CODE4.2

Reference Manual and User Guide

April 1995 (revision 18)

This document is the main reference guide to the CODE4 system. Since CODE4 is a research tool under development, this manual is not guaranteed to be completely up-to-date or accurate.

Language Analysis and Knowledge Engineering Group, University of Ottawa

Ottawa, Ontario, Canada K1N 6N5

613-562-5800 ext. 6722

Chief CODE4 Designer: Prof. Timothy C. Lethbridge (tcl@csi.uottawa.ca)

Other Authors: Karen Eck (eck@csi.uottawa.ca),Judy Kavanagh (kavanagh@csi.uottawa.ca)

Lab Manager: Prof. Doug Skuce (doug@csi.uottawa.ca)

Table of Contents

1 Introduction

CODE4 is a knowledge management system that can be used for many purposes, including:

• Designing or specifying something
• Resolving disagreements in complex domains
• Documenting complex structures, procedures, rules, etc.
• Setting standards and defining terminology
• Stimulating creative thinking

Traditionally, knowledge associated with the above tasks is represented in documents containing text or diagrams. The knowledge is often poorly structured, ambiguous, contradictory and incomplete. There is typically both conceptual and terminological confusion. A major goal of CODE4 is to allow people to represent and manage knowledge in a more structured way, so that the computer can actively assist in preventing the above types of problem.

Other computerized tools exist that share this goal. There are special purpose tools in various domains (e.g. CAD and CASE). There are general purpose tools that manage informal knowledge (e.g. integrated document processing tools, hypertext tools). There are also other tools for knowledge representation and knowledge acquisition that, like CODE4, have been developed by artificial intelligence researchers (e.g. KL-ONE derivatives, expert system shells). We believe that CODE4 takes a well balanced approach: It can be used in a wide variety of domains (both general and specific); the user can represent knowledge both formally and informally, with both text and limited graphics.

Version 2 of CODE was used by various corporations and researchers in such domains as software engineering, metallurgy, optical storage technology and philosophical thought. Version 3 was an experimental prototype. Version 4 (a complete redesign) adds flexibility, an enhanced and rationalized knowledge structure, powerful user interface paradigms and much more. Version 4 has been used in industrial and university settings since late 1991.

This document describes release 4.2 of CODE. This release runs under ObjectWorks 4.1 or VisualWorks 1.0.

Comments, criticisms and suggestions are always welcome.

1.1 Organization of this document

This document is designed as a reference document in that it contains a more-or-less complete technical coverage of CODE4. Nevertheless, it should be possible for an experienced person to read it sequentially, exploring the document and the software at the same time.

The beginner should read sections 1, 2 and 3 sequentially, skipping over subsections that are indicated to be for experts.

The rest of this introduction outlines key concepts and the environment in which CODE4 runs. Section 2 describes Smalltalk, and how CODE4 has been integrated into that environment. Section 3 is a step-by-step guide to help the beginner get started. Section 4 describes major user interface components. Section 5 describes how to perform various actions to manipulate knowledge. Finally, section 6 describes built-in knowledge. There are also several useful appendices including a list of literature references, a detailed glossary and a complete index.

1.2 Key concepts to understand

The key concepts to understand in CODE4 are briefly listed below. For a greater conceptual understanding the user should refer to other papers listed in the bibliography. For a more complete list of terms, refer to the glossary. On first reading, the beginner should only skim this subsection.

- **Concept**: A unit of knowledge. Specialized types of concepts include properties, statements, instance concepts, type concepts, metaconcepts, temporary concepts, etc. Concepts can become the *subjects* about which things can be said. To say something about a concept, one combines it with a *property* making a *statement*. Concepts can be arranged in many types of relationships.
- **Knowledge base**: A group of concepts that is loaded from a file into the on-board memory of a computer (RAM). Multiple knowledge bases may be loaded at a time. In the future, knowledge bases will be linkable in dependency relationships, but for now they are all independent.
- **Knowledge map**: A specification of a network of related concepts. Types of knowledge maps include isa hierarchies (taxonomies), property hierarchies, partonomies, etc. Knowledge maps are treated as directed graphs and are displayed (textually or graphically) in subwindows of browsers (see below). Typically, the concepts in knowledge maps form hierarchies, although this is not necessary.
- **Knowledge mask**: A filter that determines whether a concept will be included in a knowledge map (and thus displayed to the user). It contains a logical expression (in future this will be arbitrary CNF – conjunctive normal form – but for now it is just a list of conditions that must all be true) relating a set of boolean conditions that are applied to each concept. Masks control the *visibility* of concepts; they are used for hiding and showing specific sets of concepts, as well as more detailed patterns of knowledge. Each knowledge map contains a knowledge mask.
- **Interaction paradigm**: A specification of look and feel of a subwindow. Examples include 'outline', 'graphical', 'user language' and 'matrix'.
- **User language**: Text that a user types when describing a property of a concept (i.e. a statement). A user may restrict the language to a specialized syntax called Cleartalk that the system can parse and process.
- **Browser**: A window containing one or more subwindows that are linked together. A selection in one subwindow causes an update of what is displayed in another subwindow. Each subwindow operates on a knowledge map.
- **Driving browser subwindow**: A browser subwindow which is linked to one or more other (dependent) browser subwindows. A selection in the driving subwindow causes an update of what is displayed in the dependent subwindow. Closing the window of the driver causes the window of the dependent to close as well..
- **Dependent browser subwindow**: A browser subwindow updates automatically when the selection in its driving subwindow changes.
- **Browser template**: A specification of how to build a browser describing its subwindows and how they are interconnected. Each subwindow description includes the type of knowledge map, the interaction paradigm, etc. There is a set of default browser templates. In the future it will be possible for the user to add new templates
- **Format**: Information associated with each subwindow that refines its appearance. It includes such attributes as font, alphabetical *vs.* hierarchical, graph layout algorithm, etc.
- **Control panel**: A window through which the user controls aspects of the CODE4 session. The control panel allows loading and saving of files and the setting of various types of default parameters.
- **Feedback panel**: A window associated with a knowledge base that indicates to the user what commands have been executed and what problems have occurred. In the future it will make active suggestions about actions the user can take to solve various problems.

1.3 Typical pattern of use

The following describes briefly a typical user session. See section 3 (Getting Started) for more detail.

1) The user starts the CODE4 Smalltalk image and uses the control panel to load a top-level knowledge base containing general knowledge (an *ontology*).

2) The user opens a browser, and sets display parameters so the browser is displaying knowledge the ways the user wants.

3) The user queries the knowledge base by using masks and dependent browser subwindows.

4) Upon finding missing or incomplete knowledge, the user adds new concepts and refines existing ones.

5) The user saves the knowledge base and quits the image.

1.4 Platforms

CODE4 is intended to run on any platform with Objectworks™\Smalltalk-80 Release 4.1 from ParcPlace. (It will also run using the VisualWorks version 1 virtual machine)

At least 5MB of disk space is required for installation and a minimum of about 8M of RAM is required to run the software. A fast machine (e.g. Sparc-10, or Pentium) is recommended, although the software is usable on slower machines, particularly for the creation of small knowledge bases..

A high-resolution display is recommended. Color is optional, but can be useful if available. On some machines the user of colour can degrade performance.

At the current time CODE4 is known to run properly on Sun4 (SunOS, but not Solaris), MS-Windows and Macintosh (not Power-PC) platforms.

1.5 Availability

A demonstration version of CODE4 available by ftp from ai1.csi.uottawa.ca. The version is kept in a subdirectory called 'democode'. You must log on with a special password to get this version. Please contact the one of the people listed on the cover page to obtain such a password. It is not possible to save or load knowledge base files with the demo version; nor is it possible to browse the source code.

For information about a full-fledged version, including cost, please contact us at the address on the cover page of this document. We charge for copies of CODE4 in order to subsidize our research. Educational institutions pay only a minimal fee though, and furthermore we may be able to arrange reciprocity agreements to exchange each others' software.

2 Smalltalk Basics

Users of CODE4 need relatively little knowledge about Smalltalk; however, it would be useful to know the following points:

- Smalltalk applications are delivered as two files: an *image* file (e.g. code4.im), and a *sources* file (code4.sources). You don't need the sources file unless you expect to modify the software yourself. CODE4 is started by running a program called a *virtual machine* (VM). On most machines, you tell the VM which image to run by typing the VM program name followed by the image name (e.g. dost80 KBname.im). On the Macintosh, you simply click on an image icon. In some environments your system administrator may have set up a special program called a script that may slightly change the way you start CODE4.
- As you work with CODE4 you will notice an additional file in your directory with a suffix of '.changes'. This accumulates any changes made to the Smalltalk software in your image. For most users it is unimportant.
- In Smalltalk applications, the buttons on a three-button mouse have the following standard usages:

Left (operate): Selecting whatever is pointed to.	**Middle (menu)**: Displaying a menu that is application-specific.	**Right (window)**: Displaying a menu that performs window-manager operations.

- For machines that do not have three-button mice, the middle and right buttons have been made redundant. The application-specific menu can be displayed by clicking on the horizontal region above any subwindow (this region is called the *menu bar*).
- It is possible to cut and paste between other applications and Smalltalk applications. The Smalltalk end of such operations is accomplished through menu selections.
- In order to save your work, simply save the image (using the Launcher, see below); this ensures that window layouts, etc. have been saved. In CODE4, it is also advisable to save your knowledge base (i.e. the raw knowledge) from time to time, especially after you have performed a major operation; the version number is automatically incremented with every save. Although it is possible to load a knowledge base into CODE4 every time you want to work on it, window layouts and similar traits are not saved unless you save the image.

2.1 CODE4 enhancements to basic Smalltalk-80

This subsection is intended for experts only.

- It is possible to adjust the relative positions and sizes of subwindows. To do this, hold down the control (ctrl) key and press the left mouse button while pointing to a scroll bar or menu bar (you will see a double-tipped arrow aligned in the direction that the window can move). Then move the mouse to a new location in the window and release the mouse button. You can only adjust subwindow borders that are in the interior of a window (i.e. you cannot change those on a window's edge).
- Several major software development tools have been added, including a hierarchical inspector and a hierarchical call browser. The general CODE4 user need not be concerned about these facilities. See appendix 6 for more information.

- Cut and paste operations of the paragraph editor have been generalized to allow for any object to be copied, cut and pasted. When you cut or copy, the object is added to the set of paste buffers as before. When you paste, a particular class of object is requested. The selected paste buffer is converted, if possible, into the requested class. The previous mechanism is unchanged from the user's perspective in that pre-existing controllers use text only (a very general class). CODE4, however, may place *concepts* in paste buffers; these can be pasted as text into a paragraph (or even an application external to Smalltalk), or they can be pasted as concepts into some other element of CODE4. For example, the concept *car* may have a property of *engine* that has no value. The concept *car engine* can be pasted in as the value, so that the resulting property is *engine: a car engine*.
- Several Smalltalk bugs have been fixed, e.g. a bug that caused a delayed system crash when a window in a different project was closed.
- In all Smalltalk list panes, cursor arrow keys can be used to move around. In basic Smalltalk, you can only use arrow keys to move around in text panes..

3 Suggestions About What You Should Learn

This section is intended to give general guidance for people learning how to use CODE4. Subsection 3.1 is an introduction for beginners. Subsection 3.2 is intended to point out features that are probably the most useful to the average user – i.e. features that you should make sure you 'do not miss!'.

3.1 Getting Started

This subsection provides a simple step-by-step approach so the beginner can get started. Simply follow the directions. It is assumed that a system administrator has put a version of CODE4 in your directory and has made sure the appropriate version of Smalltalk can be run. It is also assumed that you are using the virgin image as delivered, with no local modifications.

1) Start CODE4: **On unix platforms, type 'st80 code4'** (or 'dost80 code4' or some other command as specified by your system administrator). **On Macintosh or MS Windows platforms, double-click on the CODE4 icon.**

> *At this point, you may have to wait a while while Smalltalk starts up. After 5-30 seconds, several windows should appear on your screen (if they do not seek help!) The* <u>*Launcher*</u> *(section 4.1) should be at the top-left; this is a small window with a menu. The* <u>*Control Panel*</u> *(section 4.2) should be at the top-centre; this is a larger window with many facilities for manipulating aspects of CODE4. You may move these windows around at any time, and you may collapse the control panel. To learn how to do these things, read the documentation of your window manager software (see your system administrator).*
>
> *We will now focus on the control panel. On its left are two sets of buttons. The upper set allows you to tell CODE4 how expert you are in its use.*

2) Tell CODE4 that you are a beginner: **Using the left mouse button, click on 'beginner' in the control panel.**

> *Now any menus that subsequently appear will be as simple as possible, although you won't have access to some powerful facilities.*

3) Go to the knowledge bases control panel: **Click on 'KBs' in the control panel.**

> *The right hand portion of the control panel now shows information about* <u>*knowledge bases*</u>*. At the top-right is a list of knowledge bases loaded into memory. A knowledge base contains a set of* <u>*concepts*</u> *which describe things in the world. The whole purpose of CODE4 is to allow you to create and edit knowledge bases.*
>
> *There should be one knowledge base in memory when you start a new image. We will now create another.*

4) Create a new knowledge base:

- **Position the cursor over the 'Loaded KBs' list.**
- **Press and hold the middle mouse button to bring up the menu.**
- **Keeping the middle mouse button down, move the cursor to 'remove and create'.**
- **Release the middle mouse button down while pointing to 'remove and create'**

(The cursor will move to the first item in a submeu)

- **Click on 'create a new default KB' in the submenu**

(on the Macintosh, where there is just one mouse button, you will need to position the cursor in the region called the menu bar that is just above the Loaded KBs list, or else use 'shift' while you click the mouse button to bring up the menu)

You have just issued your first CODE4 <u>command</u>!

CODE4 now wants to know what you will call this new knowledge base. A small window appears with a prompt saying 'Enter the new name...', the word 'default' appears in black below the prompt. 'default' is a poor name for a knowledge base, so we will change it to something better. This will be your first experience editing text in CODE4.

5) Respond to the prompt: **Type 'my first kb' into the prompt window, and hit the 'return' key.**

The prompt window disappears and a new knowledge base is created.

The name of the new knowledge base appears in the 'Loaded KBs' list of the control panel. You can also see that we have created version 0 of knowledge base 'my first kb', and that it has 87 concepts already. Don't worry about these concepts for now, most are <u>system concepts</u> that are only important to experts.

6) Indicate that we want to operate on your new knowledge base: **In the loaded KBs list, check that 'V0 my first kb...' is highlighted** (i.e. has white text on a black background)**.** If it is not, click on it with the mouse.

Commands issued from the menu will now operate on your new knowledge base.

We will now make a change to the knowledge base. To do this we will open a window called a <u>browser</u>.

7) Open a browser: **Select 'open browser on _'** from the 'loaded KBs list' menu.

Now a new menu appears, entitled 'Choose a template'. CODE4 wants to know what type of browser you want to open.

8) Request a simple textual browser: **Choose 'Outline isa hierarchy from X'.**

CODE4 now wants you to tell it where the new browser window should be placed on the screen. A rectangle appears at the cursor position. We will make a big browser.

9) Specify the shape of the new window:

- **Press and hold the left mouse button near the top-left of your screen.**
- **Holding the left mouse button down, move the cursor to near the bottom right of your screen.**
- **Release the mouse button.**

A new window now appears. It contains an indented list of concepts.

At the top of the list is a concept called 'thing'. This concept is the most general concept in every knowledge base. Everything is a thing! For now, ignore everything else in the list.

We will now prepare add a new concept to the knowledge base.

10) Indicate that we are going to perform an operation on 'thing': **Select 'thing' in the browser.**

We will now issue a command to add a subconcept (a specialised 'thing'). We could do this through the menu, but instead we will show you another way to issue a command: using a hot key.

11) Issue the 'add child' hot key: **While the cursor is pointing to the browser, hold down the 'control' key while you press 'a'**. (In the rest of this manual we will simply say, 'type ^a')

A new concept is now created by CODE4. You can tell this has been done because 'specialized thing' appears in the list. The new concept is a subconcept of thing; it is indented more deeply than 'thing'. 'Thing' is a superconcept of the new concept.

We will now give this new concept a reasonable name.

12) Rename the new concept: **Type 'car' into the browser and hit 'return'.**

The new concept is now called 'car'. Note that 'car' appears at the bottom of the list, and also at the top of the browser in a separate subwindow. The currently selected concept always appears in this small subwindow, so that you can make detailed changes to its name using the mouse. We won't worry about this for now.

We will now tell CODE4 that we want to focus on editing just the section of the knowledge base to do with 'car' (which is the currently selected concept).

13) Restrict the display to only show only the hierarchy of ‘car’ (i.e. its superconcepts and subconcepts):

- **Select ‘visibility’ from the browser menu.**
- **Select ‘exclude all but ..._... hierarchies’ from the submenu**

Now, the only concepts remaining on display are ‘thing’ and ‘car’. (The others have been temporarily hidden.)

14) Add a subconcept to ‘car’ using the menu: **Select ‘add child to ...’ from the ‘edit’ menu.**

We could have also done this using hot key ‘^a’.

The concept ‘specialised ‘car’ now appears, and is the selected concept.

15) **Rename ‘specialised car’ to ‘My trusty auto’** by typing into the browser.

As before, the concept is renamed.

We will now show you how to put the list of concepts in alphabetical order, and at the same time show you a third way of issuing commands in CODE4: Using action buttons.

16) Alphabetize the list of concepts: **Click on the word ‘alpha’ at the top right of the browser**.

The concept list no longer appears indented, but instead is in alphabetical order.

We will now open a new browser that shows concepts graphically. This new browser will show the subtree of the concepts selected in the first browser.

17) Issue the ‘open graphical subtree’ command:

- **Select concept ‘car’**
- **Type the hot key ^k**

(the outline for a new window appears)

- **Position the new window on the screen using the left mouse button**

This new browser presents concepts graphically, but has most of the same capabilities as the browser we have been using so far (called an outline paradigm browser).

We will now explore a few capabilities of the graphical paradigm.

18) Rearrange the appearance of the graph using the left mouse button:

• **Point to the background of the graph.**

• **Click and grag the background of the graph.**

(you will see that the concepts move around)

• **Select 'car' in the graph by clicking on it.**

• **Drag 'car' to the middle of the graph.**

(you will see that the concept moves but the others are left behind)

• **Use the menu item 'reformat>>reformat graph' to make the graph look good.**

• **While holding down the control key, click on 'car'**

(you will see that the whole hierarchy below 'car' is selected'

• **Hold down the mouse over 'car' and drag the mouse**

(you will see that all the selected concepts are dragged together)

We will now see that the new graphical subtree browser changes dynamically when we select concepts in the outline browser.

19) **Select the concepts in the <u>outline</u> browser one-by-one and see how the graphical subtree browser changes.**

When 'thing' is selected, its two children are shown below them, When 'My trusty auto' is selected, it appears alone, because it has no children.

We say that the outline browser is <u>driving</u> the graphical subtree browser.

There are many other kinds of browsers you can open in CODE4, but for now we are finished looking at concepts.

20) Close the outline browser: **Select close from the menu that appears when you press the <u>right</u> mouse button** (in MS-Windows, just click on the window close box).

Note that the graphical subtree browser also closes because it <u>depends</u> on the outline browser.

We will now prepare to save the knowledge base.

21) **In the control panel 'loaded KBs' list, issue the 'save _ to disk' command.**

[This will not work if you are using a special demo version of the software because you are not authorised to save knowledge bases, so skip to step 24 in that case].

You will be prompted for a file name (the default will be 'my_first_kb.ckb'

22) Specify the name of the file.**Type ‘first’ and hit return**

Note that the file name and knowledge base name do not have to be the same. Also, all knowledge base files have the suffix ‘.ckb’.

The knowledge base is now saved. You can verify this by looking at the list of files in your current directory.

23) **In the text pane at the centre-right of the KBs control panel:**

- **select the existing text by dragging the mouse pointer through it**
- **type ‘*.ckb’**
- **hit ‘return’**

You should now see ‘first.ckb’ in the ‘Files on disk’ list. At some later point you could load this into your image and continue work.

We will now save the CODE4 Smalltalk image (so we can continue where we left off), and end the session.

24) **From the launcher window, select ‘Quit’, and then ‘Save then Quit’ from the menu that appears**.

At the prompt “Enter name for image file" you will see the current name of the image (i.e. code4). Type in the name you wish to call your image (e.g. first); this will overwrite the current name.

Hit <Enter> to accept this new name and quit your image.

After 10-50 seconds the CODE4 windows will close.

Congratulations, you have just completed your first CODE4 session!

To learn more about CODE4, you can try exploring the other menu items in the launcher, control panel and browsers. While doing that you can refer to the rest of this manual. One of the most important things to learn about is adding properties to concepts. To do that, try opening a browser of type ‘outline isa hierarchy from X with editable outline statements’.

Once you become familiar with the capabilities of ‘beginner’ level, you can upgrade yourself to an ‘intermediate’ user. The next section gives you tips about what to learn.

Good luck!

3.2 Important Material to Learn in Order to be Productive

Use this section to learn to use CODE4 productively fast!

Sections of the manual listed below contain information with which an intermediate user of CODE4 should be familiar. Sections not mentioned can be saftely omitted by most users. As you learn these sections, try them out on the system, preferably with an expert to assist you.

Items in **bold** are CODE4 features to learn. **Underlined** items are critical productivity features – learn these first if you have limited time.

1. Read the introduction

2. Read about Smalltalk basics (the first part of chapter 2)

3. Read the ‘getting started’ section above (section 3.1)

4. Read about user interface basics (parts of chapter 4 as follows):

    • **Expertise level** (start of section 4.1)

    • Parts of the **environment control panel** (section 4.21): **Default KB path**, **Confirming when deleting** and **Confirming when closing**.

    • All about the **KBs control panel** (section 4.2.2): **Loading and saving files**, etc.

    • Many details about **browsers** (parts of section 4.3 as follows):

        - Browser components (4.3.1): several ways of **selecting items** (4.3.2 and 4.3.4.1), **changing subwindow size** (4.3.4.2), **editing labels** (first part of 4.3.4.3).

        - Performing **commands** in browsers (4.3.4.4 to 4.3.4.8): **Repetition factors**, **menus**, **hotkeys**, **action buttons**, **copying concepts**.

        - Basics of the **outline paradigm**: **union/isect** and **update prop set**: (start of 4.3.5.1).

        - Basics of the **graphical paradigm**: **scrolling**, **selecting** and **rearranging** (4.3.6.1). Key format options (parts of 4.3.6.3): **save layout**, **use layout**, **node font**, **sibling space**, **child space**.

        - Basics of **value panes**: **show/remove English** (in 4.3.7.1); **copying** and **pasting** (4.3.7.2).

        - Editing values (parts of 4.3.7.3): **accept** and **tab completion**; **set value to copied concept**; **add copied concept to value**; **delete concept from set.**

        - Basics of the **matrix paradigm**: how it works (start of 4.3.8) and **matrix format options** (4.3.8.1), and **editing cells** (4.3.8.6).

        - **Masks** (all of section 4.4).

        - **Feedback panel** (all of section 4.5).

5. Read about important commands (parts of chapter 5 as follows):

- **<u>Opening browsers</u>** (all of 5.1.3)
- **<u>Saving</u>**, **<u>loading</u>** and **<u>creating</u>** knowledge bases (5.2.1 to 5.2.6)
- Editing knowledge bases (parts of 5.3 as follows)
    - **<u>Adding</u>** and **<u>deleting</u>** concepts (5.3.1 to 5.3.4)
    - **<u>Renaming</u>** concepts (5.3.5)
    - **<u>Editing values</u>** and **facets** (5.3.6 and start of 5.3.7)
    - **<u>Changing the parents</u>** of concepts (5.3.9 and 5.3.10)
    - **Detailed manipulations of properties** (5.3.11 and 5.3.12)
    - **<u>Types</u>** vs. **<u>instances</u>** (5.3.15 and 5.3.16)
    - **<u>Editing general hierarchies</u>** other than isa and property (5.3.18.1)
- Querying knowledge bases
    - **<u>Masks</u>** (all of 5.4.1)
    - **Hardcopy output** (5.4.3)

6. Read chapter 7 about interpreting error messages

4 Major User Interface Components

This section describes the look and feel of the CODE4 interface components. Discussion of specific commands that the user can employ (e.g. to query or edit the knowledge base) is deferred to section 5.

4.1 The Launcher

When the Smalltalk-80 system is running, one window that should always be present is the Launcher (Figure 1). This is a menu that is used to open other windows. The top menu item in the launcher (named *CODE*) provides a few simple commands (see section 5). Most commands are better performed from the control panel, browsers or other windows, although the control panel must be opened from the launcher.

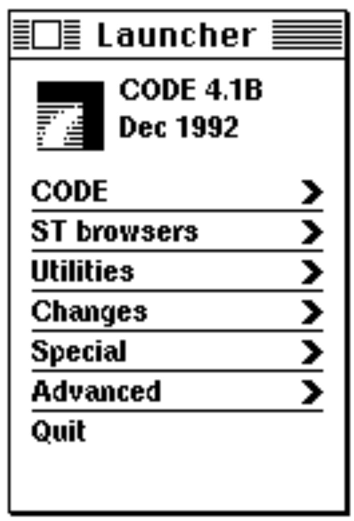


Figure 1: The Launcher.

You should not close the launcher. If you do close it, you can re-open it by carrying out the following steps: (1) find another Smalltalk window into which you can enter text, (2) type 'LauncherView openLauncher', (3) select this text and, from the middle mouse button, choose the 'do it' menu item. If you cannot find a Smalltalk window that can enter text, press <ctrl-C> in some Smalltalk window and get a debugger; you can enter text there.

The following summarizes the the functions of the CODE4 submenu items in the launcher:

control panel: opens the control panel (see section 4.2)

open browser: lists a set of browser templates. Selecting one of these will open a browser on the 'current' knowledge base (the knowleledge base selected in the control panel – see section 4.2, from which this menu can also be opened).

report problem: brings up a window into which the user can describe a problem or suggestion (figure 2). Upon hitting carriage-return in this window, a detailed report is sent to CODE4 developers by email (on unix systems only; on the Mac or PC, the report is saved to a file).

upgrade software: searches for any changes to the software that may have been issued and applies them to the current image. This feature will only work if you made arrangements for access to these upgrades.

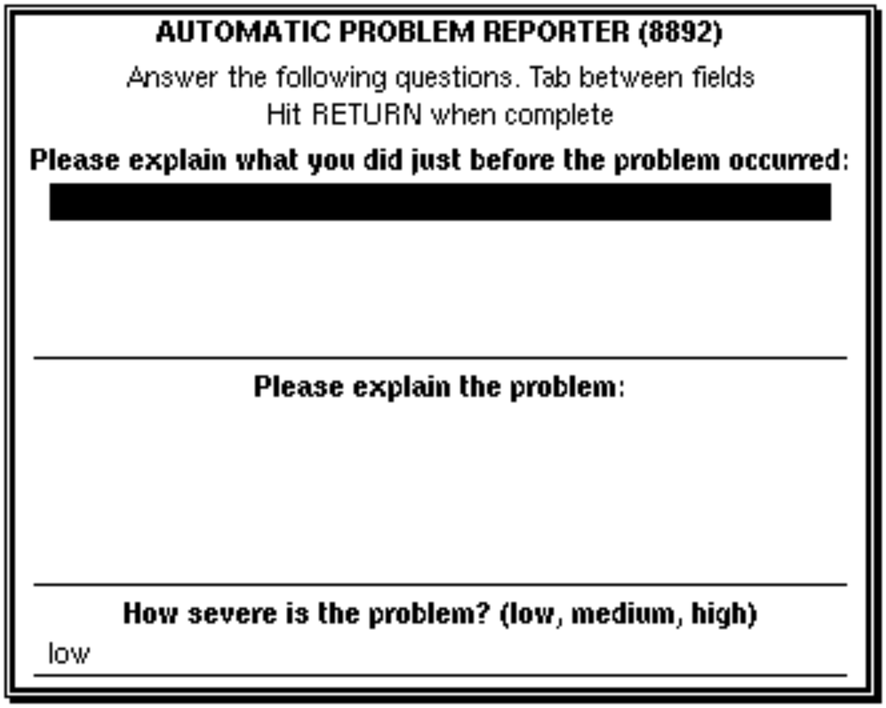


Figure 2: The problem reporter window.

All of the other menus available from the launcher are general Smalltalk utilities, including capabilities to edit ordinary files and save the 'image' of the whole CODE4 session.

4.2 The Control Panel

The control panel (figure 3) is a window used to configure the CODE session to a user's taste. It is opened from the launcher (from the first menu item in the launcher's *CODE* menu - see section 4.1).

At the left of the control panel are two subwindows where the user can select from two sets of buttons:

• **User Expertise Level**:

These buttons allow the user to select the expertise level: beginner, intermediate, expert or developer. This expertise level determines how many items will be displayed in menus, and therefore how complex CODE will be to use. Beginners should ensure 'beginner' is selected. This manual describes all operations, including many not available to beginner or intermediate users.

• **Control Panel Mode**:

These buttons determine what will appear on the right side of the control panel. The following subsections describe the specialized control panels.

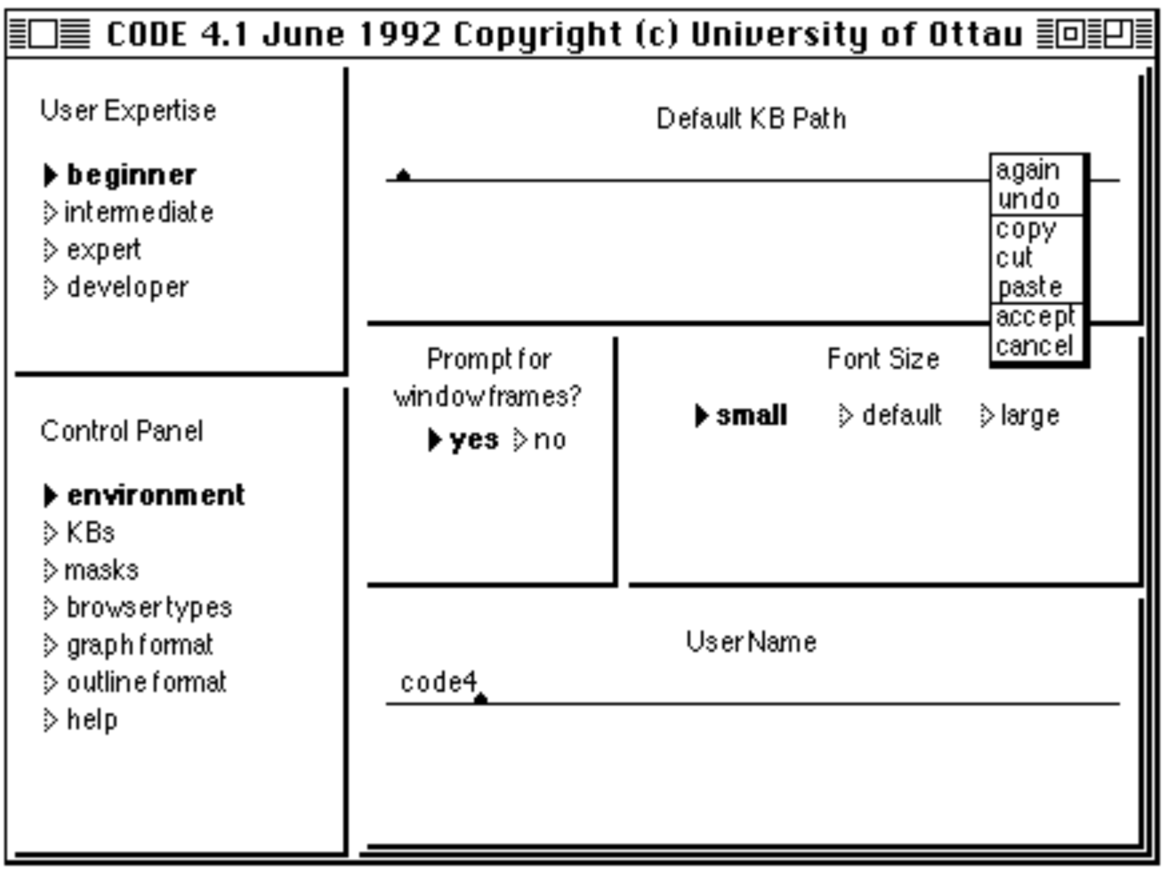


Figure 3: A Control Panel. Beginner expertise level and environment mode are selected.

