KLONE Reference Manual
R. Brachman, E. Ciccarelli, N. Greenfeld, and M. Yonke

July 1978

DISTRIBUTION STATEMENT A
Approved for public release; Distribution Unlimited

DTIC ELECTION DEC 16 1982
KLONE Reference Manual

By

Ronald Brachman
Eugene Ciccarelli
Norton Greenfield
Martin Yonke

July 1978

Sponsored by
Defense Advanced Research Projects Agency (DOD)
ARPA Order No. 3175 (Amendment 1)
Monitored by Naval Electronic Systems Command
Under Contract #N00039-77-C-0598

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.
KLONE Reference Manual

This research was supported by the Advanced Research Projects Agency under ARPA Order #3175 and monitored by Naval Electronic Systems Command under Contract #NO0039-77-C-0398.

KLONE is being developed to be an epistemologically-explicit language for representing conceptual knowledge and structured inheritance; this manual provides user documentation for the current state of the INTERLISP implementation. Documented are: types of KLONE entities and relationships; procedural and data attachment; conceptual "meta-description" of KLONE entities; implementation naming conventions; all user-accessible KLONE primitives.
Preface

This manual is intended to serve two kinds of readers: the reader who is new to the KLONE implementation; and the reader who is familiar with KLONE, but who needs particulars of a function on occasion. Its organization accommodates three basic kinds of lookup: getting familiar with what functions are available; deciding which function to use for some task (i.e., to see at a glance which functions might apply); and finding details of a particular function. While the manual concentrates on the third type of lookup, section 3 lists KLONE functions grouped logically; together with the introductory sections, this should facilitate the first two kinds of lookup.

A few words about diagrams: The diagrams in this manual are intended to capture the flavor of the KLONE functions in their full generality, and not necessarily represent typical uses. For instance, if a function returns a list of values instead of a single value, the diagram for that function will try to show two things at once: the different kinds of values that the function deals with, and also illustrate a case for which a list rather than a single value is returned. (Sometimes it is not obvious why there should be several.) Controlling this yen for generality, however, is a conscious effort to keep the diagrams simple enough to use quickly. Please note in particular the list of diagram abbreviations in Appendix 2.

KLONE is currently in a preliminary stage of development. While new functions will appear, its authors expect the functionality described here to remain fairly stable.
Acknowledgments

The epistemology of Structured-Inheritance Networks (SI-Nets) embodied in KLONE was presented in Ron Brachman's Harvard PhD dissertation, reproduced as BBN Report No. 3605. He, Norton Greenfeld, and Martin Yonke are implementing KLONE; KLONE's surface interface was designed by them and Austin Henderson.

This manual was primarily organized and written by Eugene Ciccarelli; however Ron, Norton, and Martin contributed many comments and suggestions about the manual's structure and wording. Also, some parts of the text were drawn almost intact from [Brachman 78a,b].
List of Tables

Table 1. KLONE Object Types and Subtypes 6
Table 2. Relations between KLONE Objects 7
Table 3. Role Facet Defaults Through Inheritance 11
Table 4. Arguments to Attached Procedures 13
1. Introduction to KLONE

KLONE is a language designed for representing conceptual knowledge. This section briefly describes the basic entities of KLONE; for further discussion of the epistemological principles underlying Structured-Inheritance Networks (SI-Nets), see [Brachman 78a,b,c].

1.1. The KLONE Implementation

The preliminary KLONE implementation was designed to develop and test out functional properties -- it is not optimized for speed or space. The implementation is layered, and the underlying data structures can be replaced with no changes in the user level (in fact the current implementation is the third of this type). A goal of the current implementation is that the data structures are totally inaccessible through normal LISP means -- available instead only through the KLONE functions. This decision was prompted by the desire to ensure the completeness of the user interface, and this encapsulation technique is a means of forcing such completeness. The implementation is currently being used by projects on intelligent display management and natural language systems.

"Stable" versions of KLONE will reside in the <KLONE> directory on BBN-TENEXD, as KLONE.EXE. This directory will also contain other useful library functions. These files are accessible to all users, including anonymous FTP users. However, we would like to keep track of who is using KLONE and generally for what purpose, both for interest in the applicability of KLONE and to ensure that users get on the mailing list for future improvements; therefore, we would like anyone who uses KLONE to announce their presence by sending a message to KLONE@BBN-TENEXD.

Any complaints or suggestions should be mailed to KLONE@BBN-TENEXD, and are welcome.

We expect that over the next six months or so, the underlying structure of KLONE will be reimplemented to: (1) take less space (perhaps even have KLONE structures migrate between disk and core), (2) perhaps be more time-efficient, and (3) perhaps be optimized for implementation on LISP machines.

At some time in the future we also expect to have a two-dimensional graphical editor for KLONE structures, and a library of search and pattern-match functions.

Several issues are still unresolved, and therefore are left somewhat unmotivated
and mechanical in this manual. These include specification of the inner structure of SDs, the operation of validity checking, and the nature of Number and Modality facets. These issues should be resolved and lead to changes in the top level of KLONE with time.

1.2. Overview of KLONE Objects

Tables 1 and 2 below provide a schematic overview of the objects within KLONE and the diagram symbols used to represent them. KLONE has three general types of objects: Concepts, Roles, and Structural Descriptions (SDs). These have further subtypes as shown in Table 1. Table 2 lists the kinds of inter-object relations in KLONE; the table lists for each relation the two kinds of objects being related, and illustrates the schematic symbols used in diagrams throughout this manual. (The relations are generally some form of arrow.) A shorthand in the table diagrams (not appearing in other diagrams) is to show one or two kinds of objects in parentheses beside a different kind which is participating in some relation (arrow); this indicates that the parenthesized objects could also participate in the same kind of relationship. Finally, note that [] denotes a non-network entity (a LISP S-expression).

Table 1. KLONE Object Types and Subtypes

| Concept: |  
|----------|---
| Generic: | ![Generic Concept](image)
| Individual: | ![Individual Concept](image)
| ParaIndividual: | ![ParaIndividual Concept](image)

| Role: |  
|--------|---
| Generic: | ![Generic Role](image)
| Instance: | ![Instance Role](image)
| Coref: | ![Coref Role](image)

| Focus/SubFocus Chain (Indirect Role): | ![Focus/SubFocus Chain](image)

| SD: |  
|-----|---
| ![SD Symbol](image) |
Table 2. Relations between KLONE Objects

<table>
<thead>
<tr>
<th></th>
<th>Source:</th>
<th>Destination:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Intra-Concept:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RoleD</td>
<td>Generic Concept</td>
<td>Generic Role</td>
</tr>
<tr>
<td>RoleF</td>
<td>Concept</td>
<td>Instance or Coref Role</td>
</tr>
<tr>
<td>Structure</td>
<td>Generic Concept</td>
<td>SD</td>
</tr>
<tr>
<td>2) Intra-Role:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RoleName</td>
<td>Generic Role</td>
<td>[atom]</td>
</tr>
<tr>
<td>Facets:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V/R</td>
<td>Generic Role</td>
<td>Generic Concept</td>
</tr>
<tr>
<td>Number</td>
<td>Generic Role</td>
<td>[number or pair]</td>
</tr>
<tr>
<td>Modality</td>
<td>Generic Role</td>
<td>Obligatory, Inherent.</td>
</tr>
<tr>
<td>Value</td>
<td>Instance Role</td>
<td>Optional, Derivable</td>
</tr>
<tr>
<td>CorefValue</td>
<td>Coref Role</td>
<td>Individual Concept</td>
</tr>
<tr>
<td>Focus or SubFocus</td>
<td>Focus/SubFocus chain</td>
<td>Generic Role or Concept,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ParIndividual Concept,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Focus/SubFocus chain</td>
</tr>
<tr>
<td>3) Intra-SD:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Check</td>
<td>SD</td>
<td>ParIndividual Concept</td>
</tr>
<tr>
<td>Derive</td>
<td>SD</td>
<td>ParIndividual Concept</td>
</tr>
<tr>
<td>NonActive</td>
<td>SD</td>
<td>ParIndividual Concept</td>
</tr>
<tr>
<td>4) Inter-Concept:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SuperC</td>
<td>Generic Concept</td>
<td>Generic Concept</td>
</tr>
<tr>
<td>Individuates</td>
<td>Individual Concept</td>
<td>Generic Concept</td>
</tr>
<tr>
<td>ParIndividuates</td>
<td>ParIndividual Concept</td>
<td>Generic Concept</td>
</tr>
<tr>
<td>5) Inter-Role:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Satisfies</td>
<td>Instance Role</td>
<td>Generic Role</td>
</tr>
<tr>
<td>Modifies</td>
<td>Generic Role</td>
<td>Generic Role</td>
</tr>
<tr>
<td>Differentiates</td>
<td>Generic Role</td>
<td>Generic Role</td>
</tr>
<tr>
<td>CorefSatisfies</td>
<td>Coref Role</td>
<td>Generic Role</td>
</tr>
<tr>
<td>6) Inter-SD:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Preempts</td>
<td>SD</td>
<td>SD</td>
</tr>
<tr>
<td>7) Hooks:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MetaHook</td>
<td>Concept, Role, SD</td>
<td>Individual Concept</td>
</tr>
<tr>
<td>Interpretive Hook</td>
<td>Concept, Role, SD</td>
<td>[LISP function]</td>
</tr>
<tr>
<td>Datum [by tag]</td>
<td>Concept, Role, SD</td>
<td>[LISP form]</td>
</tr>
</tbody>
</table>

- 7 -
The diagram below schematically illustrates the internal structure of a single Concept:

![Diagram 1. Schematic Concept Structure](image)

1.3. Concepts

Concepts are the basic elements of KLONE; they are formal objects used to represent objects, attributes, and relationships of the domain being modelled. There are three types of Concept: Generic, Individual, and Parametric Individual. A Generic Concept implicitly represents a class of individuals - it is principally a description of a prototypical member of the class. An Individual Concept represents a specific object, relationship, etc. fitting a Generic Concept’s description — individuating the Generic Concept. The third type of Concept, the Parametric Individual (ParaIndividual) Concept, represents a type of existential and is discussed below in connection with SDs.

Each Concept has a name (a LISP atom); no two Concepts may have the same name. The primary use for Concept names is to allow readable identification of Concepts when network information is being printed.

1.4. Roles

Roles allow several Concepts to take part in the definition of another Concept; Roles are the conceptual subpieces of a Concept. The Roles represent the various kinds of generalized attributes, parts, etc. that things in the world (and therefore the Concepts modelling them) are considered to "have". There are two main kinds of Roles (a third is described in the following section):
A Concept's Generic Roles describe\(^1\) generalized attributes of a Generic Concept: they specify properties that are expected to be true of the ultimate fillers (Individual Concepts) of those attributes of the Concept.

An Instance Role is a binding of a particular Individual Concept to (1) a Generic Role it fills\(^2\) and (2) to the Individual Concept in which it fills that Role. In other words, the filler represents a choice of a particular world entity to serve as the value of the generalized attribute.

Roles may have names (LISP atoms). Unlike Concept names, Role names need not be unique: two Roles may have the same name, even if part of the same Concept.\(^3\) A Role may have several names through inheritance (see section 1.6).

A Generic Role's attribute description is provided by the Role's facets:

- **The V/R** ("value restriction") facet specifies a Generic Concept, which is a description that any filler must satisfy.

- **The Number** facet indicates the number of fillers of the particular Role to be expected. It may be either a single number or a pair specifying a range: e.g. \((n \, m)\) specifies \(n\) through \(m\) inclusive. Either element of the pair may be NIL to indicate "don't care", e.g. \((\text{NIL} \, n)\) for "at most \(n\".

