_stylehint_set(wintype_TextGrid, sty, stylehint_ReverseColor, 1);
            gg_statuswin =
                glk_window_open(gg_mainwin, winmethod_Fixed + winmethod_Above,
                    statuswin_cursize, wintype_TextGrid, GG_STATUSWIN_ROCK);
        }
    }
    ! It's possible that the status window couldn't be opened, in which case
    ! gg_statuswin is now zero. We must allow for that later on.
    glk_set_window(gg_mainwin);
    InitGlkWindow(1);
];
[ GGRecoverObjects id;
    ! If GGRecoverObjects() has been called, all these stored IDs are
    ! invalid, so we start by clearing them all out.
    ! (In fact, after a restoreundo, some of them may still be good.
    ! For simplicity, though, we assume the general case.)
    gg_mainwin = 0;
    gg_statuswin = 0;
    gg_quotewin = 0;
    gg_scriptfref = 0;
    gg_scriptstr = 0;
    gg_savestr = 0;
    statuswin_cursize = 0;
    gg_foregroundchan = 0;
    gg_backgroundchan = 0;
    #Ifdef DEBUG;
    gg_commandstr = 0;
    gg_command_reading = false;
    #Endif; ! DEBUG
    ! Also tell the game to clear its object references.
    IdentifyGlkObject(0);
    id = glk_stream_iterate(0, gg_arguments);
    while (id) {
        switch (gg_arguments-->0) {
            GG_SAVESTR_ROCK: gg_savestr = id;
            GG_SCRIPTSTR_ROCK: gg_scriptstr = id;
            #Ifdef DEBUG;
            GG_COMMANDWSTR_ROCK: gg_commandstr = id;
```

```
                                gg_command_reading = false;
            GG_COMMANDRSTR_ROCK: gg_commandstr = id;
                                gg_command_reading = true;
            #Endif; ! DEBUG
            default: IdentifyGlkObject(1, 1, id, gg_arguments-->0);
        }
        id = glk_stream_iterate(id, gg_arguments);
    }
    id = glk_window_iterate(0, gg_arguments);
    while (id) {
        switch (gg_arguments-->0) {
            GG_MAINWIN_ROCK: gg_mainwin = id;
            GG_STATUSWIN_ROCK: gg_statuswin = id;
            GG_QUOTEWIN_ROCK: gg_quotewin = id;
            default: IdentifyGlkObject(1, 0, id, gg_arguments-->0);
        }
        id = glk_window_iterate(id, gg_arguments);
    }
    id = glk_fileref_iterate(0, gg_arguments);
    while (id) {
        switch (gg_arguments-->0) {
            GG_SCRIPTFREF_ROCK: gg_scriptfref = id;
            default: IdentifyGlkObject(1, 2, id, gg_arguments-->0);
        }
        id = glk_fileref_iterate(id, gg_arguments);
    }
    if (glk_gestalt(gestalt_Sound, 0)) {
        id = glk_schannel_iterate(0, gg_arguments);
        while (id) {
            switch (gg_arguments-->0) {
                GG_FOREGROUNDCHAN_ROCK: gg_foregroundchan = id;
                GG_BACKGROUNDCHAN_ROCK: gg_backgroundchan = id;
            }
            id = glk_schannel_iterate(id, gg_arguments);
        }
        if (gg_foregroundchan ~= 0) { glk_schannel_stop(gg_foregroundchan); }
        if (gg_backgroundchan ~= 0) { glk_schannel_stop(gg_backgroundchan); }
    }
    ! Tell the game to tie up any loose ends.
    IdentifyGlkObject(2);
];
```

§**7. Enable Acceleration.** This enables use of March 2009 extension to Glulx which optimises the speed of Inform-compiled story files by moving the work of I6 veneer routines into the interpreter itself. It should have no effect on earlier versions of the Glulx VM, which will lack the gestalt for this feature, but nor should it do any harm.

```
[ ENABLE_GLULX_ACCEL_R addr res;
    @gestalt 9 0 res;
    if (res == 0) return;
    addr = #classes_table;
    @accelparam 0 addr;
    @accelparam 1 INDIV_PROP_START;
    @accelparam 2 Class;
    @accelparam 3 Object;
    @accelparam 4 Routine;
    @accelparam 5 String;
    addr = #globals_array + WORDSIZE * #g$self;
    @accelparam 6 addr;
    @accelparam 7 NUM_ATTR_BYTES;
    addr = #cpv__start;
    @accelparam 8 addr;
    @accelfunc 1 Z__Region;
    @accelfunc 2 CP__Tab;
    @accelfunc 3 RA__Pr;
    @accelfunc 4 RL__Pr;
    @accelfunc 5 OC__Cl;
    @accelfunc 6 RV__Pr;
    @accelfunc 7 OP__Pr;
    rfalse;
];
```

§**8. Release Number.** Like all software, IF story files have release numbers to mark revised versions being circulated: unlike most software, and partly for traditional reasons, the version number is recorded not in some print statement or variable but is branded on, so to speak, in a specific memory location of the story file header.

`VM_Describe_Release()` describes the release and is used as part of the "banner", IF's equivalent to a title page.

```
[ VM_Describe_Release i;
    print "Release ";
    @aloads ROM_GAMERELEASE 0 i;
    print i;
    print " / Serial number ";
    for (i=0 : i<6 : i++) print (char) ROM_GAMESERIAL->i;
];
```

§9. Keyboard Input. The VM must provide three routines for keyboard input:

(a) `VM_KeyChar()` waits for a key to be pressed and then returns the character chosen as a ZSCII character.
(b) `VM_KeyDelay(N)` waits up to $N/10$ seconds for a key to be pressed, returning the ZSCII character if so, or 0 if not.
(c) `VM_ReadKeyboard(b, t)` reads a whole newline-terminated command into the buffer `b`, then parses it into a word stream in the table `t`.

There are elaborations to due with mouse clicks, but this isn't the place to document all of that.

```
[ VM_KeyChar win nostat done res ix jx ch;
    jx = ch; ! squash compiler warnings
    if (win == 0) win = gg_mainwin;
    if (gg_commandstr ~= 0 && gg_command_reading ~= false) {
        done = glk_get_line_stream(gg_commandstr, gg_arguments, 31);
        if (done == 0) {
            glk_stream_close(gg_commandstr, 0);
            gg_commandstr = 0;
            gg_command_reading = false;
            ! fall through to normal user input.
        } else {
            ! Trim the trailing newline
            if (gg_arguments->(done-1) == 10) done = done-1;
            res = gg_arguments->0;
            if (res == '\') {
                res = 0;
                for (ix=1 : ix<done : ix++) {
                    ch = gg_arguments->ix;
                    if (ch >= '0' && ch <= '9') {
                        @shiftl res 4 res;
                        res = res + (ch-'0');
                    } else if (ch >= 'a' && ch <= 'f') {
                        @shiftl res 4 res;
                        res = res + (ch+10-'a');
                    } else if (ch >= 'A' && ch <= 'F') {
                        @shiftl res 4 res;
                        res = res + (ch+10-'A');
                    }
                }
            }
            jump KCPContinue;
        }
    }
    done = false;
    glk_request_char_event(win);
    while (~~done) {
        glk_select(gg_event);
        switch (gg_event-->0) {
        5: ! evtype_Arrange
            if (nostat) {
                glk_cancel_char_event(win);
                res = $80000000;
                done = true;
                break;
            }
```