4.2.1 The ‘environment’ control panel

The ‘environment’ control panel mode is used to set parameters that apply to the system as a whole, including such things as font size, user name, prompt for window frames, scroll bar position etc.

The following summarizes the functionality of the items in the environment control panel:

Default KB Path: is a field into which the user can enter the directory where he or she normally keeps knowledge bases. This can be a pattern with * as a the wildcard character. This field is used when the user selects 'default KB path' in the KBs control panel.

Prompt for window frames: allows the user to choose whether the user will be prompted to position newly opened windows (yes), or whether newly opened windows will be placed on the screen with a location and size chosen by the system (no).

Font Size: allows the user to determine the size of most text in the system. After changing the font size some windows may need to be refreshed. See also section 4.3.6 which describes the fonts used in graphs.

User name: is a field into which the user can type his or her name (or userid). On unix systems, the login name can be automatically filled in by clearing the field and hitting carriage-return.

Confirm when deleting concepts?: If yes, brings up a prompt whenever the user deletes a concept. This can be useful to prevent knowledge from being lost accidentally. it can also be annoying if you delete concepts often, hence you have a choice of whether you want the prompt or not

Confirm when closing browsers?: allows the user to specify whether he or she wants to be protected from accidentally closing browsers (in which the user may have carefully set up formatting parameters and masks). If 'yes', then whenever the user asks to close a browser (and any dependent browsers), a confirmation prompt will be presented. Note: At the current time, if you edit text but do not save it, it is possible to close windows containing such text without being warned about possible loss of incomplete edits.

Scroll bar position: Allows a choice between left (probably easier to access, but non-standard) and right (standard on MS-Windows and Macintosh)

Cleartalk parsing?: Allows the user to specify whether the system should always automatically try to interpret values of statements.

When a user types in a concept name while editing a value(see section 5.3.6), the system is capable of automatically looking to see if a concept with that name exists. If ‘Cleartalk parsing?’ is set to ‘yes’, the this lookup will always be done when the user presses ‘enter’. If a match is found, a reference to the concept will be stored (not just the name the user typed). There are several advantages to this: 1) If the user changes the name of the concept, the reference will change appropriately; 2) The user can draw ‘relation’ graphs (see section 5.3.18) or use the delegation feature.

If ‘cleartalk parsing?’ is set to ‘no’, then the user must explicitly request the concept lookup be performed (e.g. by pressing the tab key or using the ‘parse as cleartalk’ menu option). This can be an advantage if there are many concepts with the same name. Regardless of whether this switch is set to 'yes' or not, *invalid* Cleartalk will result in informal text being stored in statement values.

Speed up by using more memory?: Allows the user to specify whether the system should optimize use of memory (at the expense of speed). On a fast machine with limited memory, use 'no'.

Speed up by removing details?: If set to 'yes', certain small details of the user interface (such as icons, link labels and bullets in browsers) will be omitted.

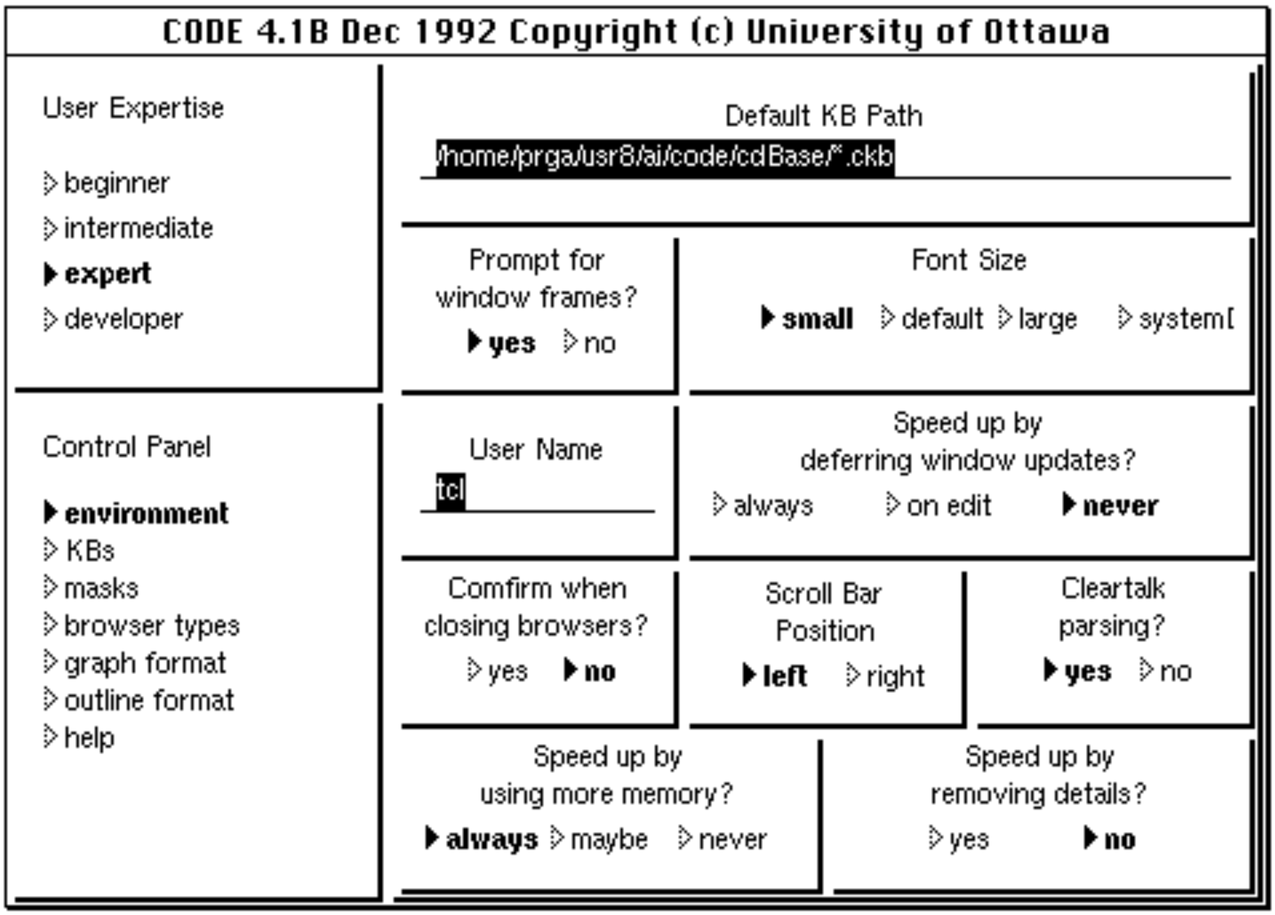


Figure 4: The 'Environment' control panel subwindow.

Speed up by deferring window updates?: allows the user to manipulate the compromise between feedback and response time. On fast machines, 'never' is the most desirable choice. On slower machines, or when dealing with very large knowledge bases, the user may choose to change this switch to 'on edit' or 'always'.

never: means that any time the user makes a selection in a browser, or edits a knowledge base, many other browser windows will be updated to reflect changing query results or changing knowledge.

on edit: means that dependent subwindows will still update when selections are made in driver subwindows, but if a knowledge base is edited, changes will not automatically reflected in other subwindows. To cause such updates to be reflected, other subwindows can be refreshed, or the control panel switch can be set back to 'never'. Windows that are not up-to-date are displayed with a grey background.

always: means that subwindows are never updated unless explicitly requested by the user.

Check integrity after every update: is a developer feature. If 'yes' then any edit will result in a 'sanity check' of the knowledge base. This operation can take considerable time.

4.2.2 The ‘KBs’ control panel

The ‘KBs’ control panel is used to save and load knowledge bases, as well as to perform operations on loaded knowledge bases. Figure 5 is an example.

Loaded KBs

At the top of the KBs subwindow (Figure 5) is a list of loaded knowledge bases (KBs). Along with the KB name is its version number (before the KB name) and the number of concepts and types in the KB (in parentheses, after the KB name), e.g. v2 ‘minimal KB’ (147 conc., 23 types).

There is always at least one KB loaded (removing all knowledge bases causes the system to create a default, top-level knowledge base). If no knowledge base is selected, the only operation available is to remove all KBs from memory and then construct a default ‘current’ KB.

The KB that is selected becomes the ‘current’ KB. See section 5.2 for details of operations that can be performed.

Any knowledge base that has been edited is marked ‘edited’, e.g. v10 'minimal KB' (298 conc., 53 types, edited). This informs the user that the knowledge base should be saved. Conversely, a knowledge base that is not marked ‘edited’ can be considered the most up-to-date version (assuming nobody else has also loaded and edited the knowledge base on another machine).

Path

In the middle of the ‘KBs’ control panel is the path box, which contains the path to the directory containing knowledge base files; the bottom of the subwindow lists all files in that directory. The pathname and file list act in a similar manner to the Smalltalk ‘file list’ tool: you simply edit the text of the pathname and hit <return> to update the file list. Select ‘load _ into memory’ from the menu to load a knowledge base (knowledge base file names have the suffix .ckb).

In the path box, it is possible to specify a pattern, using asterisks to match any arbitrary string of characters. A useful pattern is ‘*.ckb’. This will cause all knowledge base files in the current directory to be listed. Specifying no path in the path box (i.e. deleting the path and hitting return) causes the system to fill in the current directory.

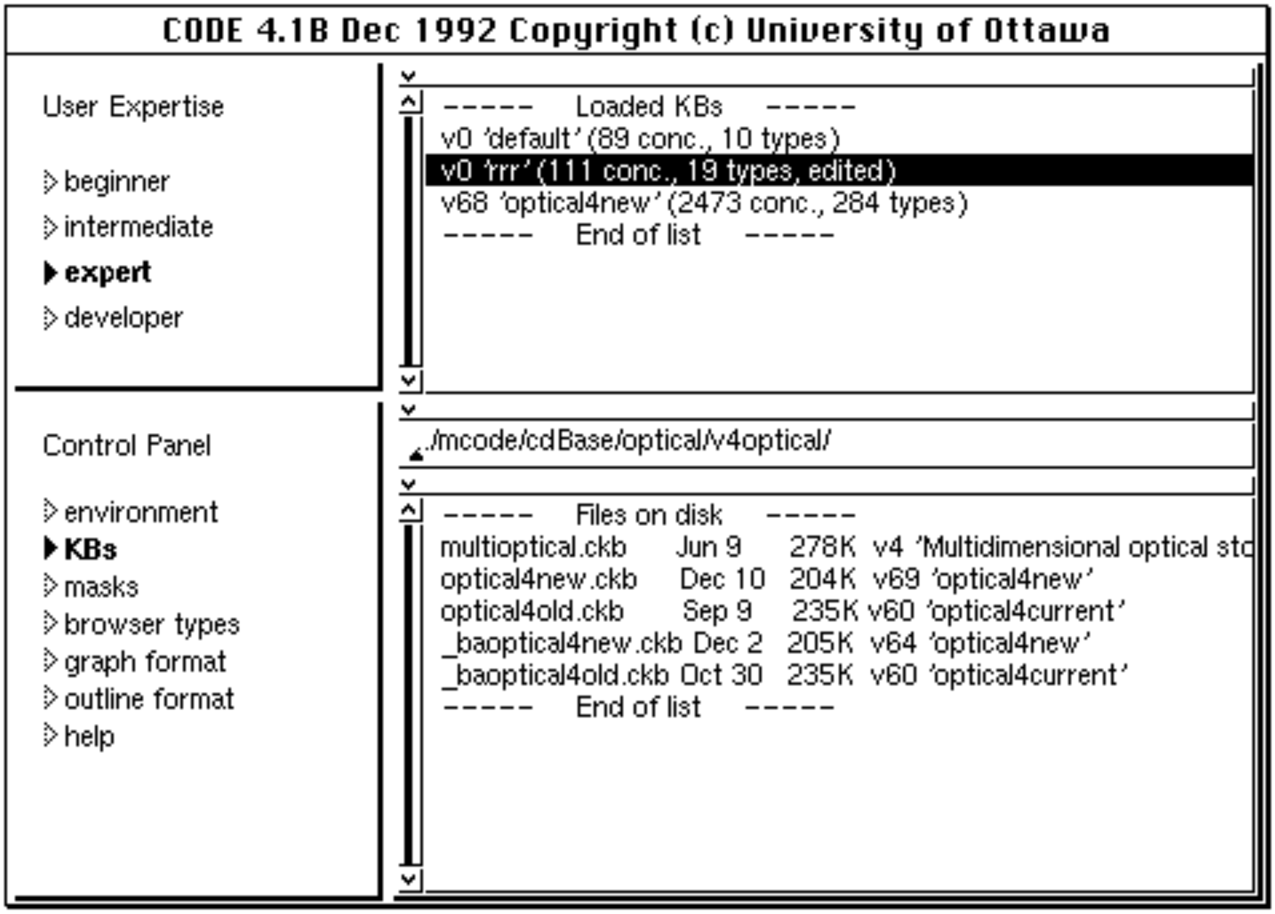


Figure 5: A KBs control panel subwindow. Three knowledge bases are loaded; the second one has been edited. There are also several knowledge bases on disk.

The following summarizes the menu items found in the path box (standard Smalltalk items are omitted)

default kb path: sets the path to equal the path specified in the environment control panel

default image path: sets the path to equal the directory from which the image was started

parent: changes the path to equal the directory of the current path

child: prompts the user to select one of the subdirectories of the current path (if any) and sets the path to equal this subdirectory

Files on disk

At the bottom of the 'KBs' control panel is the list of files on disk that match the path. Those items that have the '.ckb' suffix also display the knowledge base name and version (determined by reading the file) and the date the file was saved.

Note that filenames and knowledge base names are often similar, but need not be. For example, as seen in the bottom subwindow of figure 5, 'optical4new' is the name of both the knowledge base (with the .ckb extension) and the file. However, another knowledge base has the name 'optical4current', while its filename is 'optical4old(.ckb)'.

The following menu items are available in 'Files on disk' when a file is selected (illustrated in figure 6). All but the first work on ckb and non ckb files.

Figure 6: The Files on disk menu

load _ into memory: reads the selected .ckb file into memory. It will appear in the 'Loaded KBs' part of the control panel.

make backup of _: makes a copy of the file giving it a name with an initial '_b' followed by a letter. The letter is incremented each time you make a backup of the same file. The loading underscore of backup filenames ensures that they are always listed at the end of the directory.

rename _ to...: prompts for a new name and renames the selected file. This does not cause renaming of the knowledge base. To do the latter, it is necessary to load the knowledge base into memory.

copy _ to...: prompts for a new name and copies the selected file.

delete _ from disk: deletes the selected file after requesting confirmation.

open editor on _: opens a Smalltalk file editor on the file. This is for developer use.

If the selected file is a directory, the following option is available:

new pattern: changes the path to this subdirectory. This is the same functionality as in a Smalltalk file browser.

4.2.3 The ‘masks’ control panel

Future: This will be used to save and load masks, which can be used as ‘canned queries’. See also sections 4.4 and 5.4.1

4.2.4 The ‘browser types’ control panel

Future: This will be used to save and load browser templates. See also sections 4.3.4.2 and 5.4.2

4.2.5 The 'graph format' control panel

The 'graph format' control panel is used to determine the appearance of browser subwindows using the graphical interaction paradigm. Various items control both the 'look' of graphs and the algorithms used to draw them.

Figure 7 shows an example graph format control panel.

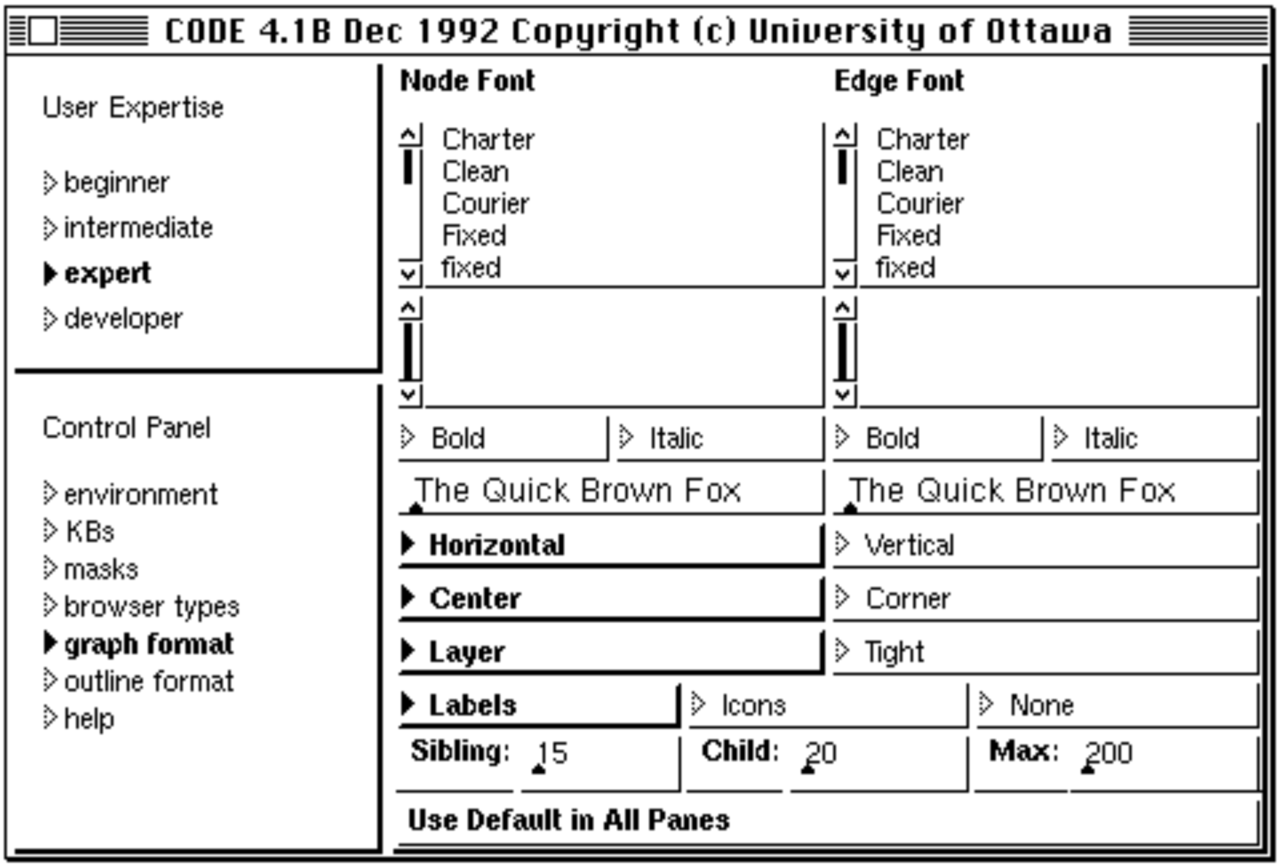


Figure 7: An example of the graph format control panel

The 'graph format' control panel contains the default specification for graph format. Each newly opened graph will be given this format (with the exception of newly opened driven subwindows which get their format from their driver). After a graph is open, its format can be changed using a format dialog that has most of the same items as the control panel. Therefore, see section 4.3.6 for a detailed description of these items.

Use Default in All Panes: This item on the graph format control panel causes all graphs to be redrawn using the format specified.

4.2.6 The 'outline format' control panel

The 'outline format' control panel is used to determine the appearance of browser subwindows using the outline interaction paradigm.

Figure 8 shows an example outline format control panel.

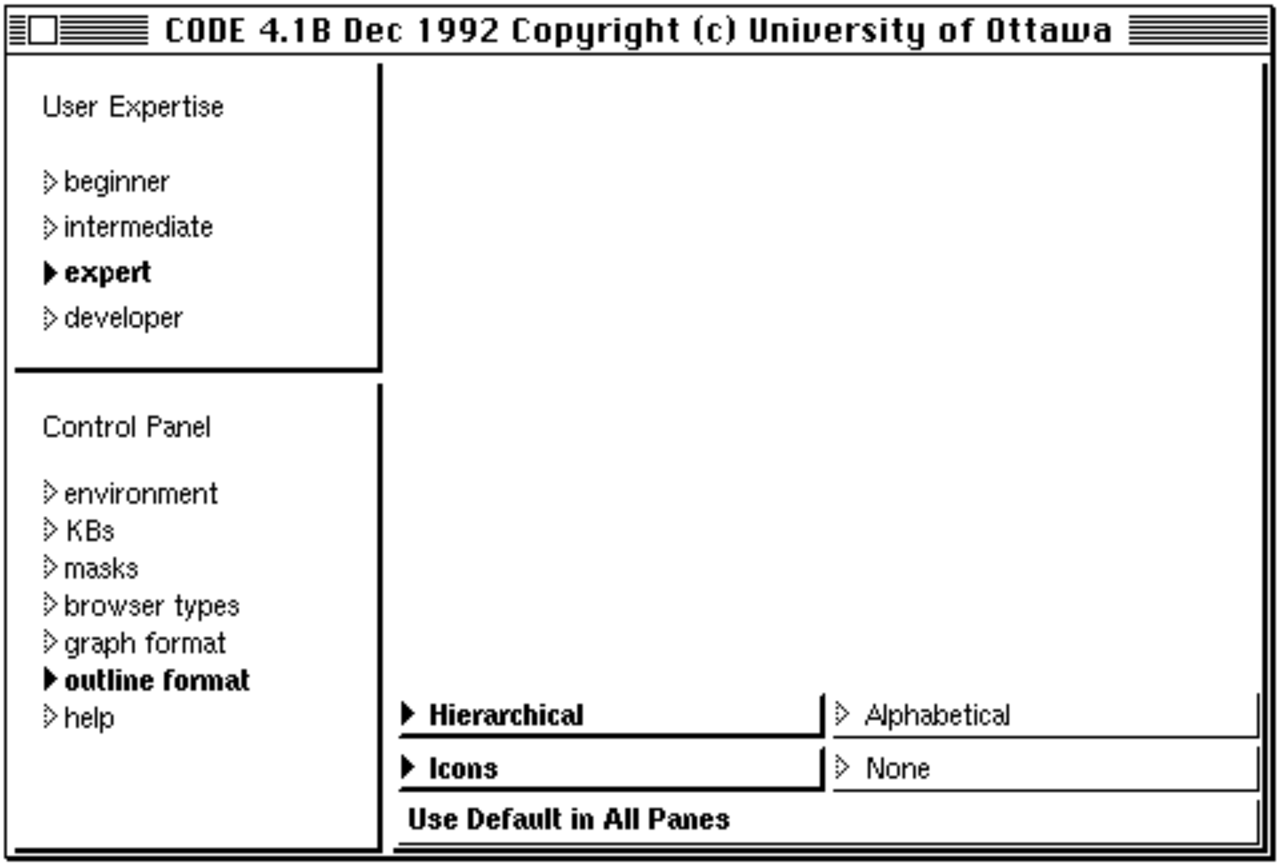


Figure 8: An example of the outline format control panel

Just like graph formats (section 4.2.5), outline formats can be set globally in the control panel, or locally in each outline pane. See section 4.3.5 for details of the available options.

Use Default in All Panes: This item on the outline format control panel causes all outline subwindows to be redrawn using the format specified.

4.2.7 The 'help' control panel

The help control panel is intended to provide help about many aspects of the system. At the current time it is only a prototype.

Help information is maintained as a hierarchy of topics, and the user's selections are recorded to allow backtracking. The following buttons are available:

index: displays a popup menu of available menu items.

previous: goes back to the panel most recently displayed.

super: goes to the more general panel

sub: displays a menu of more specific panels

related: displays a menu of panels that are related to the current one

top: goes to the top panel.

Future: Help about particular commands can be obtained as follows: Issue the command in the normal way (using a menu, hot key or action button) *while holding down the shift key*. Not all commands yet have help panels.

One of the help panels, seen under 'Index' and other buttons, is called ‘Current Command’. This displays the help for the last command for which you requested help by holding the shift key down.

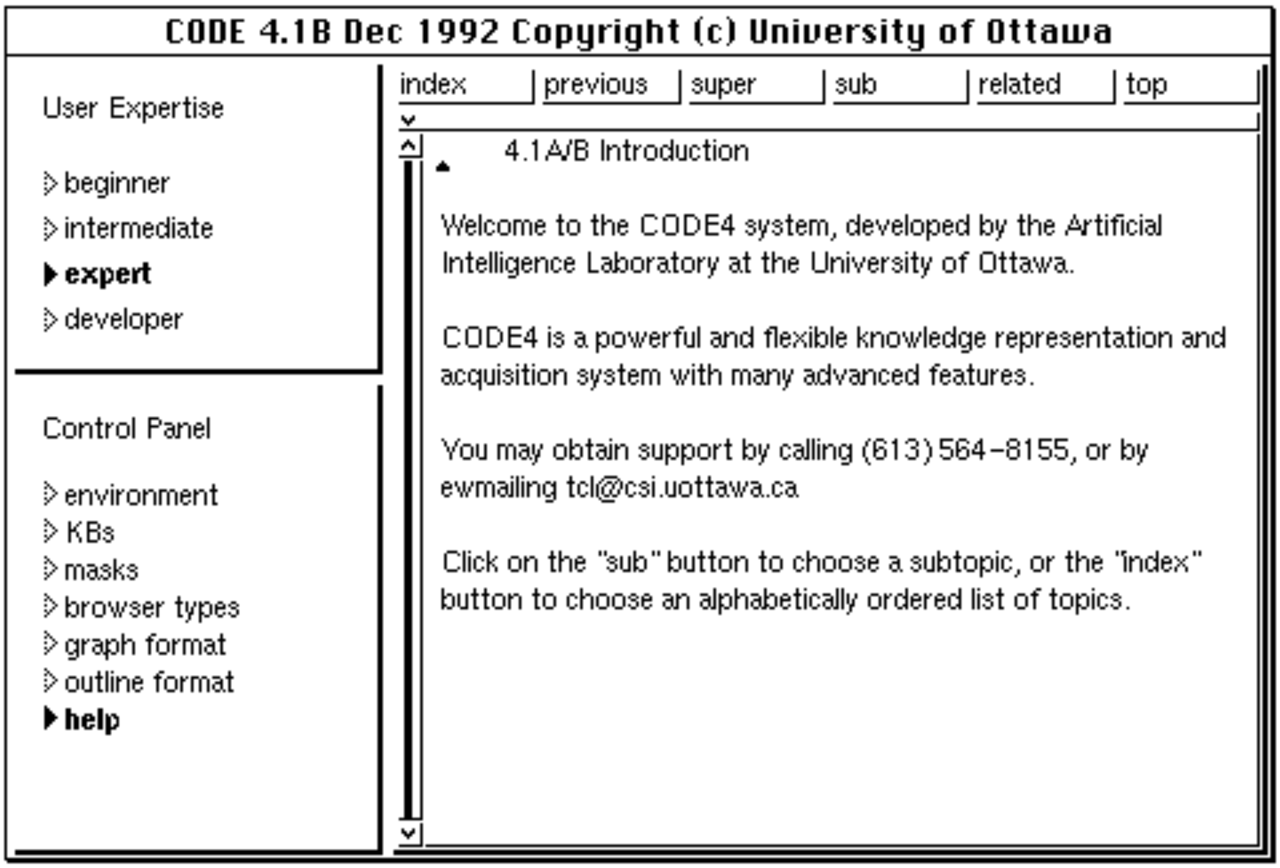


Figure 9: An example of the help control panel (incomplete fuctionality)

4.3 Browsers

Browsers are used to view and manipulate portions of a knowledge base (groups of concepts organized by a set of *knowledge maps*). Each browser is composed of one or more subwindows.

Each browser subwindow has what we call an *interaction paradigm*, i.e. it displays knowledge as one of (1) a graph, (2) an outline processor, (3) user language (i.e. simple text input by the user) (4) a matrix (like a spreadsheet), or (5) in flowing text. Where possible, however, operations are done in the same way, regardless of what interaction paradigm a subwindow is using.

Figure 10 illustrates the main features of a single-subwindow outline browser. Figures 11 and 12 illustrate graphical browsers.

