LIN

Leonardo Implementation Note

Cognitive Autonomous Systems Laboratory
Department of Computer and Information Science
Linköping University, Linköping, Sweden

Erik Sandewall

Self-description of Agents and Sessions

This implementation note pertains to the development of the Leonardo system.
Identified as PM-lin-004, it is disseminated through the CAISOR website
and has URL `http://www.ida.liu.se/ext/caisor/pm-archive/lin/004/`

Related information can be obtained via the following www sites:

CAISOR website:	`http://www.ida.liu.se/ext/caisor/`
CASL website:	`http://www.ida.liu.se/ext/casl/`
Leonardo system infosite:	`http://www.ida.liu.se/ext/leonardo/`
The author:	`http://www.ida.liu.se/~erisa/`
Date of manuscript:	2011-02-27

1 Introduction

This note describes the entities and global variables that are used in a Leonardo session for describing its current state. This information is identified and assigned during the session startup process. Its representation has evolved over time, with the effect that there are both *currently recommended* entities and *legacy* entities with the same or related contents. The legacy entities are intended to be phased out but are allowed to remain for the time being, but they should not be used when new code is written.

The reader should first be familiar with the general character of the Leonardo system, in particular those aspects that are described in the memo "Leonardo Installation, Startup and Self-Description."

We shall describe currently recommended entities and legacy entities in separate sections. After this there are some additional sections describing, for example, the entities and global variables that are used by the facility for message-passing.

The term "entities" here and henceforth will include both Leonardo entities and global Lisp variables. We distinguish between the following kinds of such *session-describing entities and attributes:*

- *Persistent entities and attributes* whose values are stored in entityfiles and obtained to the session when those entityfiles are loaded.
- *Session entities and attributes* that are assigned and used for the duration of the session.
- *Startup entities and attributes* that are used during the startup process and unused thereafter.
- *Legacy entities and attributes* that have belonged to some of the previous groups but which are no longer in use or are being phased out. They are documented briefly here in case there should be remaining usage of them somewhere in the code.

A number of entityfiles are loaded in sequence in order to start a session. This *session startup sequence* includes the following entityfiles, so these are the ones to look at in order to understand in more detail how the session-describing entities are assigned their values.

- The boot ef (entityfile), e.g. `startleo.leo` if the session is started using `startleo.bat`
- The ef `bootfuns.leo` in the `Process/Boot/` directory
- The ef `session-init.leo` in `core-kb` i.e. the `Core` directory
- The ef `load-envir.leo` in `indiv-kb`
- The ef `clonemon-defs.leo` also in `indiv-kb`
- The ef `indivmap-oper.leo` in `els-kb`

These are the 'procedural' files in the sense of ef that they contain definitions of commands, functions, and other procedural constructs. They may also contain assignments to the session-describing entities mentioned above. In addition, the following ef are loaded during that early stage of the session startup sequence merely in order to load declarative information:

- `selfdescr.leo`
- `clonemon.leo`
- `self-kb.leo`
- `kb-catal.leo`
- `core-kb.leo`

The first one of these contains the attribute assignments for the agent's proper name, described below as the value of the global variable `*my-id*`. The `clonemon` file contains information for detecting when a Leonardo individual has been cloned, and the last three are catalogs describing the locations of other entityfiles.

Many of these entityfiles are loaded from their `.leos` representation since they are loaded before the KRE parser, or for reasons of speed.

Finally there is the file `/Process/main/State/cl/sessionmode.cl` which contains parameters specifying some aspects of the conduct of the command-line dialog. It is described in Section 7.1.

The following text will specify, for each self-description entity, the ef where it is assigned its value.

2 Entities for Agents and Individuals

Note: the first two subsections of this section are identical to subsections in the memo *Self-description of Agents and Sessions.*

2.1 The Persistent Representation of Agents

The persistent representation of an agent is as a directory with its subdirectories and files in the computer's memory, or on a detachable memory device (DMD) such as a USB stick. This will be called the *agent directory.* A Leonardo agent can only execute if it is located in a *Leonardo individual* containing one or more, related agents and some auxiliary information for them. An *individual directory* is therefore a directory containing one or more agent directories as immediate subdirectories, plus an additional subdirectory called `Indivmap` that contains a number of catalogs with information about the computing environment and neighboring individuals.