```
                DrawStatusLine();
            2: ! evtype_CharInput
                if (gg_event-->1 == win) {
                    res = gg_event-->2;
                    done = true;
                    }
            }
            ix = HandleGlkEvent(gg_event, 1, gg_arguments);
            if (ix == 2) {
                res = gg_arguments-->0;
                done = true;
            } else if (ix == -1)  done = false;
        }
        if (gg_commandstr ~= 0 && gg_command_reading == false) {
            if (res < 32 || res >= 256 || (res == '\' or ' ')) {
                glk_put_char_stream(gg_commandstr, '\');
                done = 0;
                jx = res;
                for (ix=0 : ix<8 : ix++) {
                    @ushiftr jx 28 ch;
                    @shiftl jx 4 jx;
                    ch = ch & $0F;
                    if (ch ~= 0 || ix == 7) done = 1;
                    if (done) {
                        if (ch >= 0 && ch <= 9) ch = ch + '0';
                        else                    ch = (ch - 10) + 'A';
                        glk_put_char_stream(gg_commandstr, ch);
                    }
                }
            } else {
                glk_put_char_stream(gg_commandstr, res);
            }
            glk_put_char_stream(gg_commandstr, 10); ! newline
        }
    .KCPContinue;
        return res;
    ];
    [ VM_KeyDelay tenths  key done ix;
        glk_request_char_event(gg_mainwin);
        glk_request_timer_events(tenths*100);
        while (~~done) {
            glk_select(gg_event);
            ix = HandleGlkEvent(gg_event, 1, gg_arguments);
            if (ix == 2) {
                key = gg_arguments-->0;
                done = true;
            }
            else if (ix >= 0 && gg_event-->0 == 1 or 2) {
                key = gg_event-->2;
                done = true;
            }
        }
        glk_cancel_char_event(gg_mainwin);
        glk_request_timer_events(0);
        return key;
    ];
    [ VM_ReadKeyboard  a_buffer a_table done ix;
        if (gg_commandstr ~= 0 && gg_command_reading ~= false) {
            done = glk_get_line_stream(gg_commandstr, a_buffer+WORDSIZE,
                (INPUT_BUFFER_LEN-WORDSIZE)-1);
            if (done == 0) {
                glk_stream_close(gg_commandstr, 0);
                gg_commandstr = 0;
                gg_command_reading = false;
                ! L__M(##CommandsRead, 5); would come after prompt
                ! fall through to normal user input.
            }
            else {
                ! Trim the trailing newline
                if ((a_buffer+WORDSIZE)->(done-1) == 10) done = done-1;
                a_buffer-->0 = done;
                VM_Style(INPUT_VMSTY);
                glk_put_buffer(a_buffer+WORDSIZE, done);
                VM_Style(NORMAL_VMSTY);
                print "^";
                jump KPContinue;
            }
        }
        done = false;
        glk_request_line_event(gg_mainwin, a_buffer+WORDSIZE, INPUT_BUFFER_LEN-WORDSIZE, 0);
        while (~~done) {
            glk_select(gg_event);
            switch (gg_event-->0) {
            5: ! evtype_Arrange
                DrawStatusLine();
            3: ! evtype_LineInput
                if (gg_event-->1 == gg_mainwin) {
                    a_buffer-->0 = gg_event-->2;
                    done = true;
                }
            }
            ix = HandleGlkEvent(gg_event, 0, a_buffer);
            if (ix == 2) done = true;
            else if (ix == -1) done = false;
        }
        if (gg_commandstr ~= 0 && gg_command_reading == false) {
            glk_put_buffer_stream(gg_commandstr, a_buffer+WORDSIZE, a_buffer-->0);
            glk_put_char_stream(gg_commandstr, 10); ! newline
        }
    .KPContinue;
        VM_Tokenise(a_buffer,a_table);
        ! It's time to close any quote window we've got going.
        if (gg_quotewin) {
            glk_window_close(gg_quotewin, 0);
            gg_quotewin = 0;
        }
        #ifdef ECHO_COMMANDS;
        print "** ";
        for (ix=WORDSIZE: ix<(a_buffer-->0)+WORDSIZE: ix++) print (char) a_buffer->ix;
        print "^";
        #endif; ! ECHO_COMMANDS
    ];
```

§**10. Buffer Functions.** A "buffer", in this sense, is an array containing a stream of characters typed from the keyboard; a "parse buffer" is an array which resolves this into individual words, pointing to the relevant entries in the dictionary structure. Because each VM has its own format for each of these arrays (not to mention the dictionary), we have to provide some standard operations needed by the rest of the template as routines for each VM.

`VM_CopyBuffer(to, from)` copies one buffer into another.

`VM_Tokenise(buff, parse_buff)` takes the text in the buffer `buff` and produces the corresponding data in the parse buffer `parse_buff` – this is called tokenisation since the characters are divided into words: in traditional computing jargon, such clumps of characters treated syntactically as units are called tokens.

`LTI_Insert` is documented in the DM4 and the `LTI` prefix stands for "Language To Informese": it's used only by translations into non-English languages of play, and is not called in the template.

```
[ VM_CopyBuffer bto bfrom i;
    for (i=0: i<INPUT_BUFFER_LEN: i++) bto->i = bfrom->i;
];
[ VM_PrintToBuffer buf len a b c;
    if (b) {
        if (metaclass(a) == Object && a.#b == WORDSIZE
            && metaclass(a.b) == String)
            buf-->0 = Glulx_PrintAnyToArray(buf+WORDSIZE, len, a.b);
        else if (metaclass(a) == Routine)
            buf-->0 = Glulx_PrintAnyToArray(buf+WORDSIZE, len, a, b, c);
        else
            buf-->0 = Glulx_PrintAnyToArray(buf+WORDSIZE, len, a, b);
    }
    else if (metaclass(a) == Routine)
        buf-->0 = Glulx_PrintAnyToArray(buf+WORDSIZE, len, a, b, c);
    else
        buf-->0 = Glulx_PrintAnyToArray(buf+WORDSIZE, len, a);
    if (buf-->0 > len) buf-->0 = len;
    return buf-->0;
];
[ VM_Tokenise buf tab
    cx numwords len bx ix wx wpos wlen val res dictlen entrylen;
    len = buf-->0;
    buf = buf+WORDSIZE;

    ! First, split the buffer up into words. We use the standard Infocom
    ! list of word separators (comma, period, double-quote).

    cx = 0;
    numwords = 0;
    while (cx < len) {
        while (cx < len && buf->cx == ' ') cx++;
        if (cx >= len) break;
        bx = cx;
        if (buf->cx == '.' or ',' or '"') cx++;
        else {
            while (cx < len && buf->cx ~= ' ' or '.' or ',' or '"') cx++;
        }
        tab-->(numwords*3+2) = (cx-bx);
        tab-->(numwords*3+3) = WORDSIZE+bx;
        numwords++;
        if (numwords >= MAX_BUFFER_WORDS) break;
    }
    tab-->0 = numwords;

    ! Now we look each word up in the dictionary.

    dictlen = #dictionary_table-->0;
    entrylen = DICT_WORD_SIZE + 7;

    for (wx=0 : wx<numwords : wx++) {
        wlen = tab-->(wx*3+2);
        wpos = tab-->(wx*3+3);