- **The Modality** facet controls the action of individuation. Modalities are either single members of the list (Obligatory Inherent Optional), or two-element lists with one of those values as the first element, and a SubModality as the second member. At the moment, the only SubModality is Derivable. Individuation fails if any Obligatory Role is not filled. The Derivable SubModality indicates whether individuation is expected to deduce the filler from the structure of the Concept itself (in particular, from SDs). An Inherent Role indicates something that a Concept is considered to always "have", but it may be left unfilled without causing individuation to fail.

---

1. Hence the name "RoleD" for the relation between a Generic Concept and its Generic Roles.
2. Hence the name "RoleF" for the relation between a Concept (either Generic or Individual) and its Instance Roles.
3. This is one reason KLFindNamedRoles (q.v.) returns a list of Roles, rather than a single Role. The other reason is that Role names are inherited by SubRoles. See section 2.4.
1.5. Structural Descriptions (SDs)

The set of Structural Descriptions (SDs) for a Concept is the source of information about how its Role fillers interact with each other as part of the Concept's definition. Each SD is a set of relationships between two or more of the Concept's Roles. These relationships are expressed by ParaIndividual Concepts ("parametric Individual:") PICs. One may think of a PIC as a template for the construction of an individual relationship that will hold for a given individuator of the enclosing Concept and its particular fillers. The PIC therefore has CorefRoles which refer parametrically to Individual Concepts as their CorefValue ($V_C$). There are five kinds of CorefValues:

1. the enclosing Generic Concept, thus referring to the given individuator;
2. a Generic Role of the enclosing Concept, thus referring to the particular filler for that Role in the given individuator;
3. a Role of another ParaIndividual Concept in the same SD;
4. another PIC in the same SD, thus referring to that particular relationship; or
5. a Focus/SubFocus chain, which is a list of Generic Roles: ($R_1C, R_2C, ..., R_nC$). $R_iC$ must be a Role of the enclosing Concept, and every other $R_iC$ must be a Role of the value restriction of $R_{i-1}C$. (This allows reference to parts of parts of...)

1.6. Relations for Inheritance

Inheritance relations connect formal objects of the same type -- Concept to Concept, Role to Role, SD to SD -- and allow a Concept to inherit parts of other Concept's definitions: SDs, Roles, Role facets, or certain hooks. Inter-Concept relations may be thought of as "cables" of inter-Role and inter-SD inheritance relation "wires". There are two kinds of inter-Concept relations: Individuates connecting an Individual Concept to the Generic Concept it individuates; and SuperConcept, connecting one Generic Concept to another Generic Concept from which to inherit. The SubConcept is said to specialize the SuperConcept.

The inheritance relations within an in-Concept relation (the "wires" within a
"cable") allow parts of a Concept definition to be inherited intact (as if the
inheritor Concept had copies of the parts), or modified somewhat: a Role may be
modified (some of its facets replaced) by the Mods relation, a Role may be
differentiated into several subroles by the Diffs relation, and an SD may take the
place of a higher one by the Preempts relation.

A Generic Concept may have several SuperC relations to other Concepts. In
such a case, a SubRole may have several SuperRoles.

When a SuperRole has some facets when inherited by a SubRole, other facets
not mentioned may be inherited intact or may be replaced by new defaults, as
Table 3 specifies: 5

Table 3. Role Facet Defaults Through Inheritance

<table>
<thead>
<tr>
<th>Inheritance Relation</th>
<th>V/R</th>
<th>Modality</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>No parents:</td>
<td><strong>ANYTHING-</strong></td>
<td>Optional</td>
<td>1</td>
</tr>
<tr>
<td>1 Diffs:</td>
<td>(inherit)</td>
<td>Optional</td>
<td>1</td>
</tr>
<tr>
<td>1 Mods:</td>
<td>(inherit)</td>
<td>(inherit)</td>
<td>(inherit)</td>
</tr>
<tr>
<td>n Diffs:</td>
<td>(intersect)</td>
<td>Optional</td>
<td>1</td>
</tr>
<tr>
<td>n Mods:</td>
<td>(intersect)</td>
<td>(strongest)</td>
<td>(overlap)</td>
</tr>
<tr>
<td>n Mods + m Diffs:</td>
<td>(intersect)</td>
<td>Optional</td>
<td>1</td>
</tr>
</tbody>
</table>

The "overlap" of two Number Facets is simply the overlap of the Facets'
ranges. "Strongest" is defined using the ordering (strong to weak): Obligatory,
Inherent, Optional. And finally, V/R "intersection" is defined as requiring fillers to
be individuators of each V/R.

Role names are inherited by SubRoles, allowing any of the names of any
SuperRoles or any locally-specified names (see the description of KLAddRoleName)
to be used interchangeably.

5 There is a built-in Concept named "**ANYTHING**".

- 11 -
KLONE provides two kinds of "hooks" which can be used to attach various kinds of entities to KLONE objects in the SI-Net: MetaHooks to MetaDescriptions, wherein knowledge about knowledge is expressed in the same network language as the primary knowledge; and IHooks (Interpretive hooks), in which direct instructions to the interpreter are expressed in the language that implements the interpreter itself.

A MetaDescription of a Concept, Role, or SD is always expressed by an Individual Concept whose meaning is the entity as a Concept, Role, or SD. For instance, an ARCH Concept (a SubConcept of PHYSICALOBJECT, say) might have a MetaDescription of the Individual Concept of name "ARCHCONCEPT" that individuates the Generic Concept named "CONCEPT". This is illustrated in the diagram below:

The IHook allows a LISP procedure to be attached to a KLONE object and invoked at certain times when that object is being processed, as specified by a pair of either Before or After, and the name of a KLONE function: for example, (Before KLIndividuate) or (After KLVindicate) for Concepts; and (Before KLRemoveRole) or (Before KLSpecializeRole) for Roles.

For instance, a procedure attached to a Generic Concept by a (Before KLIndividuate) IHook will be run at the start of the KLIndividuate operation -- before the new Individual Concept is created. An (After KLIndividuate) procedure will be run after the new Individual Concept is created and established as an individuator, and its Roles successfully filled.
A procedure attached to an entity by an IHook is passed a single argument, which is either a single KLONE entity or a list of KLONE entities. The following are the currently available IHooks and the arguments that are passed; the “Entity” column indicates the kinds of KLONE entities to which the IHook may attach a procedure (in some cases there are more than one):

**Table 4. Arguments to Attached Procedures**

<table>
<thead>
<tr>
<th>IHook:</th>
<th>Entity</th>
<th>Argument passed:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Before KLAddCorefRole)</td>
<td>SuperRole</td>
<td>SuperRole</td>
</tr>
<tr>
<td>(After KLAddCorefRole)</td>
<td>SuperRole</td>
<td>SuperRole</td>
</tr>
<tr>
<td>(Before KLAddParalndlvidual)</td>
<td>GenericConcept</td>
<td>GenericConcept</td>
</tr>
<tr>
<td>(After KLAddParalndlvidual)</td>
<td>SD</td>
<td>SD</td>
</tr>
<tr>
<td>(Before KLChangeRoleName)</td>
<td>Role</td>
<td>Role</td>
</tr>
<tr>
<td>(After KLChangeRoleName)</td>
<td>Role</td>
<td>Role</td>
</tr>
<tr>
<td>(Before KLChangeRoleValue)</td>
<td>Role</td>
<td>Role</td>
</tr>
<tr>
<td>(After KLChangeRoleValue)</td>
<td>Role</td>
<td>Role</td>
</tr>
<tr>
<td>(Before KLDeriveRoles)</td>
<td>Ind. Concept</td>
<td>Ind. Concept</td>
</tr>
<tr>
<td>(After KLDeriveRoles)</td>
<td>Ind. Concept</td>
<td>Ind. Concept</td>
</tr>
<tr>
<td>(Before KLEestablishAsSatisfier)</td>
<td>SuperRole</td>
<td>&lt;SubRole SuperRole&gt;</td>
</tr>
<tr>
<td>(After KLEestablishAsSatisfier)</td>
<td>SubRole</td>
<td>&lt;SubRole SuperRole&gt;</td>
</tr>
<tr>
<td>(Before KLEestablishAsSpecializer)</td>
<td>SuperRole</td>
<td>&lt;SubRole SuperRole&gt;</td>
</tr>
<tr>
<td>(After KLEestablishAsSpecializer)</td>
<td>SubRole</td>
<td>&lt;SubRole SuperRole&gt;</td>
</tr>
<tr>
<td>(Before KLEestablishAsSubConcept)</td>
<td>SuperConc</td>
<td>&lt;SubConc SuperConc&gt;</td>
</tr>
<tr>
<td>(After KLEestablishAsSubConcept)</td>
<td>SubConc</td>
<td>&lt;SubConc SuperConc&gt;</td>
</tr>
<tr>
<td>(Before KLIndlvlduate)</td>
<td>GenericConcept</td>
<td>GenericConcept</td>
</tr>
<tr>
<td>(After KLIndlvlduate)</td>
<td>Ind. Concept</td>
<td>Ind. Concept</td>
</tr>
<tr>
<td>(Before KLRemoveRole)</td>
<td>Role</td>
<td>Role</td>
</tr>
<tr>
<td>(Before KLSatisfyRole)</td>
<td>GenericRole</td>
<td>GenericRole</td>
</tr>
<tr>
<td>(After KLSatisfyRole)</td>
<td>Ind. Role</td>
<td>Ind. Role</td>
</tr>
<tr>
<td>(Before KLSpecializeRole)</td>
<td>SuperRole</td>
<td>SuperRole</td>
</tr>
<tr>
<td>(After KLSpecializeRole)</td>
<td>NewSubRole</td>
<td>NewSubRole</td>
</tr>
</tbody>
</table>
2. Naming Conventions for KLONE Functions

KLONE is implemented as a set of INTERLISP functions for accessing and manipulating the KLONE data base. Each function guarantees structural integrity, and the set of functions together constitute the only possible access to the KLONE structures. This section discusses some conventions used in naming the KLONE functions.

2.1. Prefixes: 'KL', 'KLP', '#', 'KLZ'

There are several libraries of functions relating to KLONE. The use of name prefixes distinguishes between several of these libraries:

Each KLONE primitive has a name beginning with "KL", e.g. "KLFindRoles"; these are described in section 4. (There are also sets of internal functions with prefixes "#" and "KLZ". They are not meant to be user-accessible, and are not described in this manual.)

The KLONEPRINT library contains several functions for printing the structure of a Concept, Role, or entire network in a readable text format. These functions are prefixed by "KLP".

In addition, there is a set of functions in KLONELIBRARY built on top of the KLONE primitives. These are less-well-established, and likely to change. These and other higher-level libraries do not have prefixed names.

2.2. Upper and Lower Case

Each KLONE function name contains both upper and lower case letters (e.g. "KLFindAllRoles"), as contrasted with INTERLISP function names which are all upper case (e.g. "CONS").

2.3. 'Find' versus 'Get'

There are many KLONE primitives with names of the form "KLFindx" or "KLGetx". The difference is in the use of the inheritance paths: KLFindx will search inheritance paths, while KLGetx will remain local to the immediate relations of the Concept specified (or implied) by the argument list.

6 Up or down as appropriate.
2. Naming Conventions for KLONE Functions

For example: (KLGetSuperConcepts C) will return a list of the immediate SuperConcepts of C, while (KLFindSuperConcepts C) will return a list of all Concepts above C in the inheritance paths (i.e. following SuperC relations) — all the Concepts from which C inherits Roles, SDs, and hooks.