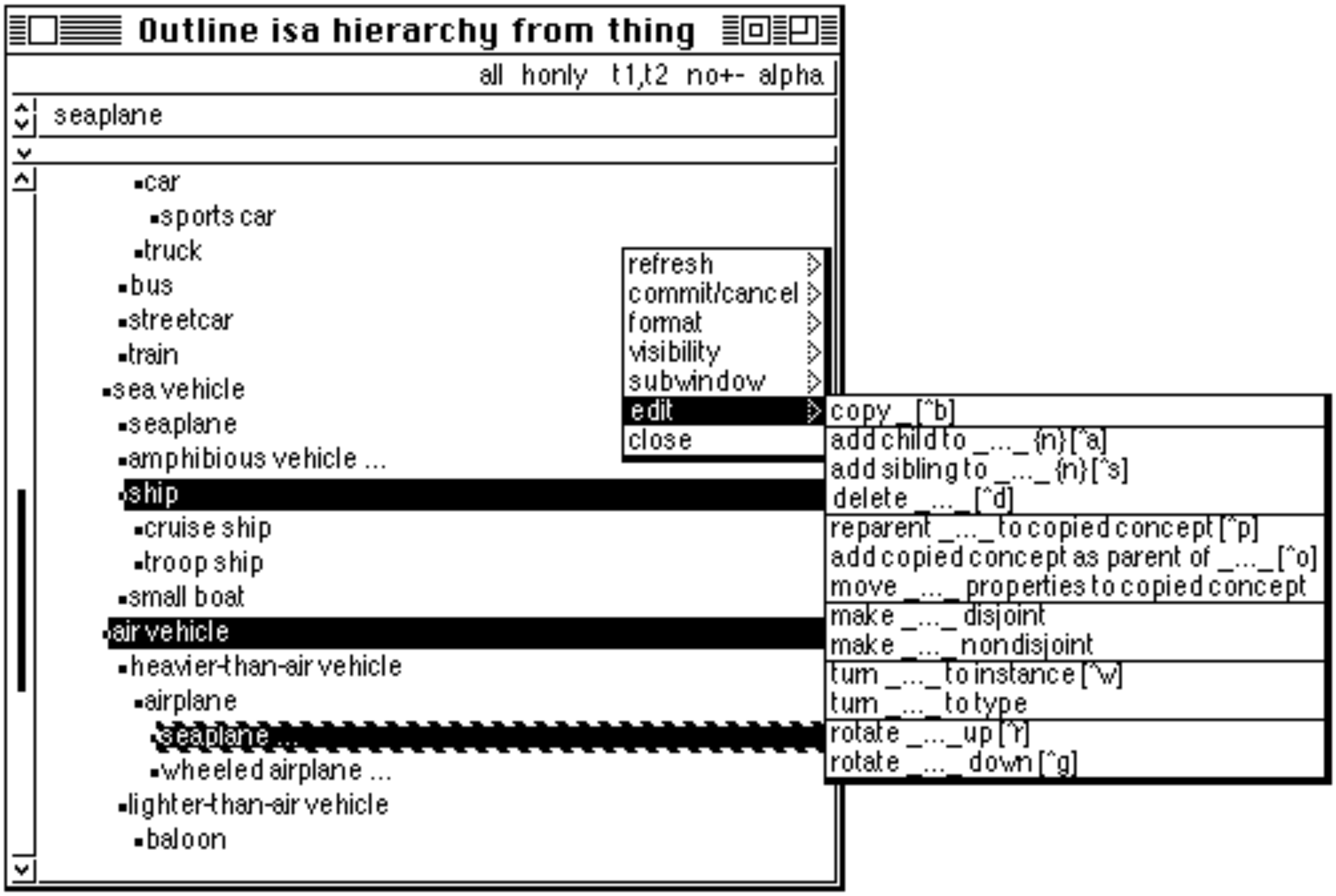


Figure 10: A single-subwindow browser using the outline interaction paradigm (showing an open edit menu)

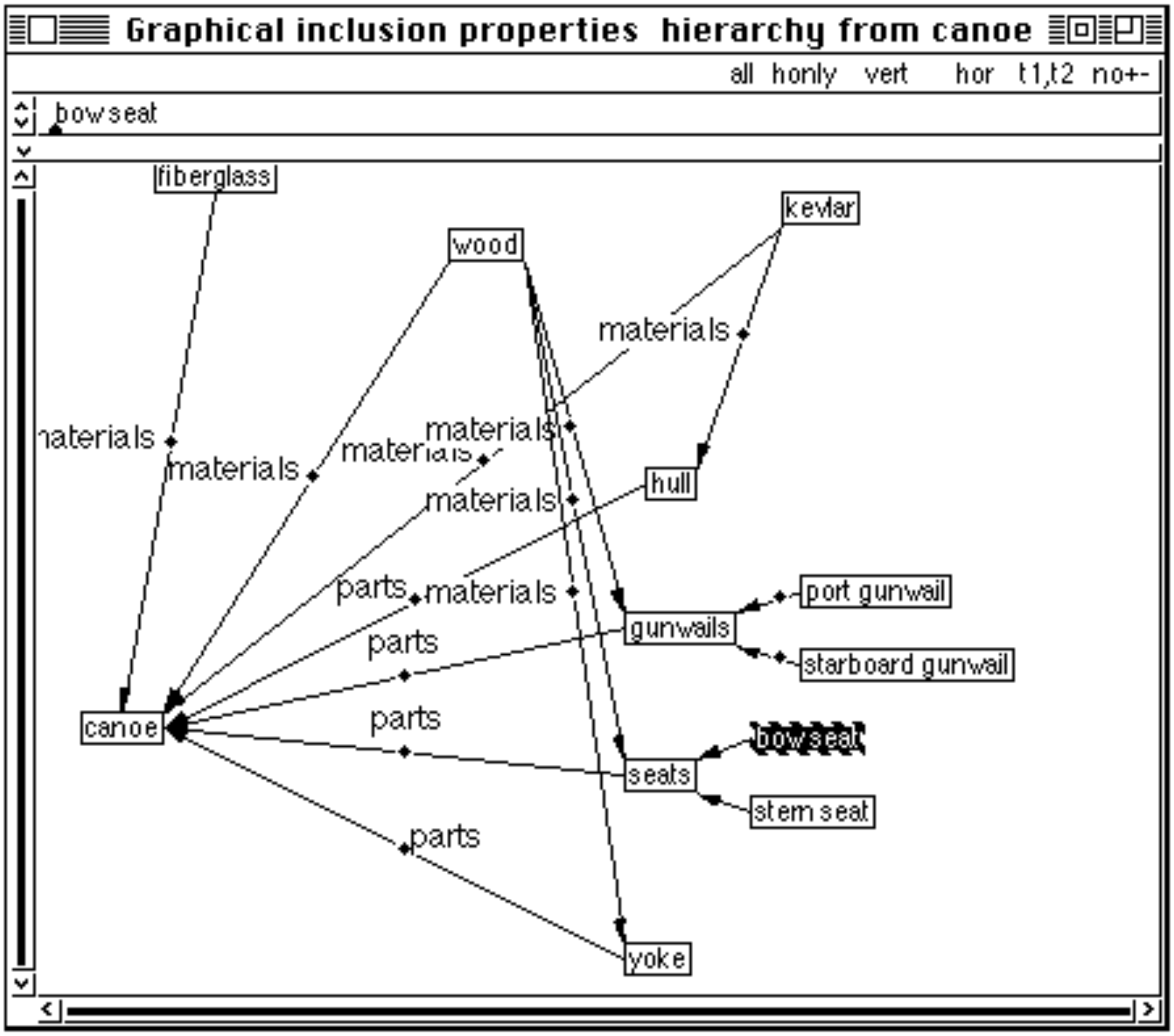


Figure 11: A single-subwindow browser using the graphical interaction paradigm. The graph is showing both the 'materials' and 'parts' relations among several concepts. See section 5.3.18 for information about how to generate such a graph.

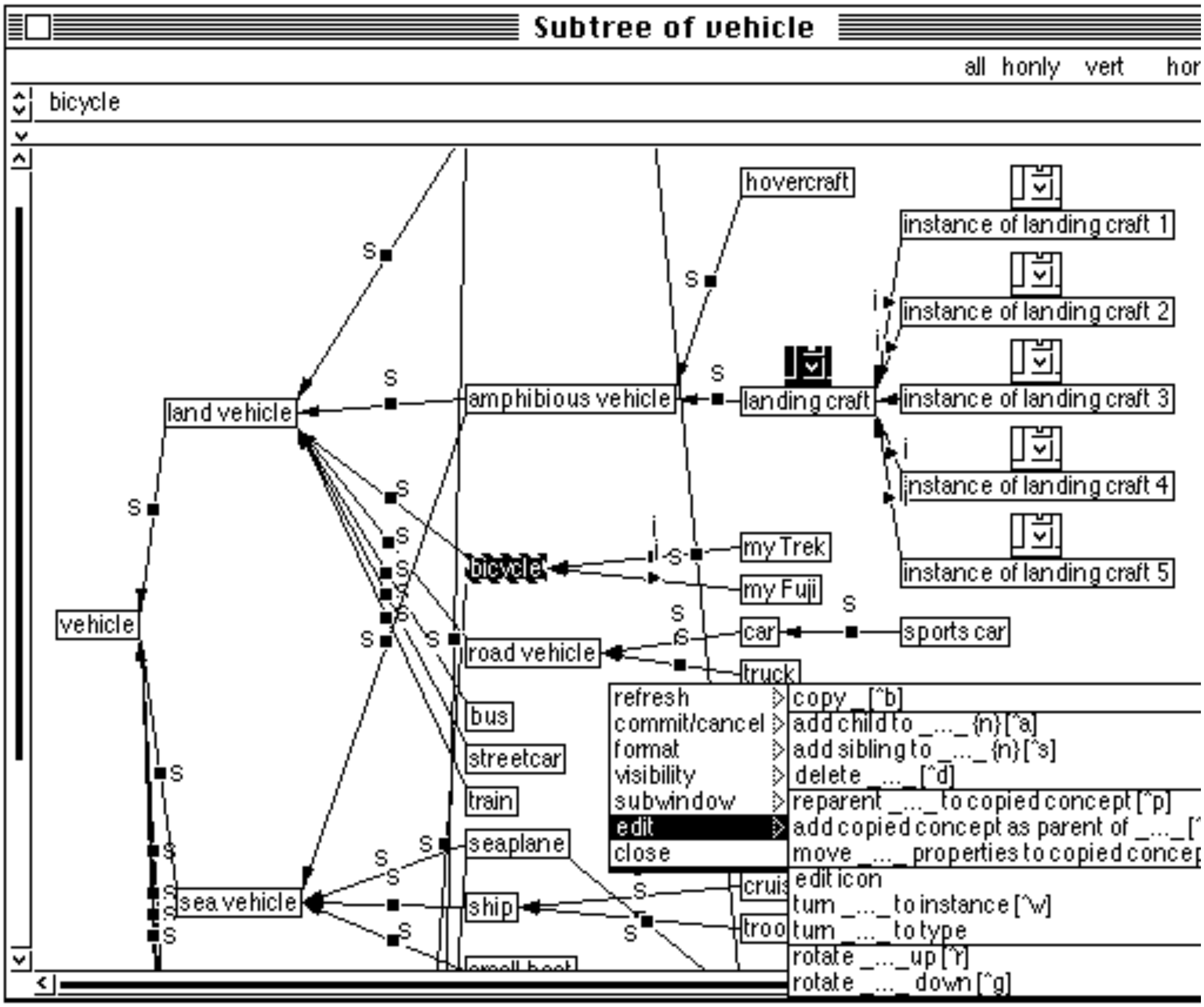


Figure 12: Another single-subwindow browser using the graphical interaction paradigm. The graph is displaying an isa hierarchy. Note the use of icons. Note also that the same edit menu item is available as in the outline paradigm (figure 10)

4.3.1 Components of browser subwindows

Most browser subwindows have the following visual components:

- At the top, an area called the *editing area*, with a single editable text string (i.e. the text of the selected item in the list, graph, etc.). If no item is selected, the string 'No item selected...' appears. The user can type over a concept name and hit enter to change the name (see section 5.3.5). The user can also use the '>' symbl followed by a concept name (with possible wildcards) to cause a search for matching concepts (see section 5.4.1.4).
- Below this (in most cases) an area called the *navigating area*, with a collection (list, graph, etc.) of links and nodes. Links and nodes represent concepts, and are referred to as *items* in the navigating area. It is the navigating area that changes most between paradigms.

Above the editing area of a browser is an *action button bar*. This contains symbols for some of the most frequently useful operations currently available (see section 4.3.4.7). The action button bar serves as the vertical boundary between two browser subwindows.

4.3.2 Selections

The navigating area can have a subset of its items currently *selected*. There are two kinds of selection.

- *multi-selection*: any subset of the navigating area items (either nodes or links).
- *master selection*: a single member of the multi-selection (a node or a link).

(Note that what might be considered a *single selection* of the navigating area items is really a simultaneous master selection and multi-selection, i.e. it is a master selection of a subset containing only one navigating area item.)

In figure 11 and figure 12 there is a single selection ('bow seat' and 'bicycle' respectively). In figure 10, there is a multi-selection ('ship', 'air vehicle' and 'seaplane') of which 'seaplane' is the master selection.

Most operations are performed on the multi-selection; such operations are then performed repetitively on every multi-selected item. Some operations (e.g. renaming) can only be logically performed on a single selected item; in such cases the master selection is used. Each command description indicates if it applies only to the master selection. In menu items, the symbol '..._...' indicates that the command is to be performed on the multi-selection, whereas '_' indicates that the command is to be performed on the master selection only.

The editing area always displays the full text of the master selection. This may not be the same as the ‘name’ of the concept in the navigating area because the editable text item may represent a *formula* for computing the name (see “specifying substitution” in section 5.3).

4.3.3 Nodes and links as concepts

As mentioned above, nodes and links can be selected independently (the multi-selection can contain some of each, although this is rarely useful). Both nodes and links represent concepts, but links always represent statements (e.g. in an isa hierarchy, they represent the statement of ‘subconcept’ in a particular metaconcept).

See the descriptions of different interaction paradigms for a description of how nodes and links appear.

It is possible to prevent links from appearing at all. The environment control panel item 'speed up by removing details' has this effect, as does selecting 'None' within 'Labels/Icons/None' in the graph format control panel.

4.3.4 Generic operations

The following subsections describe commands that apply to all browser interaction paradigms:

4.3.4.1 Selecting one or more items on which to perform operations

• **Selecting a single item**

Simply click on any unselected item with the left mouse button. The item becomes the sole selected item.

The arrow keys can also be used to change the selection. If there is nothing selected, the down arrow key will select the first item, while the up arrow key will select the last item. If there is a single selection, the up and down arrow keys will select the next or previous item, respectively. The left and right arrow keys will select the next parent or child, respectively.

• **Selecting multiple items by dragging**

Select an item and drag (move) the cursor over the other items with the left mouse button held down. The first item selected becomes the master selection. (See also ‘Selecting a subhierarchy’ and ‘Shift-selecting’ below; and ‘Marquee selecting’ in the graphical paradigm).

Multiple items can also be selected by pressing the shift key and clicking on each item to be selected with the left mouse button (see ‘Shift-selecting’ below).

• **Changing the master selection within the multi-selection**

Simply click on any selected item (except the existing master-selection) within the multi-selection. The master selection is moved to the clicked-on item and the multi-selection remains unchanged.

• **Selecting a subhierarchy**

Press the control key when you click on an item with the left button. The item and all its children are selected. This is combinable with dragging and shift-selecting (next point).

• **Shift-selecting (selecting disjoint groups of items)**

Press the shift key when you click on an item. If the item has already been selected, it is de-selected; if the item is not already selected, it is selected. The selection status of other items is unchanged.

When shift-selection is combined with dragging or subhierarchy selection (i.e. both shift and control are pressed), the *entire* dragged-out area or subhierarchy is changed to the opposite selection status (selected or unselected) from the previous status of the first selected item.

Both nodes and links can be selected at the same time by shift-selecting.

• **De-selecting**

Shift-selecting any selected item deselcted it. Clicking on the *master* selection, if it is the only item selected, de-selects it.

Clicking on the master selection, if there are other items in the multi-selection, removes the other multi-selected items from the selection.

As a logical consequence of the above statements, repeatedly clicking on the same item reduces the selection to nothing.

4.3.4.2 Changing the size of a subwindow

Press the control key while clicking on a scroll bar or menu bar that does *not* lie at the edge of a window. The pointer will become a horizontal or vertical 2-way arrow, for scroll bars (e.g. <-->) and menu bars respectively. With the mouse button held down, slide the mouse to reposition the scroll bar or menu bar, and hence resize the two adjacent subwindows separated by that bar.

This functionality applies to all windows, not just browsers.

At the current time, you should avoid dragging a subwindow so far that the subwindow is reduced in size to zero (or less).

4.3.4.3 Editing the text of an item (renaming)

Master-select the item and type a replacement name (the changes will be seen in the editing area). When done hit <return>. There is no need to actually point to the editing area unless fine text editing is required (e.g. if you want to change only a few characters). If detailed editing is required, standard Smalltalk text editing techniques may be used. The 'copy text' and 'paste text' menu items are available in the editing area.

When an item is added, it is given a default name by internal rules for automatic name generation. For example, a new instance concept of the type concept ‘car’ is called ‘instance of car n’, where n is the number of the instance.

Until renaming is performed, this automatic name generation is performed every time the concept is displayed. So, for example, if the concept ‘car’ above were changed to ‘vehicle’, the automatically named instance would change to ‘instance of vehicle n’.

In the user language paradigm (section 4.3.7) the only editing that can be done is renaming. Such renaming has the effect of editing statement values.

- **Accented characters**

It is possible to enter accented characters in concept names (and statement values):

To enter accents in CODE 4, press Control-Z followed by one of the accent characters listed below. Then type the letter that should receive the accent. On Sun machines, the Alternate key can be used in place of Control-Z. Mac users have the option of using the method presented here or using the Mac's built-in accent support, as set out in the Word manual.

ACCENT CHARACTER	ACCENT PRODUCED
' (normal single quote)	accent aigue or acute (á, é, í, ó,ú)
` (back-quote)	accent grave (à, è, ì, ò, ù)
^	accent circonflexe or circumflex (â, ê, î, ô, û)
~	accent tilde (ñ)
- (dash)	accent stroke
u	accent breve
.	dot above
" (double quote)	accent umlaut or diaresis
*	ring above
, (comma)	accent cedille or cedilla
_ (underscore)	accent underline
:	accent double acute
;	accent ogonek
v	accent caron

To enter other special characters, press Control-Z followed by one of the character sequences listed below.

<table>
<tr><td>TM</td><td>trademark</td></tr>
<tr><td>xo</td><td>currency sign</td></tr>
<tr><td>xx</td><td>cross</td></tr>
<tr><td>+-</td><td>plusminus</td></tr>
<tr><td>+d</td><td>lowercase eth, Icelandic</td></tr>
<tr><td>++</td><td>number sign, #</td></tr>
<tr><td>/^</td><td>vertical line, |</td></tr>
<tr><td>/u</td><td>micro sign</td></tr>
<tr><td>/o</td><td>o slash</td></tr>
<tr><td>/O</td><td>O slash</td></tr>
<tr><td>//</td><td>backslash, \</td></tr>
<tr><td>((</td><td>left bracket, [</td></tr>
<tr><td>(-</td><td>left brace, {</td></tr>
<tr><td>))</td><td>right bracket,]</td></tr>
<tr><td>)-</td><td>right brace, }</td></tr>
<tr><td>>></td><td>>></td></tr>
<tr><td>34</td><td>3/4</td></tr>
<tr><td>12</td><td>1/2</td></tr>
<tr><td>ae</td><td>ae dipthong</td></tr>
<tr><td>IJ</td><td>IJ ligature</td></tr>
<tr><td>ij</td><td>ij ligature</td></tr>
<tr><td>kk</td><td>Greenland small K</td></tr>
<tr><td>ss</td><td>German double s</td></tr>
<tr><td>OR</td><td>registered</td></tr>
<tr><td>OC</td><td>copyright</td></tr>
<tr><td>OE</td><td>OE dipthong</td></tr>
<tr><td>oe</td><td>oe dipthong</td></tr>
<tr><td>om</td><td>ohm sign</td></tr>
<tr><td>=Y</td><td>yen sign</td></tr>
<tr><td>=L</td><td>pound sign</td></tr>
<tr><td>-<</td><td>arrow left</td></tr>
<tr><td>-></td><td>arrow right</td></tr>
<tr><td>-D</td><td>D with stroke</td></tr>
<tr><td>-d</td><td>d with stroke</td></tr>
<tr><td>-h</td><td>h with stroke</td></tr>
<tr><td>-H</td><td>H with stroke</td></tr>
<tr><td>-l</td><td>l with stroke</td></tr>
<tr><td>-:</td><td>divided</td></tr>
<tr><td>-L</td><td>L with stroke</td></tr>
<tr><td>-t</td><td>t with stroke</td></tr>
<tr><td>--</td><td>horizontal bar</td></tr>
<tr><td>-T</td><td>T with stroke</td></tr>
<tr><td>_a</td><td>a underlined, feminine ordinal</td></tr>
<tr><td>_o</td><td>o underlined, masculine ord.</td></tr>
<tr><td>|c</td><td>cent sign</td></tr>
<tr><td>|S</td><td>dollar sign</td></tr>
<tr><td>|^</td><td>arrow up</td></tr>
<tr><td>|v</td><td>arrow down</td></tr>
<tr><td>|O</td><td>uppercase thorn, Icelandic</td></tr>
<tr><td>N)</td><td>uppercase eng, Lapp</td></tr>
<tr><td>n)</td><td>lowercase eng, Lapp</td></tr>
<tr><td>'1</td><td>single quote right</td></tr>
<tr><td>``</td><td>double quote left</td></tr>
<tr><td>!s</td><td>section sign</td></tr>
<tr><td>!p</td><td>paragraph sign</td></tr>
<tr><td>!!</td><td>inverted !</td></tr>
</table>

4.3.4.4 Performing a command (in general)

Commands are operations that typically appear on menus.

To perform a nilary command (i.e. one that takes no selected items as arguments), simply request the command using a menu, hot key or action button (one displayed on the screen, see below). Closing a window is an example of performing a nilary command.

To perform a unary command (i.e. one that takes a single set of selected items as arguments), select the items and then request the command as above.

To perform a binary command (i.e. one that takes two sets of selected items as arguments), ‘copy’ the first set of selected items (see below), then select the second set of items and proceed as for a unary operation.

• **Repetition factors (using the ESC key to perform a command several times)**

Some commands optionally take a repetition factor (an integer that tells how many times to repeat the command). There is a special command for specifying a repetition factor; one executes this command prior to executing the command to which the factor is to apply. For example, to add three child concepts to the type concept ‘car’, select ‘car’, press <Esc> + <3>, then press ^a (the hot key for adding child concepts; see below). The result will be three identical child concepts of ‘car’, each called ‘specialized car’. To specify a repetition factor greater than 9 (in other words, with more than one digit) you must press <Esc> before each digit. For example, a repetition factor of 12 is indicated by pressing <Esc> + 1, then <Esc> + 2. Commands that accept a repetition factor have the symbol {n} in their menu item.

4.3.4.5 Performing a command from a menu

Simply press the middle mouse button to bring up the menu, and release the button over the desired menu item. Repeat if the menu item brings up a sub-menu. Figure 13 shows a typical browser main menu.

The menu will change dynamically from time to time as the state of the browser causes certain items to become available or to disappear. For example, the ‘edit’ submenu only appears when items are selected.

Browser menus have the following top-level menu items. Details are deferred to later sections.

refresh: Commands to update the display (figure 13). Normally, refreshes are automatic, but this is not the case if either 1) the environment control panel selection "Speed up by deferring window updates' is set to something other than 'Never' (see section 4.2.1) , or 2) the commit/cancel items 'defer all updates' or 'defer edit updates' are in effect for this window.

refresh pane: Causes the subwindow to redraw itself incorporating any knowledge base updates, but retaining the same layout.

reformat graph (in the graphical paradigm): causes the subwindow to refresh using the automatic layout algorithm.

full update: causes any driving subwindows to refresh first, and then the current subwindow to refresh. This may be needed when window updates are deferred.

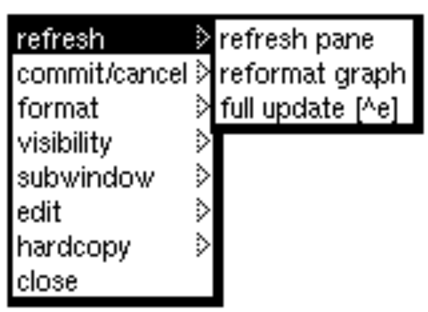


Figure 13: A refresh submenu

commit/cancel: Commands to permanently commit changes to the KB in RAM, or to back them out, or to cause deferral of updates (figure 14). See section 5.2.7 for details of the commands. See also additional commands in the user language interaction paradigm (section 4.3.7)

defer all updates: Has the same effect locally (in this subwindow) as the environment control panel option 'Speed up by deferring window updates/always' has globally. This option overrides the global setting (see section 4.2.1)

defer edit updates: Has the same effect locally (in this subwindow) as the environment control panel option 'Speed up by deferring window updates/on edit' has globally. This option overrides the global setting

defer no updates: Has the same effect locally (in this subwindow) as the environment control panel option 'Speed up by deferring window updates/never' has globally. This option overrides the global setting

file out kb: Saves the knowledge base to disk. If the knowledge base was previously read from disk, the original file is overwritten. If this is a new knowledge base, a file name is composed from the knowledge base name. Default hot key: ^f.

format: Commands to change the look of a browser subwindow (Figure 15). Different in each interaction paradigm. See the sections on the individual interaction paradigms (sections 4.3.5 to 4.3.9). See also 'visibility' below.

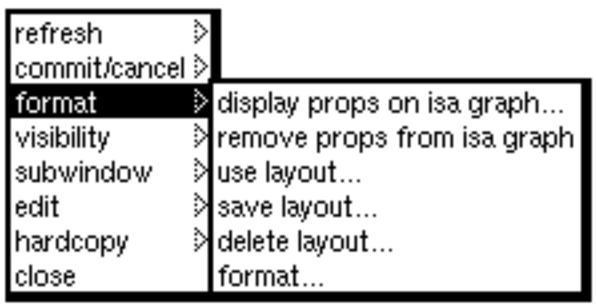


Figure 15: A format submenu (graphical interaction paradigm)

visibility: Commands to alter the mask in order to filter out (or "hide") certain concepts (Figure 16). See section 5.4.1 for command details. Note that nodes that are filtered out can either be not shown (hidden) or shown as mere placeholders (minimized). The format action buttons '+/- and 'no +/-' control this. See section 4.3.4.9.

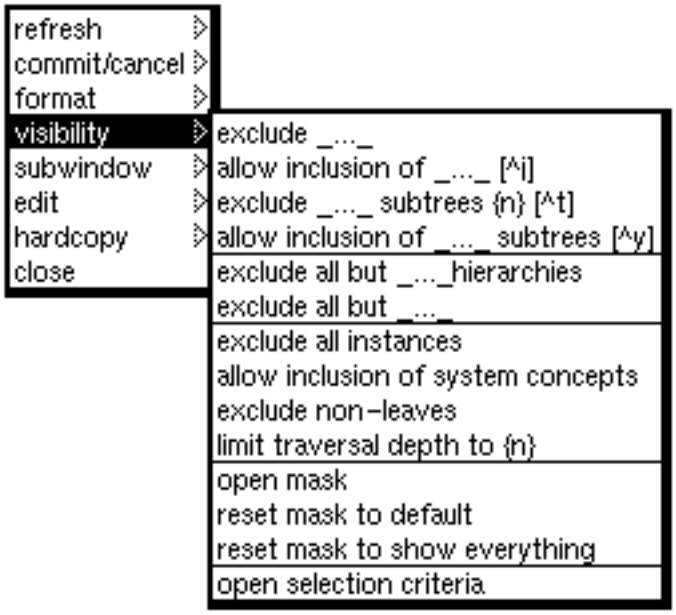


Figure 16: A visibility submenu.

subwindow: Commands to open subwindows whose contents depends on the currently selected item(s) (Figure 17). See section 5.1 for command details.

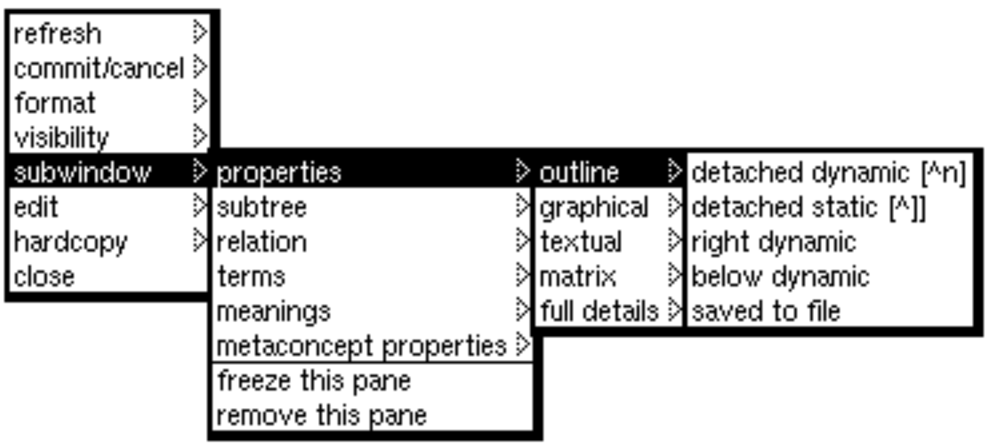


Figure 17: A subwindow submenu.

edit: Commands to change the knowledge base (i.e. modifying, adding or deleting concepts) (Figure 18). See section 5.3 for command details. See also ‘copying items’ below.

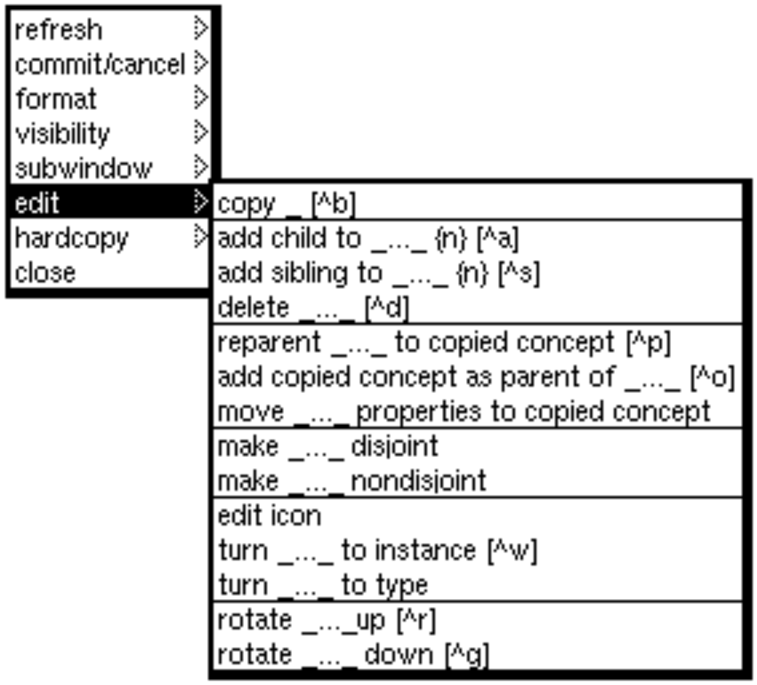


Figure 18: An edit submenu.

hardcopy: Facilities for outputting knowledge to a file (see section 5.4.3). This file can be printed by the user (Figure 19)

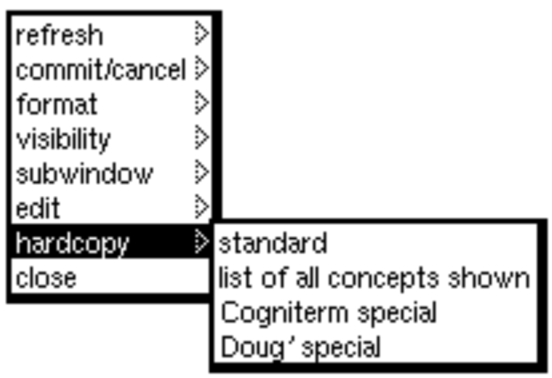


Figure 19: A hardcopy submenu.

inspect/debug: Development and maintenance utilities that allow access to system internals (Figure 20). Only appears when control panel interaction mode is set to 'developer'. Not documented further here.

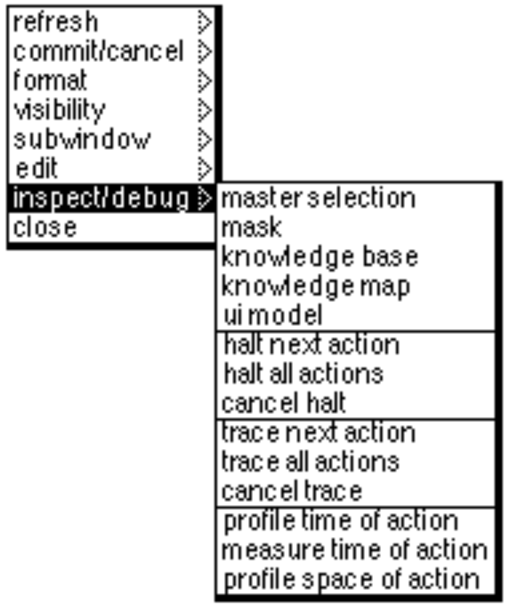


Figure 20: the inspect/debug menu

close: Command to close the current window and any dependent windows. This can also be accomplished using the right-button menu or windowing-system-specific methods.

• **Absent menu commands**

If a documented command does not appear on any submenu, either of the following may be true: (1) the command cannot be performed because it is illogical at the current time (e.g. deleting when nothing is selected, or adding a subproperty to a type concept); (2) the control panel 'user expertise' mode is set too low (e.g. to intermediate or beginner); or (3) the command is only available through keys or action buttons (e.g. specifying a repetition factor — this is fully tailorable, however).

4.3.4.6 Performing a command with a hot key

Most important commands can be performed by typing a two-key combination on the keyboard. This two-key combination, referred to as a “hot key”, combines the control key (^) and the control character (i.e. another key). Both are pressed simultaneously. Figure 21 lists the hot keys, along with the default command they run. These hot keys are fully tailorable: in the Loaded KBs pane of the KBs control panel, select ‘set a hotkey’ from the middle mouse-button menu.

For some hot keys, the control character serves as a mnemonic for the command. Such mnemonics are highlighted in figure 21. Where a control character does not serve as a direct mnemonic for a command, an attempt has been made to use one that is related to a nearby mnemonic key. For example, ^t excludes one or more subtrees, whereas ^y (adjacent on the qwerty keyboard) allows the inclusion of one or more subtrees.