Leonardo agents and individuals are *mobile* in the sense that they can be moved from one host to another and execute sessions on each of these. The internal information in the agent is preserved as the agent is moved. It can obtain information from each host that it visits, and accumulate it, and the agent can also leave information behind when it moves on. For example, an agent that is located on a USB stick can be moved between hosts simply by moving this memory device. It is also possible to move an agent in the sense of moving its entire file structure from one memory device to another, for example from a USB to a hard disk, or from the hard disk of one host to the hard disk of another one. These possibilities are used very frequently in our own work.

There is also a stronger sense of "mobile agent" where the agent decides *by itself* to move from one host to another. This is useful in some applications and the Leonardo system design is prepared for this functionality, but it has not yet been needed in any of our own applications.

2.2 Configurations and Directory Names

It is the nature of things that there are different agents with different contents, including both procedural contents and 'data' contents, but there are also many situations where it is desirable to have several *exemplars* of the same *species* of agents, that is, several copies that differ in some of the details but whose overall structure is the same. For example, if an application is realized as a Leonardo individual containing one or more agents, then one may wish to have one exemplar for production use and another one for further development and testing. [1]

The following structure is used in order to organize the resulting multitude of agents. Each species of agents is given a particular name; `remus` and `leoregistrar` are examples of currently used species. Each individual has a particular *configuration* which specifies what species must or may be present in an individual with that configuration. For example, the configuration `Registrar` is specified to contain agents of the species `remus` and `leoregistrar.`

An individual can not contain more than one agent of the same species. Therefore, for communication within an individual it is sufficient to refer to agents using their respective species name.

For communication between individuals it is necessary to distinguish between different individuals with the same configuration. Therefore, each individual has a *directory name* which is formed by appending a serial number to the name of its configuration, obtaining for example the directory name `Registrar-2.` There is a server facility for registration of individuals, which is a mechanism for obtaining unique directory names within a certain context.

Agents have *proper names,* similar to the directory names of individuals. However, it is not possible to use the same serial number for the individual and its agents, since some species are used in several configurations. The assignment of proper names for agents is based on a particular aspect of the design, namely, that some species are used as *kernel agents* that provide the basic machinery for the individual, and each configuration of individuals designates one particular agent species that is to be used as kernels. Consequently, kernel agents are assigned proper names consisting of their species name plus a serial number that is assigned by the registrar, and other agents in the same individual obtain a proper name consisting of their species name followed by the serial number of the kernel agent in the same individual.

For example, since the individual `Registrar-2` consists of agents of the species `remus` and `leoregistrar,` if the former agent has the proper name `remus-43` then the latter one will have the proper name `leoregistrar-43.` One will then assume that there are other `remus` agents with numbers from

[1] In the English language the word 'exemplar' has a dual meaning, so it can refer both to one out of many copies of e.g. a book, and also to a prototype that copies are made from. We use the word in the former sense.

1 to 42, but there is no similar assumption for agents of the `leoregistrar` species.

As already mentioned, the persistent representation of a Leonardo individual is as a directory having immediate subdirectories for each of the agents in the individual. It is strongly recommended that the directory of the individual is named by the individual's directory name. It is also required that the directory of each of the included agents is named by the agent's species (not its proper name). In our example, the directory `Registrar-2` will have subdirectories `remus` and `leoregistrar.`

2.3 Attributes of Agent Proper Names

Each agent contains an entityfile that is located at

```
Process/main/Defblock/selfdescr.leo
```

as seen from the agent's main directory. This entityfile contains one significant entity, namely, the agent's proper name. The type of this entity is `leo-agent.` This entity and its attributes are persistent.

`*my-id*` - Within each session, the global variable `*my-id*` has the agent's proper name as its value. Notice that this design makes it possible for a session to load the entity-descriptions for the proper names of other agents, without interfering with its own entity-description, but at the same time it is well defined how it can identify the entity for its own proper name. This entity is the starting point for arriving at a species of other information about the entity. The following are its most important attributes.