2.4. 'Named'

Some KLONE functions have names of the form "KLFindNamedx" or "KLGetNamedx", as opposed to "KLFindx" or "KLGetx". The former, "Named" versions, allow the caller to refer to Concepts or Roles by name, rather than by the object itself. Functions using Role names take a Concept as an argument to provide some context (the inheritance paths for that Concept) in which to evaluate the (non-unique) Role name. These functions generally return the object or set of objects that match the name.

For example, consider the following KLONE structure:

```
  "foo"
  C1  R1
  "foo"
  R2

  M:O

  C2
  R3
```

Here are some different function calls and their values:

- (KLFindRoles C1 R1) -> (R1)
- (KLFindRoles C2 R1) -> (R3)
- (KLFindRoles C1 R2) -> (R2)
- (KLFindRoles C2 R2) -> (R2)
- (KLFindNamedRoles C1 'foo) -> (R1 R2)
- (KLFindNamedRoles C2 'foo) -> (R3 R2)
- (KLFindNamedRoles C2 'bar) -> (R3)
2.5. 'Add/Remove' versus 'Establish/DisEstablish'

In a KLONE function name, "EstablishAs" indicates that a relation is being created between two already existing objects. "Add", on the other hand, indicates that a relation is being created between an existing object and a new object to be created. The "add" primitives return the newly-created object.

Similarly, "DisEstablishAs" destroys a relation between two objects, but they are still valid KLONE entities. "Remove" destroys not only a relation, but one of the objects as well (and any other relations that object has).

2.6. Compound Function Verbs

Several function names contain verbs which indicate compound operations, both creating new objects and establishing relations. These verbs are: "Derive" (add Instance Roles, establish them as satisfiers), "Individuate" (create an Individual Concept, establish it as an individuator), "Satisfy" (add an Instance Role, establish it as a satisfier), "SpecializeConcept" (create a Generic Concept, establish its SuperC relation), and "SpecializeRole" (add a Generic Role, establish it as specializer).
Appendix 1. KLONE Keywords

The following list shows (in boldface) the keywords used to indicate certain KLONE object and relation types, used in arguments or returned values of KLONE functions:

**General Types:**
- 'Concept
- 'Role
- 'SD

**Concept Types:**
- 'Generic
- 'Individual
- 'Paraindividual

**Role Types:**
- 'Generic
- 'Instance
- 'Coref

**Role Specialization Types:**
- 'Diffs or 'Mods (for Generic Roles),
- 'Satisfies (for Instance Roles),
- 'CorefSatisfies (for Coref Roles).

**Facet Types:**
- 'Number
- 'Modality
- 'V/R

**Modality Types:**
- 'Obligatory
- 'Inherent
- 'Optional

**Modality Subtype:**
- 'Derivable

**Validity Types:**
- 'NotTested
- 'Valid
- 'NotValid
- 'Can'tTell
Appendix 2. Diagram Abbreviations

The following abbreviations are used in the diagrams of this manual:

- ****: Number facet
- CG: Generic Concept
- CI: Individual Concept
- C: Concept
- Co: Coref
- D: Differentiates (Diffs)
- FT: Facet Type
- FV: Facet Value
- M: Modifies (Mods)
- NC: Concept Name
- NR: Role Name
- N: Name
- CP: Coref Individual Concept
- RCCo: Coref Role
- RG: Generic Role
- RI: Instance Role
- R: Role
- SCo: Coref Satisfies
- S: Satisfies (Sats)
- SD: Structural Description
- Rs: SuperRole
- VC: Coref Value
- V: Value
Appendix 3. KLONE Network of KLONE Objects

The following is a text file produced by KLSaveNet for the network partially (only its SuperC structure) diagramed below. Besides illustrating the format used in such files, it shows the classification of most of the KLONE object types, and the reader may find it helpful in determining which KLONE objects fit the types called for by KLONE function descriptions. (If a function description calls for some type, any object at or below that type in the SuperC structure here may be used.) The Role structure described by this file shows, for instance, that in general, Concepts have Roles; Generic Concepts have Generic Roles and Instance Roles, while Individual Concepts have only Instance Roles.
'CorefSatisfies
  type: Individual
  individuates: Keyword
  has extranet tags:
    Validity     Valid

'Derivable
  type: Individual
  individuates: Keyword
  is role value of Keyword{SubModality}
  has extranet tags:
    Validity     Valid

'Diffs
  type: Individual
  individuates: Keyword
  has extranet tags:
    Validity     Valid

'Inherent
  type: Individual
  individuates: Keyword
  is role value of Keyword{InherentModality}
  has extranet tags:
    Validity     Valid

'Modality
  type: Individual
  individuates: Keyword
  is role value of Keyword{ModalityFacetType}
  has extranet tags:
    Validity     Valid

'Mods
  type: Individual
  individuates: Keyword
  has extranet tags:
    Validity     Valid
Appendix 3

'Number
   type: Individual
   individuates: Keyword
   is role value of Keyword{NumberFacetType}
   has extranet tags:
      Validity       Valid

'Obligatory
   type: Individual
   individuates: Keyword
   is role value of Keyword{ObligatoryModality}
   has extranet tags:
      Validity       Valid

'Optional
   type: Individual
   individuates: Keyword
   is role value of Keyword{OptionalModality}
   has extranet tags:
      Validity       Valid

'Satisfies
   type: Individual
   individuates: Keyword
   has extranet tags:
      Validity       Valid

'V/R
   type: Individual
   individuates: Keyword
   is role value of Keyword{VRFacetType}
   has extranet tags:
      Validity       Valid

''ANYTHING''
   type: Generic
   has extranet tags:
      Validity       NotTested
anything
  type: Generic
  has specializers: KLONEEntity, LISP-Sexp
  in role facet V/R of Value{RoleFacet}
  has extranet tags:
    Validity       Valid

CheckPIC
  type: Generic
  specializes: ParaIndividualConcept
  has extranet tags:
    Validity       NotTested

Concept
  type: Generic
  specializes: KLONEEntity
  has specializers: GenericConcept, IndividualConcept,
                    ParaIndividualConcept
  has extranet tags:
    Validity       NotTested

ConceptName
  type: Generic
  specializes: LISP-Atom
  has extranet tags:
    Validity       NotTested
CorefRole
type: Generic
specializes: Role

roles:
  Value
    facet V/R = InstanceRole
  Value
    facet V/R = Focus/SubFocusChain
  Value
    facet V/R = ParaIndividualConcept
  Value
    facet V/R = CorefRole
  Value
    facet V/R = GenericRole
  Value
    facet V/R = GenericConcept

in role facet V/R of Value(CorefRole)
has attached procedures:
(After KLIndividuate)
  EnsureOnlyOneValueRoleFilled

has extranet tags:
  Validity NotTested

CorefSatisfiesType
type: Generic
specializes: RoleSpecializationType

has extranet tags:
  Validity NotTested

DerivePIC
type: Generic
specializes: ParaIndividualConcept

has extranet tags:
  Validity NotTested

DiffsType
type: Generic
specializes: RoleSpecializationType

has extranet tags:
  Validity NotTested
Focus/SubFocusChain
type: Generic
specializes: LISP-List
in role facet V/R of Value(CorefRole)
has extranet tags:
  Validity NotTested

GenericConcept
type: Generic
specializes: Concept
in role facet V/R of Value(V/RFacet), Value(CorefRole)
has extranet tags:
  Validity NotTested

GenericRole
type: Generic
specializes: Role
in role facet V/R of Value(CorefRole)
has extranet tags:
  Validity NotTested

IndividualConcept
type: Generic
specializes: Concept
has specializer: MetaDescription
in role facet V/R of Value(InstanceRole)
has extranet tags:
  Validity NotTested

InherentModality
type: Generic
roles:
  Keyword
    value = 'Inherent'
in role facet V/R of Type(Modality)
has extranet tags:
  Validity NotTested
InstanceRole
  type: Generic
  specializes: Role
  roles:
    Value
      facet V/R = IndividualConcept
    Value
      facet V/R = LISP-Sexp
  in role facet V/R of Value(CorefRole)
  has attached procedures:
    (After KIIndividuate)
      EnsureOnlyOneValueRoleFilled
  has extranet tags:
    Validity: Valid

Keyword
  type: Generic
  specializes: LISP-Atom
  has individuators: 'Optional, 'Obligatory, 'Inherent, 'Derivable,
    'Satisfies, 'CorefSatisfies, 'Mods, 'Diffs, 'Number,
    'V/R, 'Modality
  in role facet V/R of Keyword(CorefType)
  has extranet tags:
    Validity: Valid

KLONEEntity
  type: Generic
  specializes: anything
  has specializers: Role, Concept, SD
  has extranet tags:
    Validity: Valid

LISP-Atom
  type: Generic
  specializes: LISP-Sexp
  has specializers: SubModality, Keyword, LISP-Number, ConceptName,
    RoleName
  has extranet tags:
    Validity: Valid
LISP-List
  type: Generic
  specializes: LISP-Sexp
  has specializer: Focus/SubFocusChain
  has extranet tags:
    Validity        NotTested

LISP-Number
  type: Generic
  specializes: LISP-Atom
  has extranet tags:
    Validity        NotTested

LISP-Sexp
  type: Generic
  specializes: anything
  has specializers: Modality, NumberOrNumberPair, LISP-List, LISP-Atom
  in role facet V/R of Value{InstanceRole}
  has extranet tags:
    Validity        Valid

MetaDescription
  type: Generic
  specializes: IndividualConcept
  has extranet tags:
    Validity        NotTested
Appendix 3  

Modality

type: Generic
specializes: LISP-Sexp
roles:
  Type
    facet V/R = InherentModality
  Type
    facet V/R = OptionalModality
  Type
    facet V/R = ObligatoryModality
  SubModality
    facet V/R = SubModality
in role facet V/R of Value{ModalityFacet}
has attached procedures:
  (After KLIndividuate)
    EnsureOnlyOneTypeRoleFilled
has extranet tags:
  Validity        NotTested

ModalityFacet

type: Generic
specializes: RoleFacet
roles:
  Value
    Mods Value{RoleFacet}
    facet V/R = Modality
  Type
    Mods Type{RoleFacet}
    facet V/R = ModalityFacetType
has extranet tags:
  Validity        NotTested

ModalityFacetType

type: Generic
specializes: RoleFacetType
roles:
  Keyword
    Satisfies Keyword{RoleFacetType}
    value = 'Modality
in role facet V/R of Type{ModalityFacet}
has extranet tags:
  Validity        NotTested
ModsType
type: Generic
specializes: RoleSpecializationType
has extranet tags:
  Validity     NotTested

NonActivePIC
type: Generic
specializes: ParaIndividualConcept
has extranet tags:
  Validity     NotTested

NumberFacet
type: Generic
specializes: RoleFacet
roles:
  Value
    Mods Value{RoleFacet}
    facet V/R = NumberOrNumberPair
  Type
    Mods Type{RoleFacet}
    facet V/R = NumberFacet|Type
has extranet tags:
  Validity     NotTested

NumberFacet|Type
type: Generic
specializes: RoleFacet|Type
roles:
  Keyword
    Satisfies Keyword{RoleFacet|Type}
    value = 'Number
in role facet V/R of Type{NumberFacet}
has extranet tags:
  Validity     NotTested