Key	Command	Section
^a	add child to _..._	5.3.1
^b	copy _	4.3.4.8
^c	(reserved for Smalltalk - user interrupt)	
^d	delete _..._	5.3.2
^e	refresh - full update	4.3.4.5
^f	file out kb	5.2.3
^g	rotate _..._ down	5.3.19
^h	(reserved for Smalltalk - delete previous character)	
^i	(reserved for Smalltalk - tab)	
^j	allow inclusion of _..._	5.4.1.1
^k	detached dynamic, graphical, subtree	5.1.3.2
^l	detached dynamic, outline, subtree	5.1.3.2
^m	(reserved for Smalltalk - return)	
^n	detached dynamic, outline, properties	5.1.3.2
^o	add copied concept as parent of _..._	5.3.10
^p	reparent _..._ to copied concept	5.3.9
^q	paste text (all Smalltalk windows)	4.3.7
^r	rotate _..._ up	5.3.19
^s	add sibling to _..._	5.3.1
^t	exclude _..._ subtrees	5.4.1.1
^u	undo	5.3.17
^v	detached dynamic, graphical, relation	5.1.3.2
^w	turn _..._ to instance	5.3.15
^x	copy text (all Smalltalk windows)	4.3.7
^y	allow inclusion of _..._ subtrees	5.4.1.1
^z	enter accented character	4.3.4.3
^\	show collapsed	5.3.4.9
^/	hide collapsed	5.3.4.9
^]	detached static, outline, properties	5.1.3.2

Figure 21: Default assignments of hot keys.

Menus always indicate if a hot key can be used to select commands. The default hot key is also indicated in command descriptions later in this document.

There are two main advantages to using hot keys instead of using menus: (1) time is saved by not moving the hand to the mouse, selecting the submenu and picking the item; and (2) there is a type-ahead buffer for keystrokes. You can therefore perform operations as fast as you can type.

4.3.4.7 Performing a command with an action button.

Action buttons are displayed in a row in each browser's menu-bar (at the top right corner of the appropriate subwindow). Only currently applicable commands are displayed (the commands differ between interaction paradigms). Like hot keys, action buttons are fully tailorable. Currently the buttons are visually presented as a text string (e.g. all honly hiera); however, they can be tailored to display an arbitrary icon.

4.3.4.8 Copying items

Select one or more items and use the 'copy _' command in the edit submenu. The paste buffer will then contain the selected items. The buffer can be used to perform binary commands, or the items it holds can be pasted into text inside or outside Smalltalk.

Default hot key: ^b

Currently, only the master selection is copied.

Note that there is also a 'copy text' command available when editing text. The latter copies the textual representation, not the actual concept.

4.3.4.9 Minimizing and hiding

Nodes that are masked out (see section 5.4.1) can be made completely absent from the navigation area ('hidden'), or else can appear minimized to mere place-holders. The action button command '+/-' requests that place-holders be shown for masked-out items. The action button 'no+/-' requests that masked-out items be completely hidden. Hot keys ^\ and ^/ perform the same functions respectively.

Each interaction paradigm shows minimized items differently (see figures 22 and 24). Minimized items can be selected, although several items may coalesce into one selectable region on the screen: clicking such a region logically multi-selects all the items.

4.3.5 Outline interaction paradigm

The following are the primary distinctive features of the outline interaction paradigm:

4.3.5.1 Format options in outline panes

union/isect *(action buttons)*: ‘Union’ causes property lists to show the properties of all the concepts selected in a driving pane. ‘Isect’ causes property lists to show properties *common* to the concepts selected in the driving pane (i.e. the intersection of properties).

update prop set *(format menu command)*: Looks for a 'properties' dependent subwindow, finds the set of properties selected in it, and causes every concept in the outline interaction paradigm to be followed by a list of statement values (generated from the properties). The statement values are contained in square brackets. 'n/a' indicates that there is no statement for a given property about a given concept (i.e. the property does not inherit to the concept). This option is useful in order to observe how statement values differ throughout a hierarchy.

Note: to return the outline interaction paradigm to its original state, carry out the same process with no properties selected in the ‘properties’ dependent subwindow.

update metaconcept prop set.*(format menu command)*: The same as 'update prop set' except that it looks for a dependent subwindow that is displaying metaconcept properties. The statement values displayed after each concept name are generated using the selected set of metaconcept properties. Note that it is possible to display both a property set and a metaconcept property set.

Format *(format menu command)*: Brings up the dialog box similar to the outline format control panel shown in figure 8. The following points describe elements of the format dialog:

• **Hierarchical/Alphabetical** *(format dialog box item and action button command)*: *Hierarchical* shows the list of concepts as an indented hierarchy. *Alphabetical* shows the concepts in alphabetical order. The latter is useful for rapidly locating a concept in a long list, although the mask (section 5.4.1) provides an alternate mechanism for this. This item can be altered using the dialog box, or the 'alpha/hiera' action button.

• **Icons/None** *(format dialog box item)*: Icons displays visually distinct icons at the beginning of each line that identifies the relationship between the node and its parent (the icons are how links are represented in the outline paradigm). None indicates that these icons are not to be displayed.

• **Get Default From Control Panel/Ok/Apply/Cancel**: These work the same way as in the graphical interaction paradigm's format dialog box (see section 4.3.6)

4.3.5.2 Appearance of items in outline panes

• **Appearance of nodes:**

Nodes are represented as lines of text.

If the outline browser is displaying an isa hierarchy, then each line contains a concept. Indentation signifies the ‘subconcept’ relation.

If the outline browser is displaying statements, then each line represents a statement, and the value of any statement is shown following a colon. Formal statements are shown in bold.

• **Appearance of minimized items (nodes and subtrees):**

Minus (-) signs represent one node or several sibling nodes; plus (+) signs represent one or more entire sub-hierarchies.

Where there are ‘gaps’ in a chain of related nodes, a ‘...’ prefix appears before nodes whose parents are hidden. For example, nodes at indentation levels 1 and 3 are shown, but a child of the level-1 node (which is thus a level-2 node and a parent of the level-3 node) is hidden. In this case, the level-3 node is prefixed with ‘...’.

Where a subtree should appear under multiple parents (e.g. in multiple inheritance), the entire subtree is shown only once. Subsequent occurrences are marked by showing the root only, followed by a ‘...’ suffix.

• **Appearance of links:**

Links are represented as distinctively-shaped bullets (e.g. •). Selecting anywhere in the indentation to the left of a node selects the node’s link to the preceding parent (i.e. the item above at one less indentation level). In the isa hierarchy, the shape of the link icons allows the user to tell whether the associated node is an instance concept, or a type. Property and relation hierarchies use their own distinctive link icons.

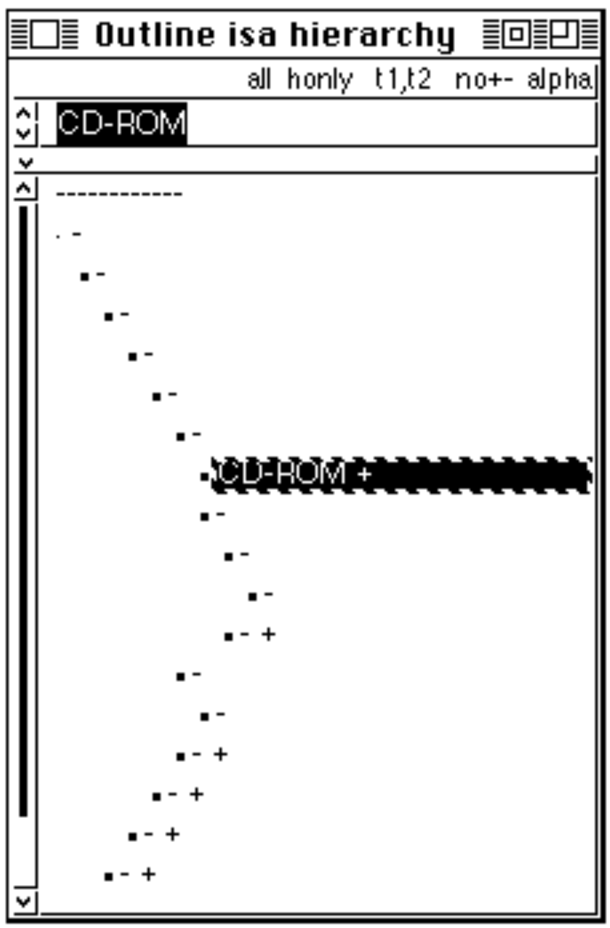


Figure 22: Minimized nodes and subtrees in the outline paradigm.

4.3.6 Graphical interaction paradigm

The following are the primary distinctive features of the graphical interaction paradigm:

4.3.6.1 Navigating, selecting and rearranging a graph

The graph is drawn in a conceptually infinite 2-D plane. You can navigate in this plane to any region whose area equals the subwindow size. The *used area* of the plane is defined as the smallest rectangle that contains all the graph's nodes. The black bars in the scroll-bars indicate what relative portion of the used area is currently visible.

To navigate around the graph, use one of the following methods:

By scroll-bar: Click on and optionally slide either the horizontal or vertical scroll bar.

By background drag: Click on the background and move the mouse while holding down the left mouse button; the graph will move correspondingly.

By hot-scroll: Click on a node and move the mouse. The node will move (see 'rearranging nodes', below). Move the node off the side of the window and the graph will move in the opposite direction. Hot-scrolling also occurs when selecting by marquee (see below).

- **Selecting by marquee**

Press <control> and click on a corner of a rectangular graph region (but not on an item). Move the mouse and release the mouse button at the opposite corner of the region. All nodes in the region are selected. (Note that the control key has a dual use: when the pointer is on a node, the control key selects a subhierarchy; when it is off a node, it commences selection by marquee.)

Marquee selection is combinable with shift-selection, i.e. one can select (or de-select) multiple regions. Press <control> and <shift> when clicking on the first corner.

- **Rearranging nodes**

Move the cursor while clicking on any selected node of a group of selected nodes in order to move *all* selected nodes as a block. This feature is very useful for moving whole subtrees on the graph.

Once a graph is arranged in a suitable manner, you may want to save that arrangement so that it can be restored later. To do that, see section 4.3.6.3.

4.3.6.2 Displaying icons (pictures) at graph nodes

Icons are stored as term concepts just like textual terms. Icons are edited using the 'edit icon' menu item. This prompts the user to drag out an area of the screen that contains a picture the user wants to save as an icon representing the concept.

Since terms are concepts, icons can be given properties. When an instance concept is added to a type concept with an icon (see section 5.3), a default icon is created for that instance concept. This is similar to automatic name generation (see 'editing the text of an item', above).

Warning: At the current time, *color* icons are not supported. Do not attempt to create an icon on a color machine.

4.3.6.3 Format options for the entire graph

display props on isa graph *(format menu command - isa hierarchy only)*: If the current subwindow is driving a property hierarchy, this command shows links (representing properties selected in the dependent subwindow) between selected concepts and other concepts. Repeatedly issuing this command with different concepts or properties selected causes additional links to be drawn. To remove all links, use remove props from isa graph.

remove props from isa graph *(format menu command - isa hierarchy only)*: If property links are being displayed, removes them.

save layout *(format menu command)*: After a user has manually laid out a graph (normaly by adapting the layout provided by the automatic algorithms), this layout may be saved. The 'save layout' menu item prompts the user for the name of the layout (no-name is the default). Any number of layouts may be saved using different names. The 'use layout' command is used to restore a saved layout. Note that if any nodes are selected while the layout is being saved, only their *particular* layout will be saved (if nodes are selected, CODE4 does query the user as to whether that is intentional). Hence, you will normally want to ensure no concept is selected when issuing this command.

use layout *(format menu command)*: Displays a menu of possible layouts. When the user picks a layout, the graph is reformatted. Any nodes that were not on display when the layout was saved are automatically positioned. Note that if any nodes are selected when this command is issued, the saved layout is only applied to those nodes. Hence, you will normally want to ensure no concept is selected when issuing this command.

format...*(format menu command)*: Brings up the dialog box shown in figure 23. This is very similar to the 'graph format' control panel (section 4.2.5). The following points describe elements of the format dialog:

> **Node font** *(dialog box item)*: Allows the user to pick the font that is used to display nodes. The user picks both a font family name and a size. An example of the font appears below the selections.
>
> **Edge font** *(dialog box item)*: The same as 'Node font' except it specifies the font used to render labels on the links (arcs) of graphs.
>
> **Horizontal/Vertical**: *(action button command or dialog box item)*: *Horizontal* aligns the top nodes of the graph along the left of the used area. *Vertical* aligns them along the top. Hierarchies grow to the right or bottom respectively. These two options may be selected from the action buttons (*vert, hor*) found in the top right corner of the subwindow, or from the dialog box brought up from the ‘format’ submenu found in figure 15 (i.e. by selecting ‘Horizontal’ or ‘Vertical’).
>
> **Center/Corner** *(dialog box item)*: *Corner* skews the graph so that the top node is always placed at the top left of the used area. *Center* ensures that the top node is always placed at the centre of the used area (either at the top in vertical mode, or on the left in horizontal mode).
>
> **Layer/Tight** *(dialog box item)*: *Layer* ensures that nodes that are at the same depth in a hierarchy are placed parallel with each other (either horizontally or vertically, depending on mode). *Tight* conserves graph real estate by placing nodes at the next level of the hierarchy as close as possible to their parents.
>
> **Labels/Icons/None** *(dialog box item)*: *Labels* causes link labels and their icons to be displayed. *Icons* removes link labels, but continues to display link icons (link labels can still be seen in the editing area, however). *None* results in links being shown as simple

arrows with no label. If none is selected, the user cannot select links in the graph, only nodes.

Sibling Space *(dialog box item)*: An integer field specifying the minimum number of pixels that can separate sibling nodes.

Child Space *(dialog box item)*: An integer field specifying the minimum number of pixels that can separate a parent node from its child node.

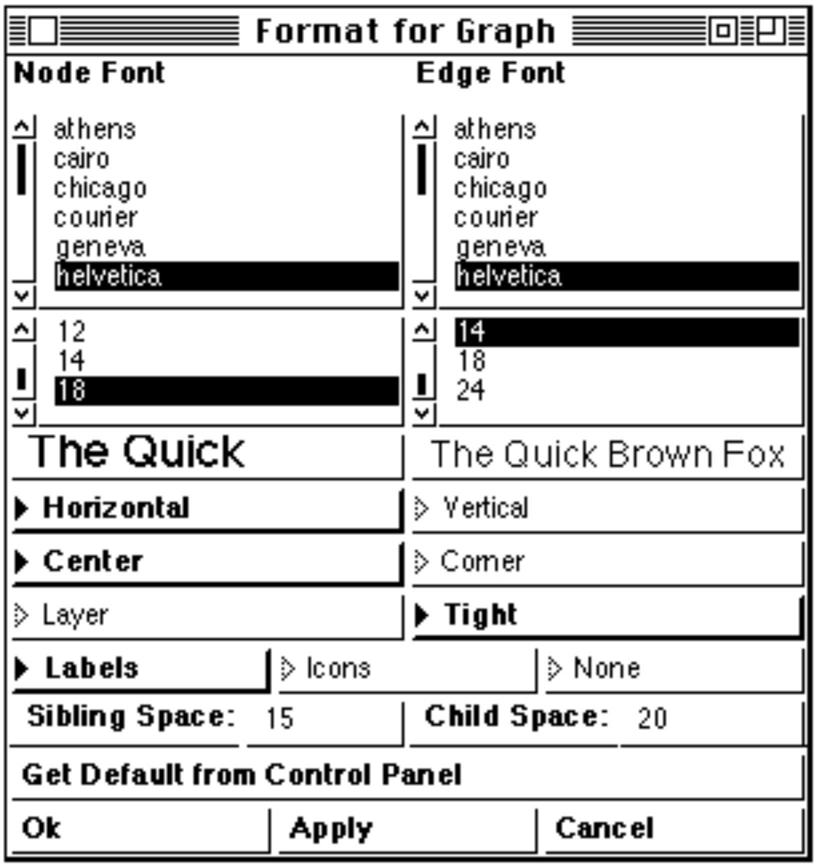


Figure 23: Graphical paradigm format dialog.

Get default from Control Panel *(dialog box item)*: Causes the format to be set to the format specified in the control panel.

Ok *(dialog box item)*: Causes the graph to be reformatted using the format items specified in the dialog box, and then closes the dialog box.

Apply *(dialog box item)*: Causes the graph to be reformatted using the format items specified, but does not close the dialog box. After changing any format item in the dialog box, it is necessary to select 'Ok' or 'Apply' before the new specification takes effect.

Cancel *(dialog box item)*: Closes the dialog box without applying any changes to the graph.

4.3.6.4 Apprarance of items on a graph

• Appearance of minimized items

When '+-' is selected, a small square represents a single node, and a small rectangle represents several sibling nodes. A fan-shape emanating from a node represents an entire sub-hierarchy (see figure 24). When 'no +-' is selected, minimized itmes do no appear on the graph at all.

• Appearance of links:

Links are represented as lines between nodes. Link labels are displayed by default. To select a link, click on the bullet that appears in the centre of the link. When an intermediate node is missing (i.e. excluded), a link between a parent node and a non-immediate descendant is shown by a dotted line.

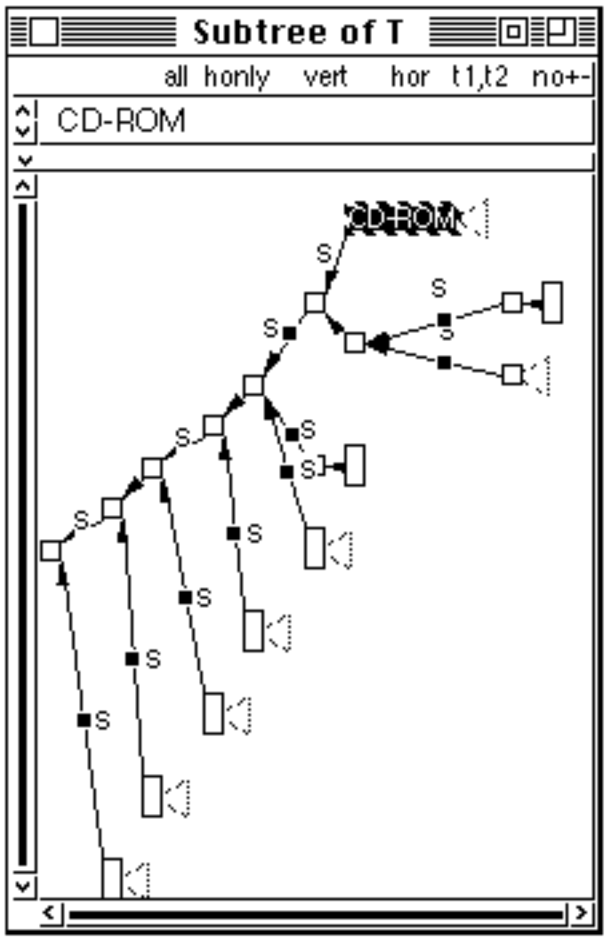


Figure 24: Minimized nodes and subtrees in the graphical paradigm. Compare this with figure 22. Clicking on 'no+-' would remove all but the 'CD-ROM' node, because then masked-out nodes would be completely hidden.

4.3.7 User language interaction paradigm (value panes)

A window using user language interaction paradigm (usually called a value pane) has only a single node (i.e. a statement) and no links. It has no navigation area. The editing area contains the value of the statement expressed as 'user language', i.e. whatever the user wants. For an example, see the bottom-right of figure 48.

4.3.7.1 Format options in value panes

There is no format dialog in this paradigm. Instead, there are two format commands that can be issued using both action buttons and the menu (figure 25)

Expand delegation & symbols/do not expand... *(format menu command and +/- delg action buttons)*: The former shows the result of applying delegation and symbol substitution to Cleartalk expressions. Actually editing this result makes it local, i.e. substitution is no longer dynamic. The latter shows pre-substitution Cleartalk. These two options may be selected from the action button (+/- *delg*) found in the top right corner of the interaction paradigm, or from the 'format' submenu.

See section 5.3.8 for details on the use of delegation.

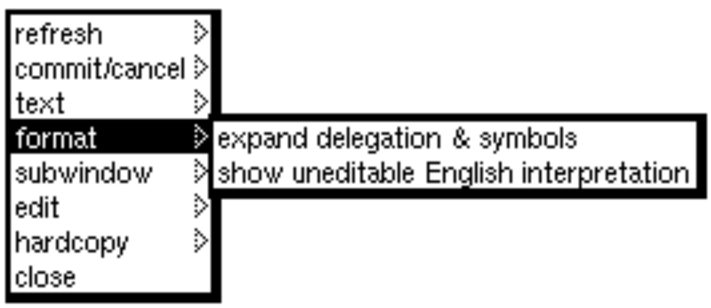


Figure 25: The user language paradigm format menu

Show/remove uneditable English interpretation *(format menu command and +/- Engl action button)* The former shows a full sentence expansion of the statement enclosed in double quotation marks. Edits to this part of the text are ignored. These two options may be selected from the action button (+/- *Engl*) found in the top right corner of the interaction paradigm, or from the 'format' submenu.

See section 4.3.7.4 for details of **update prop set**

4.3.7.2 Text manipulation options in value panes

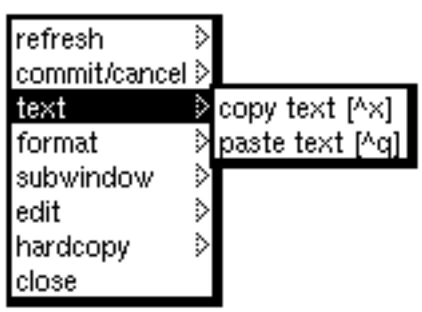


Figure 26: The user language paradigm (value pane) text menu

copy text [^x]: Copies the text into the Smalltalk (and external) text buffers. Not to be confused with 'copy _' (see section 4.3.4.8) which copies a concept. (Note for Macintosh users: we could not use ^c because this is assigned to the Smalltalk function ‘user interrupt’).

paste text [^q]: Pastes whatever is in the Smalltalk (or external) text buffer. If the shift key is held down simultaneously, a menu will appear showing the last few copied items. 'paste text' will paste text if text was copied, but it will also paste a textual representation of a concept if a concept was copied using 'copy _'.

4.3.7.3 Edit options in value panes

accept *(edit menu command and enter key):* Causes edited text to be ‘accepted’, i.e. entered into the knowledge base. This can also be accomplished by hiting ‘enter’ unless the ‘-crAcc’ action button command has been issued (see below). See also ‘tab completion’ below for a more intellegent mechanism of entering a value fast.

If the value entered matches the name of a concept, and if cleartalk parsing is switched on, then an actual reference to the concept is made. Otherwise, whatever is typed is entered into the knowledge base as informal text. To find out whether the text is formal or informal, use ‘+Engl’ format option – the word ‘informal’ will appear if the value contains informal text. Another way to determine whether the value is formal is to look at the property list (which usually is situated above the value pane): Bold items are formal.

tab completion *(using the tab key only)*: This mechanism helps users to enter knowledge rapidly into the knowledge base. To use it, enter one or more characters of a known concept name into the value pane and then press ‘tab’. If there is only one concept whose initial characters match what is entered, the concept name will be completed and entered into the knowledge base. If there is no matching concept then, the user will be notified by a simple error message.

If there are multiple matching concepts, then a menu is presented. The user can choose from the menu – the resulting concept is entered into the knowledge base. This is probably the fastest way to edit values so they refer to other concepts.

cancel *(edit menu command)*:This causes entered text to be reset to its original state, as long as ‘accept’ or ‘tab completion’ has not been successfully performed.

parse as cleartalk *(edit menu command)*: Causes the value to be explicitly parsed. This may be necessary if the default parsing setting is off (see the environment control panel or the ‘subsequently parse as cleartalk’ action button below). It may also be necessary if knowledge base changes that have been made would result in a different parse than was previously computed.

set value to copied concept *(edit menu command):* Explicitly sets the statement value (which must be a concept) to be the contents of the copy buffer. This is useful in the following circumstances: 1) When default Cleartalk parsing is switched off; 2) When the results of Cleartalk parsing would be ambiguous (e.g. several concepts with the same name).

add copied concept to value *(edit menu command):* Adds the concept (which *must* be a concept) in the copy buffer to the existing concept in the value. If there was no previous concept in the value, or if the value was informal, then this has the same effect as ‘set value to copied concept’. Note that if the value is inherited, this command causes a new local override of the inherited value that includes the inherited concepts plus the newly added one: If the inherited concepts are later changed, the local override will not be affected.

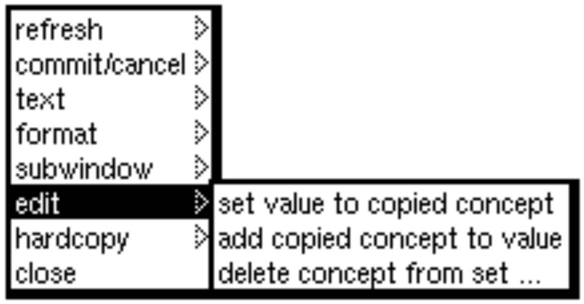


Figure 27: The user language paradigm edit menu

delete concept from set *(edit menu command):* Allows selective editing of a set of concepts in a statement value. Displays a popup menu from which the user can choose a concept to delete. This option is particularly useful when Cleartalk parsing gives ambiguous results (due to several concepts having the same name)

carriage return accepts *(+/-crAcc action button):* +crAcc causes subsequent use of the carriage return to enter (i.e. accept) whatever is typed as an update to the knowledge base. -crAcc causes enter to be just a character in the value. In the latter case, the 'accept' command or 'tab' command must be used to actually update the knowledge base.

subsequently parse as cleartalk *(+/- parse action button):* +parse causes subsequent entries using 'accept' to be interpreted as concept names (values that have concept names or 'set of' followed by a concept name are said to be *formal*. -parse causes subsequent entries to be *informal*.. If parsing is not switched on, then the 'parse as cleartalk' mechanism can be used to explicitly make a vallue formal.

4.3.7.4 Displaying and editing facets in the value pane

Normally, only the value of one statement is displayed in the value pane (perhaps also with the uneditable English interpretation – see section 4.3.7.1).

It is also possible to show and edit the values of other facets directly in the value pane. To do this:

• Open a properties subwindow from the value pane – this will display *facets* of the statement.

• Select the facets you are interested in displaying in the value pane.

• Issue the 'update prop set' command in the value pane

The selected facets will appear in the value pane, separated by the '%' character. Any facet can be edited. Note: Editing facets in the value pane has the same effect as editing them in a properties subwindow that us showing the facets individually.

4.3.8 Matrix interaction paradigm

The matrix paradigm allows the editing of inherently 2-dimensional data; for example concepts may be displayed on one axis and properties on another. Cells in the matrix contain values of the statement involving a given concept and property

In other words, it allows the user to view and compare properties for two or more concepts in tabular format (similar to a spreadsheet). These concepts may be siblings, coordinates or arbitrarily chosen. The properties of a singly-selected concept may also be viewed.

The most typical uses of the matrix would be to either (1) view the properties of coordinates or siblings of one selected concept or (2) view the properties of 2 or more arbitrarily chosen concepts.

• **How the matrix paradigm works**

A concept (or characteristic) is selected in either a hierarchical outline browser or graphical browser. (Although more than one concept may be initially selected, this document assumes that only *one* concept is selected from the browser.) The matrix is generated by selecting **subwindow>>properties>>matrix>>detached dynamic** (or detached static, right dynamic or right static) from the middle mouse button.

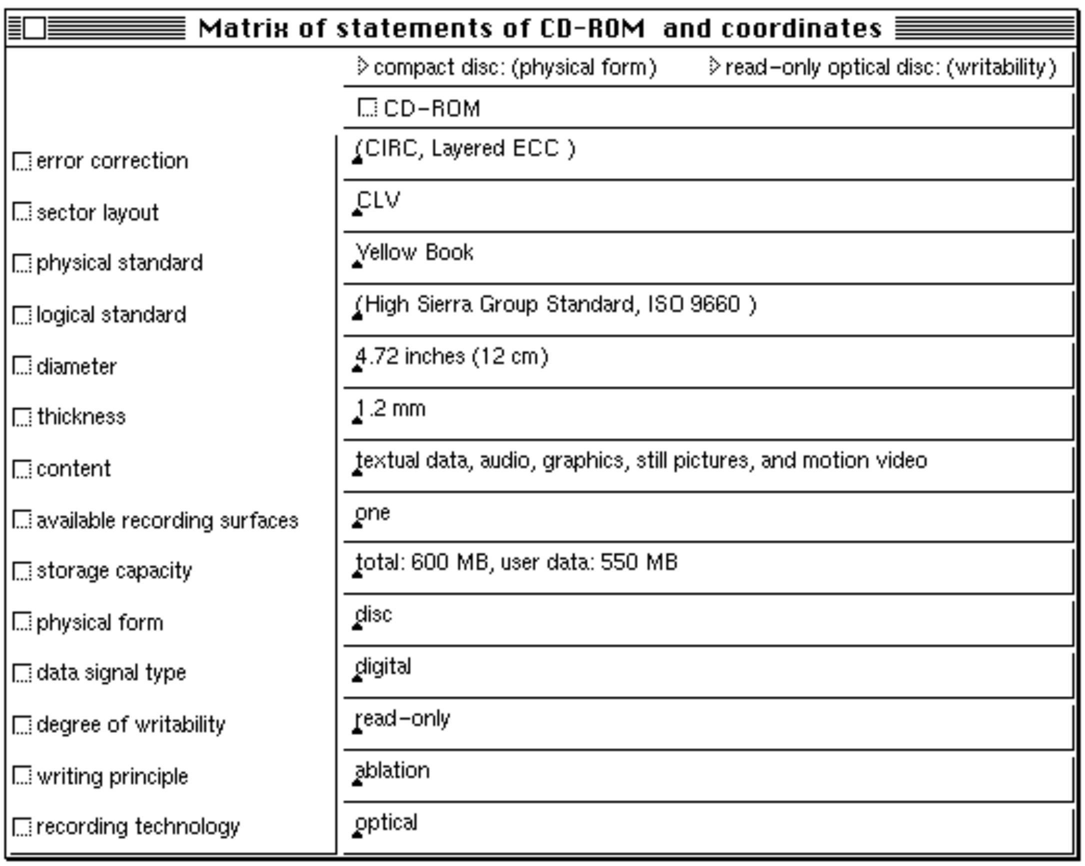


Figure 28. Typical matrix interaction paradigm, inital phase.

The typical matrix initially shows the selected concept and its properties. It may also show the concept's parent(s) and dimensions.[1] As figure 28 shows, the various elements are displayed as follows:

- •along the top: superconcept(s) (dimension)
    - •under the superconcept(s): selected concept. Note that the selected concept is reflected in the matrix label as well
- •along the left side: property names of the selected concept
- •within the remaining cells: property values

4.3.8.1 Format options in the matrix paradigm

The default format of the matrix is set in the **matrix format control panel** (figure 29). As seen in the figure below, it has three main parameters which can be set in a variety of ways: (1) subjects; (2) properties and (3) property computation operator (PCO). These three parameters are all taken into account by the matrix when it computes what will appear in its various cells at various stages. The order in which they are taken into account is (1), (3), (2); i.e. the subject parameter is considered first, then the PCO parameter, and lastly, the properties parameter. This format can be changed at any time (see 'Changing the default format within a matrix', below).

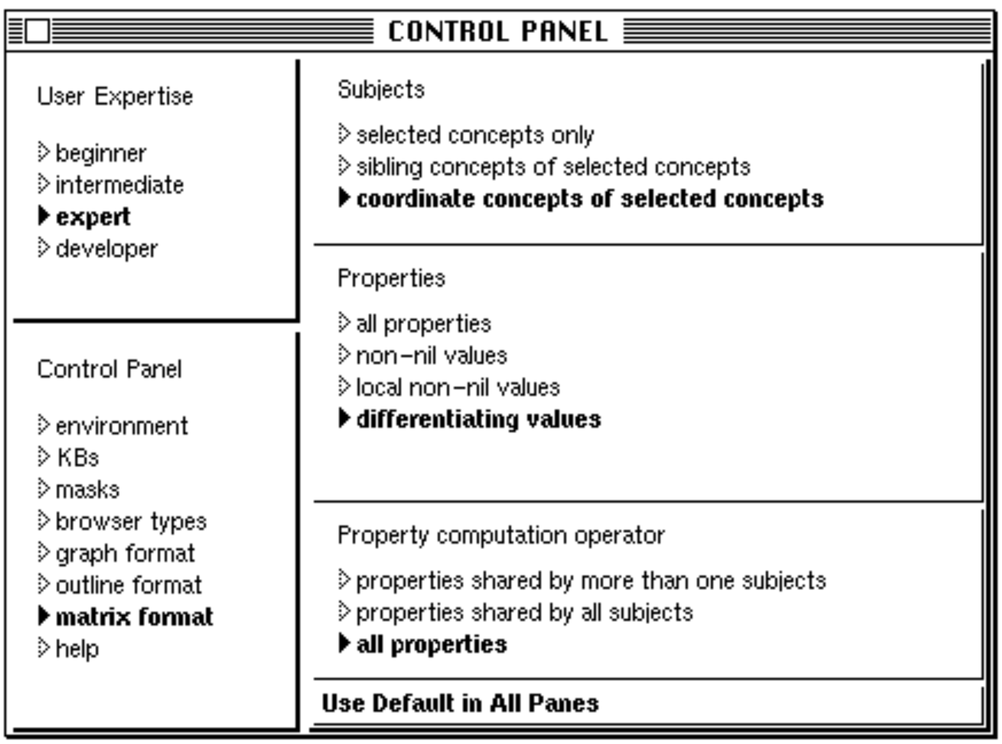


Figure 29. Matrix Format in Control Panel.