- `leoname` - The name of its species, e.g. `remus`
- `has-owner` - An identifier for the person 'owning' the agent, for example `/Larsson.Lars`
- `created-on` - Date of creation of this agent, e.g. `2011-02-22`
- `in-society` - Described below
- `indivmap-name` - Described below

2.4 Network Level Names

Directory names of individuals and proper names of agents are convenient to use, but unfortunately they can not be guaranteed to be unique in a larger scale system. The Leonardo architecture makes use of *leosocieties* as a way of organizing situations involving many users. A leosociety is a structure for the maintenance of a family of individuals and their constituent agents, including registration, assignment of the numbers for directory names and agent proper names, agent catalogs that provide addressing information for message-passing, software updates, and so forth. For example, if each student in a class receives her or his own software individual for use in course projects, then it may be appropriate to create a separate leosociety for this purpose.

It follows that directory names and agent proper names are unique within each leosociety, but not necessarily between them. Therefore there is a

more elaborate naming scheme, called *network names* or *indivmap names* that guarantees uniqueness and that is also used for registration purposes. These network names and their attributes are persistent.

Network names are *composite entities,* i.e. expressions consisting of an operator and its argument(s). The network name for an individual is formed like in the following example for `Registrar-2`

```
(leoind: Registrar soc.root 2)
```

where the second argument of the operator specifies the leosociety that has issued the serial number for this individual. The network name for a kernel agent is formed like in the following example for `remus-43`

```
(kercl: soc.root 1 soc.root 43)
```

where `soc.root 1` is the network characterization for the species `remus,` formed as the leosociety that has registered the species and the number it obtained, and then again `soc.root` for specifying that it was the same leosociety that assigned the number 43 to the present examplar of the species.

Finally, the network name for a non-kernel agent is formed like in the following example for `leoregistrar-43`

```
(agda: (kercl: soc.root 1 soc.root 43) leoregistrar)
```

representing that this is the `leoregistrar` agent in the individual whose kernel agent is `remus-43.` The acronym `agda` stands for 'agent daughter'.

`indivmap-name` - The proper name of an agent refers to the agent's network name using tye attribute `indivmap-name`

`*curagent*` - The value of this global session variable shall be the network name of the current agent, i.e. it can be obtained as the value of

```
(get *my-id* 'indivmap-name)
```

The entities that are network names for individuals, kernel agents and other agents have their entity descriptions in entityfiles with the following names, respectively.

```
individuals-catal
kernel-agents-catal
agents-catal
```

These entityfiles are not stored within each agent, but in the separate directory `Indivmap` which is an immediate subdirectory of the individual directory. In this way it is shared by all the agents in the individual. The following are the most important attributes of an individual network-name entity.

- `directory-name` - its directory name
- `has-owner` - the owner, represented like in the agent's attribute with the same name
- `in-host` - the host where this individual is persistently located, in those cases where this is the case. (Not applicable for individuals that are stored on detachable memory devices).
- `has-kernel` - the network name of the individual's kernel agent.

The following is one of the attributes of kernel-agent and non-kernel-agent network names.

- `in-individual` - the network name of the individual that the present agent belongs to.

A number of other attributes for the network names of agents are used for specifying how to reach the agent in case of message-passing. The entityfiles for catalogs of agents, individuals, hosts and so forth shall therefore contain information about other individuals, so that message-passing becomes possible. The registrar of the leosociety is the point of contact for exchanging this information between individuals in the society, as well as (when applicable) between leosocieties.

This design means in particular that the information about an agent is partitioned between attributes for the agent's proper name and attributes for its network name. The former part is local to the agent itself whereas the network name and its attributes are made public to other members of the same leosociety by appearing in the `Indivmap` structure.

The one-to-one correspondence between the proper name and the network name [2] is represented by the `indivmap-name` attribute on the proper name. It is duplicated by the `refers-to` attribute on the proper name which shall have the same value as the `indivmap-name` attribute. [3] In the reverse direction there is an attribute `referred-as` on network names that have the corresponding proper name as its value.

The `indivmap-name` attribute is persistent i.e. it is saved in the entityfile containing the proper name, whereas the other two attributes are session attributes whose values are constructed during session startup but not preserved.