        ! Copy the word into the gg_tokenbuf array, clipping to DICT_WORD_SIZE
        ! characters and lower case.
        if (wlen > DICT_WORD_SIZE) wlen = DICT_WORD_SIZE;
        cx = wpos - WORDSIZE;
        for (ix=0 : ix<wlen : ix++) gg_tokenbuf->ix = VM_UpperToLowerCase(buf->(cx+ix));
        for (: ix<DICT_WORD_SIZE : ix++) gg_tokenbuf->ix = 0;

        val = #dictionary_table + WORDSIZE;
        @binarysearch gg_tokenbuf DICT_WORD_SIZE val entrylen dictlen 1 1 res;
        tab-->(wx*3+1) = res;
    }
];
[ LTI_Insert i ch  b y;

    ! Protect us from strict mode, as this isn't an array in quite the
    ! sense it expects
    b = buffer;

    ! Insert character ch into buffer at point i.
    ! Being careful not to let the buffer possibly overflow:
    y = b-->0;
    if (y > INPUT_BUFFER_LEN) y = INPUT_BUFFER_LEN;

    ! Move the subsequent text along one character:
    for (y=y+WORDSIZE : y>i : y--) b->y = b->(y-1);
    b->i = ch;

    ! And the text is now one character longer:
    if (b-->0 < INPUT_BUFFER_LEN) (b-->0)++;
];
```

§11. Dictionary Functions. Again, the dictionary structure is differently arranged on the different VMs. This is a data structure containing, in compressed form, the text of all the words to be recognised by tokenisation (above). In I6 for Glulx, a dictionary word is represented at run-time by its record's address in the dictionary.

`VM_InvalidDictionaryAddress(A)` tests whether `A` is a valid record address in the dictionary data structure. In Glulx, dictionary records might in theory be anywhere in the 2 GB or so of possible memory, but we can rule out negative addresses. (This allows −1, say, to be used as a value meaning "not a valid dictionary word".)

`VM_DictionaryAddressToNumber(A)` and `VM_NumberToDictionaryAddress(N)` convert between word addresses and their run-time representations: since, on Glulx, they are the same, these are each the identity function.

```
[ VM_InvalidDictionaryAddress addr;
    if (addr < 0) rtrue;
    rfalse;
];
[ VM_DictionaryAddressToNumber w; return w; ];
[ VM_NumberToDictionaryAddress n; return n; ];
Array gg_tokenbuf -> DICT_WORD_SIZE;
[ GGWordCompare str1 str2 ix jx;
    for (ix=0 : ix<DICT_WORD_SIZE : ix++) {
        jx = (str1->ix) - (str2->ix);
        if (jx ~= 0) return jx;
    }
    return 0;
];
```

§12. SHOWVERB support. Further VM-specific tables cover actions and attributes, and these are used by the SHOWVERB testing command.

```
#Ifdef DEBUG;
[ DebugAction a str;
    if (a >= 4096) { print "<fake action ", a-4096, ">"; return; }
    if (a < 0 || a >= #identifiers_table-->7) print "<invalid action ", a, ">";
    else {
        str = #identifiers_table-->6;
        str = str-->a;
        if (str) print (string) str; else print "<unnamed action ", a, ">";
    }
];
[ DebugAttribute a str;
    if (a < 0 || a >= NUM_ATTR_BYTES*8) print "<invalid attribute ", a, ">";
    else {
        str = #identifiers_table-->4;
        str = str-->a;
        if (str) print (string) str; else print "<unnamed attribute ", a, ">";
    }
];
#Endif;
```

§13. Command Tables. The VM is also generated containing a data structure for the grammar produced by I6's `Verb` and `Extend` directives: this is essentially a list of command verbs such as DROP or PUSH, together with a list of synonyms, and then the grammar for the subsequent commands to be recognised by the parser.

```
[ VM_CommandTableAddress i;
    return (#grammar_table)-->(i+1);
];
[ VM_PrintCommandWords i wd j dictlen entrylen;
    dictlen = #dictionary_table-->0;
    entrylen = DICT_WORD_SIZE + 7;
    for (j=0 : j<dictlen : j++) {
        wd = #dictionary_table + WORDSIZE + entrylen*j;
        if (DictionaryWordToVerbNum(wd) == i)
            print "'", (address) wd, "' ";
    }
];
```

§14. Random Number Generator. No routine is needed for extracting a random number, since I6's built-in `random` function does that, but it's useful to abstract the process of seeding the RNG so that it produces a repeatable sequence of "random" numbers from here on: the necessary opcodes are different for the two VMs.

```
[ VM_Seed_RNG n;
    @setrandom n;
];
```

§15. Memory Allocation. This is dynamic memory allocation: something which is never practicable in the Z-machine, because the whole address range is already claimed, but which is viable on recent revisions of Glulx.

```
[ VM_AllocateMemory amount i;
    @gestalt 7 0 i;
    if (i == 0) return i;
    @malloc amount i;
    return i;
];
[ VM_FreeMemory address i;
    @gestalt 7 0 i;
    if (i == 0) return;
    @mfree address;
];
```

§16. Audiovisual Resources. The Z-machine only barely supports figures and sound effects, so Glulx is the preferred VM to choose if they are wanted. Properly speaking, it's not Glulx which supports these, but its I/O layer Glk, and implementations of Glk are free to support them or not as they please: "cheapglk", a dumb terminal version, does not, for instance. We therefore have to investigate the "gestalt" to find out.

```
[ VM_Picture resource_ID;
    if (glk_gestalt(gestalt_Graphics, 0)) {
        glk_image_draw(gg_mainwin, resource_ID, imagealign_InlineCenter, 0);
    } else {
        print "[Picture number ", resource_ID, " here.]^";
    }
];
[ VM_SoundEffect resource_ID;
    if (glk_gestalt(gestalt_Sound, 0)) {
        glk_schannel_play(gg_foregroundchan, resource_ID);
    } else {
        print "[Sound effect number ", resource_ID, " here.]^";
    }
];
```

§17. Typography. Glk makes an attempt to present typographic styles as being a matter of semantic markup rather than controlling the actual appearance of text: the idea is that the story file should want to print something in a heading kind of way, and then the interpreter – guided by the player's reading preferences – might set that in bold, or larger type, or red ink, or any combination of the three, or with other effects entirely. This is not the place to discuss whether that was a wise decision for Glk to take (it really, really, *really* wasn't): we can only play along.

```
[ VM_Style sty;
    switch (sty) {
        NORMAL_VMSTY:     glk_set_style(style_Normal);
        HEADER_VMSTY:     glk_set_style(style_Header);
        SUBHEADER_VMSTY:  glk_set_style(style_Subheader);
        NOTE_VMSTY:       glk_set_style(style_Note);
        ALERT_VMSTY:      glk_set_style(style_Alert);
        BLOCKQUOTE_VMSTY: glk_set_style(style_BlockQuote);
        INPUT_VMSTY:      glk_set_style(style_Input);
    }
];
```

§18. Character Casing. The following are the equivalent of `tolower` and `toupper`, the traditional C library functions for forcing letters into lower and upper case form, for the ZSCII character set. Note that Glulx can also use Unicode characters for some purposes (Unicode was a relatively late addition to the Glulx standard), and we make good use of this when storing indexed text.