The various matrix format options are explained below:

---

[1] Whether or not parents and dimensions show up in the matrix depends on the selected Subjects format (see below).

1. Subjects

•selected concepts only: allows user to view properties of the concept(s) selected in the browser

•sibling concepts of selected concept: allows user to view the properties of **all** the subconcepts of the selected concept's superconcept when the superconcept is clicked on in the matrix

•coordinate concepts of selected concept: allows user to view the properties of all the subconcepts of the selected concept's superconcept within the same dimension when the superconcept is clicked on in the matrix

2. Properties

•all properties: shows all properties, whether they have a value or not

•non-nil values: shows all properties with existing values

•local non-nil values: shows properties whose value has been changed at that particular concept level

•differentiating values: shows properties whose values are different for one or more of the selected concepts (only valid where 2 or more concepts are being compared)

3. Property computation operator (PCO)

•properties shared by more than one subject (not relevant for single-select concepts): compares properties shared by more than one of the selected subjects

•properties shared by all subjects (not relevant for single-select concepts): compares properties shared by all selected subjects

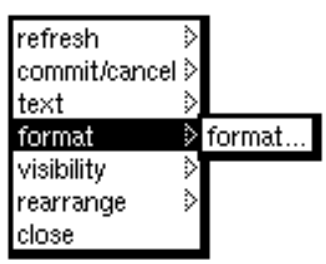
•all properties: compares all properties, regardless of subject

Changing the default format within a matrix

As with the other interaction paradigms, the format can be changed within an opened matrix at any time by one of two ways:

(1) In the matrix format control panel, change the desired parameter(s) and select **use default in all panes**: this allows you to make the new format effective for all currently opened matrices.

(2) Within an opened matrix, select **format>>format** to bring up a dialog box with the same parameters listed as in the matrix format control panel.



After making the desired changes, select either

•**apply:** to make changes to the matrix and keep the format box open

•**ok**: to make changes to the matrix and close the format box

•**use default in all panes**: to make changes effective for all opened matrices

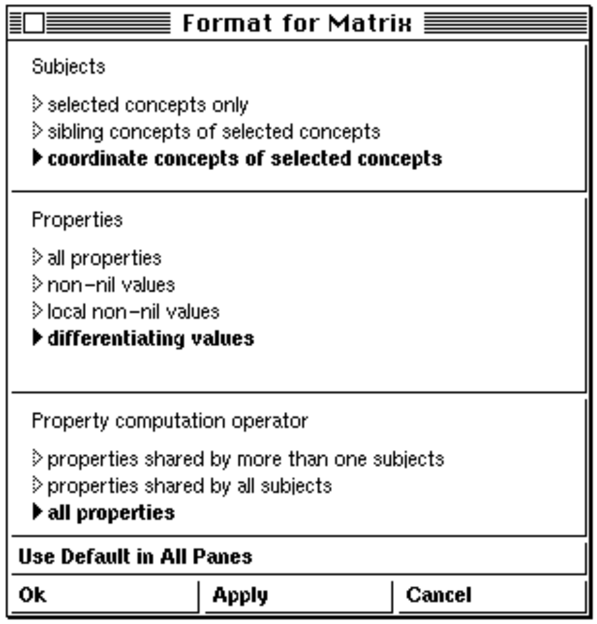


Figure 30. Matrix Format, selected from within a matrix.

4.3.8.2 Matrix example 1: Statements about a concept

This section and the next present some typical “matrix scenarios” for one concept (CD-ROM). Each scenario uses different matrix formats.

First, ensure that the default formats are set to those specified in figure 31.

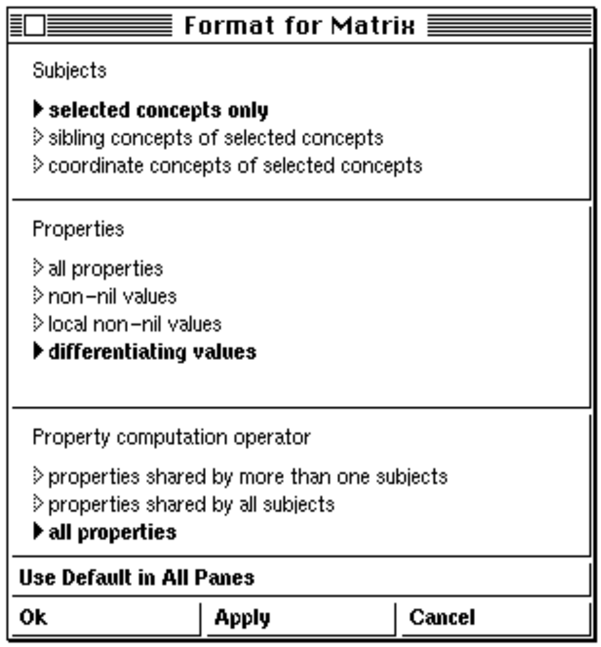


Figure 31. Default format (1).

(a) Select CD-ROM from an outline isa hierarchy, and then open a detached dynamic matrix. The Matrix of statements of CD-ROM appears, as seen below.

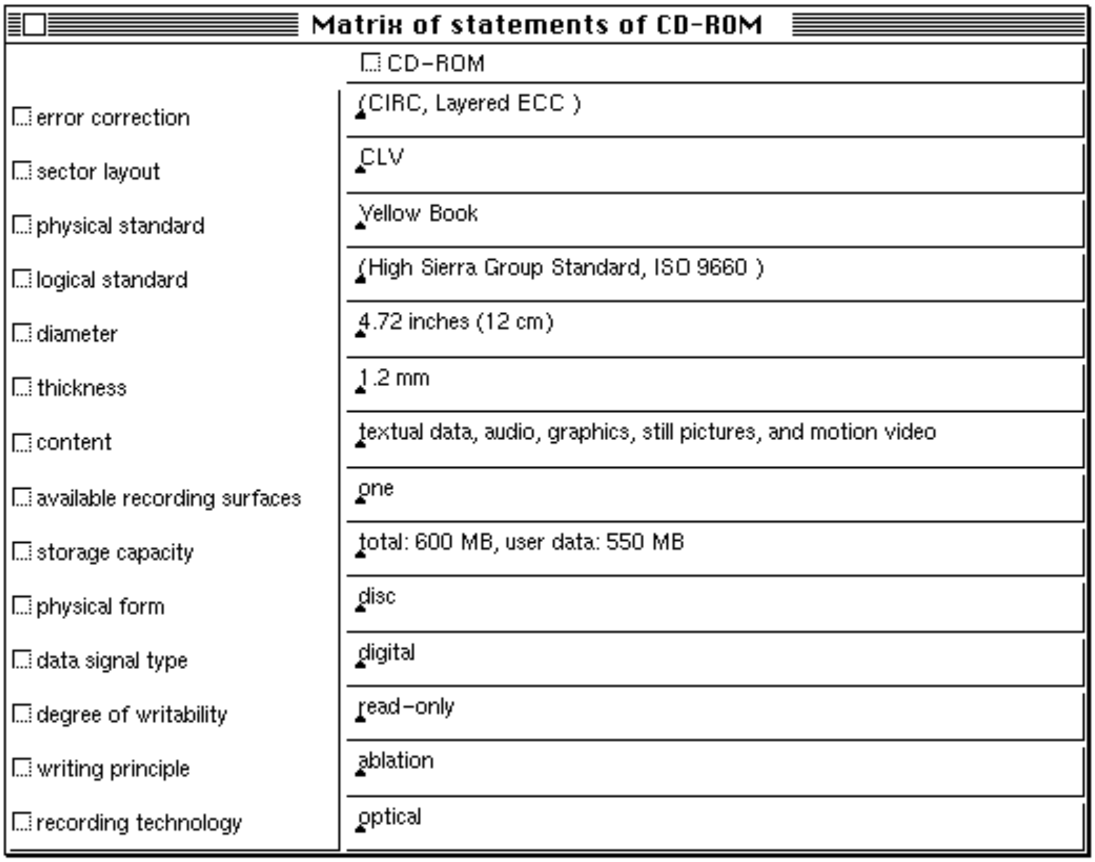


Figure 32. Matrix of statements of CD-ROM.

Note:

• at the top: because it has been specified (via the matrix format) that only the selected concept be viewed, no parents are visible. (Note: it is through the parent that coordinate concepts and sibling concepts can be made to appear in the matrix.)

• along the side: property names

• in the remaining cells: property values

*although the default parameter for properties is ‘differentiating values’, this obviously does not apply when only one concept is selected. Instead, properties with non-nil values appear.

Usefulness of format

This format is useful for comparing two or more arbitrarily selected concepts. If the user decides to compare the already selected concept with another concept, she can simply shift-select a second concept in the outline isa hierachy or on the graph; because this is a dynamic matrix, the second concept will automatically appear.

If the user decides that she would like to compare properties for coordinate concepts or sibling concepts of CD-ROM, she can open a format dialog box (**format>>format** from the middle mouse button menu) and make the appropriate format changes.

(b) To close the matrix, the user clicks on ‘close’ from the right mouse-button menu.

4.3.8.3 Matrix example 2: Statements about a concept and siblings

First ensure that the default formats are set to those specified in figure 33.

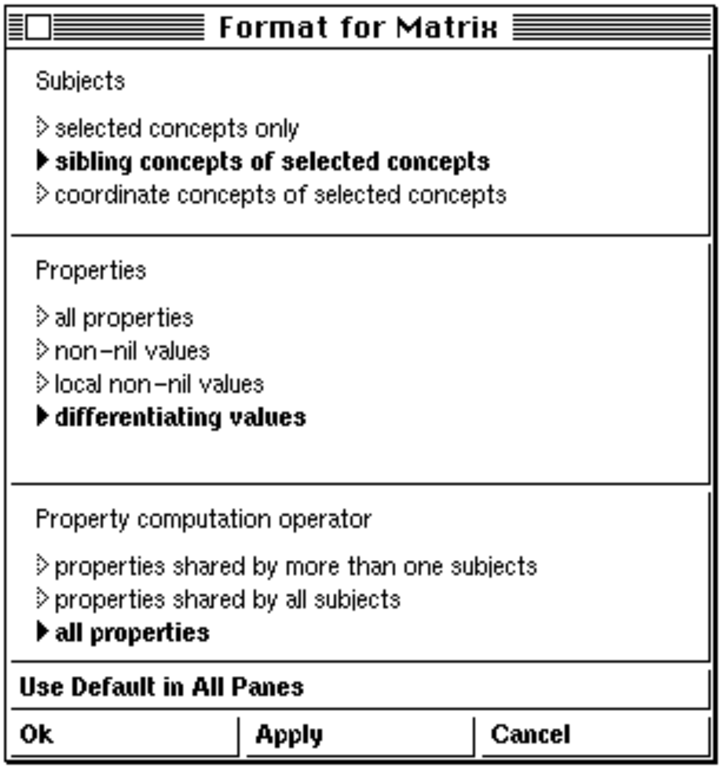


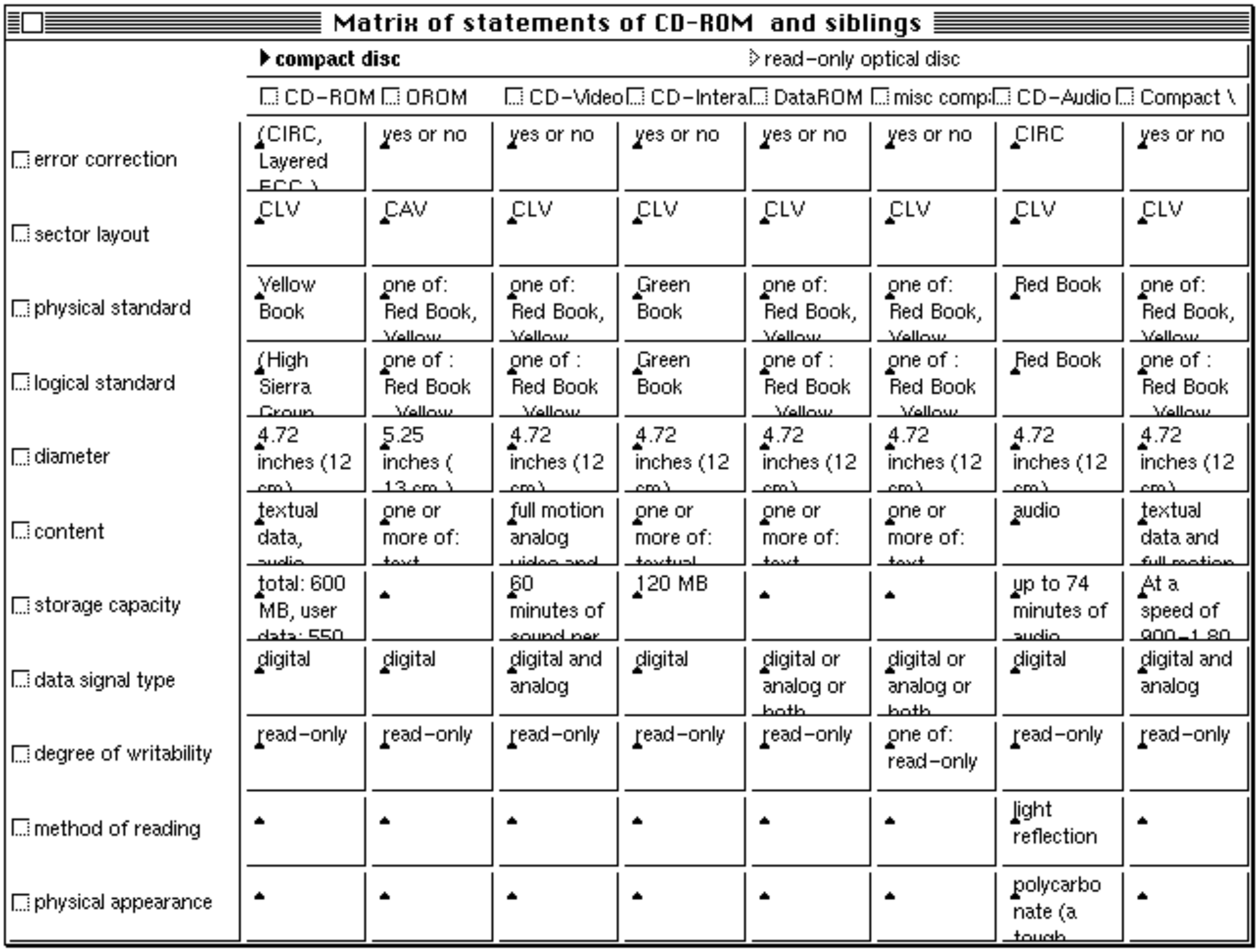
Figure 33. Default format (2).

(a) Select CD-ROM from an outline isa hierarchy and then open a detached dynamic matrix. The Matrix of statements of CD-ROM and siblings appears, with only CD-ROM showing.

Note:

- at the top: because the user has requested (via the matrix format) that sibling concepts of the selected concept be viewed, the parents of CD-ROM are now visible, i.e. compact disc and read-only optical disc. By clicking on a given parent, the user will be able to see the siblings of CD-ROM under that parent.

(b) To see the sibling concepts of CD-ROM under compact disc (i.e. all the subconcepts of compact disc regardless of dimension), the user clicks on 'compact disc'. The sibling concepts of CD-ROM under compact disc and their differentiating properties (predicate along the left side, values in the remaining cells) appear, as seen below.



Matrix of statements of CD-ROM and siblings

▸ compact disc ▹ read-only optical disc

	CD-ROM	OROM	CD-Video	CD-Intera	DataROM	misc comp	CD-Audio	Compact \
error correction	(CIRC, Layered ECC)	yes or no	yes or no	yes or no	yes or no	yes or no	CIRC	yes or no
sector layout	CLV	CAV	CLV	CLV	CLV	CLV	CLV	CLV
physical standard	Yellow Book	one of: Red Book, Yellow	one of: Red Book, Yellow	Green Book	one of: Red Book, Yellow	one of: Red Book, Yellow	Red Book	one of: Red Book, Yellow
logical standard	(High Sierra Group	one of : Red Book , Yellow	one of : Red Book , Yellow	Green Book	one of : Red Book , Yellow	one of : Red Book , Yellow	Red Book	one of : Red Book , Yellow
diameter	4.72 inches (12 cm)	5.25 inches (13 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)
content	textual data, audio	one or more of: text	full motion analog video and	one or more of: textual	one or more of: text	one or more of: text	audio	textual data and full motion
storage capacity	total: 600 MB, user data: 550		60 minutes of sound per	120 MB			up to 74 minutes of audio	At a speed of 900–1 80
data signal type	digital	digital	digital and analog	digital	digital or analog or both	digital or analog or both	digital	digital and analog
degree of writability	read-only	read-only	read-only	read-only	read-only	one of: read-only	read-only	read-only
method of reading							light reflection	
physical appearance							polycarbonate (a tough	

Figure 34. Matrix of statements of CD-ROM and siblings, under superconcept compact disc.

Note that compact disc is now highlighted (i.e. in bold text), so that the user is always aware of which superconcept is dictating the siblings being shown.

(c) To see the sibling concepts of CD-ROM under read-only optical disc (i.e. all the subconcepts of read-only optical disc regardless of dimension), the user would click on 'read-only optical disc'. After a brief delay, the sibling concepts of CD-ROM under read-only optical disc and their differentiating properties (names along the left side, values in the remaining cells) would appear.

(d) To close the matrix, the user clicks on close from the right mouse-button menu.

4.3.8.4 matrix example 3: Statements of a concept and its coordinates

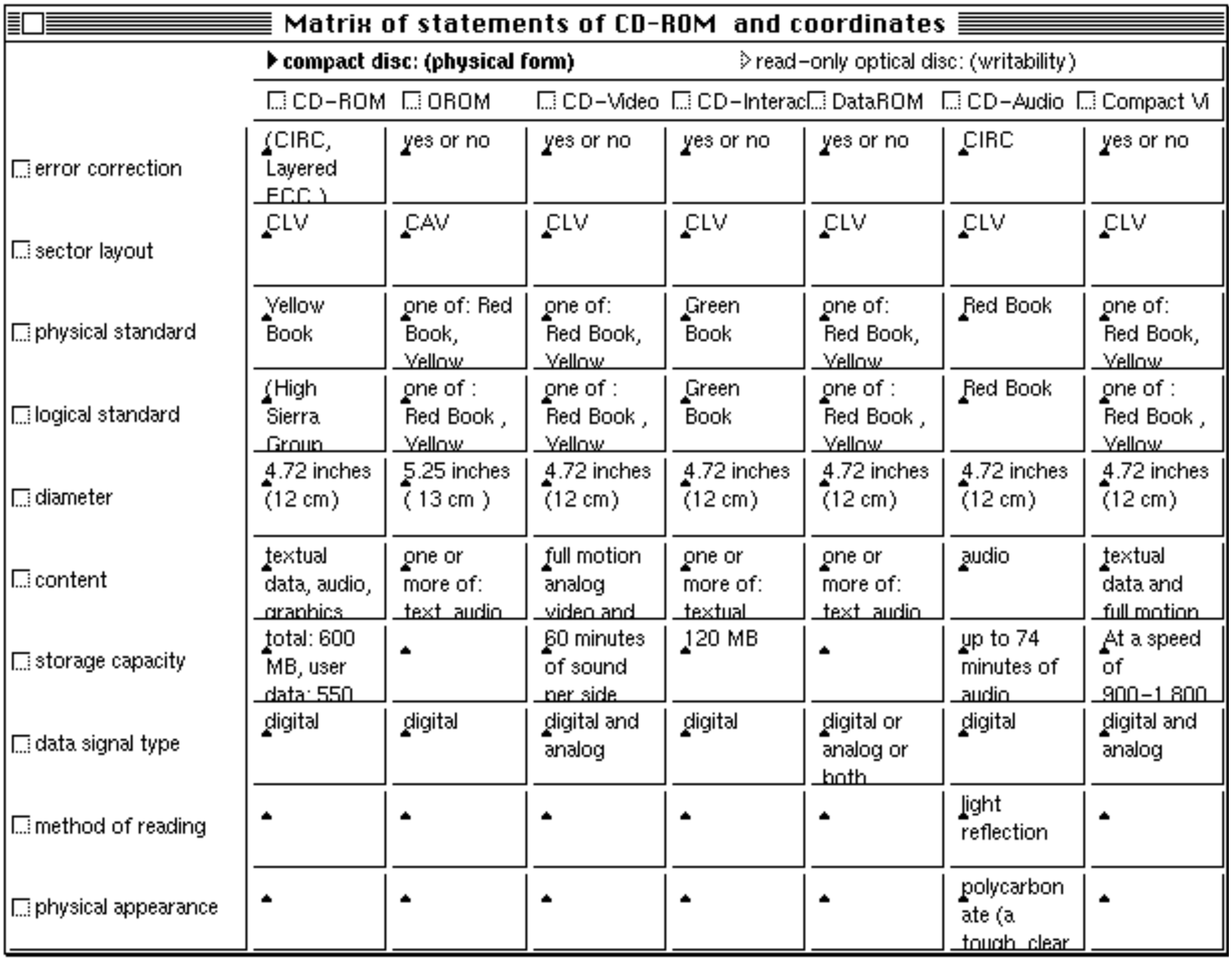
First ensure that the default formats are set to those specified in figure 35.

(a) Select CD-ROM from an outline isa hierarchy and open a detached dynamic matrix. The Matrix of statements of CD-ROM and coordinates appears, with only CD-ROM showing, along with its two parents, compact disc and read-only optical disc, and their respective dimensions.

Note:

- at the top: because the user has requested (via the matrix format) that coordinate concepts of the selected concept be viewed, the parents of CD-ROM *plus their dimensions* are now visible, i.e. compact disc (physical form) and read-only optical disc (writability). By clicking on a given parent, the user will be able to see the coordinates of CD-ROM under that parent.

(b) To see the coordinate concepts of CD-ROM under compact disc (physical form) (i.e. all the subconcepts of compact disc within the dimension ‘physical form’), the user clicks on ‘compact disc (physical form)’. After a brief delay, the coordinate concepts of CD-ROM under compact disc and their differentiating properties (names along the left side, values in the remaining cells) will appear, as seen below.



Matrix of statements of CD-ROM and coordinates

compact disc: (physical form) | read-only optical disc: (writability)

	CD-ROM	OROM	CD-Video	CD-Interac	DataROM	CD-Audio	Compact Vi
error correction	(CIRC, Layered ECC)	yes or no	yes or no	yes or no	yes or no	CIRC	yes or no
sector layout	CLV	CAV	CLV	CLV	CLV	CLV	CLV
physical standard	Yellow Book	one of: Red Book, Yellow	one of: Red Book, Yellow	Green Book	one of: Red Book, Yellow	Red Book	one of: Red Book, Yellow
logical standard	(High Sierra Group	one of : Red Book , Yellow	one of : Red Book , Yellow	Green Book	one of : Red Book , Yellow	Red Book	one of : Red Book , Yellow
diameter	4.72 inches (12 cm)	5.25 inches (13 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)	4.72 inches (12 cm)
content	textual data, audio, graphics	one or more of: text, audio	full motion analog video and	one or more of: textual	one or more of: text, audio	audio	textual data and full motion
storage capacity	total: 600 MB, user data: 550		60 minutes of sound per side	120 MB		up to 74 minutes of audio	At a speed of 900–1,800
data signal type	digital	digital	digital and analog	digital	digital or analog or both	digital	digital and analog
method of reading						light reflection	
physical appearance						polycarbon ate (a tough, clear	

Figure 36. Matrix of statements of CD-ROM and coordinates, under superconcept compact disc (physical form).

Note that compact disc is now highlighted (i.e. in bold text), so that the user is always aware of which superconcept is dictating the coordinates being shown.

(c) To see the coordinates of CD-ROM under read-only optical disc (writability) (i.e. all the subconcepts of read-only optical disc in the dimension writability), the user would click on 'read-only optical disc'. After a brief delay, the coordinate concepts of CD-ROM under read-only optical disc (writability) and their differentiating properties (names along the left side, values in the remaining cells) would appear.

(d) To close the matrix, the user clicks on 'close' from the right mouse-button menu.

4.3.8.5 Manipulating matrix information

Once the matrix is opened and the arbitrarily selected, sibling or coordinate concepts and their properties are displayed, the user may manipulate the matrix information in a variety of ways. All menu commands are accessed from the middle mouse-button (note: to access the middle mouse button, the cursor must be in the property value cells area).

4.3.8.6 Modifying statement values in matrix cells

Statement values in cells can be modified within the matrix. Any modifications made in the matrix are instantly reflected in the list of conceptual properties (if one is open), and vice-versa.

Properties cannot be added directory to the matrix, but if a property is added to a concept (e.g. in an outline isa hierarchy with editable statements), it will automatically appear (as long as the selected matrix format dictates that this property *should* appear on the matrix).

Any statement value modification can be accepted (i.e. entered into the knowledge base) by hitting <return>. Note that several statements can be changed before hitting <return> to accept all of them at once; this is the most time-saving way to edit using the matrix.

Copy text (^x or text>>copy text) and paste text (^q or text>>paste text) functions are also available. Text may be copied from elsewhere in the KB and pasted into the matrix, and vice-versa.

4.3.8.7 Masking out concepts and properties from a matrix

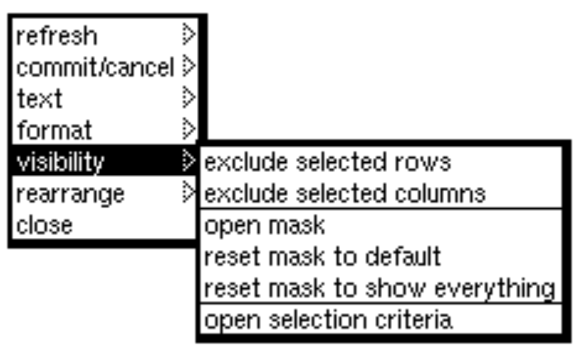
Both concepts and properties in the matrix can be masked out to make it easier for the user to focus on specific information. For more information about the mask, see sections 4.4 and 5.4.1.

a) To mask out concepts

• Select the concepts to be masked out by clicking on the concept name (it will become bolded when selected)

• Select **visibility>>exclude selected columns**

The selected concepts will then be masked out (disappear) from the matrix.



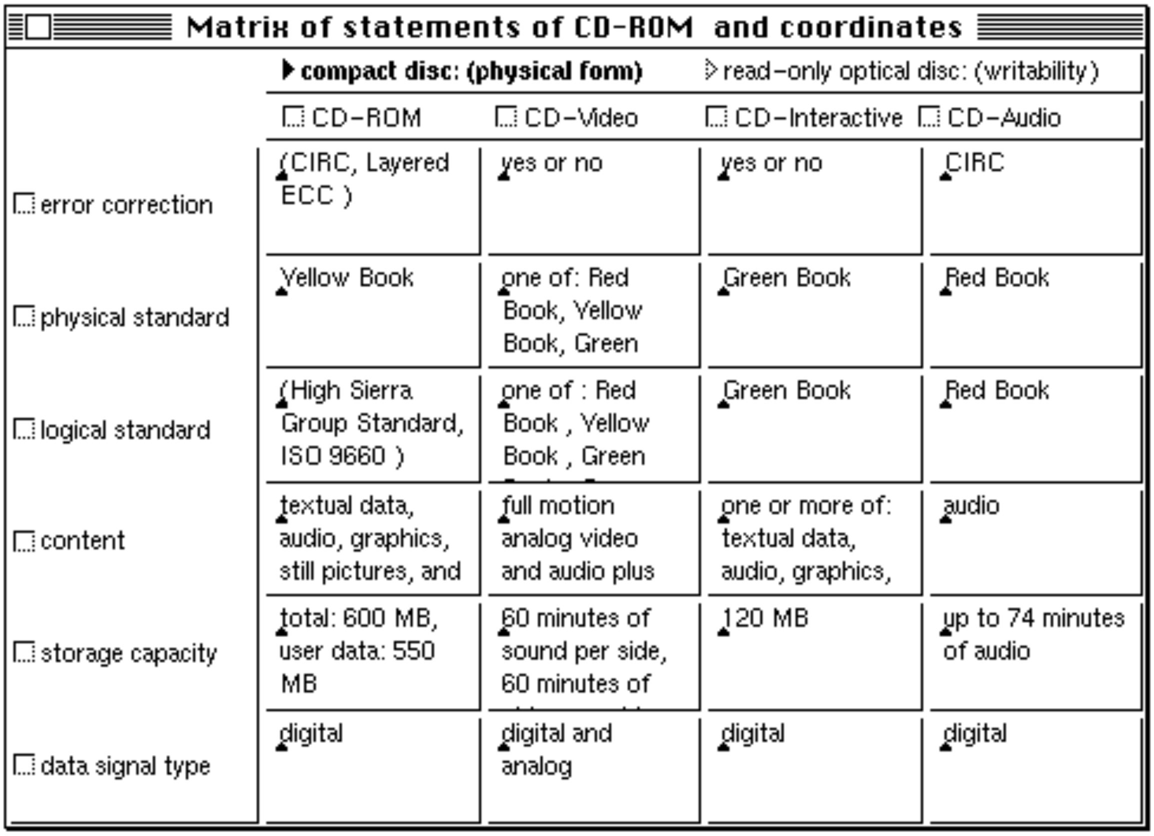
b) To mask out properties

•select the properties to be masked out by clicking on the property name (it will become bolded when selected)

• Select **visibility>>exclude selected rows**

The selected rows will then be masked out (disappear) from the matrix.

Figure 37 shows the matrix seen in figure 36 with masked out subjects and predicates.


Matrix of statements of CD-ROM and coordinates

	compact disc: (physical form)		read-only optical disc: (writability)	
	CD-ROM	CD-Video	CD-Interactive	CD-Audio
error correction	(CIRC, Layered ECC)	yes or no	yes or no	CIRC
physical standard	Yellow Book	one of: Red Book, Yellow Book, Green	Green Book	Red Book
logical standard	(High Sierra Group Standard, ISO 9660)	one of : Red Book , Yellow Book , Green	Green Book	Red Book
content	textual data, audio, graphics, still pictures, and	full motion analog video and audio plus	one or more of: textual data, audio, graphics,	audio
storage capacity	total: 600 MB, user data: 550 MB	60 minutes of sound per side, 60 minutes of	120 MB	up to 74 minutes of audio
data signal type	digital	digital and analog	digital	digital

Figure 37. Matrix of statements of CD-ROM and coordinates under superconcept compact disc (physical form), with masked-out subjects and predicates.