3 Entities for Sessions and Episodes

3.1 Session Configuration Entities

The term 'configuration' is used for two different purposes in the Leonardo architecture. Besides configurations for individuals that were introduced in Subsection 2.2, there are also *session configurations* for use within agents. Each session configuration specifies one particular way of starting a session. Starting a session in a Linux environment or in a Windows environment requires different configurations, for example. It is also possible to define special configurations where a particular set of knowledgeblocks and entityfiles are loaded during startup.

Each available configuration is represented by an entityfile in the directory `Process/main/Boot/ .` Two session configurations are used almost universally, namely `startleo` for use in Windows environments and `linuleo` for Linux and Mac-OS environments.

[2] Notice however that this correspondence is only one-to-one within one specific leosociety.

[3] This redundancy is for legacy reasons.

Each session configuration has a corresponding startup file, in particular `startleo.bat` and the Linux executable file `linuleo.lex` in the directory `Process/main/ .` The startup file `startleo.bat` invokes the CommonLisp interpreter with `Boot/cl/startleo.leos` as its starting-file parameter [4] and similarly for other session configuration entities. The `.leos` version of the session configuration entityfile is therefore always the first file in the startup sequence.

Session configuration entities are persistent.

`*myconfig*` - the global variable `*myconfig*` has the currently used configuration as its value. It is merely a startup entity, i.e. it is assigned during the session startup phase and should not be used elsewhere. The attribute `configuration` that is defined in the next section should be used instead.

3.2 Session Entities

The following global variables, entities and attributes are introduced and assigned in ef `session-init.` These are session entities, i.e. they apply only within one session. (The same applies for episode entities and session history entities that are described in later subsections of this section).

`*cursession*` The value shall be a composite entity of type `leosession` that denotes the current session; it shall look like e.g.

```
(session: leoregistrar-43 119)
```

specifying that it is the 119-th time that a session is started by the agent whose proper name is `leoregistrar-43,` which presumably is the present agent. (However, it is perfectly possible for one agent to load and analyze session information from other agents in the same leosociety, or from its own earlier sessions).

The session entity is used as a carrier for miscellaneous status information within the session, so it is a session entity in the sense of Section 1. It has the following attributes, among others.

- `inhost` - the identity of the current host, specified as an entity such as `host.aldebaran` which is defined in ef `hosts-catal.`
- `configuration` - the entity representing the current session configuration as described in Subsection 3.1 above. The value shall therefore be an entity of type `startup-file.`
- `init-leos-files` - a Leonardo sequence of the first entityfiles that were loaded during startup, and that were loaded in their `.leos` format.
- `runstart-ms` - the time when the session started, expressed in global seconds i.e. seconds since the beginning of year 1900.
- `runstart-datetime` - the time when the session started, expressed as a Leonardo sequence e.g. `<2009 8 10 23 0 3 0 t -1>`

[4] The session is started in the directory `Process/main/` relative to the top-level directory of the agent, so the `startleo` file is located at `Process/main/Boot/cl/startleo.leos.`

- `cur-time` - initialized as the value 0 (zero), this attribute is used by some parts of the system as a discrete "clock" that makes one tick for every main "action" being performed.
- `session-nr` - an integer specifying the number of the present session of the present agent.
- `already-started` - a flag that is used for system purposes, so that the banner shown when a session starts is different from the banner that is shown after recovery from an execution error.
- `uses-language` - one of the entities `English, Swedish,` or any other member of the set that is provided as the value of the global variable `*phrasecode-languages*` (see Subsection 6.2). This controls the selection of language-variable phrases.
- `globvar-bindings` - contains the binding of global variables that are made using the `ssv` command.

The session number that is used as the second argument of `session:` is maintained in the file `State/session-nr.txt` relative to the starting directory. The value is picked up and incremented by the session starting routines.

The value of the attribute `inhost` is obtained in either of two ways. In a Windows environment it is obtained by asking the underlying Lisp system to fetch the value of the operating-system environment variable that contains this information. In the case of Linux and Mac-OS environments this option is not available in Allegro CommonLisp [5] so the hostname is obtained instead by reading the single line in the file `Process/temp/_compnam.sii` This file must therefore be edited by the user if an agent is moved to a Linux or Mac-OS environment.

Note: Couldn't we set up the linuleo.lex file as a script that assigns contents to this file?