```
[ VM_UpperToLowerCase c; return glk_char_to_lower(c); ];
[ VM_LowerToUpperCase c; return glk_char_to_upper(c); ];
```

§19. Glulx-Only Printing Routines. Partly because of the smallness of the range of representable values in the Z-machine, there is little run-time type-checking that can be done: for instance a dictionary address cannot be distinguished from a function address because they are encoded differently, so that a function address (which is packed) could well coincide with that of a dictionary word (which is not). On Glulx these restrictions are somewhat lifted, so that it's possible to write a routine which can look at a value, work out what it must mean, and print it suitably. This is only possible up to a point – for instance, it can't distinguish an integer from a function address – and in I7 the use of this sort of trick is much less important because type-checking in the NI compiler handles the problem much better. Still, we retain some Glulx-only features because they are convenient for writing external files to disc, for instance, something which the Z-machine can't do in any case.

`Glulx_PrintAnything` handles strings, functions (with optional arguments), objects, object properties (with optional arguments), and dictionary words.

`Glulx_PrintAnyToArray` does the same, but the output is sent to a byte array in memory. The first two arguments must be the array address and length; subsequent arguments are as for `Glulx_PrintAnything`. The return value is the number of characters output. If the output is longer than the array length given, the extra characters are discarded, so the array does not overflow. (However, the return value is the total length of the output, including discarded characters.) The character set stored here is ZSCII, not Unicode.

`Glulx_ChangeAnyToCString` calls `Glulx_PrintAnyToArray` on a particular array, then amends the result to make it a C-style string – that is, a sequence of byte-sized characters which are null terminated. The character set stored here is once again ZSCII, not Unicode.

```
! Glulx_PrintAnything()                    <nothing printed>
! Glulx_PrintAnything(0)                   <nothing printed>
! Glulx_PrintAnything("string");           print (string) "string";
! Glulx_PrintAnything('word')              print (address) 'word';
! Glulx_PrintAnything(obj)                 print (name) obj;
! Glulx_PrintAnything(obj, prop)           obj.prop();
! Glulx_PrintAnything(obj, prop, args...)  obj.prop(args...);
! Glulx_PrintAnything(func)                func();
! Glulx_PrintAnything(func, args...)       func(args...);

[ Glulx_PrintAnything _vararg_count obj mclass;
    if (_vararg_count == 0) return;
    @copy sp obj;
    _vararg_count--;
    if (obj == 0) return;

    if (obj->0 == $60) {
        ! Dictionary word. Metaclass() can't catch this case, so we do it manually
        print (address) obj;
        return;
    }

    mclass = metaclass(obj);
    switch (mclass) {
    nothing:
        return;
    String:
        print (string) obj;
        return;
    Routine:
        ! Call the function with all the arguments which are already
        ! on the stack.
        @call obj _vararg_count 0;
        return;
    Object:
        if (_vararg_count == 0) {
            print (name) obj;
        }
        else {
            ! Push the object back onto the stack, and call the
            ! veneer routine that handles obj.prop() calls.
            @copy obj sp;
            _vararg_count++;
            @call CA__Pr _vararg_count 0;
        }
        return;
    }
];
[ Glulx_PrintAnyToArray _vararg_count arr arrlen str oldstr len;
    @copy sp arr;
    @copy sp arrlen;
    _vararg_count = _vararg_count - 2;
    oldstr = glk_stream_get_current();
    str = glk_stream_open_memory(arr, arrlen, 1, 0);
    if (str == 0) return 0;
    glk_stream_set_current(str);
    @call Glulx_PrintAnything _vararg_count 0;
    glk_stream_set_current(oldstr);
    @copy $ffffffff sp;
    @copy str sp;
    @glk $0044 2 0; ! stream_close
    @copy sp len;
    @copy sp 0;
    return len;
];
Constant GG_ANYTOSTRING_LEN 66;
Array AnyToStrArr -> GG_ANYTOSTRING_LEN+1;
[ Glulx_ChangeAnyToCString _vararg_count ix len;
    ix = GG_ANYTOSTRING_LEN-2;
    @copy ix sp;
    ix = AnyToStrArr+1;
    @copy ix sp;
    ix = _vararg_count+2;
    @call Glulx_PrintAnyToArray ix len;
    AnyToStrArr->0 = $E0;
    if (len >= GG_ANYTOSTRING_LEN)
        len = GG_ANYTOSTRING_LEN-1;
    AnyToStrArr->(len+1) = 0;
    return AnyToStrArr;
];
```

§20. The Screen. Our generic screen model is that the screen is made up of windows: we tend to refer only to two of these, the main window and the status line, but others may also exist from time to time. Windows have unique ID numbers: the special window ID −1 means "all windows" or "the entire screen", which usually amounts to the same thing.

Screen height and width are measured in characters, with respect to the fixed-pitch font used for the status line. The main window normally contains variable-pitch text which may even have been kerned, and character dimensions make little sense there.

```
[ VM_ClearScreen window;
    if (window == WIN_ALL or WIN_MAIN) {
        glk_window_clear(gg_mainwin);
        if (gg_quotewin) {
            glk_window_close(gg_quotewin, 0);
            gg_quotewin = 0;
        }
    }
    if (gg_statuswin && window == WIN_ALL or WIN_STATUS) glk_window_clear(gg_statuswin);
];
[ VM_ScreenWidth  id;
    id=gg_mainwin;
    if (gg_statuswin && statuswin_current) id = gg_statuswin;
    glk_window_get_size(id, gg_arguments, 0);
    return gg_arguments-->0;
];
[ VM_ScreenHeight;
    glk_window_get_size(gg_mainwin, 0, gg_arguments);
    return gg_arguments-->0;
];
```

§21. Window Colours. Our generic screen model is that the screen is made up of windows, each of which can have its own foreground and background colours.

The colour of individual letters or words of type is not controllable in Glulx, to the frustration of many, and so the template layer of I7 has no framework for handling this (even though it is controllable on the Z-machine, which is greatly superior in this respect).

```
[ VM_SetWindowColours f b window doclear  i fwd bwd swin;
    if (clr_on && f && b) {
        if (window) swin = 5-window; ! 4 for TextGrid, 3 for TextBuffer

        fwd = MakeColourWord(f);
        bwd = MakeColourWord(b);
        for (i=0 : i<style_NUMSTYLES: i++) {
            if (f == CLR_DEFAULT || b == CLR_DEFAULT) {  ! remove style hints
                glk_stylehint_clear(swin, i, stylehint_TextColor);
                glk_stylehint_clear(swin, i, stylehint_BackColor);
            } else {
                glk_stylehint_set(swin, i, stylehint_TextColor, fwd);
                glk_stylehint_set(swin, i, stylehint_BackColor, bwd);
            }
        }