4.3.8.8 Unmasking concepts and properties in a matrix

Concepts and properties that have been masked out can be easily unmasked, either individually or all at once.

a) To unmask individual concepts and/or properties

- Select **visibility>>open mask** to get the Mask on knowledge map

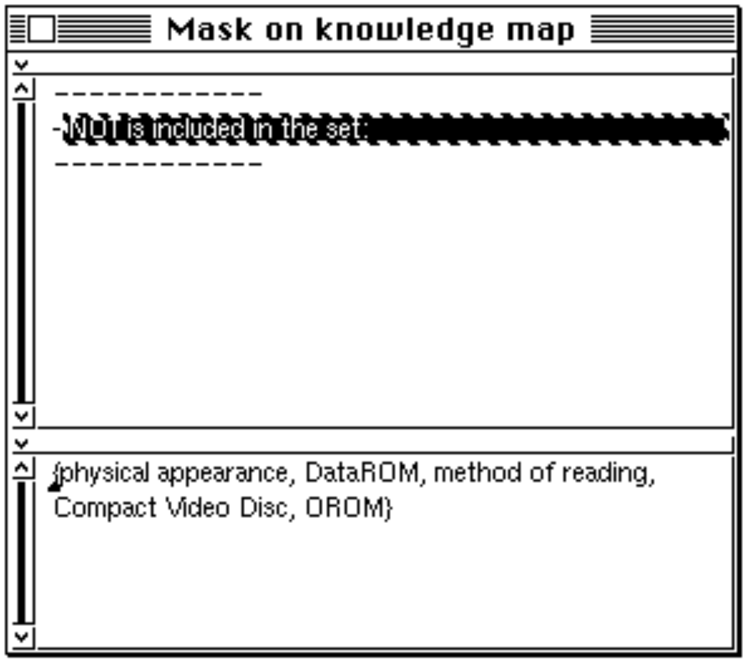


Figure 38. Marix mask with masked-out concepts.

- Select 'NOT is included in the set': masked-out concepts and/or properties will appear in the lower subwindow in curly brackets, as seen in figure 38

To remove a given item (e.g. physical appearance):

- Place your cursor in the lower subwindow and select **middle mouse button>>remove from set**

- A list of the items in the set appears

- Select the item to be deleted from the set (i.e.unmasked); this procedure must be repeated for each item to be removed from the list

- Select **middle mouse button>>accept and apply changes**

The items removed from the list will reappear in the matrix.

b) To unmask all masked-out concepts and/or properties

- Select **visibility>>reset mask to show everything**

4.3.8.9 Rearranging the order of concepts and properties in the matrix

The order of concepts and properties can be rearranged in any manner according to the user's wishes. The various ordering functions are relatively self-explanatory.

To rearrange concepts

- Select the concept(s) to be rearranged
- Select **rearrange>>subjects**, and desired method of arranging subjects (see figure 39)

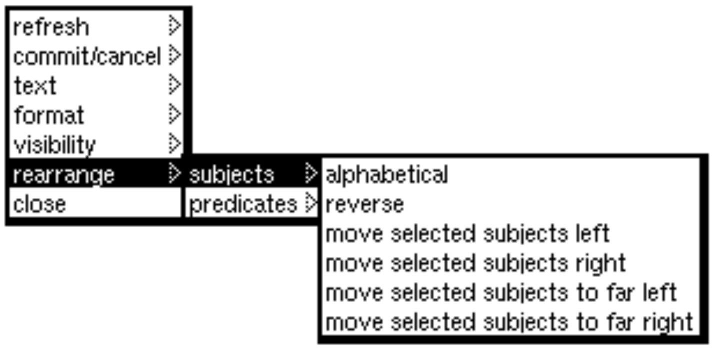


Figure 39. Rearrange subjects submenu.

To rearrange properties

- Select **rearrange>>predicates**, and desired method of arranging predicates (see figure 40)

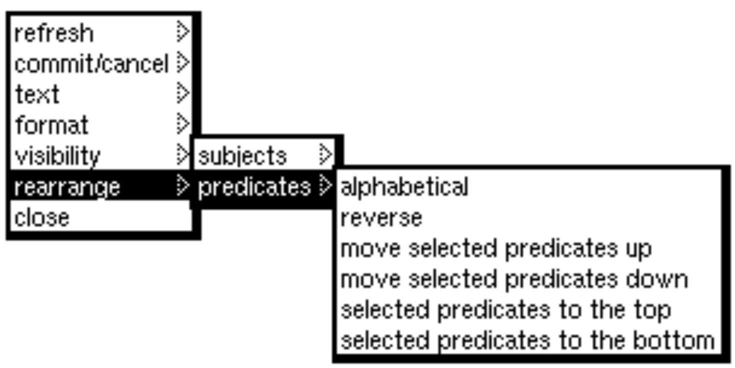


Figure 40. Rearrange predicates submenu.

4.3.8.10 The property history matrix

The material described above showed how to compare various concepts that are related 'horizontally'. I.e. they have a common parent.

Another useful capability is to find out how a chain of ever-more-specialized concepts becomes specialized. To do this:

• Select a concept and a property

• Open a detached dynamic history matrix. This kind of matrix can be edited in a manner similar to regular property matrices

4.4 Mask Views

Associated with each knowledge map (and with its browser subwindow) is a *knowledge mask*. The user can select 'open mask' from the browser visibility menu to display a view of the mask. An example is shown in figure 41.

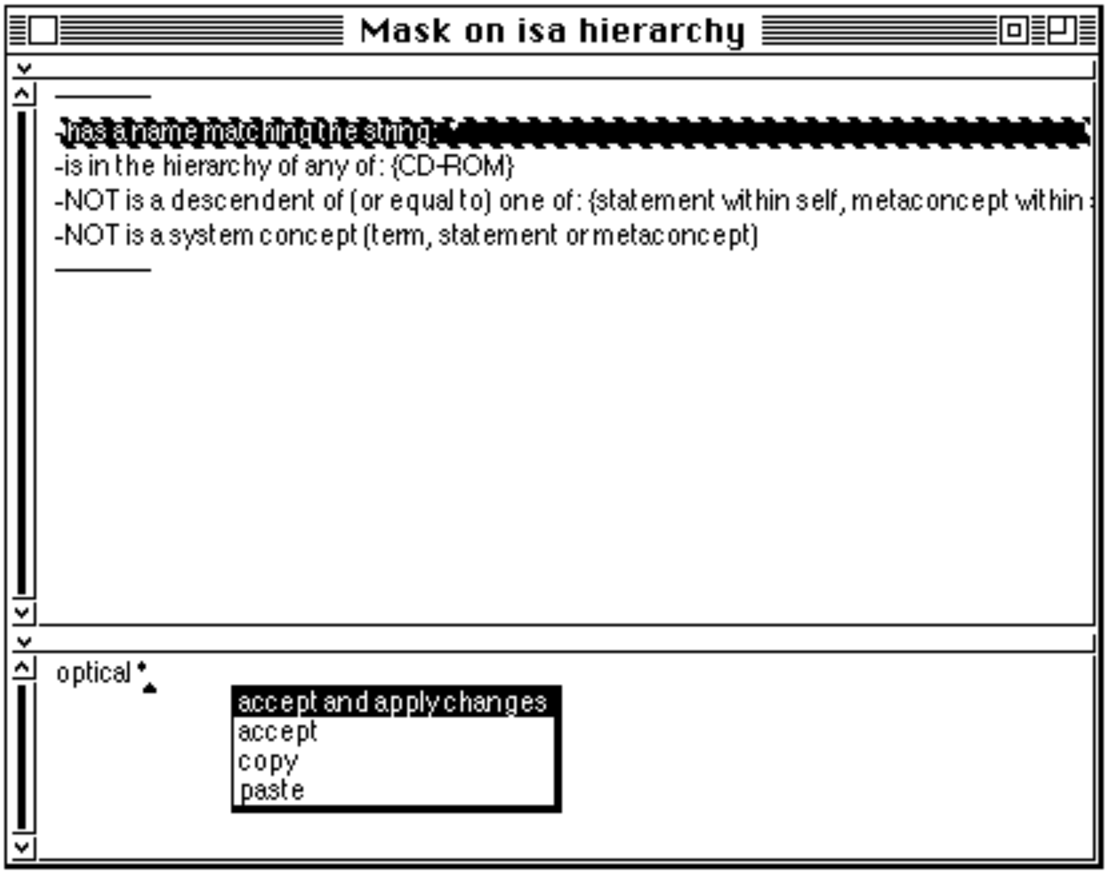


Figure 41: An example mask with four predicates. A predicate with a textual argument is selected This mask will show everything in its knowledge map's hierarchy that match the following criteria: 1) They must have names starting with 'optical', 2) they must be in the hierarchy of 'CD-ROM' (above or below), 3) they must not be system concepts or descendants of system concept types..

Also associated with each knowledge map is a set of *selection criteria*. The user can select ‘open selection criteria’ from the browser visibility menu to display a view of the selection criteria.

The regular mask is used to control which nodes are displayed in a paradigm. The selection criteria is used to cause a set of nodes to be highlighted (selected).

Both mask and selection criteria use mask views, as described in this section.

A mask view is composed of two subwindows:

• The mask predicate hierarchy (top subwindow)

• The argument editing field (bottom subwindow)

The mask predicate hierarchy currently contains a mere list of mask predicates. To determine whether a given concept is displayed, all of the mask predicates must evaluate to true when applied to that concept. In future the mask predicate hierarchy will become a full hierarchy that will permit 'and', 'or' and 'not' nodes — this user will this be able to specify an arbitrary logical expression.

The mask window updates under two circumstances:

1). The user issues a command from one of the mask menus described below, or

2) The user issues a command from a knowledge map's 'visibility' menu.

Details of both these ways of controlling the set of displayed concepts are described in section 5.4.1.

4.4.1 Editing the mask predicate hierarchy

One or more predicates can be selected in the mask predicate hierarchy (there are master selection and multi-selection capabilities just like in the browser - see section 4.3.2). The user may add or delete from this list. Figure 42 shows a mask with the 2 predicates selected in the mask predicate hierarchy, the predicate hierarchy menu (left-hand menu) and the ‘add predicate’ submenu.

The predicate hierarchy for mask and selection criteria menu has the following items:

apply changes: (mask only) Causes the browser subwindow to update with concepts being masked out as appropriate (this is only effective if updates are not deferred). Refreshing the subwindow has the same effect as this command. Until either of these actions is taken, the user may edit the mask without causing immediate change in the browser.

generate new selection: (selection criteria only) Causes concepts in the browser subwindow to be selected. The mask is temporarily overridden if any of the selected concepts are masked out (subsequently refreshing the browser subwindow restores the normal mask).

add predicate: Adds a predicate from a list of available predicates, increasing the restriction about what is displayed. Use 'apply changes' to see the effect. The predicates are described in section 5.4.1.

delete _..._: Deletes the selected predicates, removing their restrictions about what is displayed. Use 'apply changes' to see the effect.

negate _..._: Reverses the effect of the selected predicates. The prefix 'NOT' is displayed.

refresh: Updates the mask display (this should not be needed as the mask display is supposed to always be up-to-date.

inspect: (developer-only feature) Allows direct editing of the mask.

browse _ implementors: (developer-only feature) Allows access to the Smalltalk source code implementing the master selected predicate.

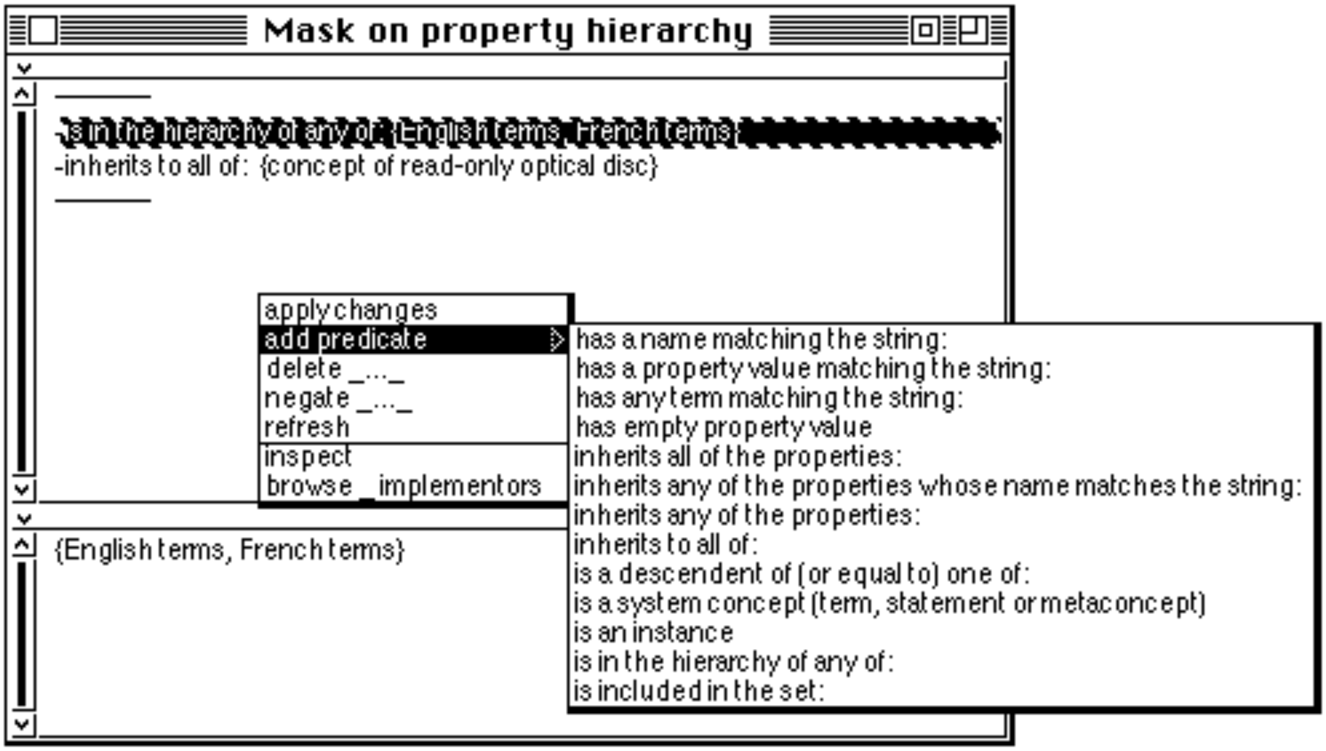


Figure 42: An example mask showing the add predicate menu. This particular mask is requesting to show only 'English terms' and 'French terms' subproperties that inherit to the metaconcept of 'read-only optical disc'.

4.4.2 Editing the mask predicate arguments

There are three types of mask predicates: 1) Those with no arguments, 2) Those with arguments that are string patterns, and 3) Those with arguments that are sets of concepts.

In future, some predicates might be able to have more than one argument.

When a predicate with an argument is master selected, its argument is presented for editing in the argument editing field. The menu options available depend on whether the argument is a string or concept set.

• **Editing a string argument**

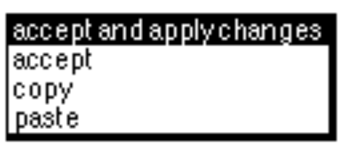


Figure 43: The menu of the mask predicate argument editing field for string predicates

Arguments of this type are used for pattern matching (e.g. to find all the concepts whose names or statement values match a certain pattern). Figure 43 shows the menu that is presented. Strings can use the asterisk character as a wildcard. In the future, more sophisticated regular expressions will be made available. In general, if the predicate argument is a single asterisk, and the predicate is not negated, the predicate has no effect.

The following menu items are available:

accept and apply changes: (mask only) Updates the predicate with the entered string and causes the browser subwindow to refresh, presenting the results of the 'query', i.e. hiding those concepts that do not match

accept and generate new selection: (selection criteria only) Updates the predicate with the entered string and causes the browser subwindow selection to show the results of the ‘query’.

accept: Updates the predicate with the entered string, but does not cause the browser subwindow to refresh.

copy: Standard Smalltalk text copy

paste: Standard Smalltalk text paste

• Editing a set argument

Figure 44 shows the menu that is presented when the selected predicate accepts a set of concepts as its argument. This type of predicate is typically used for such masking operations as preventing display of a specific set of concepts, or *only* displaying a specific set of concepts.

Figure 44: The menu of the mask predicate argument editing field for concept set predicates

The following menu items are available

add copied concept to set: Takes the concept copied from some browser subwindow using 'Copy _ ' or ^b, and adds it to the set. Use 'apply changes' in the predicate hierarchy to see the effect.

remove from set: Presents a menu of items currently in the set. The item selected is removed.

4.5 Feedback Panels

Associated with each knowledge base is a feedback panel. Often it is not displayed, but it will appear any time the user makes a knowledge base editing request that cannot be satisfied. The user may also explicitly request that the panel be displayed. This is done using the KBs control panel (see section 5.1.4).

Warning: In version 4.0 of Smalltalk, you should not collapse a feedback panel or else it will not appear when an editing error occurs. If you don't want to see the feedback panel, you should close it.

Figures 45 and 46 give examples of feedback panels.

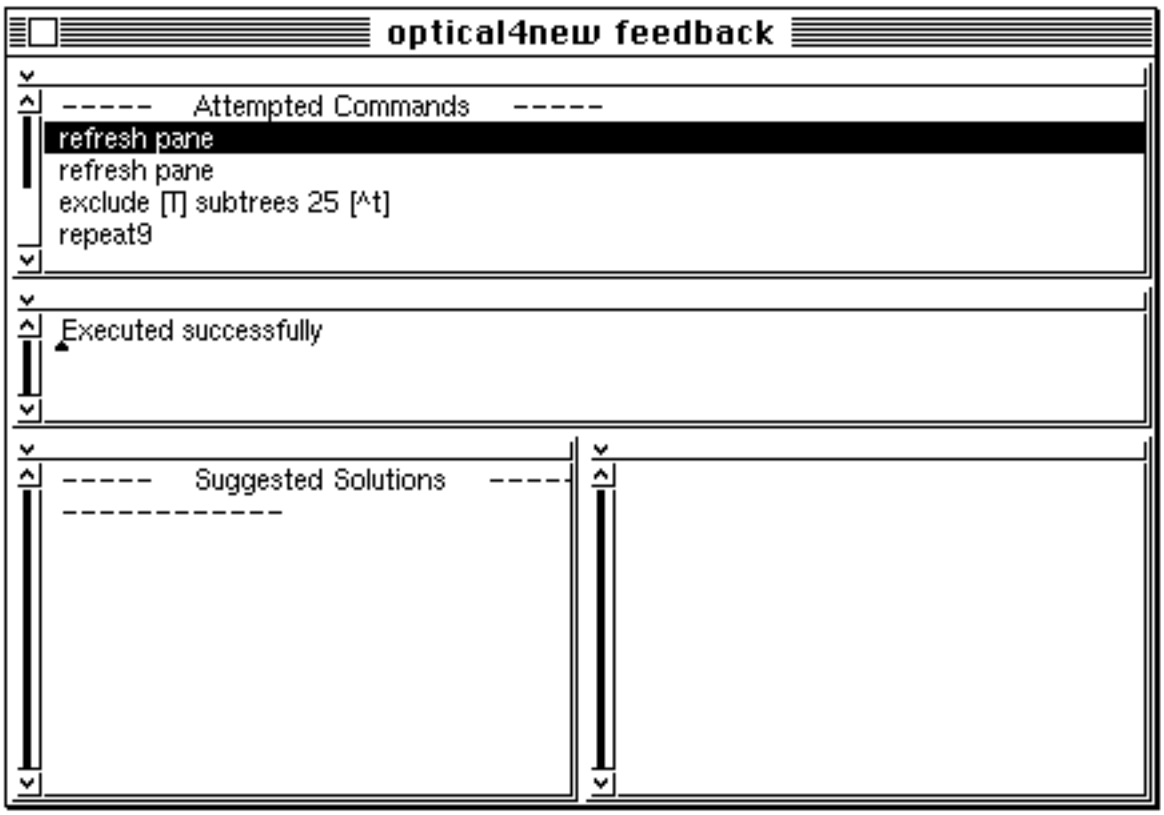


Figure 45: A feedback panel on a knowledge base called 'optical4new'.

A feedback panel has four subwindows

1) The list of attempted commands (at the top): This lists each command applied to the knowledge base. In future there will be capabilities for reversing and repeating commands. Currently this acts as a useful reference and reminder of what the user has done.

2) Command results subwindow (center): This describes textually the results of performing the selected command. The user may select any command in the attempted-commands list in order to see its results — by default the command results subwindow shows the results of the latest command. This subwindow often says 'executed successfully'. If the command did not execute successfully, then a detailed explanation of why not is presented.

3) Suggested actions subwindow (bottom left). This is currently unused. In future it will list possible courses of action the user may take, especially as a result of failed commands.

4) Suggested commands subwindow (bottom right). This is also not used. It will list the commands to be executed based on a selection in the suggested actions subwindow.

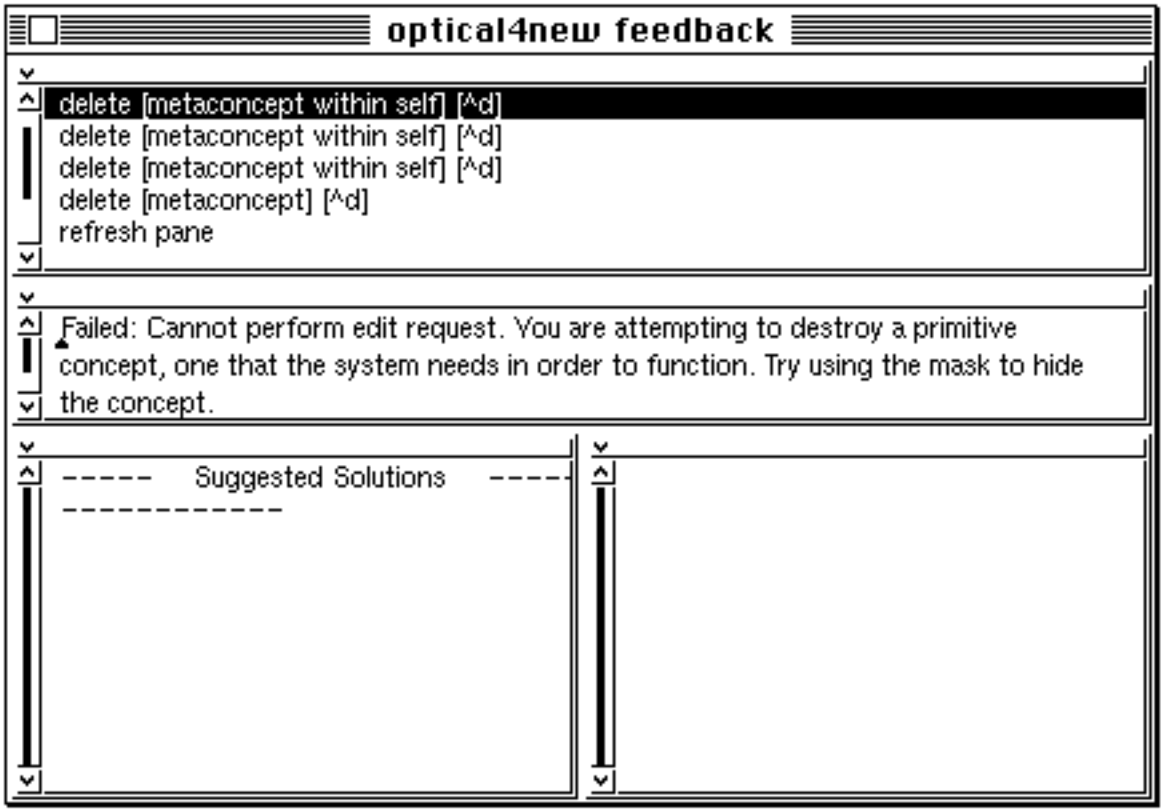


Figure 46: A feedback panel showing a situation where an error occurred.

Figure 47 shows a menu that exists in the current release. When some commands fail, a solution is presented in the 'suggested solutions' subwindow. The perform solution menu item can be used to carry out the suggested command. After executing 'perform solution' successfully, select the command that originally failed and issue 'retry'.

In a near-future release of CODE4, significant enhancements are planned to this functionality.

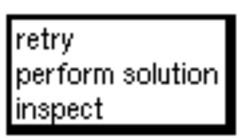


Figure 47: The feedback panel attempted commands subwindow

5 Commands

This section describes commands available for manipulating and editing knowledge. It assumes the user is familiar with the user interface as described in the last section.

5.1 Opening windows - basics

5.1.1 Opening the launcher

The launcher should always be open, but if it is not, typing the Smalltalk expression 'LauncherView openLauncher' in a Smalltalk window will open a new one. See more details in section 4.1

5.1.2 Opening the control panel

From the launcher's CODE menu, choose 'control panel'. Any existing control panel will be brought to the front. If there is no existing control panel, one will be opened. For more details see section 4.2

5.1.3 Opening a browser

There are three ways to open a browser:

(1) From the 'KBs' control panel, select a knowledge base and choose 'open browser on _'.

(2) From the launcher, choose 'open browser' (operates on the knowledge base selected in the control panel).

(3) From any browser, choose 'subwindow'.

For details of the user interface of browsers see section 4.3

5.1.3.1 Opening a top level browser

Opening a top-level browser is achieved from the launcher or from the 'Loaded KBs' control panel menu (figure 51).

In either case, a popup menu is presented from which the user may pick from a set of templates that combine the following options:

• **graphical or outline**

The interaction paradigm of the first subwindow.

• **... isa hierarchy or property hierarchy:** The knowledge map of the first subwindow. An isa hierarchy lists all concepts in the system. A property hierarchy lists all the properties in the system.

• **... with editable outline statements, graphical statements, full details or none:** The interaction paradigm and knowledge map of the second (driven) subwindow.

Figures 48 and 49 show two examples of top-level browsers, each with several subwindows.

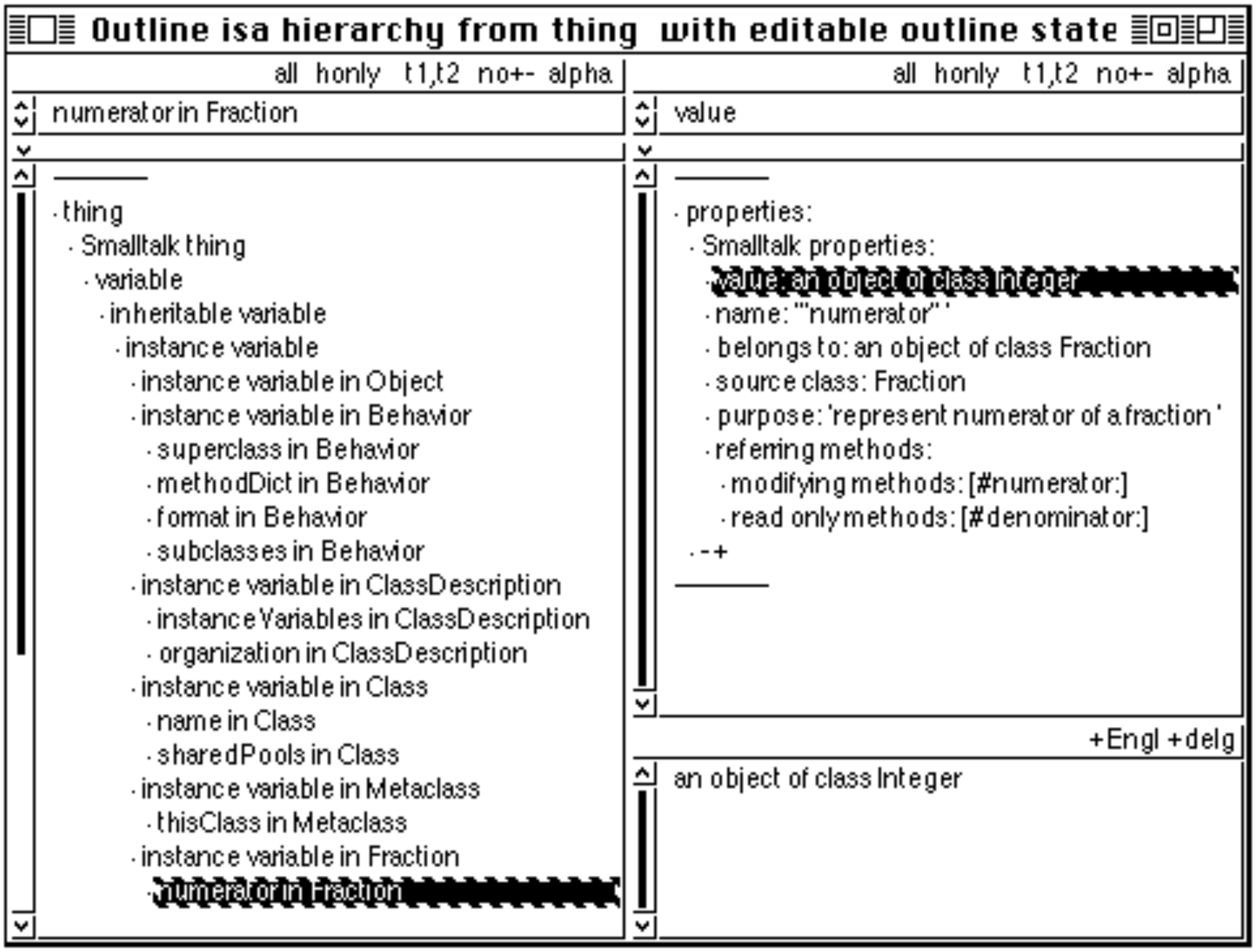


Figure 48: An 'Outline isa hierarchy with editable outline statements': the most popular kind of top-level browser. On the left are concepts in an isa hierarchy. At the top right is a statement hierarchy about the selected concept. At the bottom right is a user language view showing the value of the statement whose subject is the selected concept and whose predicate is the selected property. The right-hand half of this browser may also be opened from another browser by selecting 'subwindow>>properties>>outline>>...'

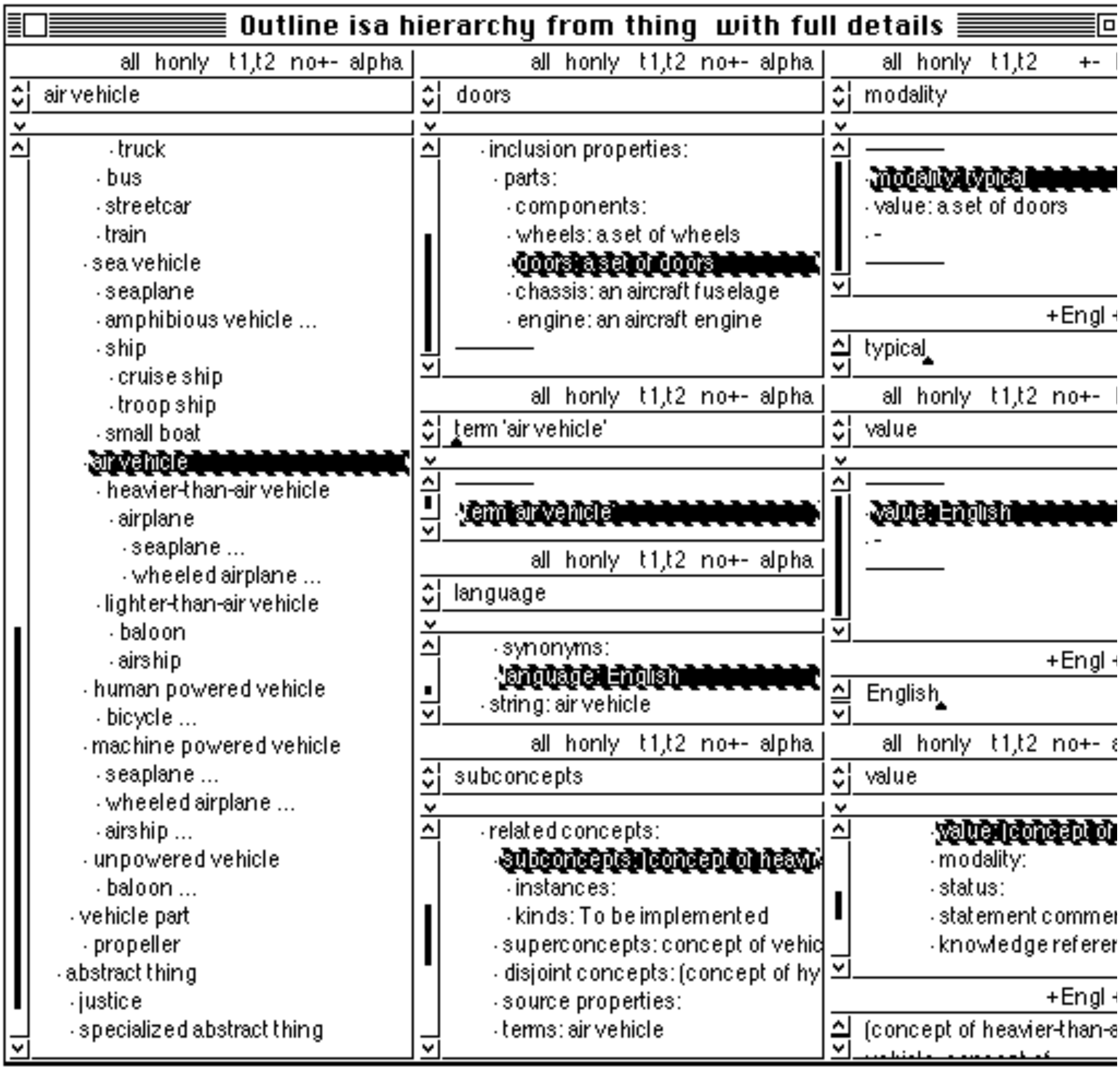


Figure 49: A fully-detailed top-level browser. At the left is a list of concepts. In the centre are (from top to bottom) properties, terms, term properties, and metaconcept properties. On the right are (from top to bottom) facets, term facets and metaconcept facets. The right-hand two-thirds of this browser may also be separately opened from another browser by selecting 'subwindow>>properties>>full details'.