NumberOrNumberPair
type: Generic
specializes: LISP-Sexp
in role facet V/R of Value{NumberFacet}
has extranet tags:
  Validity     NotTested
ObligatoryModality
type: Generic
roles:
  Keyword
  value = 'Obligatory
  in role facet V/R of Type[Modality]
has extranet tags:
  Validity  NotTested

OptionalModality
type: Generic
roles:
  Keyword
  value = 'Optional
  in role facet V/R of Type[Modality]
has extranet tags:
  Validity  NotTested

ParaIndividualConcept
type: Generic
specializes: Concept
has specializers: NonActivePIC, DerivePIC, CheckPIC
in role facet V/R of Value{CorefRole}
has extranet tags:
  Validity  NotTested

Role
type: Generic
specializes: KLONEEntity
has specializers: CorefRole, InstanceRole, GenericRole
has extranet tags:
  Validity  Valid
RoleFacet
type: Generic
has specializers: NumberFacet, V/RFacet, ModalityFacet
roles:
   Value
       facet V/R = anything
       is modified by Value{ModalityFacet}, Value{NumberFacet}, Value{V/RFacet}
   Type
       facet V/R = RoleFacetType
       is modified by Type{ModalityFacet}, Type{NumberFacet}, Type{V/RFacet}
has extranet tags:
   Validity NotTested

RoleFacetType
type: Generic
has specializers: V/RFacetType, NumberFacetType, ModalityFacetType
roles:
   Keyword
       facet V/R = Keyword
       is satisfied by Keyword{V/RFacetType}, Keyword{NumberFacetType}, Keyword{ModalityFacetType}
in role facet V/R of Type{RoleFacet}
has extranet tags:
   Validity NotTested

RoleName
type: Generic
specializes: LISP-Atom
has extranet tags:
   Validity NotTested

RoleSpecializationType
type: Generic
has specializers: SatisfiesType, CorefSatisfiesType, ModsType, DiffsType
has extranet tags:
   Validity NotTested
SatisfiesType
  type: Generic
  specializes: RoleSpecializationType
  has extranet tags:
    Validity      NotTested

SD
  type: Generic
  specializes: KLONEEntity
  has extranet tags:
    Validity      NotTested

SubModality
  type: Generic
  specializes: LISP-Atom
  roles:
    Keyword
      value = 'Derivable
    in role facet V/R of SubModality{Modality}
  has extranet tags:
    Validity      NotTested

V/RFacet
  type: Generic
  specializes: RoleFacet
  roles:
    Value
      Mods Value{RoleFacet}
      facet V/R = GenericConcept
    Type
      Mods Type{RoleFacet}
      facet V/R = V/RFacetType
  has extranet tags:
    Validity      NotTested
V/RFacetType
  type: Generic
  specializes: RoleFacetType
  roles:
    Keyword
      Satisfies Keyword(RoleFacetType)
      value = 'V/R
  in role facet V/R of Type(V/RFacet)
  has extranet tags:
    Validity   NotTested
Appendix 4. Logical Index of KLONE Functions

Functions here are grouped by two properties: first, by the kind of object the function primarily concerns (Concept, Role, SD, or hook); second, by the general use of the SI-Net: retrieving information, adding new information to the network, removing information. But first, a few that don't fit those categories:

GENERAL

KLGetType [KLONEEntity] \rightarrow \text{a pair of a KLONETYPEName and a subtype.}
KLLoadNet [File]
KLSaveNet [File; NoTxtFileFlg]
KLPPrintAllConcepts [File; ImmediateOnlyFlg] \rightarrow \text{anything.}
ppc or
PPC ConceptName-or-SExpEvaluatingToAConcept (LISPXMACRGC)
KLPPrintConcept [Concept; ImmediateOnlyFlg] \rightarrow \text{a Concept.}
CONCEPTS

MISCELLANEOUS:

KLGetType [KLONEEntity] \rightarrow a pair of a KLONETYPEName and a subtype.
KLGetValidityState [Concept] \rightarrow a ValidityState.
KLMapSubConcepts [GenericConcept; Function; IndividuatorFlag]
KLMapSuperConcepts [Concept; Function]
KLOValidate [Concept; BreakFlag]

PREDICATES:

KLOConceptP [Anything] \rightarrow either NIL or a Concept.
KLOGenericConceptP [Anything] \rightarrow either a Generic Concept or NIL.
KLOIndividualConceptP [Anything] \rightarrow either an Individual Concept or NIL.
KLIsConceptDescendantP [SubConcept; GenericConcept] \rightarrow either NIL or a Concept.
KLOTrueInSomeAncestor [Concept; Predicate] \rightarrow either NIL or a Concept.

RETRIEVAL:

KLOFindIndividuators [GenericConcept] \rightarrow a set of Individual Concepts.
KLOFindSuperConcepts [Concept] \rightarrow a set of Generic Concepts.
KLOGetConceptName [Concept] \rightarrow a ConceptName.
KLOGetNamedConcept [ConceptName] \rightarrow either a Concept or NIL.
KLOGetSuperConcepts [Concept] \rightarrow either a set of Generic Concepts or NIL.

ADDITION:

KLOCreateConcept [ConceptName; ConceptType] \rightarrow a Concept.
KLOEstablishAsIndividuator [IndividualConcept; SuperConcept]
KLOEstablishAsSubConcept [GenericConcept; SuperConcept]
KLOIndividuate [GenericConcept; ConceptName; GenericRoles&Fillers] \rightarrow an Individual Concept.
KLOSpecializeConcept [GenericConcept; II ConceptName; WiringList] \rightarrow a Generic Concept.

REMOVAL:

KLODeleteConcept [Concept]
KLODisEstablishAsIndividuator [IndividualConcept; SuperConcept]
KLODisEstablishAsSubConcept [SubConcept; SuperConcept]
ROLES

MISCELLANEOUS:

KLGetType  [KLONEEntity] -> a pair of a KLONETypeName and a subtype.

PREDICATES:

KLCheckNumberForSingleRole  [Concept: InstanceRole] -> a Boolean.
KLCheckNumberForSingleRole  [Concept: InstanceRole] -> a Boolean.
KLFillIsRoleInSubConceptP  [IndividualConcept; RoleName; GenericConcept] -> a Boolean.
KLGenericRoleP  [Anything] -> either NIL or a Generic Role.
KLInstanceRoleP  [Anything] -> either NIL or an Instance Role.
KLIsInheritedRoleP  [Concept; Role] -> either NIL or a Role.
KLIsRoleDescendantP  [SubRole; GenericRole] -> either NIL or a Role.
KLRoleP  [Anything] -> either NIL or a Role.

RETRIEVAL:

KLFindAllRoles  [Concept] -> a set of Roles.
KLFindDifferentiableRoles  [Concept; RoleName] -> a set of Generic Roles.
KLFindFacetOfRole  [GenericRole; FacetType] -> a set of FacetValues.
KLFindNamedGenericRoles  [GenericConcept; RoleName] -> either a set of Generic Roles or NIL.
KLFindNamedInstanceRoles  [Concept; RoleName] -> either a set of Instance Roles or NIL.
KLFindNamesOfRole  [Role] -> either NIL or a set of RoleNames.
KLFindParentsOfRole  [Role] -> a set of Generic Roles.
KLFindRoles  [Concept; Role] -> a set of Roles.
KLFindRoleValues  [Concept; Role] -> a set of RoleValues.
KLFindSatisfiableRoles  [Concept; RoleName] -> a set of Generic Roles.
KLFindSatisfiersOfRole  [Concept;GenericRole] -> either a set of Instance Roles or NIL.
KLFindSpecializersOfRole  [GenericConcept; GenericRole]
KLGetConceptOfRole  [Role] -> a Concept.
KLGetRoleFacetInverses  [Concept; FacetType] -> either a set of Generic Roles or NIL.
KLGetRoleValue  [InstanceRole] -> a RoleValue.
KLGetRoleValueInverses  [IndividualConcept] -> either a set of Instance Roles or NIL.
ADDITION:

KLAddInstanceRole [Concept; RoleValue] -> an Instance Role.
KLAddRole [GenericConcept; Facet&ValuePairs] -> a Generic Role.
KLAddRoleName [Role; RoleName] -> a RoleName.
KLDeriveRoles [IndividualConcept]
KLEstablishAsSatisfier [InstanceRole; SuperRole]
KLEstablishAsSpecializer [GenericRole; SuperRole; RoleSpecializationType]
KLSatisfyRole [Concept; GenericRole; RoleValue] -> an Instance Role.
KLSpecializeRole [GenericConcept; GenericRole; RoleSpecializationType; Facet&ValuePairs] -> a Generic Role.

REMOVAL:

KLRemoveRole [Role]
KLRemoveRoleName [Role]

CHANGE:

KLCChangeRoleFacet [GenericRole; FacetType; NewFacetValue] -> a FacetValue.
KLCChangeRoleName [Role; NewRoleName] -> a RoleName.
KLCChangeRoleValue [InstanceRole; NewRoleValue] -> a RoleValue.
STRUCTURAL DESCRIPTIONS (SDs)

MISCELLANEOUS:

- **KLGetType** [KLONEEntity] → a pair of a KLONETYPEName and a subtype.

PREDICATES:

- **KLCorefRoleP** [Anything] → either NIL or a Coref Role.
- **KLPARIndividuapP** [Anything] → either NIL or a PAraIndividuap Concept.
- **KLSDP** [Anything] → either NIL or a SD.

RETRIEVAL:

- **KLFindCorefSpecializersOfRole** [PARIndividuap; GenericRole] → either a set of Coref Roles or NIL.
- **KLFindNamedCorefRoles** [PARindividuap; RoleName]
- **KLFindParalndividuators** [GenericConcept] → a set of PAraIndividuap Concepts.
- **KLFindSDs** [Concept] → a set of SDs.
- **KLFindValueForCoreflndividuator** [CorefRole; IndividualConcept] → a RoleValue.
- **KLFfindConceptOfSD** [SD] → a Concept.
- **KLFindRoleCoref** [CorefRole] → a Coref Value.
- **KLFindRoleCoreflnverses** [CorefValue] → either a set of Coref Roles or NIL.
- **KLFindSDDerives** [SD] → either a set of PARindividuap Concepts or NIL.
- **KLFindSDOfParalndividual** [PARindividuap] → a SD.

ADDITION:

- **KLAAddCorefRole** [PARIndividuap; CorefValue, SuperRole] → a Coref Role.
- **KLAAddParalndividual** [SD; ConceptName; GenericConcept; WiringList] → a PARindividuap Concept.
- **KLAAddSD** [GenericConcept] → a SD.
- **KLEstablishAsPreemptor** [PreemptingSD; SuperSD]
- **KLEstablishAsSDDerive** [PARindividuap]
- **KLEstablishAsSDDerive** [PARindividuap]
REMOVAL:

KLDisEstablishAsSDDCheck [ParaIndividual]
KLDisEstablishAsSDDerive [ParaIndividual]
KLRemoveParaIndividual [ParaIndividual]
KLRemoveSD [SD]

CHANGE:

KL changeRoleCoref [CorefRole; NewCorefValue] → a CorefValue.
Hooks

**RETRIEVAL:**

KLGetData  [KLONEEntity; Tag] \(\rightarrow\) either a set of anythings or NIL.
KLGetDefaultValue [GenericRole] \(\rightarrow\) either a RoleValue or NIL.
KLGetMarks  [KLONEEntity] \(\rightarrow\) a set of anythings.
KLGetMetaDescriptions [KLONEEntity] \(\rightarrow\) either a set of Individual Concepts or NIL.
KLGetMetaDescriptionInverse [DescriptiveIndividualConcept] \(\rightarrow\) either a KLONEEntity or NIL.