`current-host` The value of `(get 'current-host 'value)` shall be the entity representing the current host, i.e. it is the same as the value of `(get *cursession* 'inhost).` (The use of `current-host` shall be phased out).

3.3 Episode Entities

The episode construct in Leonardo is used for representing a segment of the command-line interactions within a session, and more generally for representing a sequence of actions that are performed during the session. Episodes can be used, for example, for representing a sequence of interactions that is to be extracted and used for demonstration and publication purposes. It is also intended that they shall be used for managing sequences of actions in case-based planning systems.

`episode-0000` This entity of type `episode` is the top node in a tree of subepisodes. Further nodes in this tree can be generated by applications as a way of organizing past events during a session. The entity `episode-0000` is initialized with the following attributes, among others.

[5] It is supposed to be available, but it apparently does not work.

- `in-session` - the value of `*cursession*` is also the value of this attribute.
- `has-sub-episodes` - a sequence of other entities of type `episode` which are the immediate daughter epsiodes.
- `max-episode` - initialized as 0 (zero), this is the number of the latest created episode, and it is used as a counter for setting the number of each additional sub-episode.
- `has-chronicle` - the *chronicle record* corresponding to the episode `episode-0000`
- `cur-time` - the present time setting in the current episode, usually represented as an integer that is incremented from zero and up.

The chronicle record is a large record containing much of the history information. Immedidately after its creation it looks like in the following example:

```
[chronicle :clock (session: leoregistrar-43 119)
           :occurs-in episode-0000]
```

Successive events (e.g. input commands) will then be added as arguments in that record. There is a one-to-one correspondence between episode entities and chronicle records, which is represented using the `has-chronicle` attribute and the `occurs-in` parameter.

The `clock` parameter makes it possible to use the `cur-time` attribute of either the current session or the current episode as the time counter. In addition there is in fact a global variable that can be used for the same purpose, namely:

`*tick*` This global variable is set to 0 when ef `lite-exec` is loaded. It shall be phased out.

The following global variables are used for identifying the most important points in the episode hierarchy.

`*top-epi*` The entity for the top-level episode, normally selected as the entity `episode-0000.`

`*cur-epi*` The entity for the current episode. It is initially selected as the value of `*top-epi*` but it changes when the agent creates and switches to a sub-episode. This may be done in order to create a record of a particular sequence of events, for example in case-based reasoning.

`*server-episode*` Intended for separate episodes as used by server processes. Not actively used at present; it is initially set to `episode-0000` and there is nothing in the system that changes it.

`*top-chronicle*` The value of this global variable is the chronicle associated with `*top-epi*` as the value of its `has-chronicle` attribute.

`*cur-chronicle*` The value of this global variable is the chronicle associated with `*cur-epi*` as the value of its `has-chronicle` attribute.

3.4 Session History Entities

The following global variables accumulate certain information about what has been done during the current session.

`*loaded-blocks*` Initially set to the empty Leonardo set, the value of this global variable shall be a list of the knowledgeblocks that have been loaded into the session using the operator `loadk,` not including those that are listed in the value of the `init-leos-files` attribute.

`*loaded-bundles*` Similar to `*loaded-blocks*` but little used. The idea is/was that a bundle should represent a subset of the entityfiles in a knowledgeblock, and there should be a load-bundle operation similar to `loadk.`

These variables shall be reorganized as additional attributes of the current session entity.

4 Legacy Entities for Sessions and Episodes

The following constructs are used by some of the existing code in Leonardo and are described here for reference, but they should not be used when new code is written.

`*curactor*` The value of this global variable is initially set to the symbol `IA.` It is more or less a legacy facility since it is little used, but some facilities that need to represent the "current X" for some X assign them as attributes of the current value of `*curactor*.` They should be replaced by attributes for the current value of current session and episode entities.

`(pickup-host)` Evaluation of this form obtains the entity representing the host where the session is presently running. `(get *cursession* 'inhost)` should be used instead.

`*top-episode*` Has the same value as `*top-chronicle*` - the variable `*top-episode*` was the original one but it was inappropriate since the value is a chronicle record and not an episode entity, and it was therefore renamed. There may still be remaining occurrences of the old variable name in some places in the code.

`*cur-episode*` Has the same value as `*cur-chronicle*` and is analogous to the previous item.