5.1.3.2 Opening a browser or subwindow from another browser

The 'subwindows' menu (figure 17) in any browser subwindow allows the user to create a new browser, or add subwindows to the current browser. This menu has a series of submenus, that are combined to produce the desired subwindow(s). The user selects from the first submenu, then from the second submenu, and finally from the third submenu. After that, the resulting command is performed. Not all combinations are possible.

• The first submenu selects the kind of related knowledge to be displayed; it has the following choices:

properties: A subwindow or subwindows are opened that will show statements about the selected concept(s) in the current window. If the current subwindow itself is showing statements, then the new subwindow(s) will show statements about statements (i.e. facets).

subtree: A subwindow is opened with the same type of knowledge map as the driving subwindow, except that subhierarchies of the selected nodes are shown.

relation: A subwindow is opened that can display an arbitrary set of relationships. For more details, see section 5.3.18. If this option is selected from a statement hierarchy, then the selected properties determine the relations to be displayed, and the master selection determines the relation to be edited. Figure 11 is an example of this kind of subwindow.

terms: A subwindow is opened containing all terms of the selected concepts. The user can open a further ‘properties’ subwindow from this to obtain linguistic information, etc.

meanings: A subwindow is opened showing the concepts referred to by any selected terms (their meanings) or statements (the related concepts of the value).

metaconcept properties: Similar to ‘properties’ except that the properties of the metaconcept of selected concepts is displayed

freeze this pane: This does not open a new window, but rather alters the current subwindow so that it is no longer dependent (i.e. it is no longer driven by its driver subwindow).

remove this pane: Removes the current subwindow entirely.

• The second submenu selects the types of subwindows in which the related knowledge will be presented; it has the following choices:

outline: A composite of two subwindows is opened: (1) an outline-paradigm hierarchy of the statements of the selected concept(s) (section 4.3.5) , and (2) a user language subwindow to allow editing of the value of the selected statement (section 4.3.7). For an example, see the right half of figure 48.

graphical: A composite of two subwindows is opened as above, except that the first subwindow uses the graphical interaction paradigm (section 4.3.6)

textual: A subwindow is opened using the textual interaction paradigm (section 4.3.9). This is currently under development.

matrix: A subwindow is opened using the matrix interaction paradigm (section 4.3.8).

full details: A composite of a number of subwindows is opened, showing statements and their facets, terms and their properties and metaconcept statements and their facets. All these subwindows use the outline interaction paradigm. See the right two-thirds of figure 49 for an example of the full-details composite.

• The third submenu selects where the new subwindow(s) will be put with respect to the current subwindow, and whether the new subwindows will be dependent or not. The menu has the following choices:

detached dynamic: The new subwindow(s) will be dependent on the current subwindow (i.e. the current subwindow will 'drive' the new one(s); selections made in the current subwindow will cause their content to change dynamically). The new subwindow(s) will also be in their own window, separate from the driving subwindow.

detached static: The knowledge in the new subwindow(s) will be determined by the selection in the current subwindow, but will not be updated when that selection is changed. The new subwindow(s) will also be in their own window.

right dynamic: The new subwindow(s) will appear in the same window as the current subwindow, and to the right of it. The current subwindow will be shrunk in size to accommodate the new subwindow(s). They user may therefore have to subsequently increase the size of the whole window, or adjust the internal borders (section 4.3.4.2). The new subwindow(s) are dependent on the current subwindow

below dynamic: The new subwindow(s) will appear in the current window, below the current subwindow, and will be dependent on the current subwindow.

saved to file: This option is currently inoperative. It is planned that the subsidiary information be savable to a file, instead of being saved in a window.

Several combinations of the above options have hot keys (section 4.3.4.6). These are:

properties, outline, detached dynamic: ^k

properties, outline, detached static: ^]

relation, graphical, detached dynamic: ^v

subtree, outline, detached dynamic: ^l

5.1.4 Opening a feedback panel

To see the history of commands issued to a knowledge base, select '**open feedback panel on _**' from the 'Loaded KBs' control panel menu (figure 51). Figure 45 shows an example of a feedback panel. The user interface is described in section 4.5. See also section 5.4.3 for other hardcopy mechanisms.

5.1.5 Displaying statistics about a knowledge base

To see detailed statistics about the number and types of concepts in a knowledge base, select '**display statistics about _**' from the 'Loaded KBs' control panel menu (figure 51). Figure 50 shows an example of a statistics window.

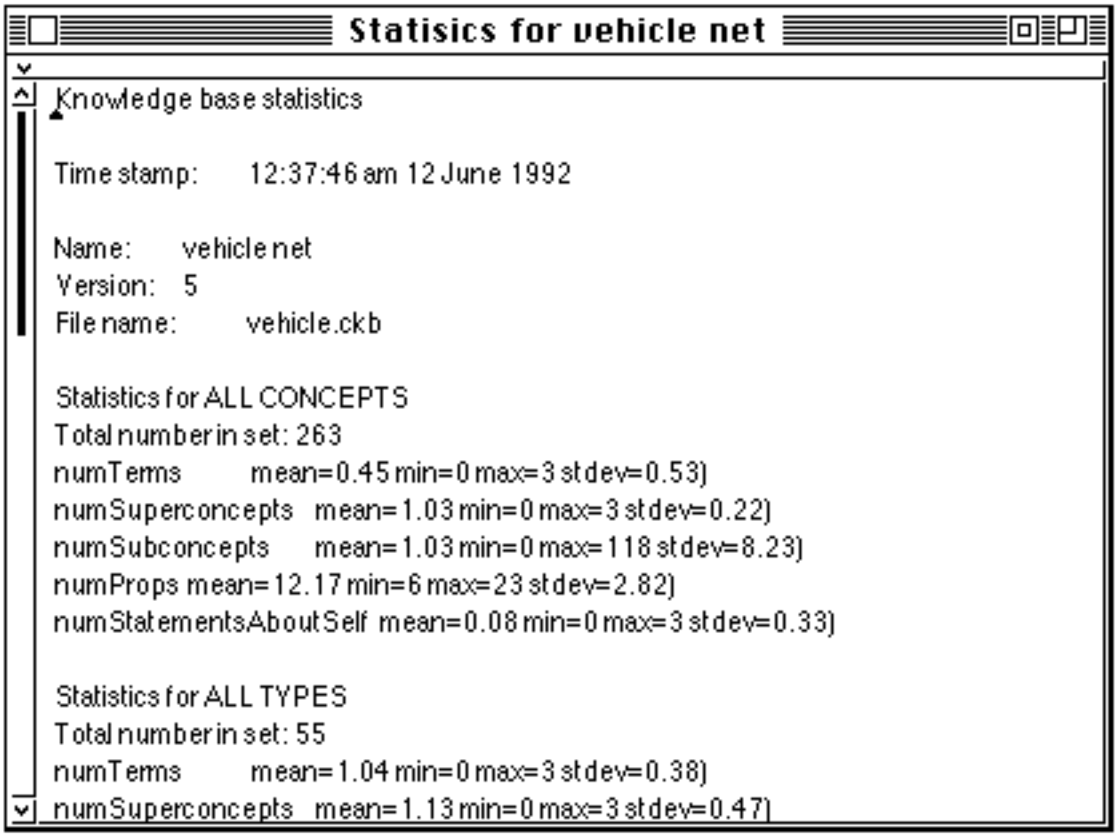


Figure 50: A Statistics window generated from the 'vehicle net' knowledge base.

5.2 Manipulating knowledge bases

The KBs control panel (section 4.2.2) is used to manipulate knowledge bases. Figure 51 shows the menu of available operations (all accessed from the middle mouse button).

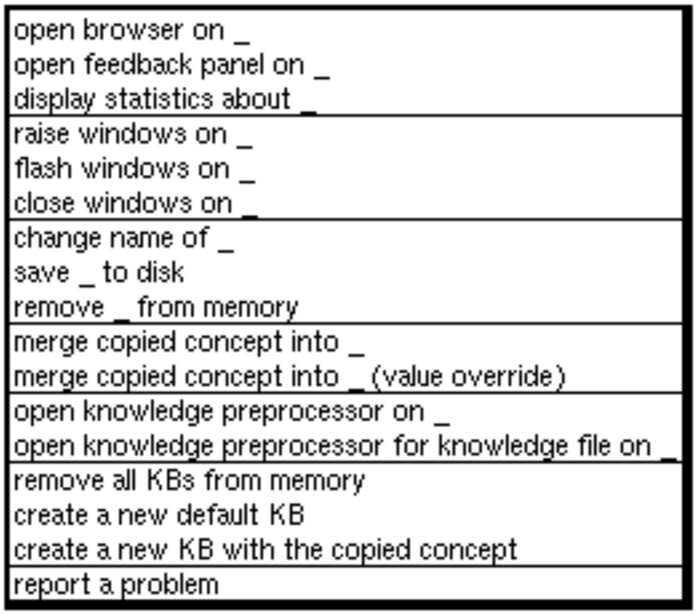


Figure 51: The 'Loaded KBs' control panel menu

5.2.1 Loading a knowledge base

Select a file with a ‘.ckb’ suffix in the bottom part of the ‘KBs’ control panel, and choose ‘l**oad _ into memory**’. The system will ask you to confirm your choice. This may take a minute or longer for a very large knowledge base.

Beware that loading too many large knowledge bases can cause the system to run out of memory.

Also beware of loading multiple copies of the same knowledge base: you might lose track of which copy you have edited (even though the control panel indicates when a knowledge base has been edited).

5.2.2 Changing a knowledge base name

Select a loaded knowledge base in the ‘KBs’ control panel (found under ‘Loaded KBs’ in the upper part of the subwindow) and choose ‘**change name of _**’. Type in a new name at the prompt and hit <return>.

Knowledge bases have both a version number and a name, e.g. v1 ‘minimalKB’ (145 conc., 22 types). The version number is automatically updated whenever the knowledge base is saved. The name of a knowledge base is in fact distinct from the name of the file in which it is saved. This is done because file names can easily be changed by external operating system facilities (especially when copies are made), whereas the name of the knowledge base should remain constant.

5.2.3 Saving a knowledge base to disk

A knowledge base which has been edited (e.g. v1 ‘minimalKB’ (145 conc., 22 types, edited)) should be saved to ensure that any edits made are not lost.

Select a loaded knowledge base in the ‘KBs’ control panel and choose ‘**save _ to disk**’. You will be prompted for the file name (which defaults to the knowledge base name). The version number will be incremented before the saving takes place.

Note that a KB marked ‘edited’ before it is saved will not contain that note once it is saved, e.g. v1 ‘minimalKB’ (145 conc., 22 types, edited) becomes v2 ‘minimalKB’ (145 conc., 22 types). Therefore any KB not marked ‘edited’ can be assumed to be the most up-to-date version of that KB.

5.2.4 Removing a knowledge base from memory

Select a loaded knowledge base in the ‘KBs’ control panel and choose ‘**remove _ from memory**’.

If a KB is removed without the edits having been saved, the edits will be lost.

5.2.5 Removing all KBs from memory and creating a new default

In the ‘KBs’ control panel, select ‘**remove all KBs from memory**’. All knowledge bases are removed and a default one is created.

The default knowledge base that is created contains a minimal set of system concepts needed for the system’s internal functioning (i.e. knowledge about self; see section 6). It is expected, however, that users will typically want to load a top-level (ontology) knowledge base rather than starting from an empty default

5.2.6 Creating a new default (empty) knowledge base

In the ‘KBs’ control panel, select ‘**create a new default KB**’. A default KB is created. (See section 5.2.5, above, for more details on the default KB.)

5.2.7 Merging one knowledge base with another

To merge two knowledge bases, first decide what isa-hierarchy subtree of the source knowledge base is to be merged into the destination knowledge base. Then, use 'copy _ [^b]' (section 4.3.4.8) to place the root concept of the source subtree in the copy buffer. Then select the destination knowledge base in the KBs control panel (section 4.2.2) and select menu item '**merge copied concept into _**'.

CODE4 will transfer all concepts in the source subtree into the destination knowledge base. It makes some heuristic decisions about where to put the destination subtree. It looks for concepts in the destination KB that match the superconcept(s) of the root of the subtree being copied. If it finds such a concept, it will merge the subtree as a subconcept. This is useful because it allows multiple people to work on different hierarchies, and then combine their work periodically. If no match can be found, the incoming subtree and its superconcept chain will be copied under the top concept of the knowledge base. The superconcepts must be copied too because they contain properties that inherit to the copied subtree.

During the merge process, all properties, statements, metaconcepts and facets of the copied subtree are also merged.

The merge mechanism described above will not destroy any knowledge in the destination knowledge base. If there are conflicting statements, the destination knowledge base has priority, and statements from the source knowledge base may not be transferred as expected.

Another option in the KBs control panel menu, '**merge copied concept into _ (value override)**', gives priority to the source knowledge base. In this case, conflicts are resolved in favour of the source KB, and it can be guaranteed that all knowledge from the source subtree is transferred. Which of the two menu items is most desirable depends on what kind of collaborative work is being persued, and how much overlap of work has taken place.

In the future a third merge option is planned. This will display all conflicts and allow statement-by-statement resolution.

5.2.8 Duplicating a knowledge base or subtree in memory

Sometimes it is desirable to take a knowledge base and develop it in two different directions (so that subsequently, the descendants of the two copies contain different knowledge). This is useful, for example when there exists a knowledge base with top-level knowledge that can be specialized for various domains or subdomains.

It is always possible to copy a file and thus create a new knowledge base on disk (section 4.2.2). Likewise it is always possible to load two copies of the same knowledge base into memory (section 5.2.1). A third option is to duplicate a knowledge base in memory. To do this select the top concept of the knowledge base to be duplicated and use menu item '**copy _ ^b**' (section 4.3.4.8). Then use menu item '**create a new KB with the copied concept**' from the KBs control panel (section 4.2.2). You will be prompted to give a name to the new knowledge base. This operation is equivalent to merging (section 5.2.7) one knowledge base into a default knowledge base (section 5.2.6).

As a variation on the above, it is often useful to just duplicate a subtree. This can be useful when there exists a large master knowledge base, and for purposes of collaborative work, it is desired to have various people work on different subtrees. The easiest way to achieve this is to make several new knowledge bases containing the individual subtrees, have the collaborators work on these subtree knowledge bases, and later merge the edited subtrees back into the master knowledge base using '**merge copied concept into _ (value override)**'. To create the initial subtree knowledge bases, first select the concept at the roots of the desired subtree and put it in the buffer using '**copy _ ^b**', then repeatedly use '**create a new KB with the copied concept**'.

5.2.9 Manipulating knowledge base windows

There are three options on the 'Loaded KBs' control panel menu (figure 51 and section 4.2.2) that can be used to manipulate all the windows on a knowledge base:

raise windows on _: brings all browsers displaying knowledge about the selected knowledge base to the front. This can be useful when windows get lost behind others, or when it becomes unclear which windows belong to which knowledge base. Warning: In release 4.0 of Smalltalk, collapsed windows may not appear.

flash windows on _: momentarily blackens browsers displaying knowledge about the selected knowledge base. This is useful to determine which windows belong to which knowledge base.

close windows on _ : closes all windows on the selected knowledge base

5.2.10 Other knowledge base operations

See ‘Opening a browser’ in section 5.1.

See also the file manipulation options in the control panel (section 4.2.2).

5.3 Editing knowledge bases

Editing knowledge bases is primarily accomplished by issuing commands to a set of selected concepts. It is also possible to edit textual lists (e.g. item1, item2, item3) in metaconcepts properties to accomplish many of the same effects as described in this section.

Before you start to make edits, ensure the correct knowledge base is loaded and selected (sections 4.2.2, 5.2.1), and that you have one or more browsers open on it (section 5.1.3).

The edit menu is illustrated in figure 18.

Many edit commands are binary in that they not only require a selection, but also require a 'copied concept' in a buffer. The copying process is described in section 4.3.4.8.

Some edit commands can be automatically repeated (e.g. you can add 4 subconcepts at once). See section 4.3.4.4

5.3.1 Adding concepts

In any browser subwindow select one or more concepts and then choose one of the following menu items from the ‘edit’ submenu (found within the middle button menu, figure 18):

add child to _..._ [^a]: for each selected item, creates a concept that is a child in the current hierarchy (i.e. a subconcept in an isa hierarchy, a subproperty in a property hierarchy etc.)

add sibling to _..._[^s]: For each selected item, creates a concept that is a sibling in the current hierarchy. Where a selected item had multiple parents, the sibling will have the same parents.

It is not possible to add a child to an instance concept. If this is attempted, a feedback panel will appear. If you really want to do this, try changing the concept to a type first. (Some people feel that instance concepts should be allowed to have subconcepts; this is because they have a broader notion of what it means to be an instance concept.)

When subconcepts are added in an isa hierarchy, they are created as types by default. However, if all of the existing concepts **???**

5.3.2 Deleting concepts

Select one or more concepts and choose ‘**delete _..._[^d]**’ from the ‘edit’ submenu. There are many cases where it is not possible to immediately delete a concept because of various dependencies (e.g. the concept is referred to, or is the source of properties that cannot be readily moved to some other concept). In these circumstances the feedback panel will appear explaining why (section 4.5).

Deleting a concept in the middle of a hierarchy causes every parent of the deleted concept to be made a parent of every child of the deleted concept.

Future: The feedback panel will list the steps necessary to actually delete a concept, if immediate deletion cannot be accomplished.

5.3.3 Adding properties to a concept

This is a special case of adding concepts (see ‘Adding concepts’ above). It can only be performed in a property hierarchy knowledge map, driven from some other knowledge map. A concept must be selected in the latter, and one or more properties in the former.

The easiest way to accomplish this is to open an 'outline isa hierarchy from x with editable outline statements' (section 5.1.3). Then select a concept in the left hand pane. This concept will be the most general subject of the new property. Next select a property in the top-right pane. This will be the superproperty of the new property. Finally, select the menu item '**add child to _..._[^a]**' in the top-left pane. You may then want give a value to the statement involving the most general subject and the new property; to do this, see section 5.3.6.

5.3.4 Deleting properties from a concept

This is a special case of deleting concepts. See ‘Deleting concepts’ and ‘Adding properties to a concept’, above, for more details. Note that a property must be deleted from the most general subject (i.e. the source of the property).

5.3.5 Renaming a concept

Select the concept name, then either (a) type the new name or (b) perform detailed editing in the editing area of the subwindow, and hit <return>. Applies only to the master selection. See section 4.3.4.3 for more details.

5.3.6 Changing the value of a statement

This is a special case of renaming a concept. It can only be performed in a user language knowledge map, driven by a property hierarchy knowledge map.

Simply select the statement, edit the value text and hit <return>. See section 4.3.7 for more details.

5.3.7 Adding and editing facets of properties

This is a special case of adding and editing properties because facets are properties of statements.

Open a property subwindow from a property subwindow so that the new subwindow is displaying statements of statements (section 5.1.3) . New facets can be added (as in section 4.3.3) and the name and value of any facet can be edited (section 5.3.6). Note that the value of the value facet of a statement is, by definition, the same as the value of the statement.

5.3.7.1 Making facets inherit

Adding a facet from a property subwindow of a statement adds a facet *in that statement only*. The facet does not inherit.

To make a facet appear in every statement in the system, open a property subwindow on the subject concept 'statement within self'. This is a system concept (see section 6). Add a property to 'statement within self'. The new facet property will then appear in every statement.

To make the **value** of a facet property inherit down the subject hierarchy as well as the existence of the facet, edit the value of the facet in 'statement within self', giving it the value: #f. This is a special delegation symbol (see section 5.3.8). Make sure you have Cleartalk parsing switched on when you define a delegation symbol (section 4.2.1).

To make a facet inherit to just a certain set of concepts, do the following:

1) Create a sibling concept of 'statement within self' in the isa hierarchy (section 5.3.1)

2) Reparent this 'specialized statement within self' to be a subconcept of 'statement with self' (section 5.3.9)

3). Reparent all the concepts to which you wish to add the facet so that they become subconcepts of the new 'specialized statement within self'.

4) Add the new facet to the new 'specialized statement within self' as described in 5.3.7.

5.3.8 Specifying substitution (delegation and special symbols)

When editing the value of a statement, special syntax can be used to cause special effects to occur (see below) if substitution has been requested. See section 4.3.7 (format options for the user language interaction paradigm) for details about how to turn substitution on and off.

Note that substituted values may be different in subconcepts: In fact this is the source of power of substitution. The *current concept* refers to the concept in which substitution is taking place.

When specifying delegation, it is necessary to have Cleartalk parsing switched on (section 4.2.1)

Future: The following description will be enhanced.

The following syntaxes cause substitution by the value indicated at the left of the '==>':

• **the <propname>** ==> the value of the property <propname> in the current concept. This is called *simple delegation*.

• **the <propname2> of the <propname1>** ==> the value of the property <propname2> in the concept related to the current concept by the property <propname1>. This is called *compound delegation*; any number of levels of 'of' are allowed. In the current release this is not guaranteed to work properly.

• **#** ==> the current concept (self)

• **#m** ==> the metaconcept of the current concept

• **#s** ==> the superconcept(s) of the current concept

• **#c** ==> the concept of the current concept if it is a statement

• **#p** ==> the property of the current concept if it is a statement

• **#u** ==> the submetaconcept of the current concept if it is a metaconcept

• **#1** ==> what the concept is 'of' if it is an associated concept (future).

• **#f** ==> the value of the same facet of the same property in the superconcept of the property's subject (see section 5.3.8). This is only effective as the value of a property of 'statement with self'

• **#v** ==> the value of a statement involving the same property

These syntaxes can be combined, e.g. 'the comment of #sm' would result in substitution of value of the comment property of the metaconcept of the superconcept of the current concept!

Useful combinations of delegation symbols:

Facet value inheritance: #f (section 5.3.8)

Metaconcept value inheritance: #usmv (take the value of the statement about the same property of the metaconcept of the superconcept of the submetaconcept). This is only effective as the value of a property of 'metaconcept within self', or a subtype.

5.3.9 Reparenting

This is a binary command which allows you to give a set of concepts new parents. First, copy a set of concepts to become the new parent(s) (temporarily only one parent can be set at a time — section 4.3.4.8). Then select one or more concepts to be reparented and choose '**reparent _..._ to copied concept [^p]**' from the 'edit' submenu.

Where no consistency constraints are violated, the second set of concepts have their old parents removed and replaced with the new parent (i.e. the copied concept).

Some consistency constraints that may be violated (and thus may result in a problem being reported in the feedback panel) are:

• In isa or property hierarchies, a child of a concept cannot also be a parent of the same concept (directly or indirectly). In a relation map this constraint is relaxed.

• The isa hierarchy or property hierarchy must always be maintained so that all concepts that inherit a given property, also inherit that property's superproperties.

Note that what 'parent' means depends on the knowledge map. In a isa hierarchy, it means 'superconcept'; in a property hierarchy map, it means 'superproperty'; etc.

5.3.10 Specifying multiple parents

This is a binary command. First, copy a set of concepts to be added as additional parent(s) (temporarily only one parent can be added at a time — section 4.3.4.8). Then, select one or more concepts to have parents added and choose **add copied concept as parent of _..._** from the 'edit' submenu.

Where no consistency constraints are violated, the first (copied) concept is added as an additional parent of the second set of concepts.

5.3.11 Moving a property to a different concept

This is a binary command. First, use '**copy _ [^b]**' to copy the concept that is to become the most general subject (source) of the properties. Then, select one or more properties to be moved and choose '**move _..._ properties to copied concept'** from the 'edit' submenu.

Where no consistency constraints are violated, the properties are moved to the copied concepts. The most common constraint violation involves leaving a statement in a concept that no longer inherits that statement's property.

5.3.12 Making a property an inverse of another

To add a property inverse, perform the following steps:

a) Set up the two properties in question that are to be inverses (for example 'parts' and 'part of')

b) In a browser, select one of the properties to have an inverse made (e.g. 'parts')

c) Open a property browser on that property.

d) Edit the value of the 'inverse' property to be the inverse (e.g. 'part of').

From now on, when any values of either property are edited, The inverse will be edited as well.

To make an inverse of an existing property: Add the inverse as above, then install all the inverse values using 'install all inverses of _..._ properties' in the 'edit' menu.

5.3.13 Making concepts disjoint

By default, two type concepts are not disjoint, i.e. instance concepts of one type can also be instances of the other. To declare that types cannot share common instances, you must make them disjoint.

Select one or more type concepts and choose **make _..._ disjoint** from the 'edit' submenu. All the selected concepts will be made disjoint from each other.

The metaconcept property 'disjoint concepts' displays the most general set of concepts that is disjoint from a given concept.

An instance concept is always disjoint from any other concept.

5.3.14 Making concepts nondisjoint

Select one or more type concepts and choose **make _..._ nondisjoint** from the 'edit' submenu. All the selected concepts will be made nondisjoint from each other.

5.3.15 Changing a type into an instance

A type concept represents a class or category of things, whereas an instance concept represents a single thing (e.g. 'city' is a type concept, while 'Paris' is an instance concept). New subconcepts of types are generally added as types; they can then be changed into instances.

Select one or more type concepts and choose **turn _..._ to instance** from the 'edit' submenu. Whether this operation will succeed is constrained by the following rule:

• An instance concept cannot have subconcepts. The feedback panel will display a failure message if an attempt is made to violate this rule.

5.3.16 Changing an instance into a type

Select one or more instance concepts and choose '**turn _..._ to type**' from the 'edit' submenu. This operation is constrained by the same rules as 'Changing a type into an instance' (above).

Concepts such as properties, statements, terms, metaconcepts, etc. are inherently instance concepts and cannot be changed into types.

5.3.17 Committing and cancelling

The 'commit/cancel' submenu is currently of most use only when editing values. See section 4.3.7 for details.

5.3.18 Editing different relations

The most popular relations (i.e. propertoes) used as links in browser subwindows are 'isa' and 'subproperty'. Isa is a metaconcept property (a relation between concepts themselves). Subproperty is a relation between properties (that can also be applied to statements because statements have properties as their predicate).

Often the user wants to display a network of arbitrary (non-isa) relations between things. This can be done by opening a 'relation' subwindow (see section 5.1.3.2). Figure 11 is an example of this kind of subwindow.

The following subsections describes how one might make use of this functionality:

5.3.18.1 Creating a simple parts hierarchy

To create a ‘parts’ hierarchy, do the following (for other kinds of hierarchy, follow similar steps with some property other than 'parts')

1) Open a browser (or set of browsers) containing an isa hierarchy and a dependent statement hierarchy. The simplest such browser is an 'isa hierarchy from x with editable outline statements'.

2) Ensure that the concept in the isa hierarchy that is to have its partitive relations shown in a partitive hierarchy has properties that are labelled as parts (or something similar). For example, the concept ‘dog’ could have the properties ‘part:legs’, ‘part:head’, ‘part:body’ and ‘part:tail’, where legs, head, body and tail exist as concepts elsewhere in the kb. The values of these properties must be found elsewhere in the kb as concepts, i.e. (in this case) legs, head, body and tail must be found elsewhere in the kb.

3) Select the concept for which you want to see the partitive relations, e.g. dog.

4) Select the property for which you want to see the relation hierarchy (in this case ‘parts’).

5) From the statement hierarchy subwindow, open a relation browser (graphical or outline; dynamic or static—see section 5.1.3.2). In a graphical browser, you will see the concept dog related by links labelled ‘part’ (the property name specified in step 2) to the concepts legs, head, body and tail. For an outline browser, you will have to click on the icons to see the link label.

Additional points:

• The concept that is the root of the parts hierarchy should inherit the 'parts' property. If it does not, you should go to a superconcept and add the property. It is important that the 'parts' property not have the root of the part-of hierarchy as its most general subject. If it does then unexpected circularities can occur in the parts relation.

• In step 3, you can select several concepts to see several hierarchies (possibly overlapping). See section 5.3.18.2

• In step 4 you can select several properties to see several relations graphed at once. If you select a superproperty, relations involving all its subproperties will be graphed. See section 5.3.18.2

• If you perform step 5 starting from the concept subwindow (instead of the statement subwindow), then *all* the relations involving the selected concepts will be displayed.

Adding concepts (section 5.3.1): If you add a child concept or sibling concept, it will appear as a child or sibling in the parts hierarchy. It will also appear as a new concept in the isa hierarchy: It will be created as a subconcept of the most-general-subject of the parts property. The user can subsequently go to an isa subwindow and reparent the concept in the isa hierarchy (section 5.3.9)

Reparenting concepts (sections 5.3.9 and 5.3.10): This is how you get existing concepts into the part-of hierarchy. a) First, get a concept so that it is displayed in the relation subwindow , but not connected to anything: To do this, simply select the concept in the driving isa hierarchy. b) Then select the superpart in the relation hierarchy, copy it using '**copy _ [^b]**, select the new part, and '**add copied concept as a parent of ..._... [^o]**'.

7) You can also note that the values of statements involving the parts property change dynamically as the relation hierarchy is updated.

5.3.18.2 More complex hierarchies

In reality, there are many subtle shades of relations between concepts. For example, in some people's ontologies, 'inclusion properties' may be subdivided into 'parts' and 'materials'.

It is possible to show several relations at the same time in a relation hierarchy. All you have to do is to select *all* the properties you want to show in the driving statement hierarchy. You can select the properties individually, or you can select the one or more superproperties all of whose subproperties you want to use.

When showing multiple relations it is best to use a graphical relation hierarchy, because the links will be explicitly labelled with the type of relation. It is also important to ensure that the graph format is set to show link labels (section 4.3.6, figure 23). If link labels do not appear, it may be because nodes are shown too close together – try dragging the nodes farther apart.

5.3.18.3 Non hierarchical relations

It is possible to use the above methods to show one or more non-hierarchical relations. At the current time it is necessary to manually lay out the graph because the graph algorithm assumes a hierarchy.

Examples of non-hierarchical relation diagrams that can be drawn using CODE4 are:

• State transition diagrams

• Network interconnection diagrams

• Physical interconnectedness diagrams.

An example of how to display non-hierarchical relations on a CODE4 graph

The example below shows how “holding tank” concepts called relatedTo X (where X is a concept in an isa hierarchy, and relatedTo X has a non-hierarchical relation to X) have been set up for COGNITERM optical storage technology TKB, a terminological knowledge base built using CODE4.

<u>Creating "relatedTo concepts" and displaying them on the graph</u>

RelatedTo X concepts are used as holding tanks within COGNITERM to display concepts about which little is known except that they are somehow related to X, but they do not have an isa relation with X.