**ADDITION:**

KLAttachDatum  [KLONEEntity; Tag; Datum]
KLAddDefault [GenericRole; RoleValue] \(\rightarrow\) a RoleValue.
KLAttachProcedure [KLONEEntity; IHook; Procedure]
KLEstablishAsMetaDescription [BaseKLONEEntity; DescriptiveIndividualConcept]
KLMarkEntity [KLONEEntity; Anything]

**REMOVAL:**

KLDisEstablishAsMetaDescription [DescriptiveIndividualConcept]
KLRemoveAllData [KLONEEntity; Tag]
KLRemoveAllProcedures [KLONEEntity; IHook]
KLRemoveDatum [KLONEEntity; Tag; DatumOrIndex]
KLRemoveDefault [GenericRole]
KLRemoveProcedure [KLONEEntity; IHook; ProcedureOrIndex]
KLUunMarkEntity [KLONEEntity; Anything]

**CHANGE:**

KLChangeDefaultValue [GenericRole; NewRoleValue] \(\rightarrow\) a RoleValue.
Appendix 5. KLONE Function Descriptions

Each function description in this section contains the following: an automatically-generated comment about the action performed and argument/returned-value types (produced from the KLONE source), and for most non-trivial functions a diagram.
KLAddCorefRole [ParaIndividual;CorefValue;SuperRole]

description: Adds a Coref Role to an existing Concept, expected to be a ParaIndividual. SuperRole is a Role of the Concept being ParaIndividualated, and CorefValue is the binding of that Role in the context of ParaIndividual. CorefValues are: 1: Roles of the Concept in whose SD the ParaIndividual lies, 2: that Concept itself, 3: a Coref Role of some other ParaIndividual in the same SD as the ParaIndividual, 4: some other ParaIndividual in the same SD, 5: a list of Roles such that the first is a Role of the enclosing Concept and each thereafter is a Role of the V/R of the preceding one -- this is a Focus/SubFocus chain, and 6: NIL, which means me -- the particular Individual Concept that has this SD inherited and is invoking it.

parameters: ParaIndividual
  type: a ParaIndividual Concept.
  meaning: Concept to which the new Role is to be added.
CorefValue
  type: a CorefValue.
  meaning: Source of the value of the Role when an individuator is finally constructed.
  If a Role, then must be of the Concept in whose SD ParaIndividual appears, or a Role of another ParaIndividual in the same SD; if a Concept, must be another ParaIndividual, in same SD as ParaIndividual, or the enclosing Concept itself; if a list, must be a list of Roles accessible by walking down Roles from the enclosing Concept -- i.e., Focus/SubFocus; can also be NIL.
SuperRole
  type: a Generic Role.
  meaning: Role that is being CorefSatisfied.
  value:
  type: a Coref Role.
  meaning: the new Coref Role that is created as part of ParaIndividual.

\[
[C_{pi}; V_{co}; R_{sup}] \rightarrow R_{co}
\]
KLAddDefault [GenericRole;RoleValue]

**description:** Creates a meta-description that represents the fact that the default value for GenericRole is RoleValue. Will not work if a default value has already been specified for Role. GenericRole and RoleValue are both meta-described, and the meta-descriptions are tied together in an individuator of the Generic Concept, DEFAULT. The interpreter is expected to know about DEFAULT.

**parameters:**
- **GenericRole**
  - type: a Generic Role.
  - meaning: the Role whose value is to be defaulted if it is not explicitly specified.
- **RoleValue**
  - type: a RoleValue.
  - meaning: the default value.

**KLAddInstanceRole [Concept;RoleValue]**

**description:** Adds an Instance Role -- whose value is to be bound to RoleValue -- to an existing Concept. If this new instance Role is to satisfy some Generic Role in a SuperConcept, use KL SatisfyRole.

**parameters:**
- **Concept**
  - type: a Concept.
  - meaning: Concept to which the Role is being added.
- **RoleValue**
  - type: a RoleValue.
  - meaning: value of Role.

**value:**
- **type:** an Instance Role.
- **meaning:** the new Instance Role that is created as a part of Concept.
**KLAddParaIndividual** ([SD;ConceptName;GenericConcept;WiringList])

**description:** Creates a new ParaIndividual Concept, and places it within a Structural Description of an existing Concept. GenericConcept is the Concept that the new one ParaIndividuates, and the wiring list specifies how each of its Roles are to be matched and filled. The ParaIndividual added to the SD is not considered initially to be in the Check or Derive parts.

**parameters:**
- **SD**
  - type: a SD.
  - meaning: SD to which ParaIndividual is added.
- **ConceptName(optional)**
  - type: a ConceptName.
  - meaning: name of new ParaIndividual.
  - restrictions: (KLGetNamedConcept ConceptName)
- **GenericConcept**
  - type: a Generic Concept.
  - meaning: Concept being ParaIndividuated.
- **WiringList**
  - type: a set of a triple of a Generic Role, either 'CorefSatisfies' or 'Satisfies', and a Value.
  - meaning: the set of Role-Role connections for constructing the ParaIndividual's parts.

**value:**
- type: a ParaIndividual Concept.
- meaning: the newly created ParaIndividual.
**KLAddRole** [GenericConcept; Facet&ValuePairs]

*description:* Adds a Generic Role to an existing Concept. The new Role is not considered initially to have any relationship to any other Roles. The V/R, Number, and Modality are specified in the Facet/Value pair list. To include a RoleName, use KLAddRoleName. To establish a relationship between the new Role and some SuperRole, use KLSpecializeRole.

*parameters:*

- **GenericConcept**
  - type: a Generic Concept.

- **Facet&ValuePairs**
  - type: a set of pairs of a FacetType and a RoleValue.
  - meaning: each pair contains a FacetType name and a value for that Facet. The Facets are added one at a time; if any failure occurs, only the Facet that causes that failure is aborted — all others continue.

- **value:**
  - type: a Generic Role.
  - meaning: the new Role created and added to Concept.

\[
\text{CG; } \langle \langle FT_1, FV_1 \rangle, \langle FT_2, FV_2 \rangle \rangle \rightarrow RG
\]
KLAddRoleName [Role:RoleName]

description: Adds a RoleName to any kind of Role. Cannot use if a RoleName already exists at that Role.

parameters:
- Role: type: a Role.
- RoleName: type: a RoleName.

value:
- RoleName: type: a RoleName.
- value: the new RoleName added to Role.

KLAddSD [GenericConcept]

description: Adds a blank SD to an existing Concept.

parameters:
- GenericConcept: type: a Generic Concept.
- SD: a SD.

value:
- value: the new SD created as a part of GenericConcept.

KLAttachDatum [KLONEEntity:Tag:Datum]

description: Attaches a non-network piece of data to a Concept, Role, or SD. The data is kept in a list keyed by a Tag.

parameters:
- KLONEEntity: type: a KLONEEntity.
- Tag: meaning: Concept, Role, or SD to which Datum is attached.
- Datum: type: anything.

value:
- Tag: type: anything.
- Datum: type: anything.
KLAttachProcedure [KLONEEntity;IHook;Procedure]
description: Attaches a procedure, which will be invoked by the KLONE interpreter, to an entity. The IHook designates the situation in which the procedure will be invoked.

parameters: KLONEEntity type: a KLONEEntity.
IHook type: a pair of either 'Before or 'After and a KLONEFunction.
Procedure type: either a LISPFunctionName or a LISPFunction.

KLChangeDefaultValue [GenericRole;NewRoleValue]
description: Changes the default value for a Role by finding the meta-description that represents the defaulting, and changing the value of the DefaultValue Role. If none has yet been specified, this creates a new default structure.

parameters: GenericRole type: a Generic Role.
NewRoleValue type: a RoleValue.

value:

returns the new default value for Role.

KLChangeRoleCoref [CorefRole;NewCorefValue]
description: Deletes the old CorefValue of a Coref Role, and replaces it with a new one. Does not activate Delete or Add IHooks.

parameters: CorefRole type: a Coref Role.
NewCorefValue type: a CorefValue.

value:

the CorefValue of Role after the change.
KLChangeRoleFacet [GenericRole;FacetType;NewFacetValue]

description: Deletes a Role Facet and adds a new value. Does not invoke Delete or Add IHooks.

parameters:  
GenericRole type: a Generic Role.
FacetType type: a FacetType.
NewFacetValue type: a FacetValue.

value:  
type: a FacetValue.
meaning: a legal value for the particular FacetType being changed.

KLChangeRoleName [Role;NewRoleName]

description: Changes the RoleName associated with a Role by deleting the old one -- if it exists -- and replacing it with the new one specified as argument. Does not invoke Delete or Add IHooks.

parameters:  
Role type: a Role.
NewRoleName type: a RoleName.

value:  
type: a RoleName.
meaning: the new RoleName for Role.

KLChangeRoleValue [InstanceRole;NewRoleValue]

description: Changes the value bound to an Instance Role to be the new one specified as an argument.

parameters:  
InstanceRole type: an Instance Role.
NewRoleValue type: a RoleValue.

value:  
type: a RoleValue.
meaning: the new VAL for Role.
KLCheckNumberForSingleRole [Concept;InstanceRole]

description: Takes a Concept and a single Instance Role of that Concept, and finds all Number restrictions that apply to the Role. It then counts up the Values that are considered to exist at Concept (inherited or otherwise) and applies the Number predicates until one fails, or all succeed. Returns T if no violations occur.

parameters:  
Concept type: a Concept.  
InstanceRole type: an Instance Role.  
value: 

KLConceptP [Anything]

description: Predicate for checking if an entity is a Concept. Returns NIL if it is not, or the entity itself if it is.

parameters:  
Anything type: anything.  
value: 

KLCorefRoleP [Anything]

description: Predicate to test whether some datum is a Coref Role. If so, the Role itself is returned; if not, NIL is the result.

parameters:  
Anything type: anything.  
value:  

meaning: NIL means that the entity was not a Coref Role; otherwise the Role passed in as argument is returned.
**KLCreateConcept [ConceptName;ConceptType]**

*description:* Creates a new Concept, with name and type as specified. If no name is specified, one will be supplied. If no type is specified, Generic is the default.

*parameters:*
- **ConceptName** (optional)
  - type: a ConceptName.
  - restrictions: (KLGetNamedConcept ConceptName)

- **ConceptType**
  - type: either 'Individual', 'Generic', or NIL.
  - value: a Concept.

**KLDelbeteConcept [Concept]**

*description:* Removes a Concept from any inter-Concept chains, and then deletes it. If the Concept is ParaIndividuated, then it will not be deleted, since it is required as part of some SD. Or, if it appears as a V/R or VAL, it will remain.

*parameters:*
- **Concept**
  - type: either a Generic Concept or an Individual Concept.

**KLDeriveRoles [IndividualConcept]**

*description:* Derives fillers for all Derivable Roles of an Individual Concept that have not yet been filled in, and can be derived.

*parameters:*
- **IndividualConcept**
  - type: an Individual Concept.

**KLDisEstablishAsIndividuator [IndividualConcept;SuperConcept]**

*description:* IndividualConcept ceases to be an individuator of SuperConcept.

*parameters:*
- **IndividualConcept**
  - type: an Individual Concept.

- **SuperConcept**
  - type: a Generic Concept.
**KLDisEstablishAsMetaDescription** *(DescriptiveIndividualConcept)*

description: Removes the relationship that specifies that an Individual Concept meta-describes a KLONE Entity.

parameters: DescriptiveIndividualConcept
  
type: an Individual Concept.
  
meaning: an Individual Concept which is a meta-description of some other Entity.