`*runstart-ms*` Replaced by `(get *cursession* 'runstart-ms).`

`*runstart-datetime*` Similarly replaced by the `runstart-datetime` attribute of the value of `*cursession*`

`*errset*` Set to `t` in ef `bootfuns` and is/was used for controlling how to handle trapped errors.

5 Computational Environment

5.1 Location of External Software

Facilities in Leonardo sometimes need to invoke other software systems, for example a text editor, a pdf renderer, or the LaTeX text formater. It must therefore have information about the location of such softwares in the host at hand. Although it is sometimes convenient to store this information inside each Leonardo individual, there are also situations where this is quite

inconvenient, either because an individual is regularly being moved between different hosts, or because there are a number of individuals on the same host and one does not wish to duplicate the information.

These various needs are met by having a knowledgeblock called `leohost-kb` whose directory `Leohost` may either be an immediate subdirectory of the individual's directory, or be located in a fixed position in a given host, for example `C:/Leohost/` on a Windows system. In the former case it will be addressed as `../../../Leohost/` from the point of view of the Leonardo session. The Leonardo entity for the host in question specifies the appropriate path to be used for Leonardo agents on that host. This is persistent information.

5.2 Location of the Allegro CommonLisp System

The http-related packages in the Allegro system require a separate loading command, and in order to do this Leonardo needs to know the path to the Allegro system that is being used. The following are the entities that are involved in this.

`has-allegro-path` - This is an attribute for host entities such as e.g. `host.aldebaran` where the value shall be a string showing the path to the Allegro system being used.

`(get allegrodir commandphrase)` - This is the entry for the Allegro system in the `Leohost` directory of external software.

`*acldir*` - This session variable is set to the directory that is obtained in either of the following ways:

- For the current agent, look up what individual it is in, then look up the host that the individual is supposed to be in, and finally look up the `has-allegro-path` attribute of the individual.
- In the currently used `Leohost` structure, look up the location of the presently used Allegro system, and use it

`*aserve-ok*` The session startup process attempts to load the http related packages, using the path information that is obtained from the above. If it succeeds in this then it sets `*aserve-ok*` to true, otherwise to nil.

5.3 Initial Description of Environment

At each point in time an agent is located in a particular individual, and in a particular (computer) host, but both of these may change over time. Each individual and each host contains a small amount of information describing itself which the agent picks up during session startup. If that information is missing, for example because the agent has arrived to a host that neither it nor any other Leonardo agent has ever visited before, then it asks the user for the information and stores it in the host.

Besides the catalogs of external software, the `Leohost` directory also contains a few files that are used temporarily when a new agent is being set up locally and before it and its associated entities (its owner, the host it is on, etc) have been registered properly.

`/Leohost/host-descr.leo` The ef `host-descr` in `Leohost` contains an entity `host-self` with the following attributes:

- `host-own-name` whose value is e.g. `host.aldebaran` i.e. an entity based on the host's name for itself in its operating system
- `host-owner-string` whose value is a string e.g. ``Lars Svensson''
- `has-host-type` whose value is one of the entities `stationary-host` or `detachable-host.` The latter value is normally used for laptops and in order to indicate that they may be attached to different local area networks at different times.

This information is collected when the user is queried, and later it is transferred to its final location in entityfiles in the `Indivmap` directory.

`/Indivmap/indiv-descr.leo` Besides the catalog files, the `Indivmap` directory also contains an ef called `indiv-descr` whose only purpose is to specify what the agent at hand is called in the network. It contains the entity `current-individual` whose type is `globvar` and which also has an attribute `value` whose value is the network name for the individual at hand.

These entityfiles have fixed names and they can always be read by an agent during its session startup phase (or later) in order to find out the Leonardo name of the host and the individual where it is currently located. Notice however that these ef provide only the names, but very little other information about the host and the individual. Other entityfiles in `indivmap-kb` are used for that purpose. In our examples, the ef `hosts-catal` in the `Indiv` directory would contain the entity `host.aldebaran` with its attributes, and the ef `individuals-catal` would contain the individual's network name. In this way it is possible for an agent to maintain and to use information both about itself and about other agents and individuals.

