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
! ------------------------------------------------------------------------
```