**KLDisEstablishAsPreemptor** *(SD)*

description: Removes the preemptive relationship between SD and an SD of a SuperConcept, wherein SD had previously overridden the one in the SuperConcept. The SD that was previously preempted will now be inherited intact by the Concept in which SD appears.

parameters: SD
  
type: a SD.
  
meaning: an SD that will no longer preempt one inherited from a SuperConcept of the Concept in which SD appears.

**KLDisEstablishAsSDCheck** *(ParaIndividual)*

description: Removes a ParaIndividual from the Check part of an SD; leaves it as still part of the SD.

parameters: ParaIndividual
  
type: a ParaIndividual Concept.

**KLDisEstablishAsSDDerive** *(ParaIndividual)*

description: Removes a ParaIndividual from the Derive part of an SD; leaves it as still part of the SD.

parameters: ParaIndividual
  
type: a ParaIndividual Concept.
KLDisEstablishAsSubConcept \([\text{SubConcept};\text{SuperConcept}]\)

description: Removes the SubConcept cable connecting SubConcept and SuperConcept. All inter-Role connections dependent on the SubConcept relation are broken.

parameters: 
- SubConcept type: a Generic Concept.
- SuperConcept type: a Generic Concept.

KLEstablishAsIndividuator \([\text{IndividualConcept};\text{SuperConcept}]\)

description: Makes an Individual Concept an Individuator of a Generic one. Does not establish any Role correspondences. NOTE: an Individual Concept can individuate only ONE Generic.

parameters: 
- IndividualConcept type: an Individual Concept.
- SuperConcept type: a Generic Concept.

KLEstablishAsMetaDescription \([\text{BaseKLONEEntity};\text{DescriptiveIndividualConcept}]\)

description: Establishes an Individual Concept to be a meta-description of some other entity, so that the entity can be talked about AS A DESCRIPTION.

parameters: 
- BaseKLONEEntity type: a KLONEEntity.
- DescriptiveIndividualConcept type: an Individual Concept.
- meaning: the description of BaseKLONEEntity AS A DESCRIPTION.
KLEstablishAsPreemptor [PreemptingSD:SuperSD]

description: Establishes a relationship between two SD's wherein one (the PreemptingSD) overrides the other. SuperSD would normally be inherited intact by the Concept in which PreemptingSD appears, but ceases to have effect after the establishment.

parameter
PreemptingSD type: a SD. meaning: the SD that will override an inherited one.
SuperSD type: a SD. meaning: the SD that will no longer be in effect.

KLEstablishAsSatisfier [InstanceRole:SuperRole]

description: Establishes an existing Instance Role as a satisfier of some SuperRole inherited by the Concept of the Instance Role. No previous relation between the two Role should exist.

parameters:
InstanceRole type: an Instance Role.
SuperRole type: a Generic Role.

KLEstablishAsSDCheck [ParaIndividual]

description: Takes a ParaIndividual that is part of an SD and puts it in the Check part of that SD, so that it will be used as a predicate when the enclosing Concept is individuated.

parameters:
ParaIndividual type: a ParaIndividual Concept.

KLEstablishAsSDDerive [ParaIndividual]

description: Takes a ParaIndividual that is part of an SD and puts it in the Derive part of that SD, so that it will be used as a function to derive new ...role fillers when the enclosing Concept is individuated.

parameters:
ParaIndividual type: a ParaIndividual Concept.
KLEstablishAsSpecializer [GenericRole;SuperRole;RoleSpecializationType]

description: Establishes an existing Role as a specializer of some SuperRole inherited by the Concept of the SubRole. No previous relation between the two Roles should exist.

parameters:
- GenericRole type: a Generic Role.
- SuperRole type: a Generic Role.
- RoleSpecializationType type: a RoleSpecializationType.

KLEstablishAsSubConcept [GenericConcept;SuperConcept]

description: Makes a Concept a SubConcept of a second one, provided that both are Generic. If the SubConcept already specializes other Concepts, and some of its already inherited Roles have the same source as Roles of the new SuperConcept, new Roles will be created to merge the multiple inheritance paths. This avoids duplication of identical Roles.

parameters:
- GenericConcept type: a Generic Concept.
- SuperConcept type: the Concept which will be subordinate. a Generic Concept.

KLFillsRoleInSubConceptP [IndividualConcept;RoleName;GenericConcept]

description: Tests to see whether IndividualConcept fills a Role named by RoleName in some descendant of GenericConcept.

parameters:
- IndividualConcept type: an Individual Concept.
- RoleName type: a RoleName.
- GenericConcept type: a Generic Concept.

value: type: a Boolean.
KLFindAllRoles [Concept]

description: This function produces the complete set of Roles that are considered to be applicable at Concept. They are the lowest ones down all inheritance chains that are effective at the Concept. That is, for each Role, the function finds the most immediately accessible one. Some may be instance Roles -- inherited from arbitrarily far up -- and some may actually appear directly at the Concept from which the search is initiated. They all act equally well as governing Roles for the Concept, and their facets and values can be obtained with KLFindFacetOfRole, and KLGetRoleValue.

parameters: Concept

value: Concept

KLFindCorefSpecializersOfRole [Paraindividual;GenericRole]

description: From a given Role, finds any Coref Roles that Coref Specialize it in Paraindividual. Follows down chains of other specialization types if necessary.

parameters: Paraindividual

value: Paraindividual

GenericRole

value: either a set of Coref Roles or NIL.
KLFindDifferentiableRoles [Concept;RoleName]

description: Finds the set of Roles inherited by a Concept that are available for differentiation. If RoleName is specified, only Roles with that RoleName are considered.

parameters: Concept type: a Concept.
               meaning: the Concept whose inherited Roles are being considered for differentiation.

          RoleName {optional} type: a RoleName.
               meaning: if this argument is given, the function considers only Roles whose RoleName matches it.

value: type: a set of Generic Roles.
       meaning: the set of Roles available at Concept for differentiation.

KLFindFacetOfRole [GenericRole;FacetType]

description: This is the general FIND routine for all role facets, incorporating both full inheritance and default values. Given a Generic Role and a Facet Type, it returns a list of the applicable values for that Facet. This will be a singleton list for Facets other than V/R. A conjoined V/R is returned as a list of the individual V/R's.

parameters: GenericRole type: a Generic Role.
            FacetType type: a FacetType.
            value: type: a set of FacetValues.
               meaning: NOTE: this function returns a list in all cases.

KLFindIndividuators [GenericConcept]

description: Finds all of the individuating Concepts of a given Generic Concept. This includes all of its immediate individuators, and those of all of its SubConcepts.

parameters: GenericConcept type: a Generic Concept.
            value: type: a set of Individual Concepts.

\[
[C_G] \rightarrow (C_{I_1}, C_{I_2}, C_{I_3}, C_{I_4})
\]
KLFindNamedCorefRoles [ParaIndividual;RoleName]

description: Finds all Coref Roles of ParaIndividual that are considered to be named by RoleName.

parameters: ParaIndividual type: a ParaIndividual Concept.
RoleName type: a RoleName.

KLFindNamedGenericRoles [GenericConcept;RoleName]

description: Finds all Generic Roles inherited by GenericConcept that are named by RoleName. Works like KLFindNamedRoles, except that it finds only Generic Roles.

parameters: GenericConcept type: a Generic Concept.
RoleName type: a RoleName.

KLFindNamedInstanceRoles [Concept;RoleName]

description: Finds all Instance Roles inherited by Concept that are named by RoleName. Works like KLFindNamedRoles, except that it finds only Instance Roles.

parameters: Concept type: a Concept.
RoleName type: a RoleName.

KLFindNamedRoles [Concept;RoleName]

description: Finds all Roles inherited by a given Concept whose RoleName is that specified as argument. Returns a list of these – each Role inheritance chair is represented in this list by its lowest applicable Role (in this respect, it works like KLFindAllRoles.)

parameters: Concept type: a Concept.
RoleName type: a RoleName.

value: a set of Roles.
**KLFindNamesOfRole [Role]**

**description:** Retrieves the set of RoleNames applicable to Role. More than one may be in force because of Role differentiation or multiple SuperConcepts.

**parameters:**

- **value:** Role
  - **type:** a Role.
  - **type:** either NIL or a set of RoleNames.

\[
[R_1] \rightarrow (R_{one} R_{four}) \\
[R_2] \rightarrow (R_{two} R_{three}) \\
[R_3] \rightarrow (R_{three}) \\
[R_4] \rightarrow (R_{four})
\]

**KLFindParaIndividuators [GenericConcept]**

**description:** Returns a list of all of the ParaIndividuators of a Concept. These include any ParaIndividuators of SubConcepts of the Concept, etc.

**parameters:**

- **value:** GenericConcept
  - **type:** a Generic Concept.
  - **type:** a set of ParaIndividual Concepts.

\[
[C_G] \rightarrow (C_{PI}, C_{PI2})
\]
**KLFindParentsOfRole [Role]**

description: Constructs a list of the complete set of Roles that the argument specializes or satisfies, directly or by inheritance.

parameters: Role type: a Role.
value: type: a set of Generic Roles.

\[
[R] 
\rightarrow (R_0, R_2, R_3)
\]

**KLFindRoles [Concept;Role]**

description: For a Concept and a Role that that Concept inherits from somewhere above in a SuperConcept chain, finds the lowest Roles that govern the Concept. There can be more than one by virtue of Role differentiation or multiple SuperConcepts. Works like KLFindNamedRoles, except that it starts from a Role, not a name.

parameters: Concept type: a Concept.
value: Role type: a Role.

\[
[C; R_1] \rightarrow (R_1) \\
[C; R_2] \rightarrow (R_2) \\
[C; R_3] \rightarrow (R_3) \\
[C; R_4] \rightarrow (R, R_7) \\
[C; R_5] \rightarrow (R_5) \\
[C; R_6] \rightarrow (R_6) \\
[C; R_7] \rightarrow (R_7)
\]
KLFindRoleValues [Concept;Role]

description: Finds all values that satisfy a given Role at a given Concept. The Role has to be one that the Concept inherits.

parameters: Concept type: a Concept.
Role type: a Role.

value: type: a set of RoleValues.
KLFindSatisfiableRoles [Concept;RoleName]

description:  This function finds all of the Roles inherited by a Concept that are available for satisfying. Any Generic Role that does not have its Number facet satiated may be considered for filling. If a RoleName is specified as second argument, only Roles with that name will be considered.

parameters:  Concept  type: a Concept.  meaning: this is the Concept at which a Role might be attempted to be satisfied.

RoleName{optional}  type: a RoleName.  meaning: if this argument is specified, the function finds only the satisfiable Roles that have it as RoleName.

value:  type: a set of Generic Roles.  meaning: the set of Roles available for satisfaction at Concept.
**Appendix 5**

**KLFindSatisfiersOfRole [Concept:GenericRole]**

*description:* Finds all of the Instance Roles inheritable by Concept considered to be Satisfiers of the one specified as argument. Includes those down Role inheritance chains.

*parameters:*

- **Concept**
  - type: a Concept.
  - meaning: the context in which the Instance Roles are to be found.
  - value: a Generic Role.

- **GenericRole**
  - type: either a set of Instance Roles or NIL.

**KLFindSDs [Concept]**

*description:* Returns the complete set of SDs applicable at a Concept.

*parameters:*

- **Concept**
  - type: a Concept.
  - value: a set of SDs.