6 Command-Line Functionalities

6.1 Attitudes

The command-line executive (CLE) allows for a species of command-line *attitudes.* The standard one is called `att.ses` which is shown, among other things, as the prefix of the command-line prompt which may be e.g. `ses.007)` meaning interaction number 7 in a dialog with this attitude. Different attitudes are distinguished in particular with how they react to various kinds of errors. For example, if the precondition for a command is violated then one attitude may return an error message, and another attitude may invoke a planner in order to find a sequence of auxiliary actions that achieve the precondition of the given command.

`*cur-attitude*` The value of this variable shall be the current attitude.

`*attitude-list*` The value is the list of currently available attitudes.

These two session variables are initialized in the `sessionmode` file which is described in Subsection 7.1.

6.2 Phrasecodes

There are many points in the Leonardo software where a one-line response has to be produced to the user in the command-line dialog. In order to prepare the system for the use of multiple dialog languages, there is a facility whereby such phrases are not written in-line in the code, but instead the code contains a *phrasecode* whose corresponding phrase is defined in a separate property. Moreover, each phrasecode can be associated with separate phrases for each one of a number of languages, beginning of course with English which is also the default.

The persistent global variable `phrasecode-languages` contains the list of available languages. It is defined in ef `responsedefs.` Notice however that there is no guarantee that all phrasecodes are defined for all languages, so this list is more like a wish-list.

6.3 Auxiliary Graphical Entities

The following persistent entities contain simple graphical objects that are used by the command-line dialog and by the procedure for writing entityfiles.

`*starline*` Defined in ef `bootfuns,` the value is a string consisting entirely of asterisks. It is used in the greeting sequence to the user when a session is started.

`*quoteline*` Defined in ef `actexec,` the value is a string having the same length as the value of `*starline*` and consisting entirely of single-quote characters. It is used when the command-line dialog switches from CEL input to Lisp input.

`*separator*` A line consisting entirely of dashes, used as the separator between entities in an entityfile.

`*section-separator*` A line consisting entirely of equality signs and having the same length as the value of `*separator*,` used as the separator between sections of entities in an entityfile.

`*final-separator*` A line consisting entirely of the small letter o and having the same length as the value of `*separator*,` used as the terminating line in an entityfile.

7 Session Modes and Startup

7.1 Mode Parameters

Each agent has an optional file called `sessionmode.cl` (mentioned already in Section 1) which is loaded at a very early stage in the session startup process. This file is used for setting some parameters that determine low-level aspects of the dialog behavior, for example, printing of information messages when entityfiles are loaded and written, and the behavior of the system after a Lisp-level error in execution. It is written as an ordinary CommonLisp file using S-expressions and is intended to be edited in this

way. However there is also a separate facility for converting it to standard Leonardo form for Leonardo-level processing if this should be required.

The meaning of the parameters that are set in this file is described in the documentation file at `Process/main/State/sessionmode.dok` as well as in comments in the file itself. The following are additional descriptions of some of the parameters in this file.

`*load-by-leo*` - If the value of this variable is true then the entityfiles in `indivmap-kb` i.e. in the `Indivmap` directory will be loaded from their `.leo` representation during session startup; if it is nil then they will be loaded from their `.leos` representation. The latter alternative is faster, but it has the disadvantage that if some of these files have been edited before starting the session then one must either edit the `.leos` representation as well, or start a session, load and write the edited files, and close the session before the changes will take effect. This is inconvenient in particular when working with new individuals or with hosts that have not previoously been declared in the `indivmap-kb` that one is using. The default value for this variable is therefore true.

`(get *cursession* 'uses-language)` - the value of this attribute is set in `sessionmode.cl` and the standard version of the file sets it to the value `English`.

7.2 Allegro CommonLisp Parameters

The following persistent global variables are used to control the behavior of the Allegro CommonLisp system.

`*print-startup-message*` Leonardo's assignment to this global variable suppresses some information that ACL would otherwise present to the user, such as the information the he is running a variant of CommonLisp that accepts both upper-case and lower-case characters. The user may not need to know this every time.

`*restart-init-function*` Leonardo's assignment to this variable arranges that the session starts in Leonardo's command-line executive (CLE) and not in Lisp's command-line loop.