        ! Now re-open the windows to apply the hints
        if (gg_statuswin) glk_window_close(gg_statuswin, 0);
        gg_statuswin = 0;
        if (doclear || ( window ~= 1 && (clr_fg ~= f || clr_bg ~= b) ) ) {
            glk_window_close(gg_mainwin, 0);
            gg_mainwin = glk_window_open(0, 0, 0, wintype_TextBuffer, GG_MAINWIN_ROCK);
            if (gg_scriptstr ~= 0)
                glk_window_set_echo_stream(gg_mainwin, gg_scriptstr);
        }
        gg_statuswin =
            glk_window_open(gg_mainwin, winmethod_Fixed + winmethod_Above,
                statuswin_cursize, wintype_TextGrid, GG_STATUSWIN_ROCK);
        if (statuswin_current && gg_statuswin) VM_MoveCursorInStatusLine(); else VM_MainWindow();
        if (window ~= 2) {
            clr_fgstatus = f;
            clr_bgstatus = b;
        }
        if (window ~= 1) {
            clr_fg = f;
            clr_bg = b;
        }
    }
];
[ VM_RestoreWindowColours; ! used after UNDO: compare I6 patch L61007
    if (clr_on) { ! check colour has been used
        VM_SetWindowColours(clr_fg, clr_bg, 2); ! make sure both sets of variables are restored
        VM_SetWindowColours(clr_fgstatus, clr_bgstatus, 1, true);
        VM_ClearScreen();
    }
];
[ MakeColourWord c;
    if (c > 9) return c;
    c = c-2;
    return $ff0000*(c&1) + $ff00*(c&2 ~= 0) + $ff*(c&4 ~= 0);
];
```

§**22. Main Window.** The part of the screen on which commands and responses are printed, which ordinarily occupies almost all of the screen area.

`VM_MainWindow()` switches printing back from another window, usually the status line, to the main window.

```
[ VM_MainWindow;
    glk_set_window(gg_mainwin); ! set_window
    statuswin_current=0;
];
```

§23. Status Line. Despite the name, the status line need not be a single line at the top of the screen: that's only the conventional default arrangement. It can expand to become the equivalent of an old-fashioned VT220 terminal, with menus and grids and mazes displayed lovingly in character graphics, or it can close up to invisibility.

`VM_StatusLineHeight(n)` sets the status line to have a height of `n` lines of type. (The width of the status line is always the width of the whole screen, and the position is always at the top, so the height is the only controllable aspect.) The $n = 0$ case makes the status line disappear.

`VM_MoveCursorInStatusLine(x, y)` switches printing to the status line, positioning the "cursor" – the position at which printing will begin – at the given character grid position (x, y). Line 1 represents the top line; line 2 is underneath, and so on; columns are similarly numbered from 1 at the left.

```
[ VM_StatusLineHeight hgt;
    if (gg_statuswin == 0) return;
    if (hgt == statuswin_cursize) return;
    glk_window_set_arrangement(glk_window_get_parent(gg_statuswin), $12, hgt, 0);
    statuswin_cursize = hgt;
];

[ VM_MoveCursorInStatusLine line column;
    if (gg_statuswin == 0) return;
    glk_set_window(gg_statuswin);
    if (line == 0) { line = 1; column = 1; }
    glk_window_move_cursor(gg_statuswin, column-1, line-1);
    statuswin_current=1;
];
```

§24. Quotation Boxes. On the Z-machine, quotation boxes are produced by stretching the status line, but on Glulx they usually occupy windows of their own. If it isn't possible to create such a window, so that `gg_quotewin` is zero below, the quotation text just appears in the main window.

```
[ Box__Routine maxwid arr ix lines lastnl parwin;
    maxwid = 0; ! squash compiler warning
    lines = arr-->0;

    if (gg_quotewin == 0) {
        gg_arguments-->0 = lines;
        ix = InitGlkWindow(GG_QUOTEWIN_ROCK);
        if (ix == 0)
            gg_quotewin =
                glk_window_open(gg_mainwin, winmethod_Fixed + winmethod_Above,
                    lines, wintype_TextBuffer, GG_QUOTEWIN_ROCK);
    } else {
        parwin = glk_window_get_parent(gg_quotewin);
        glk_window_set_arrangement(parwin, $12, lines, 0);
    }

    lastnl = true;
    if (gg_quotewin) {
        glk_window_clear(gg_quotewin);
        glk_set_window(gg_quotewin);
        lastnl = false;
    }

    VM_Style(BLOCKQUOTE_VMSTY);
    for (ix=0 : ix<lines : ix++) {
        print (string) arr-->(ix+1);
        if (ix < lines-1 || lastnl) new_line;
    }
    VM_Style(NORMAL_VMSTY);
    if (gg_quotewin) glk_set_window(gg_mainwin);
];
```

§**25. GlkList Command.** GLKLIST is a testing command best used by those who understand Glulx and its ways: it isn't documented in the I7 manual, because it is pretty inscrutable for "real" users, but it's probably worth keeping just the same.

```
#Ifdef DEBUG;
[ GlkListSub id val;
    id = glk_window_iterate(0, gg_arguments);
    while (id) {
        print "Window ", id, " (", gg_arguments-->0, "): ";
        val = glk_window_get_type(id);
        switch (val) {
        1: print "pair";
        2: print "blank";
        3: print "textbuffer";
        4: print "textgrid";
        5: print "graphics";
        default: print "unknown";
        }
        val = glk_window_get_parent(id);
        if (val) print ", parent is window ", val;
        else     print ", no parent (root)";
        val = glk_window_get_stream(id);
        print ", stream ", val;
        val = glk_window_get_echo_stream(id);
        if (val) print ", echo stream ", val;
        print "^";
        id = glk_window_iterate(id, gg_arguments);
    }
    id = glk_stream_iterate(0, gg_arguments);
    while (id) {
        print "Stream ", id, " (", gg_arguments-->0, ")^";
        id = glk_stream_iterate(id, gg_arguments);
    }
    id = glk_fileref_iterate(0, gg_arguments);
    while (id) {
        print "Fileref ", id, " (", gg_arguments-->0, ")^";
        id = glk_fileref_iterate(id, gg_arguments);
    }
    if (glk_gestalt(gestalt_Sound, 0)) {
        id = glk_schannel_iterate(0, gg_arguments);
        while (id) {
            print "Soundchannel ", id, " (", gg_arguments-->0, ")^";
            id = glk_schannel_iterate(id, gg_arguments);
        }
    }
];
{-testing-command:glklist}
    *                                         -> Glklist;
#Endif;
```

§**26. Undo.** These are really emulations of the Z-machine's conventions on UNDO: Glulx's undo opcodes used different result codes while providing essentially the same functionality, for reasons which are opaque, but no trouble is caused thereby.

```
[ VM_Undo result_code;
    @restoreundo result_code;
    return (~~result_code);
];
[ VM_Save_Undo result_code;
    @saveundo result_code;
    if (result_code == -1) { GGRecoverObjects(); return 2; }
    return (~~result_code);
];
```

§**27. Quit The Game Rule.**

```
[ QUIT_THE_GAME_R;
    if (actor ~= player) rfalse;
    GL__M(##Quit, 2); if (YesOrNo()~=0) quit;
];
```

§**28. Restart The Game Rule.**

```
[ RESTART_THE_GAME_R;
    if (actor ~= player) rfalse;
    GL__M(##Restart, 1);
    if (YesOrNo() ~= 0) {
        @restart;
        GL__M(##Restart, 2);
    }
];
```

§**29. Restore The Game Rule.**

```
[ RESTORE_THE_GAME_R res fref;
    if (actor ~= player) rfalse;
    fref = glk_fileref_create_by_prompt($01, $02, 0);
    if (fref == 0) jump RFailed;
    gg_savestr = glk_stream_open_file(fref, $02, GG_SAVESTR_ROCK);
    glk_fileref_destroy(fref);
    if (gg_savestr == 0) jump RFailed;
    @restore gg_savestr res;
    glk_stream_close(gg_savestr, 0);
    gg_savestr = 0;
    .RFailed;
    GL__M(##Restore, 1);
];
```

§30. Save The Game Rule.