**Diagram:**

- [Concept]
  - SD1
  - SD2
  - SD3
  - SD4
  - SD5

- [Concept; R1] \(\rightarrow (R_{I1}, R_{I2}, R_{I3})\)
- [Concept; R2] \(\rightarrow (R_{I2}, R_{I3})\)
- [Concept; R3] \(\rightarrow (R_{I1}, R_{I2}, R_{I3})\)
**KLFindSpecializersOfRole** [GenericConcept;GenericRole]

*description:* Finds all Generic Roles that are inherited by GenericConcept and are specializers of GenericRole. Works like KLFindRoles, except that it finds only Generic Roles.

*parameters:*
- **GenericConcept**
  - *type:* a Generic Concept.
- **GenericRole**
  - *type:* a Generic Role.

**KLFindSubConcepts** [GenericConcept]

*description:* Returns a list of all Concepts considered to be SubConcepts of the argument. Includes both direct SubConcepts, and those included by transitivity of the SubConcept relationship.

*parameters:*
- **GenericConcept**
  - *type:* a Generic Concept.

*value:* a set of Generic Concepts.

```
[CG1] => (CG2)
[CG3] => (CG2 CG4)
```

**KLFindSuperConcepts** [Concept]

*description:* Gathers a list of all of the SuperConcepts of a Concept. This includes immediate SuperConcepts as well as those by transitivity.

*parameters:*
- **Concept**
  - *type:* a Concept.

*value:* a set of Generic Concepts.

```
[CG] => (CG1 CG2 CG3 CG4)
[CI] => (CG1 CG2 CG3 CG4)
```

(see diagram above)
KLFindValueForCorefInIndividuator [CorefRole; IndividualConcept]

description: Finds the value in a particular Individual Concept that is indicated by a
Coref Role in one of its inherited SD's. That is, uses IndividualConcept as
the parameter for a ParIndividuator, and evaluates a coreference in that
case.

parameters: 
CorefRole type: meaning: 
meaning: a Coref Role.

IndividualConcept type: meaning: 
meaning: some Coref Role in an inherited SD of
IndividualConcept.

value: IndividualConcept type: meaning: 
meaning: an Individual Concept.

a RoleValue.

the meaning of the coreference in the
context of IndividualConcept.

KLGenericConceptP [Anything]

description: Predicate to test whether an item is a Generic Concept.

parameters: 
Anything type: meaning: 
meaning: anything.

value: 
meaning: either a Generic Concept or NIL.

returns the argument if it is a Generic
Concept, NIL otherwise.

KLGenericRoleP [Anything]

description: Returns its argument if it is a Role, and is Generic.

parameters: 
Anything type: meaning: 
meaning: anything.

value: 
meaning: either NIL or a Generic Role.

if not NIL, the Role passed in as
argument.

KLGetConceptName [Concept]

description: Returns the name of a Concept. NOTE: Concept names are unique.

parameters: 
Concept type: meaning: 
meaning: a Concept.

value: 
meaning: a ConceptName.
KLGetConceptOfRole [Role]

description: Returns the Concept of which Role is a Role. A Role is an immediate part of only one Concept.

parameters: Role
value: type: a Role.

KLGetConceptOfSD [SD]

description: Returns the Concept of which SD is an SD. An SD is an immediate part of only one Concept.

parameters: SD
value: type: a SD.

KLGetData [KLONEEntity;Tag]

description: Returns the list of attached data that is attached to a Concept, Role, or SD by Tag. Elements of the list are not interpreted as network entities.

parameters: KLONEEntity
Tag
value: type: a KLONEEntity.
type: anything.
meaning: either a set of anythings or NIL.

KLGetDefaultValue [GenericRole]

description: Locates and returns the particular value that GenericRole is expected to have if no value is explicitly specified. Defaults are implemented as meta-descriptions, and this function goes to the meta-level to find an individuator of the Concept DEFAULT which applies to GenericRole.

parameters: GenericRole
value: type: a Generic Role.
type: either a RoleValue or NIL.
KLGetMarks [KLONEEntity]

description: Retrieves the complete set of marks resident at a KLONE Concept, Role, or SD.

parameters:

KLONEEntity

type: a KLONEEntity.

type: a set of anythings.

meaning: the set of marks found at KLONEEntity.

KLGetMetaDescriptionInverse [DescriptiveIndividualConcept]

description: Returns the Concept, Role, or SD of which the input Concept is a MetaDescription. NOTE: an Individual Concept meta-describes at most one base-layer entity.

parameters:

DescriptiveIndividualConcept

type: an Individual Concept.

type: either a KLONEEntity or NIL.

KLGetMetaDescriptions [KLONEEntity]

description: Returns a list of all MetaDescriptions of an KLONEEntity (i.e. all Individual Concepts attached by a MetaHook).

parameters:

KLONEEntity

type: a KLONEEntity.

type: either a set of Individual Concepts or NIL.

KLGetNamedConcept [ConceptName]

description: Returns a Concept whose name is specified as argument. Will always be unique.

parameters:

ConceptName

type: a ConceptName.

type: either a Concept or NIL.
**KLGetRoleCoref [CorefRole]**

*description:* Given a Coref Role, return its Coref Value -- a Role, Concept, or Individual.

*parameters:*
- **value:** CorefRole  
  *type:* a Coref Role.

**KLGetRoleCorefInverses [CorefValue]**

*description:* Retrieves all Coref Roles to which the value is bound as CorefValue.

*parameters:*
- **CorefValue**  
  *type:* a CorefValue.
  *meaning:* the Concept or Role which is the CorefValue of the Roles to be found.
  *type:* either a set of Coref Roles or NIL.

**KLGetRoleFacetInverses [Concept;FacetType]**

*description:* Retrieves all Generic Roles to which the Concept is bound as a Facet's value.

*parameters:*
- **Concept**  
  *type:* a Concept.
- **FacetType**  
  *type:* a Facetype.

*value:*
- **FacetType**  
  *type:* either a set of Generic Roles or NIL.
**KLGetRoleValue [InstanceRole]**

*description:* Given a single Instance Role, returns its Value.

*parameters:*
- **InstanceRole**
  - type: an Instance Role.
  - type: a RoleValue.

**KLGetRoleValueInverses [IndividualConcept]**

*description:* Retrieves all Instance Roles to which the argument is bound as Value.

*parameters:*
- **IndividualConcept**
  - type: an Individual Concept.
  - meaning: the Individual Concept which is the Value of the Roles to be found.
  - type: either a set of Instance Roles or NIL.

![Diagram](image)

\[
[C_I] \rightarrow (R_{I_1}, R_{I_2})
\]

**KLGetSDChecks [SD]**

*description:* Returns the list of Paraiindividuals in the Check part of an SD.

*parameters:*
- **SD**
  - type: a SD.
  - type: either a set of Paraiindividual Concepts or NIL.

**KLGetSDDerives [SD]**

*description:* Returns the list of Paraiindividuals in the Derive part of an SD.

*parameters:*
- **SD**
  - type: a SD.
  - type: either a set of Paraiindividual Concepts or NIL.
**KLGetSDOfParaIndividual** (ParaIndividual)

description: Finds the SD in which a particular ParaIndividual Concept is situated. The SD is unique for a given ParaIndividual.

parameters: ParaIndividual

description: a ParaIndividual Concept.

value: type:

type:

description: a SD.

**KLGetSuperConcepts** (Concept)

description: Returns a list of the immediate SuperConcepts of a Concept (i.e. only those directly tied to the Concept).

parameters: Concept

description: a Concept.

value: type:

type:

description: either a set of Generic Concepts or NIL.
**KLGetType** [KLONEEntity]

*description:* Returns a two-element list, the first element of which is a general type (Concept, Role, or SD), and the second of which is the subtype (Individual, Generic, etc.) of an KLONEEntity.

*parameters:* KLONEEntity type: a KLONEEntity.

*value:* a pair of a KلونTypeName and a subtype.

**KLGetValidityState** [Concept]

*description:* Returns the value that KLONE thinks of as the validity state of a Concept. May not have been tested, may have been tested and succeeded, failed, or not been able to determine.

*parameters:* Concept type: a Concept.

*value:* a ValidityState.
KLIndividualConceptP [Anything]

description: Predicate to test whether an item is an Individual Concept.

parameters: Anything type: anything.

value: Anything type: either an Individual Concept or NIL.

meaning: returns the argument if it is an Individual Concept, NIL otherwise.

KLIndividuate [GenericConcept;ConceptName;GenericRoles&Fillers]

description: This function creates a new Individual Concept which is an individuator of the Concept supplied as first argument. It does so by mapping the Roles of Concept onto new Instance Roles of the new Concept, using the values supplied in the third argument (the new Concept is named by the second argument). Once it fills all Roles that are specified in the call, the function tries to validate the Individual Concept with only those Roles filled. If it succeeds, all Roles that are Derivable from the ones already specified are derived and filled. Then, the Individual Concept is revalidated. Note that (Before KLIndividuate) attached procedures are run as soon as the new Concept is first created, and the (After KLIndividuate) procedures are run only after the individuation is successfully completed and validated.

parameters: GenericConcept type: a Generic Concept.

ConceptName {optional} type: a ConceptName.

GenericRoles&Fillers type: a set of pairs of a Generic Role and a RoleValue.

value: type: an Individual Concept.
KLInstanceRoleP [Anything]

description: Predicate to check if an item is an Instance Role. Returns the Role if so, otherwise returns NIL.

parameters: Anything

type:

value: thing.

type: either NIL or an Instance Role.

meaning: if non-NIL, the Role passed in as argument.

KLIsConceptDescendantP [SubConcept;GenericConcept]

description: Predicate to see if one Concept lies on the SuperC chain of another. Works faster than using KLFindSubConcepts and MEMB, since finishes when the right Concept is found.

parameters: SubConcept

type: a Concept.

meaning: a Concept which is being tested to be a descendant of GenericConcept.

GenericConcept

type: a Generic Concept.

meaning: a purported SuperConcept of SubConcept.

value: either NIL or a Concept.

meaning: the SubConcept, if it is a descendant of the other.

KLIsInheritedRoleP [Concept;Role]

description: Checks to see if a Role is part of the inheritance of a Concept. This predicate is useful for checking whether some manipulation is legal on a Role.

parameters: Concept

type: a Concept.

meaning: a Role.

Role

type: either NIL or a Role.

meaning: the Role passed in as argument, if it is inherited by Concept; otherwise NIL.
KLIsRoleDescendantP [SubRole;GenericRole]

description: Predicate to test whether a Role is descended by a chain of specializations from some other Role.

parameters: SubRole type: a Role.
meaning: the Role which is expected to be the descendant.

GenericRole type: a Generic Role.
meaning: the Role expected to be the ancestor.

value: type: either NIL or a Role.
meaning: NIL if SubRole is not a descendant of SuperRole; otherwise, SubRole is returned.

KLLoadNet [File]

description: Loads a KLONE network from File assuming it was saved by KLSaveNet.

parameters: File type: a File.
KLMapSubConcepts [GenericConcept;Function;IndividuatorFlg]

description:  This function applies the Function passed in to each Concept considered to be a SubConcept of GenericConcept. It is different than KLFindSubConcepts in that it does not first create a list, and then walk it, but applies the function as it walks the hierarchy. The Function is not applied to GenericConcept. NOTE: Function should be allowed to apply to a Concept more than once without adverse effects, since some Concepts will be traversed more than once. IndividuatorFlg, if T, will cause individuators as well as SubConcepts to be walked.

parameters:

GenericConcept  

type:  a Generic Concept.

meaning:  this is the Concept from which the SubConcept walk is initiated. The Function is NOT applied to this Concept.