7.3 Session Startup Entities

The following are some global variables that are assigned during the session startup phrase in order to communicate between different procedures that are involved in that process. They will only be of interest if you are going to modify that startup process.

`*stars-before-acl-banner*` An auxiliary parameter that is used for obtaining the right presentation of the introductory 'banner' lines under a species of conditions.

`*load-verbose*`

`*load-ef-verbose*` (used in ef `actexec`)

`*startup-phase*`

These three are intended for controlling babble and other peculiarities of behavior during session startup, but they are also used for some other operations that apply later on in a session.

`*load-fail*` - Set to true if there is a failure during the loading of a knowledgeblock. The definitions for this are in ef `session-init` and they should be reviewed and maybe revised.

`*suppress-nullvalued*` - Set to true to indicate that nullvalued attributes are not to be shown when an entityfile is printed.

`*no-leoprint-babble*` -

`*base-server-port*` - Specifies the port used for the dynamic resource called `dres.base` with the default being 8080.

`this-session` - This entity has the attributes `runstart-datetime` and `runstart-ms` with the same contents as for the value of the same attributes in the value of `*cursession*.`

8 Session Entities for Message-Passing

Please refer to the separate memo "Agent LAN Ontology in Leonardo" for a description of how the agent network structure is represented. The following merely specifies some session entities that are used for this purpose.

`current-lan`

Entities representing computer hosts may have the type of `stationary-host` or `detachable-host.` For the former type there is an attribute `in-lan` that specifies what local area network the host is attached to, but for detachable hosts this does not apply. By definition they may be attached to different local area networks at different times.

The information about LAN attachment is significant for message-passing, since messages to another host on the same LAN can be sent using the port numbers that apply within the LAN whereas inter-LAN messages are restricted by the port assignments and mappings that are set by the server-side or target-side router.

Therefore, for use by individuals that are located on a detachable host or on a detachable memory device, there is an entity `current-lan` whose sole purpose is to have a value for the attribute `value.` Then `(get 'current-lan 'value)` shall be an entity of type `local-area-network` that is defined in ef `lan-catal` in `indivmap-kb.` This value is assigned during session startup and is obtained by asking the user whether the agent is on the same LAN as during its preceding session; the user can confirm that this is the case, or state another LAN.

The information about the preceding LAN location is obtained from the first line of the text file `../../../Indivmap/current-lan.txt.` The setting of `current-lan` is done in ef `indivmap-oper.` This implementation has been chosen since at present we have not identified any way of obtaining this information from the router that is in place during the session.

In addition there are commands for changing this value during the session; please refer to the ef `lan-config-def.`

`current-xlan`

If a session runs on the stationary memory of a stationary host then the value of `(get 'current-xlan 'value)` is the same as for `current-lan.` On the other hand if the session runs on a detachable host then the value of the expression `(get 'current-xlan 'value)` is a composite entity like e.g.

```
(lan-with-laptop: lan-ida-caslab dell-laptop-1)
```

where the first argument is the current LAN and the second argument is the current host. (The case of a detachable memory unit on stationary host remains to be documented.) Several attributes of the composite entity are derived from attributes of its two arguments. This value is set in ef `els-kb.`

For the attributes of these entities, please refer to the report on the agent LAN ontology.

These two entities should be reorganized as additional attributes of the current session entity i.e. the value of `*cursession*.`

9 Miscellaneous

9.1 Global Variables Used by the KRE Parser

The following global variables are used by the KRE parser. This design is in fact unfortunate and the parser shall be rewritten, but we document them here since they are actually in use. The user should not touch them.

`*S*` - Contains the string that is presently being parsed. (Bad design and bad variable name, admittedly. It was intended to be very temporary.)

`*stk*` - The current stack while parsing a recursively formed expression.

`*form*`

`*fun*`

`*nfun*`

The above three variables are used for temporary purposes during parsing.

`*oldtags*` - Obsolete.

`*latest-read-entity*` - As the parser reads an entityfile, it sets the variable to the entity whose description it is beginning to read, for each entity description in the file. This is used if there is an error condition when an entity-description is parsed.

`*R*` - A string containing the Return character.

`*N*` - A string containing the Newline character.

`*RN*` - A string containing the Return character followed by the Newline character.