```
[ SAVE_THE_GAME_R res fref;
    if (actor ~= player) rfalse;
    fref = glk_fileref_create_by_prompt($01, $01, 0);
    if (fref == 0) jump SFailed;
    gg_savestr = glk_stream_open_file(fref, $01, GG_SAVESTR_ROCK);
    glk_fileref_destroy(fref);
    if (gg_savestr == 0) jump SFailed;
    @save gg_savestr res;
    if (res == -1) {
        ! The player actually just typed "restore". We're going to print
        !  GL__M(##Restore,2); the Z-Code Inform library does this correctly
        ! now. But first, we have to recover all the Glk objects; the values
        ! in our global variables are all wrong.
        GGRecoverObjects();
        glk_stream_close(gg_savestr, 0); ! stream_close
        gg_savestr = 0;
        return GL__M(##Restore, 2);
    }
    glk_stream_close(gg_savestr, 0); ! stream_close
    gg_savestr = 0;
    if (res == 0) return GL__M(##Save, 2);
    .SFailed;
    GL__M(##Save, 1);
];
```

§31. Verify The Story File Rule. This is a fossil now, really, but in the days of Infocom, the 110K story file occupying an entire disc was a huge data set: floppy discs were by no means a reliable medium, and cheap hardware often used hit-and-miss components, as on the notorious Commodore 64 disc controller. If somebody experienced an apparent bug in play, it could easily be that he had a corrupt disc or was unable to read data of that density. So the VERIFY command, which took up to ten minutes on some early computers, would chug through the entire story file and compute a checksum, compare it against a known result in the header, and determine that the story file could or could not properly be read. The Z-machine provided this service as an opcode, and so Glulx followed suit.

```
[ VERIFY_THE_STORY_FILE_R res;
    if (actor ~= player) rfalse;
    @verify res;
    if (res == 0) return GL__M(##Verify, 1);
    GL__M(##Verify, 2);
];
```

§32. Switch Transcript On Rule.

```
[ SWITCH_TRANSCRIPT_ON_R;
    if (actor ~= player) rfalse;
    if (gg_scriptstr ~= 0) return GL__M(##ScriptOn, 1);
    if (gg_scriptfref == 0) {
        gg_scriptfref = glk_fileref_create_by_prompt($102, $05, GG_SCRIPTFREF_ROCK);
        if (gg_scriptfref == 0) jump S1Failed;
    }
    ! stream_open_file
    gg_scriptstr = glk_stream_open_file(gg_scriptfref, $05, GG_SCRIPTSTR_ROCK);
    if (gg_scriptstr == 0) jump S1Failed;
    glk_window_set_echo_stream(gg_mainwin, gg_scriptstr);
    GL__M(##ScriptOn, 2);
    VersionSub();
    return;
    .S1Failed;
    GL__M(##ScriptOn, 3);
];
```

§33. Switch Transcript Off Rule.

```
[ SWITCH_TRANSCRIPT_OFF_R;
    if (actor ~= player) rfalse;
    if (gg_scriptstr == 0) return GL__M(##ScriptOff,1);
    GL__M(##ScriptOff, 2);
    glk_stream_close(gg_scriptstr, 0); ! stream_close
    gg_scriptstr = 0;
];
```

§34. Announce Story File Version Rule.

```
[ ANNOUNCE_STORY_FILE_VERSION_R ix;
    if (actor ~= player) rfalse;
    Banner();
    print "Identification number: ";
    for (ix=6: ix <= UUID_ARRAY->0: ix++) print (char) UUID_ARRAY->ix;
    print "^";
    @gestalt 1 0 ix;
    print "Interpreter version ", ix / $10000, ".", (ix & $FF00) / $100,
    ".", ix & $FF, " / ";
    @gestalt 0 0 ix;
    print "VM ", ix / $10000, ".", (ix & $FF00) / $100, ".", ix & $FF, " / ";
    print "Library serial number ", (string) LibSerial, "^";
    #Ifdef LanguageVersion;
    print (string) LanguageVersion, "^";
    #Endif; ! LanguageVersion
    ShowExtensionVersions();
    say__p = 1;
];
```

§**35. Descend To Specific Action Rule.** There are 100 or so actions, typically, and this rule is for efficiency's sake: rather than perform 100 or so comparisons to see which routine to call, we indirect through a jump table. The routines called are the `-Sub` routines: thus, for instance, if `action` is `##Wait` then `WaitSub` is called. It is essential that this routine not be called for fake actions: in I7 use this is guaranteed, since fake actions are not allowed into the action machinery at all.

Strangely, Glulx's action routines table is numbered in an off-by-one way compared to the Z-machine's: hence the `+1`.

```
[ DESCEND_TO_SPECIFIC_ACTION_R;
    indirect(#actions_table-->(action+1));
    rtrue;
];
```

FileIO Template B/iot

Purpose

Reading and writing external files, in the Glulx virtual machine only.

§**1. Language.** This whole template contains material used only if the "Glulx external files" element is part of Inform's current definition, so:

```
#IFDEF PLUGIN_FILES;
```

§**2. Structure.** The I7 kind of value "auxiliary-file" is an `-->` array, holding a memory structure containing information about external files. The following constants specify memory offsets and values. Note the safety value stored as the first word of the structure: this helps protect the routines below from accidents. (16339, besides being prime, is a number interesting to the author of Inform since it was the examination board identifying number of his school, and so had to be filled in on all of the many papers he sat during his formative years.)

```
Constant AUXF_MAGIC = 0; ! First word holds a safety constant
Constant AUXF_MAGIC_VALUE = 16339; ! Should be first word of any valid file structure
Constant AUXF_STATUS = 1; ! One of the following:
    Constant AUXF_STATUS_IS_CLOSED = 1; ! Currently closed, or perhaps doesn't exist
    Constant AUXF_STATUS_IS_OPEN_FOR_READ = 2;
    Constant AUXF_STATUS_IS_OPEN_FOR_WRITE = 3;
    Constant AUXF_STATUS_IS_OPEN_FOR_APPEND = 4;
Constant AUXF_BINARY = 2; ! False for text files (I7 default), true for binary
Constant AUXF_STREAM = 3; ! Stream for an open file (meaningless otherwise)
Constant AUXF_FILENAME = 4; ! Packed address of constant string
Constant AUXF_IFID_OF_OWNER = 5; ! UUID_ARRAY if owned by this project, or
    ! string array of IFID of owner wrapped in //...//, or NULL to leave open
```

§**3. Instances.** These structures are not dynamically created: they are precompiled by the NI compiler, already filled in with the necessary values. The following command generates them.

```
{-call:Plugins::Files::arrays}
```

§**4. Errors.** This is used for I/O errors of all kinds: it isn't within the Glulx-only code because one of the errors is to try to use these routines on the Z-machine.

```
[ FileIO_Error extf err_text  struc;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES)) {
        print "^*** Error on unknown file: ", (string) err_text, " ***^";
    } else {
        struc = TableOfExternalFiles-->extf;
        print "^*** Error on file '",
            (string) struc-->AUXF_FILENAME, "': ",
            (string) err_text, " ***^";
    }
    RunTimeProblem(RTP_FILEIOERROR);
    return 0;
];
```

§5. Glulx Material.