Function  

type:  a Function.

meaning:  this function is applied to each Concept considered to be a SubConcept of Concept.

IndividuatorFlg  

type:  a Boolean.

meaning:  if non-NIL, will cause individuators as well as SubConcepts to be walked.
**KLMapSuperConcepts** [Concept;Function]

*description:* This function applies the Function passed in to each Concept considered to be a SuperConcept of Concept. It is different than KLFindSuperConcepts in that it does not first create a list, and then walk it, but applies the function as it walks the hierarchy. The Function is not applied to Concept. NOTE: Function should be allowed to apply to a Concept more than once without adverse effects, since some Concepts will be traversed more than once.

*parameters:*
- **Concept**
  *type:* a Concept.
  *meaning:* this is the Concept from which the SuperConcept walk is initiated. The Function is NOT applied to this Concept.
- **Function**
  *type:* a Function.
  *meaning:* this function is applied to each Concept considered to be a SuperConcept of Concept.

**KLMarkEntity** [KLONEEntity;Anything]

*description:* Attaches a mark to a KLONE entity. The mark can be arbitrary LISP and is never interpreted. If the same mark already occurs at the entity, it is not duplicated.

*parameters:*
- **KLONEEntity**
  *type:* a KLONEEntity.
- **Anything**
  *type:* anything.

**KLParaIndividualP** [Anything]

*description:* Predicate that tests whether a datum is a ParaIndividual or not. Returns the datum if so, otherwise NIL.

*parameters:*
- **Anything**
  *type:* anything.
  *value:* either NIL or a ParaIndividual Concept.
  *meaning:* NIL if Anything is not a ParaIndividual; otherwise returns Anything.
Appendix 5

KLPpco1 [LISPXLINIE]

description: Interprets the ppc history and break command. Prints a Concept

KLPPrintAllConcepts [File:ImmediateOnlyFig]

description: prints all Concepts in the network on a file, in user-readable form. ImmediateOnlyFig non-NIL inhibits all inheritance.

parameters: File type: anything.
            ImmediateOnlyFig type: a Boolean.

value: OrderFn type: a FunctionName.
       type: anything.

KLPPrintConcept [Concept:ImmediateOnlyFig]

description: Prints a user-understandable description of a Concept on the primary output file. If ImmediateOnlyFig is T, then prints only those aspects of the Concept that are immediately attached to it -- i.e. not inherited. This function prints the following information about the Concept -- what other Concepts it is related to, and how; all Roles in which it participates as a Facet or a Value; all of its own Role descriptions and instances; its SDs and their parts; meta-descriptions of the Concept, and if any, things meta-described by the Concept; and attached procedures and data. -- There is also a LISPXMACRO called PPC which takes a concept name or S-Expression which evaluates to a concept, and an ImmediateOnlyFig, and calls KLPPrintConcept with those parameters. PPC attempts to do spelling correction on the concept name.

parameters: Concept type: a Concept.
            ImmediateOnlyFig type: a Boolean.

value: type: a Concept.
**KLPreemptSD** [GenericConcept:SD]

**description:** Adds a new SD to GenericConcept that will override SD, whereas SD would normally be expected to be inherited directly by GenericConcept.

**parameters:**
- **GenericConcept**
  - type: a Generic Concept.
  - meaning: a Concept that would normally inherit SD, if it weren't preempted.
- **SD**
  - type: a SD.
  - meaning: the SD being preempted.
- **value:** a SD.
  - type: the new SD added to GenericConcept.

**KLRemoveAllData** [KLONEEntity:Tag]

**description:** Removes the entire list of data attached to a Concept, Role, or SD by Tag. Mostly a convenience function, since can be achieved with a loop and KLRemoveDatum; however, this works faster.

**parameters:**
- **KLONEEntity**
  - type: a KLONEEntity.
- **Tag**
  - type: anything.

**KLRemoveAllProcedures** [KLONEEntity:IHook]

**description:** Removes the entire list of Procedures attached to a Concept, Role, or SD by IHook. Mostly a convenience function, since can be achieved with a loop and KLRemoveProcedure; however, this works faster.

**parameters:**
- **KLONEEntity**
  - type: a KLONEEntity.
- **IHook**
  - type: a pair of either 'Before or 'After and a KLONEFunction.
KLRemoveDatum [KLONEEntity; Tag; DatumOrIndex]

description: Removes a single Datum from a list indexed by Tag. That is, the data attached to a Concept, Role, or SD by a Tag is stored in a list, and this function removes an element from that list. The element is specified by an argument which is EQUAL to the element, or by an index into the list -- specified by * followed by an integer.

parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>KLONEEntity</td>
<td>a KLONEEntity</td>
<td>the Concept, Role, or SD to which the datum is attached.</td>
</tr>
<tr>
<td>Tag</td>
<td>anything</td>
<td>the Tag by which the datum is attached to the entity.</td>
</tr>
<tr>
<td>DatumOrIndex</td>
<td>either a *index or anything</td>
<td>if a * followed by an integer, removes the integer'th element from the list. If anything else, removes the element which is EQUAL.</td>
</tr>
</tbody>
</table>

KLRemoveDefault [GenericRole]

description: Removes any meta-structure that represents the defaulting of GenericRole.

parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenericRole</td>
<td>a Generic Role</td>
<td></td>
</tr>
</tbody>
</table>

KLRemoveParaIndividual [ParaIndividual]

description: Removes a ParaIndividual from an SD, and throws away all of its connections to parts of the Concept it was in.

parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ParaIndividual</td>
<td>a ParaIndividual Concept</td>
<td></td>
</tr>
</tbody>
</table>
**KLRemoveProcedure** [KLONEEntity; IHook; ProcedureOrIndex]

*description:* Removes a single Procedure from a list indexed by IHook. That is, the procedures attached to a Concept, Role, or SD by an IHook are stored in a list, and this function removes an element from that list. The element is specified by an argument which is EQUAL to the element, or by an index into the list -- specified by # followed by an integer.

*parameters:*

- **KLONEEntity** type: a KLONEEntity.
  meaning: the Concept, Role, or SD to which the Procedure is attached.

- **IHook** type: a pair of either 'Before or 'After and a KLONEFunction.
  meaning: the IHook by which the Procedure is attached to the entity.

- **ProcedureOrIndex** type: either a #index, a LISPFunctionName, or a LISPFunction.
  meaning: if a # followed by an integer, removes the integer'th element from the list. If anything else, removes the element which is EQUAL.

---

**KLRemoveRole** [Role]

*description:* Removes and throws away a Role of a Concept. All ties to parent Roles, specializers, and satisfiers are removed.

*parameters:*

- **Role** type: a Role.
**KLRemoveRoleName** [Role]

*description:* Removes the RoleName from a Role. Does nothing if the Role doesn't have a name.

*parameters:* Role  type:  a Role.

**KLRemoveSD** [SD]

*description:* Removes an SD from a Concept, throwing away any ParaIndividuals that it comprises.

*parameters:* SD  type:  a SD.

**KLRoleP** [Anything]

*description:* Returns Anything if it is a Role, otherwise returns NIL.

*parameters:* Anything  type:  anything.

*value:*  type:  either NIL or a Role.

*meaning:*  if non-NIL, returns the Role passed in as argument.

**KLSatisfyRole** [Concept;GenericRole;RoleValue]

*description:* Creates a new Instance Role that is a satisfier of a Generic Role inherited from a SuperConcept. Concept is the Concept at which the Role is to be satisfied (often an Individual Concept) , and Value is the filler. GenericRole must be inheritable from some SuperConcept of Concept.

*parameters:* Concept  type:  a Concept.

GenericRole  type:  a Generic Role.

RoleValue  type:  a RoleValue.

*value:*  type:  an Instance Role.
**KLSaveNet [File:NoTxtFileFlag]**

description: Saves a loadable form of the KLONE net on File. Also creates a text version unless NoTxtFileFlag is T.

parameters:
- File type: a File.
- NoTxtFileFlag type: a Boolean.
  meaning: if non-NIL, stops the function from producing a text version of the network.

**KLSDFP [Anything]**

description: Predicate to test whether an item is an SD. If so, the item is returned; if not, NIL is returned.

parameters:
- Anything type: anything.
  value: either NIL or a SD.
  meaning: if non-NIL, returns the SD passed in as argument.
KLSpecializeConcept [GenericConcept;ConceptName;WiringList]

description: Creates a SubConcept of GenericConcept according to the list of
c connctions specified in WiringList. WiringList is used to specify what
new Roles, either Instance or Generic, to create at the new SubConcept,
and what their Facets should be. The new Concept is named by
ConceptName, if specified (otherwise a name is generated).

parameters:  

  GenericConcept  
    type: a Generic Concept.
    meaning: the SuperConcept of the Concept
             being created.
  ConceptName{optional}  
    type: a ConceptName.
    meaning: a set of a triple of a Generic Role, 
either 'Satisfies, 'Diffs, or 'Mods, and 
either a RoleValue or a set of pairs of 
a FacetType and a FacetValue.
  WiringList  
    type: a correspondence between a Generic 
          Role to be inherited by the new 
          Concept, a specialization type, and 
either a RoleValue (in the case of 
          Satisfies) or a set of 
          Facet&ValuePairs.
    meaning: a Generic Concept.
    value: the newly created SubConcept of 
           GenericConcept.
**KLSpecializeRole**

**[GenericConcept;GenericRole;RoleSpecializationType;Facet&ValuePairs]**

**description:** Creates a modification or differentiation of a Role at Concept. The new Role is a Generic Role, some of the definition of which is inherited from the Role it specializes (GenericRole). The fourth argument specifies any new values for facets of the Role.

**parameters:**

- **GenericConcept**  
  type: a Generic Concept.

- **GenericRole**  
  type: a Generic Role.

- **RoleSpecializationType**  
  type: a RoleSpecializationType.

- **Facet&ValuePairs**  
  type: a set of pairs of a FacetCode and a FacetValue.

**value:**  

- **type:** a Generic Role.
**KLTrueInSomeAncestor** [Concept;Predicate]

**description:** This function takes a Concept and applies a function to it and its SuperConcepts, until that function returns some non-NIL value. If there is no ancestor of Concept for which Function returns non-NIL, this function returns NIL, otherwise, it returns the Concept which causes Predicate to be non-NIL.

**parameters:**
- Concept
- Predicate

**value:**
- type: a Concept.
- type: a LISPFunction.
- meaning: this is a function of one argument that is APPLIED to each of Concept's ancestors until one returns a non-NIL value.
- meaning: either NIL or a Concept.

**KLUnMarkEntity** [KLONEEntity;Anything]

**description:** Removes a mark from an entity. Marks can be arbitrary LISP and the one passed in as argument must be EQUAL to one on the list of markers.

**parameters:**
- KLONEEntity
- Anything

**value:**
- type: a KLONEEntity.
- type: anything.
**KLValidate** [Concept;BreakFlg]

**description:** This function checks to see if a Concept is constructed correctly according to KLONE rules. If the Concept is an Individual Concept, then it must satisfy the SDs of its defining SuperConcepts. In addition, no Obligatory Roles may be left open, Number restrictions must be met, and V/Rs have to be satisfied in all cases. If not an Individual, then Number restrictions must be consistent (in Diffs of SuperRoles, in particular).

**parameters:**
- **Concept**
  - type: a Concept.
  - meaning: a Concept.
- **BreakFlg**
  - type: a Boolean.
  - meaning: if non-NIL, will cause the function to BREAK if the validation fails. This is for circumstances where Validate is not expected to fail, and its failure constitutes a major error in the network structuring.
REFERENCES