```
#IFDEF TARGET_GLULX;
```

§**6. Existence.** Determine whether a file exists on disc. Note that we have no concept of directories, or the file system structure on the host machine: indeed, it is entirely up to the Glulx VM what it does when asked to look for a file. By convention, though, files for a project are stored in the same folder as the story file when out in the wild; when a project is developed within the Inform user interface, they are either (for preference) stored in a `Files` subfolder of the `Materials` folder for a project, or else stored alongside the Inform project file.

```
[ FileIO_Exists extf  fref struc rv usage;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES)) rfalse;
    struc = TableOfExternalFiles-->extf;
    if ((struc == 0) || (struc-->AUXF_MAGIC ~= AUXF_MAGIC_VALUE)) rfalse;
    if (struc-->AUXF_BINARY) usage = fileusage_BinaryMode;
    else usage = fileusage_TextMode;
    fref = glk_fileref_create_by_name(fileusage_Data + usage,
        Glulx_ChangeAnyToCString(struc-->AUXF_FILENAME), 0);
    rv = glk_fileref_does_file_exist(fref);
    glk_fileref_destroy(fref);
    return rv;
];
```

§**7. Readiness.** One of our problems is that a file might be being used by another application: perhaps even by another story file running in a second incarnation of Glulx, like a parallel world of which we can know nothing. We actually want to allow for this sort of thing, because one use for external files in I7 is as a sort of communications conduit for assisting applications.

Most operating systems solve this problem by means of locking a file, or by creating a second lock-file, the existence of which indicates ownership of the original. We haven't got much access to the file-system, though: what we do is to set the first character of the file to an asterisk to mark it as complete and ready for reading, or to a hyphen to mark it as a work in progress.

`FileIO_Ready` determines whether or not a file is ready to be read from: it has to exist on disc, and to be openable, and also to be ready in having this marker asterisk.

`FileIO_MarkReady` changes the readiness state of a file, writing the asterisk or hyphen into the initial character as needed.

```
[ FileIO_Ready extf  struc fref usage str ch;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES)) rfalse;
    struc = TableOfExternalFiles-->extf;
    if ((struc == 0) || (struc-->AUXF_MAGIC ~= AUXF_MAGIC_VALUE)) rfalse;
    if (struc-->AUXF_BINARY) usage = fileusage_BinaryMode;
    else usage = fileusage_TextMode;
    fref = glk_fileref_create_by_name(fileusage_Data + usage,
        Glulx_ChangeAnyToCString(struc-->AUXF_FILENAME), 0);
    if (glk_fileref_does_file_exist(fref) == false) {
        glk_fileref_destroy(fref);
        rfalse;
    }
    str = glk_stream_open_file(fref, filemode_Read, 0);
    ch = glk_get_char_stream(str);
    glk_stream_close(str, 0);
    glk_fileref_destroy(fref);
    if (ch ~= '*') rfalse;
    rtrue;
];
[ FileIO_MarkReady extf readiness  struc fref str ch usage;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES))
        return FileIO_Error(extf, "tried to open a non-file");
    struc = TableOfExternalFiles-->extf;
    if ((struc == 0) || (struc-->AUXF_MAGIC ~= AUXF_MAGIC_VALUE)) rfalse;
    if (struc-->AUXF_BINARY) usage = fileusage_BinaryMode;
    else usage = fileusage_TextMode;
    fref = glk_fileref_create_by_name(fileusage_Data + usage,
        Glulx_ChangeAnyToCString(struc-->AUXF_FILENAME), 0);
    if (glk_fileref_does_file_exist(fref) == false) {
        glk_fileref_destroy(fref);
        return FileIO_Error(extf, "only existing files can be marked");
    }
    if (struc-->AUXF_STATUS ~= AUXF_STATUS_IS_CLOSED) {
        glk_fileref_destroy(fref);
        return FileIO_Error(extf, "only closed files can be marked");
    }
    str = glk_stream_open_file(fref, filemode_ReadWrite, 0);
    glk_stream_set_position(str, 0, 0); ! seek start
    if (readiness) ch = '*'; else ch = '-';
    glk_put_char_stream(str, ch); ! mark as complete
    glk_stream_close(str, 0);
    glk_fileref_destroy(fref);
];
```

§8. Open File.

```
[ FileIO_Open extf write_flag append_flag
    struc fref str mode ix ch not_this_ifid owner force_header usage;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES))
        return FileIO_Error(extf, "tried to open a non-file");
    struc = TableOfExternalFiles-->extf;
    if ((struc == 0) || (struc-->AUXF_MAGIC ~= AUXF_MAGIC_VALUE)) rfalse;
    if (struc-->AUXF_STATUS ~= AUXF_STATUS_IS_CLOSED)
        return FileIO_Error(extf, "tried to open a file already open");
    if (struc-->AUXF_BINARY) usage = fileusage_BinaryMode;
    else usage = fileusage_TextMode;
    fref = glk_fileref_create_by_name(fileusage_Data + usage,
        Glulx_ChangeAnyToCString(struc-->AUXF_FILENAME), 0);
    if (write_flag) {
        if (append_flag) {
            mode = filemode_WriteAppend;
            if (glk_fileref_does_file_exist(fref) == false)
                force_header = true;
        }
        else mode = filemode_Write;
    } else {
        mode = filemode_Read;
        if (glk_fileref_does_file_exist(fref) == false) {
            glk_fileref_destroy(fref);
            return FileIO_Error(extf, "tried to open a file which does not exist");
        }
    }
    str = glk_stream_open_file(fref, mode, 0);
    glk_fileref_destroy(fref);
    if (str == 0) return FileIO_Error(extf, "tried to open a file but failed");
    struc-->AUXF_STREAM = str;
    if (write_flag) {
        if (append_flag)
            struc-->AUXF_STATUS = AUXF_STATUS_IS_OPEN_FOR_APPEND;
        else
            struc-->AUXF_STATUS = AUXF_STATUS_IS_OPEN_FOR_WRITE;
        glk_stream_set_current(str);
        if ((append_flag == FALSE) || (force_header)) {
            print "- ";
            for (ix=6: ix <= UUID_ARRAY->0: ix++) print (char) UUID_ARRAY->ix;
            print " ", (string) struc-->AUXF_FILENAME, "^";
        }
    } else {
        struc-->AUXF_STATUS = AUXF_STATUS_IS_OPEN_FOR_READ;
        ch = FileIO_GetC(extf);
        if (ch ~= '-' or '*') { jump BadFile; }
        if (ch == '-')
```

```
            return FileIO_Error(extf, "tried to open a file which was incomplete");
        ch = FileIO_GetC(extf);
        if (ch ~= ' ') { jump BadFile; }
        ch = FileIO_GetC(extf);
        if (ch ~= '/') { jump BadFile; }
        ch = FileIO_GetC(extf);
        if (ch ~= '/') { jump BadFile; }
        owner = struc-->AUXF_IFID_OF_OWNER;
        ix = 3;
        if (owner == UUID_ARRAY) ix = 8;
        if (owner ~= NULL) {
            for (: ix <= owner->0: ix++) {
                ch = FileIO_GetC(extf);
                if (ch == -1) { jump BadFile; }
                if (ch ~= owner->ix) not_this_ifid = true;
                if (ch == ' ') break;
            }
            if (not_this_ifid == false) {
                ch = FileIO_GetC(extf);
                if (ch ~= ' ') { jump BadFile; }
            }
        }
        while (ch ~= -1) {
            ch = FileIO_GetC(extf);
            if (ch == 10 or 13) break;
        }
        if (not_this_ifid) {
            struc-->AUXF_STATUS = AUXF_STATUS_IS_CLOSED;
            glk_stream_close(str, 0);
            return FileIO_Error(extf,
                "tried to open a file owned by another project");
        }
    }
    return struc-->AUXF_STREAM;
    .BadFile;
    struc-->AUXF_STATUS = AUXF_STATUS_IS_CLOSED;
    glk_stream_close(str, 0);
    return FileIO_Error(extf, "tried to open a file which seems to be malformed");
];
```

§9. Close File.

```
[ FileIO_Close extf  struc;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES))
        return FileIO_Error(extf, "tried to open a non-file");
    struc = TableOfExternalFiles-->extf;
    if (struc-->AUXF_STATUS ~=
        AUXF_STATUS_IS_OPEN_FOR_READ or
        AUXF_STATUS_IS_OPEN_FOR_WRITE or
        AUXF_STATUS_IS_OPEN_FOR_APPEND)
        return FileIO_Error(extf, "tried to close a file which is not open");
    if ((struc-->AUXF_BINARY == false) &&
        (struc-->AUXF_STATUS ==
        AUXF_STATUS_IS_OPEN_FOR_WRITE or
        AUXF_STATUS_IS_OPEN_FOR_APPEND)) {
        glk_set_window(gg_mainwin);
    }
    if (struc-->AUXF_STATUS ==
        AUXF_STATUS_IS_OPEN_FOR_WRITE or
        AUXF_STATUS_IS_OPEN_FOR_APPEND) {
        glk_stream_set_position(struc-->AUXF_STREAM, 0, 0); ! seek start
        glk_put_char_stream(struc-->AUXF_STREAM, '*'); ! mark as complete
    }
    glk_stream_close(struc-->AUXF_STREAM, 0);
    struc-->AUXF_STATUS = AUXF_STATUS_IS_CLOSED;
];
```

§10. Get Character.

```
[ FileIO_GetC extf  struc;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES)) return -1;
    struc = TableOfExternalFiles-->extf;
    if (struc-->AUXF_STATUS ~= AUXF_STATUS_IS_OPEN_FOR_READ) return -1;
    return glk_get_char_stream(struc-->AUXF_STREAM);
];
```

§11. Put Character.

```
[ FileIO_PutC extf char  struc;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES)) return -1;
        return FileIO_Error(extf, "tried to write to a non-file");
    struc = TableOfExternalFiles-->extf;
    if (struc-->AUXF_STATUS ~=
        AUXF_STATUS_IS_OPEN_FOR_WRITE or
        AUXF_STATUS_IS_OPEN_FOR_APPEND)
        return FileIO_Error(extf,
            "tried to write to a file which is not open for writing");
    return glk_put_char_stream(struc-->AUXF_STREAM, char);
];
```

§**12. Print Line.** We read characters from the supplied file until the next newline character. (We allow for that to be encoded as either a single `0a` or a single `0d`.) Each character is printed, and at the end we print a newline.

```
[ FileIO_PrintLine extf ch  struc;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES))
        return FileIO_Error(extf, "tried to write to a non-file");
    struc = TableOfExternalFiles-->extf;
    for (::) {
        ch = FileIO_GetC(extf);
        if (ch == -1) rfalse;
        if (ch == 10 or 13) { print "^"; rtrue; }
        print (char) ch;
    }
];
```

§**13. Print Contents.** Repeating this until the file runs out is equivalent to the Unix command `cat`, that is, it copies the stream of characters from the file to the output stream. (This might well be another file, just as with `cat`, in which case we have a copy utility.)

```
[ FileIO_PrintContents extf tab  struc;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES))
        return FileIO_Error(extf, "tried to access a non-file");
    struc = TableOfExternalFiles-->extf;
    if (struc-->AUXF_BINARY)
        return FileIO_Error(extf, "printing text will not work with binary files");
    if (FileIO_Open(extf, false) == 0) rfalse;
    while (FileIO_PrintLine(extf)) ;
    FileIO_Close(extf);
    rtrue;
];
```

§**14. Print Text.** The following writes a given piece of text as the new content of the file, either as the whole file (if `append_flag` is false) or adding only to the end (if true).

```
[ FileIO_PutContents extf text append_flag  struc str ch;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES))
        return FileIO_Error(extf, "tried to access a non-file");
    struc = TableOfExternalFiles-->extf;
    if (struc-->AUXF_BINARY)
        return FileIO_Error(extf, "writing text will not work with binary files");
    str = FileIO_Open(extf, true, append_flag);
    if (str == 0) rfalse;
    @push say__p; @push say__pc;
    ClearParagraphing();
    PrintText(text);
    FileIO_Close(extf);
    @pull say__pc; @pull say__p;
    rfalse;
];
```

§**15. Serialising Tables.** The most important data structures to “serialise” – that is, to convert from their binary representations in memory into text representations in an external file – are Tables. Here we only carry out the file-handling; the actual translations are in “Tables.i6t”.

```
[ FileIO_PutTable extf tab rv  struc;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES))
        return FileIO_Error(extf, "tried to write table to a non-file");
    struc = TableOfExternalFiles-->extf;
    if (struc-->AUXF_BINARY)
        return FileIO_Error(extf, "writing a table will not work with binary files");
    if (FileIO_Open(extf, true) == 0) rfalse;
    rv = TablePrint(tab);
    FileIO_Close(extf);
    if (rv) return RunTimeProblem(RTP_TABLE_CANTSAVE, tab);
    rtrue;
];
[ FileIO_GetTable extf tab  struc;
    if ((extf < 1) || (extf > NO_EXTERNAL_FILES))
        return FileIO_Error(extf, "tried to read table from a non-file");
    struc = TableOfExternalFiles-->extf;
    if (struc-->AUXF_BINARY)
        return FileIO_Error(extf, "reading a table will not work with binary files");
    if (FileIO_Open(extf, false) == 0) rfalse;
    TableRead(tab, extf);
    FileIO_Close(extf);
    rtrue;
];
```

§**16. Z-Machine Stubs.** These routines do the minimum possible, but equally, they only generate a run-time problem when there is no alternative.

```
#IFNOT; ! TARGET_GLULX
[ FileIO_Exists extf; rfalse; ];
[ FileIO_Ready extf; rfalse; ];
[ FileIO_GetC extf; return -1; ];
[ FileIO_PutTable extf tab;
    return FileIO_Error(extf, "external files can only be used under Glulx");
];
[ FileIO_MarkReady extf status; FileIO_PutTable(extf); ];
[ FileIO_GetTable extf tab; FileIO_PutTable(extf); ];
[ FileIO_PrintContents extf; FileIO_PutTable(extf); ];
[ FileIO_PutContents extf; FileIO_PutTable(extf); ];
#ENDIF; ! TARGET_GLULX
```

§**17. Back To Core.**

```
#IFNOT; ! PLUGIN_FILES
[ FileIO_GetC extf; return -1; ];
#ENDIF; ! PLUGIN_FILES
```