Note: ‘Cleartalk parsing?’ must be set to **yes** in the environment control panel or the non-hierarchical links between relatedTo X and X won't show up on the graph.

In an ‘outline isa hierarchy with editable statements’ browser:

1) add the property ‘r’ (representing relatedTo) to the system concept ‘metaconcept within self’. This ensures that all concepts in the kb have this metaconcept property (although most won’t make use of it).

2) add several ‘normal’ concepts under T (i.e. with isa relations), ensuring that there is one concept under T from which a whole graph of the subject field can be generated. For example, for the optical storage technology TKB, the “true” top concept of the TKB was **storage** (a direct subconcept of T), and a graph for the whole subject field was generated from it. Storage has the subconcept optical storage, which in turn has the subconcepts optical storage media, optical storage devices and optical storage processes.

[try storage .. optical storage.. media, devices, processes]

3) create a concept called **relatedTo concepts** directly under T.

4) create the appropriate subconcepts under **relatedTo concepts**: these subconcepts will all be called relatedTo X, with X being the concept with which the appropriate relatedTo concept will link up.

5) At each concept (X), fill in the correct name of the relatedTo concept for the background characteristic r.

e.g. storage has metaconcept property **r:**relatedTo storage

optical storage has metaconcept property **r:**relatedTo optical storage

optical storage media has metaconcept property **r:**relatedTo optical storage media

6) In the outline isa hierarchy browser, select the top concept of the 'normal' hierarchy that you want to show on the graph (e.g. **storage**) and open a static subtree graph (subwindow>> subtree>>graphical>>detached static).

On the graph:

7) Ensure that all concepts on the graph are selected by ctrl-selecting the top concept (e.g. **storage**).

8) Open a metaconcept property browser (using subwindow>>metaconcept properties>>outline>> detached dynamic) while still on the graph, and select r (the "link" between each X and its relatedTo X).

9) Select **format>>display meta props on isa graph** (middle mouse button).

The various relatedTo X concepts (and any of their subconcepts) should be displayed relatively near to their respective Xs on the graph. The link between each X and relatedTo X will be labelled **r** (representing "relatedTo").

10) Rearrange the layout of the graph as desired to best display the various concepts, and save the layout using **format>>save layout**.

Note: if new concepts (Xs) are added to the classified hierarchy, they will appear on the graph. If any new relatedTo Xs are added, steps 4, 5, and 7-9 will have to be repeated. You will probably also want to rearrange and resave the graph format to accommodate these new additions.

5.3.18.4 Editing arbitrary relations textually

At the current time, editing a relation graph causes statement values to update. You can also edit these statement values directly – see section 4.3.7.

5.3.19 Specifying order

Sibling concepts in the isa and property relations can be ordered with respect to each other. The following commands on the edit submenu allow this:

rotate _..._ up [^r]: Causes all selected concepts to be moved up one position with respect to non-selected siblings. This does not change any parent-child relationships. If all siblings of a given concept are selected, this has no effect. Rotating-up a concept that is the first listed child makes it the last listed child. Rotating-up the root of a subtree rotates the whole subtree. A repetition factor may be specified (see section 4.3.4.4).

rotate _..._ down [^g]: Inverse of the above.

The effect of rotating is immediately apparent in the outline paradigm. In the graphical paradigm, an attempt is made to always keep nodes in the same place (to preserve user rearrangements). It is necessary to reformat a graph to see the effect of rotating.

At the current time, rotation is effective only with respect to the first parent. In future it will be possible to specify a particular parent about which to rotate.

5.4 Querying a knowledge base

There are three main ways by which you can query the knowledge base: 1) using the regular mask or the selection criteria, 2) using networks of dependent browsers, and 3) generating hardcopy output. These are described in the following subsections.

5.4.1 Masks

Items in the 'visibility' submenu (figure 16) control the regular mask and the selection criteria. The regular mask and the selection criteria can also be opened and edited directly (see section 4.4).

As noted in section 4.4, the regular mask is used to control what nodes are displayed, while the selection criteria is used to cause a set of nodes to be highlighted (selected). Both use objects called *knowledge masks*.

Before any node is displayed, the regular mask is asked, 'should this node be displayed?' A knowledge mask contains an *expression* that is evaluated and can return either true (yes, this concept should be displayed) or false (no, this concept should not be displayed). At the current time, the expression is merely a list of *mask predicates* (functions which also return true or false), and all mask predicates must be true before a concept is displayed. Therefore, if a concept is not displayed, and you remove a mask predicate that is causing it not to be displayed, the concept may not immediately reappear; this is because there may be other mask predicates that are also causing the concept to not be displayed.

The selection criteria (the other mask-based capability) is called into action *on demand,* i.e. the user must explicitly request that a set of concepts be selected. (Most of the time, the user selects concepts by mousing on them, instead of using the selection criteria.) The regular mask remains constantly active, restricting what will be displayed.

5.4.1.1 The visibility submenu

The visibility submenu (section 4.3.4.5, figure 16) has the following items (with default hot keys listed directory after the menu item); except where noted, these items update the regular mask:

exclude _..._ [^h]: Causes the selected concepts to be removed from display (either hidden or minimized - see section 4.3.4.9).

allow inclusion of _...._[^j]: If no other predicate is preventing display of the selected concepts, reverses the effect of 'exclude _..._' so that the selected concepts are displayed. In order to select concepts for this operation it is necessary to be showing masked out concepts as 'minimized' nodes. Otherwise, there is no way the concepts could ever be selected.

exclude _..._ subtrees {n}[^t]: Causes the subtrees of the selected concepts to be removed from display. The word 'subtrees' depends on the kind of hierarchy being displayed: it may mean subconcepts, subproperties or children or some other relation. A repetition factor, {n}, may be specified; if so, then subtrees at a depth of n below each of the selected concepts are hidden.

allow inclusion of _..._ subtrees [^y] : Causes the subtrees of the selected concepts to be redisplayed, if they are not also masked out by some other predicate.

exclude all but _..._ hierarchies: Causes any concept that is not a parent or child of one of the selected concepts to be removed from display. Displays only the hierarchies of the selected concepts. This command can be also issued from the 'honly' action button.

exclude all but _..._: Causes only the selected concepts to remain on display.

exclude all instances: Causes instance concepts to be removed from display.

allow inclusion of instances: Causes instance concepts to be redisplayed as long as they are not masked out by some other predicate.

exclude system concepts: Causes system concepts to be removed from display. This is the default in the isa hierarchy because the user rarely wants to see all the statements, metaconcepts and terms.

allow inclusion of system concepts. Causes system concepts to be redisplayed as long as they are not masked out by some other predicate.

exclude non-leaves: Only shows leaf nodes; i.e. those that have no children. This command does not use the mask.

include non-leaves: Allows non-leaves to be redisplayed as long as they are not masked out.

limit traversal depth to {n}: Prevents children deeper in a hierarchy than a certain number from being displayed. The number is specified as a repetition factor. This functionality is similar to 'exclude _..._ subtrees' with a repetition factor, except that it applies from the root of the hierarchy and does not involve the mask. Every knowledge map has a default traversal depth limit (very large in the case of isa and statement hierarchies) to prevent possible infinite regress. For a relation map, the limit is essential because new concepts can be generated dynamically and infinite regress can easily occur.

open mask: Displays a mask view on the regular mask (section 4.4). This allows more detailed mask operations than is possible with the visibility menu.

reset mask to default: Sets the mask to the default for the particular hierarchy, redisplaying most concepts. System concepts are hidden in the isa hierarchy.

reset mask to show everything: Sets the mask so that all traversable concepts are displayed.

open selection criteria: Displays a mask view (section 4.4) on the selection criteria. See section 5.4.1.4 for the ‘rapid goto’ way of using the selection criteria.

5.4.1.2 Predicates

The following describes some of the most important predicates that can be manipulated in the mask to control what is displayed in a browser subwindow. See section 4.4 for details of adding, deleting, negating and editing the arguments of predicates.

Predicates can be very easily added, and we solicit ideas from users for new predicates. We plan to add a significant number of predicates to allow such things as searching for concepts with specific metaconcept and term properties or statements.

has a name matching the string: Searches for concepts whose names match the pattern. Note that this generally is the *primary* term; this predicate will not find concepts whose other terms matches the pattern (see ‘has any term matching the string’).

has a property value matching the string: In a statement hierarchy, shows just those statements whose body matches the pattern. In a property hierarchy (no subject selected) shows just those properties that have any statement value matching the pattern. In an isa or relation hierarchy, shows just those concepts that have any property with a statement matching the pattern.

has any term matching the string: Searches for concepts any of whose terms match the pattern.

has empty property value: Same as 'has a property value matching the string:' except that the pattern is the empty string. This is a very useful predicate because it restricts the display to statements that are in some sense 'complete'. When negated, it can be used to find 'incomplete' statements.

inherits all of the properties: Searches for those concepts that have all of the properties in the set.

inherits any of the properties whose name matches the string: Searches for those concepts that have properties whose name matches the pattern.

inherits any of the properties: Searches for those concepts that have any of the properties in the set.

inherits to all of: Searches for properties that inherit to all of the concepts in the set. By default in a statement hierarchy, the mask is set to restrict display to properties that inherit to all of the selected subjects. The predicate can be removed, but an optimization will still prevent all statements from being displayed. It is often useful to add extra concepts to the set.

is a descendant of (or equal to) one of: When negated, restricts subtrees from display. See '**exclude _..._ subtrees**' and '**allow inclusion of _..._ subtrees**' above.

is a system concept (term, statement or metaconcept: When negated, restricts system concepts from display. See '**exclude system concepts**' and '**allow inclusion of system concepts**' above.

is an instance: When negated, restricts instances from display. See '**exclude all instances**' and '**allow inclusion of instances**' above.

is in the hierarchy of any of: Searches for concepts in the hierarchies of those concepts in the set. This can be set using the 'honly' action button. See '**exclude all but _..._ hierarchies**' above.

is included in the set: Searches for concepts in the set. See '**exclude all but _..._**', '**exclude _..._**' and '**allow inclusion of _..._**' above.

5.4.1.3 Typical queries

The following points describe typical queries the user may want to do. For any querying to be effective, it is essential that subwindow updates not be deferred (sections 4.2.1 and 5.3.17)

• To find all the concepts that have the word 'CD' somewhere in any of their terms:

1) Use '**reset mask to default**' to clear out any previous query

2) Open a mask (section 4.4)

3) Add the predicate '**has any term matching the string**'

4) Edit the predicate argument to: '*CD*'

5) Choose '**apply changes**' or '**accept and apply changes**'.

• To find all the concepts that do not have properties referring to cars (i.e. that don’t have ‘cars’ found anywhere in their value)

1) Reset and open a mask as above.

2) Add the predicate '**has a property value matching the string**'

3) Edit the predicate argument to '*car*'

4) Negate the predicate

5) Choose '**apply changes**' or '**accept and apply changes**'

5.4.1.4 The ‘goto’ capability

The selection criteria capability allows you to select (highlight) a concept or set of concepts based on any criteria. The graph or outline browser are also repositioned (e.g. collapsed subtrees are temporarily uncollapsed) so that any hidden selected concepts can be seen. A common use of this capability is to ‘go to’ a single concept with a known name.

There are two ways to access the selection criteria functionality:

1) Using the ‘selection criteria’ window:

• open this window from the visibility window

• edit the predicates (just as you would do with regular mask)

• issue the ‘generate new selection’ command.

2) Using the ‘quick goto’ capability

• In any browser, type a greater-than sign followed by the name of the concept that should be selected (e.g. ‘>car’ to cause ‘car’ to be selected and positioned so it is visible).

• Normally typing into a browser causes a rename operation to take place, but CODE4 knows that that ‘>‘ symbol indicates that the goto operation should be performed instead.

• It is possible that the concept you want to go to is masked out. If this is the case, then type two greater-than signs followed by the concept name, e.g. ‘>>car’. In this case, the concept will appear regardless of whether the mask is hiding it or not. Subsequently refreshing the subwindow (^e) restores the mask.

• The quick goto capability uses the ‘has a name matching the string’ mask predicate.

• Note that wildcards are allowed in the strings you specify in the quick goto (just as they are when editing the selection criteria window). Be careful when using the mask override capability (two ‘>‘ signs) with wildcards because unwanted system concepts may appear.

5.4.2 Networks of dependent browsers

By setting up networks of dependent browsers, it is possible to pose rapid queries to the knowledge base. When one browser is dependent on another, selections made in the latter cause updates to the content of the former. For more details see the following sections:

Section 5.1.3.2 on opening dependent browsers

Section 5.3.18 on various relations that can be displayed

5.4.3 Hardcopy output

To generate a text file describing a set of concepts, select the concepts to be output and then select one of the following commands from the 'hardcopy' menu:

> **standard**: outputs descriptions of the concepts as in figure 52.
>
> **list of all concepts shown**: all the concepts shown in the selected browser or subwindow
>
> **Cogniterm and Doug's special**: alternate output formats specified by some users

```
Concept name: reflective optical videodisc
Properties:
        dimensions/diameter:
                value: 8 or 12 inches
                modality: i r n
                status: in progress
                reference: ELSHAMI90 p.8&13
                original source: concept of read-only optical disc
                immediate source: concept of reflective optical videodisc

        recording technology:
                value: optical
                original source: concept of storage
                immediate source: concept of optical storage

        content:
                value: one or more of: textual data, audio, graphics, still pictures, and motion video
                original source: concept of optical storage media
                immediate source: concept of videodisc
                property description: type of information that can be stored on medium, i.e. textual data, audio,
graphics, still pictures, motion video , type of information that can be stored on medium, i.e. textual data, audio,
graphics, still pictures, motion video

        available recording surfaces:
                value: generally two, but sometimes only one.
                original source: concept of videodisc
                immediate source: concept of videodisc

        storage capacity:
                value: data: 324 MB, sound: 30 minutes per side, video: 30 minutes per side, images: 54,000
                original source: concept of optical storage media
                immediate source: concept of optical videodisc
                property description: the number of 1) bytes of data that the medium can hold, 2) minutes of audio, 3)
minutes or hours of motion video , the number of 1) bytes of data that the medium can hold, 2) minutes of audio, 3)
minutes or hours of motion video


Metaconcept properties:
        subconcepts: (concept of interactive videodisc, concept of 8-inch videodisc, concept of instant jump )
        superconcepts: concept of optical videodisc
        disjoint concepts: (concept of dye-polymer media, concept of phase-change media, concept of digital paper,
concept of optical tape, concept of optical film, concept of compact disc, concept of optical card )
        source properties: concept of material
        English terms: reflective optical videodisc
        French terms: disque optique reflechissant
        knowledge base: opticalv4_1
        classification status: classified
```

Figure 52: Output from the hardcopy menu item

5.5 Customizing browsers

There is a set of standard browsers available from the control panel KBs menu and the launcher. When one of these is open, it is possible to alter it in the following ways:

• Add new subwindows (section 5.1.3.2)

• Delete subwindows (section 5.1.3.2)

• Change the boundaries between subwindows (section 4.3.4.2)

• Set the mask on subwindows (sections 4.4 and 5.4.1)

• Set the format of subwindows (section 4.3.5 and 4.3.6)

• Change the label on the window (Smalltalk functionality - right mouse button)

• Change the size of the window (Smalltalk functionality - right mouse button)

At the current time it is not possible to save a new browser configuration, but we plan to provide such a capability in the future (section 4.2.4)

6 System knowledge

This section describes the built-in (primitive) knowledge. Without exception, these concepts can be renamed, although most can be deleted. If desired, you can mask out concepts (section 5.4) you do not want to see.

When a new knowledge base is created in the control panel using 'create a new default KB', the primitive concepts (and any new superconcepts of them) are the only ones created. When a knowledge base is loaded, primitive concepts are first created so they will exist in the loaded knowledge base even if the ckb format file was missing them. Filing out of ckb format files includes the filing out of primitives; however, other applications that generate primitives need not create all of them.

6.1 Primitive types

There are several primitive type concepts in the system that are used to hold special instances. These are: metaconcept within self, ordinary property within self, term within self, statement within self. None of these can be deleted.

Another concept that cannot be deleted is the top concept 'thing'. This is the ultimate superconcept of all other concepts in a knowledge base.

6.2 Primitive properties

There are two main types of primitive properties:

• optimized properties. These are ordinary, editable properties from the user’s point of view. Values of their statements can contain anything. They have been specially optimized because of frequent use.

• computed properties: These are properties which compute their own value. At the current time the value cannot be edited, although in future it will be possible to edit some computed values in order to make knowledge base edits. An example of a computed property is the metaconcept property 'subconcepts'.

6.2.1 Metaconcept properties

Metaconcept properties are properties of a ‘concept of a concept’. Many properties that do not logically belong with instances of a particular concept should go here. For example a ‘car’ does not have a comment, but a ‘concept of car’ might.

The following metaconcept properties are built-in. Others can be added by the normal property addition process. Built-in properties are more efficiently processed and stored; if necessary, the set of built-in metaconcept properties can be changed.

comment: An optimized property available for the user to document the purpose of the concept, etc.

changer: Computed property. Currently incomplete. Future: the system will be configurable to record information about every edit. Saved information includes what the change was, who did it and when.

related concepts (superconcepts, subconcepts, instances, disjoint concepts, source properties, terms). Computed properties that contain the same information as displayed in the hierarchical structure of browsers. The first three of these can be edited textually;

for the rest it is currently necessary to use the editing commands. When editing textually, it is only possible to add to the list.

kinds: A computed property that shows how subconcepts are broken down into different dimensions.

dimensions: A computed property that contains an informal string indicating how a concept relates to each of its superconcepts (similar to the link label on the graph).

inherited dimensions: A computed property that shows the values of dimensions of all parents.

6.2.2 Statement properties (facets)

Facets are properties of statements. Users can add their own to the following set:

value: The most important facet, and the only one that is essential. Specifies the concept at one side of the relation between two concepts, where the relation itself is represented by the property. For example, the property 'engine: a car engine', has a value 'a car engine'. Values can be informal text, or pointers to other concepts. The value of a value facet is the same thing as the value of the statement.

modality: (optimized) Specifies whether the property is necessary, typical, optional or inappropriate. In future, masking and type consistency checking will make extensive use of this facet.

status: (optimized) The degree to which the knowledge is 'approved'.

statement comment: (optimized) A comment about this statement.

knowledge reference: (optimized) Used to document the source of the knowledge (i.e. who said it, where it was written, etc.).

predicate: (computed) The property that is the predicate of the statement.

subject :(computed) The concept that is the subject of the statement.

most general subject of predicate: (computed) The concept highest in the isa hierarchy that has the property.

sources of value: (computed) The concepts which have statements whose values directly inherit to this statement without further refinement.

6.2.3 Properties of properties

Properties of properties are rarely used, but users can add to the set below. The easiest way to see properties of properties is to open an 'isa hierarchy from x with editable outline statements' and then open a separate properties subwindow from the statement hierarchy. When *no* concept is selected, the statement hierarchy reverts to a property hierarchy, at which time the subwindow shows statements of the selected property.

most general subject: (computed) The concept highest in the isa hierarchy that has this property

superproperties: (computed) The superproperties

subproperties: (computed) The subproperties (shows up as 'p' in the browser because this is how subproperty links are displayed).

inverse: (computed) Specifies which property is the inverse. This property is symmetric in the sense that the inverse will have this property as its inverse. Whenever values of the inverse are changed, values of this property are changed (and vice versa). See section 5.3.12.

6.2.4 Term properties

string: (computed) The characters making up the term. For icon terms, the string can be non-human readable.

meanings: (computed) The concept(s) the term refers to. This may be changed to only be a single meaning.

part of speech (optimized): Whether the term represents a noun, verb, adjective etc.

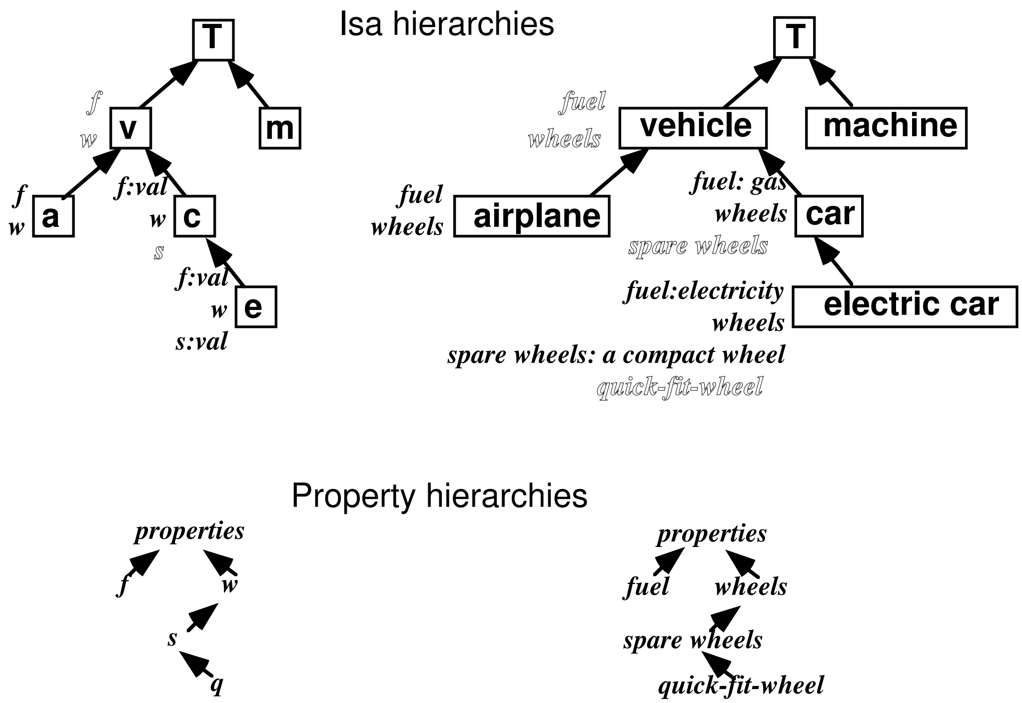
plural (optimized): The plural form of a noun

7. Explanation of Error Messages

This section illustrates some CODE4 error messages that can be hard to understand. The following figure illustrates a typical knowledge base on which seven different operations are to be attempted. In each case, a system constraint prevents the desired operation from being performed.

In the figure, the boxes contain concepts, and properties are in italics. Outline font indicates a property at its ‘most general subject’, i.e. its highest point in the inheritance hierarchy.

The hierarchies at the left contain only the first letter of each concept – these are the letters used in the error messages.



1. You are trying to break a superconcept link from <c> to <v> and create a superconcept link from <c> to <m>. This operation is called ‘reparenting’ <c>.

There is a predicate <f> that is inherited by <c> from or through <v>. But <f> is not currently inherited by <m>. Furthermore, there is a statement <f:val> (with a value) whose predicate is <f> and whose subject is <c> or below.

However, a statement’s predicate must be inherited by its subject.

Your reparenting operation would deprive <f:val> of its predicate (The proposed subject of <f:val> would not inherit <f>), therefore the operation is not allowed.

Possible solutions:

1. Change the most general subject of <f> to be a common superconcept of <v> and <m>, then reparent.

2. If no suitable common superconcept exists: create one (above <v> and <m>) and made this the new most general subject of <f>, then reparent.

3. Delete all values of <f> at or below <c>, then reparent.

4. Add <m> as a new superconcept of <c>, but leave <v> as a second superconcept.

5. (Extreme) Delete property <f> entirely, then reparent.

2. You are attempting to destroy a concept <v> that is the most general subject of a property <f>.

If there were a single subconcept of <v> the system would automatically make that the most general subject of <f>. However, <v> has either no children, or more than one, and the system does not know what to do.

Possible solutions:

1. If <v> has zero children, delete property <f> entirely, then delete <v>.

2. Pick one of the children of <v>, and move <f> there, then delete <v>.

3. Move <f> to a superconcept of <v>, then delete <v>

3. You are attempting to destroy a property <f>.

You have selected subject <c> which is not the most general subject of <f>. This is a warning in case you really only want to delete the value of <f> at <c>, or in case you do not realize that there are statements using <f> as their predicate.

Possible solutions:

1. Simply edit statements using <f> so their values are null, then select subject <c> and delete <f>.

4. You are trying to change the most general subject of property <s> from <c> to <a>. This is called ‘moving’ <s>.

But <c> would no longer inherit <s>. Furthermore, there is a statement <s:val> (with a value) whose predicate is <s> and whose subject is <c> or below.

However, a statement’s predicate must be inherited by its subject.

Your operation would deprive <s:val> of its predicate (The subject of <s:val> would no longer inherit <s>), therefore the operation is not allowed.

Possible solutions:

1. Instead, move <s> to a common superconcept of <c> and <a>

2. If no suitable common superconcept exists, create one (above <c> and <a>) and move <s> there.

3. Delete all values of <s> at or below <c>, and then perform the original move operation.

5. You are trying to change the most general subject of property <q> from <e> to <a>. This is called ‘moving’ <q>.

However, if a subject inherits property <q>, then it must also inherit all the superproperties of <q>, such as <s>.

But <a> currently does not inherit <s>, therefore your move operation is not allowed.

Possible things to do before your operation can succeed:

1. Move superproperty <s> to a common superconcept of <e> and <a>.

2. Change <q> so that it has a different superproperty that is inhertted by <a>.

6. You are trying to make property <f> a subproperty of <s>. This is called reparenting in the property hierarchy.

However, if a subject inherits property <f>, then it must also inherit all the superproperties of <f>.

But <s> is not inherited by the most general subject (<v>) of <f>, therefore your reparenting operation is not allowed.

Possible solutions:

1. Move <s> so that it is inherited by <v> (i.e. move its most general subject up), then reparent

2. Move <f> so that it has the same most general subject as <s> (i.e. move its most general subject down), then reparent

7. You are trying to break a superconcept link from <c> to <v> and create a superconcept link from <c> to <m>. This operation is called ‘reparenting’ <c>.

There is a property <s>whose most general subject is <c> or a subconcept of <c>.

However, if a subject inherits property <s>, then it must also inherit all the superproperties of <s>, such as <w>.

But <m> does not currently inherit <w>

Possible things to do before your operation can succeed:

1. Reparent <w> in the property hierarchy, so that it is no longer a superproperty of <s>.

2. Change the most general subject of <w> to be a common superconcept of <v> and <m>.

3. If no suitable common superconcept exists: create one (above <v> and <m>) and made this the new most general subject of <w>.

Appendix 1. Files Processed by CODE

A1.1 Knowledge Base Files

These are ASCII files containing a compact representation of a knowledge base. The plan is that they will be forward compatible between releases.

Their suffix is .ckb. See appendix 9 for more details on this format, which is also used by the knowledge server capability.

A1.2 Mask Files

Future: masks will be stored for on-demand loading.

A1.3 Environment Files

Future: a whole user's environment might be saved, typically in his or her home directory. Includes all parameters specified in the Environment control panel.

A1.4 Browser Template Files

Future: customized saving of user-defined browser types.

Appendix 2. Important Changes in the Latest Release

CODE4.0 was first released in January 1991. Since then there have been two major releases (4.1 and 4.2), some 25 minor releases and hundreds of incremental patches. The following lists a few of the key enhancements for release 4.2.

• Tab completion as a rapid knowledge entry mechanism

• Ability to pick a concept to be the root of a subtree when opening a browser

• Ability to display union or intersection of properties in the property hierarchy

• Capabilities to show and edit property values in value panes

• Inverse properties

• Improved knowledge server capabilities

• Some new mask predicates

• Property history matrix

• Improvements to dimensions and inherited dimensions

• Ability to edit most metaconcept properties textually

• Reversal of arrows in relation graphs

• Numerous small bug fixes

• Substantial internal cleanup of unneeded code (e.g. removal of unneeded cleartalk functionality)

Appendix 3. Some CODE4 Design Philosophies

The following points illustrate some of the ideas that the CODE4 designers have tried to adhere to:

• The user interface should be as non-modal as possible. By this we mean that by performing an operation, the user does not get into a state where another operation is inaccessible until that first operation is completed. In particular, ‘modal dialogs’ are frowned upon.

• Maximal consistency across interaction paradigms and knowledge representation components.

• Full pluggability of classes where possible (e.g. knowledge map classes, interaction paradigm classes, etc.).

• What the user sees is not necessarily how it is implemented: User documentation may frequently refer to concepts as if they were Smalltalk objects; however, not all concepts that can be referred to need necessarily exist at all times. If a concept contains no distinct information, it can be computed as required (as a temporary concept).

• Maximum availability of options to permit informal knowledge representation.

Appendix 4. Future Enhancement Plans

In addition to the following, see all the points marked ‘future’ throughout this document. These items are listed roughly in order of perceived importance. If you would like us to change our priorities, feel free to drop us a line. Anyone who gives us already-written enhancements will receive our eternal gratitude.

• Improvements to the feedback panel to better handle redo and undo requests, and to give more information and options when a command fails (work underway)

• Improved graph drawing algorithms, especially in the relation map

• Addition of sets as a fundamental class of concept
• Population of the help system with information.

• Improved merging of concepts

• Individual node shapes, fonts and highlighting in graphical view.

• Inheritance enhancements
  - concatenation of values
  - upward inheritance (building of sets)

• Loading and saving of masks, environments and browser templates.

• Making masks full-fledged concepts

Appendix 5. User Enhancement of CODE4 software — A Brief Guide.

CODE4 is designed to be flexible enough for experienced programmers to make enhancements. The classes have been written with, we hope, sufficient documentation and good design that enhancements should be relatively easy.

One of the first things to do when setting out to enhance CODE4 is to change the control panel to ‘developer mode’. This gives access to menu items for inspecting and debugging.

There are three approaches to user enhancements: (1) enhancing the existing system at the user interface level, (2) interfacing external code to the existing system and (3) writing a client to communicate with the knowledge server. The latter aproach is deferred to Appendix 9.

A5.1 Enhancing the existing system

The following points indicate places we believe have the highest potential for extension: (temporarily incomplete as to details)

• Adding knowledge maps (e.g. maps that act like ‘viewpoints’).

• Adding inference capabilities onto the knowledge engine (e.g. forward chaining to highlight consequences, detect inconsistencies)

• Parser enhancements (other languages, etc.)

• Adding specialized editing or query commands.

A useful note for those accessing system innards: It is possible to refer to concepts in the current knowledge base directly, using an extension of the Symbol syntax. (Do not, however, imbed this syntax in permanent methods.) For example:

> #’}}sports car’
>
> will return the Smalltalk object for the concept of sports car.
>
> #’}sports car’
>
> will return the Smalltalk object for the term ‘sports car’.

• Adding mask predicates

• Adding features that help extract knowledge from specific sources and enter it into the knowledge base.

We would always appreciate being sent any enhancements users may make to our system. We would also appreciate consultation about proposed enhancements to help minimize future conflicts.

A5.2 Interfacing to the existing system

Instead of enhancing CODE4’s UI features, one may choose instead to use only its knowledge engine capabilities, adding one’s own front-end.

Such interfacing can be done at two levels: (1) the knowledge map level, and (2) the concept level.

Interfacing at the knowledge map level involves writing software that creates, traverses and edits knowledge maps. The effect of such interfacing is to replace CODE's UI. Knowledge map level interfacing is simpler than concept level interfacing; however, the possible operations are more limited.

Concept level interfacing involves creating and editing concepts, bypassing knowledge maps.

Main interface methods for knowledge map level interfacing

KnowledgeMap>>TraverseAllQuick:with:

Any methods in KnowledgeMap>view-display properties

Main interface methods for concept level interfacing

LongTermConcept>>newSubType

LongTermConcept>>termed

LongTermConcept>>newSubpropertyOf:

Concept>>valueAtProperty:put:

Appendix 6. Auxiliary Tools Delivered with CODE4

This appendix describes several Smalltalk utilities primarily intended for software development that are delivered with CODE4. Separate files are available for these utilities and they will eventually be made available as shareware.

This section is currently incomplete.

A6.1 The call browser

A normal exploration of the Smalltalk environment can require a very large number of successive calls to 'senders', 'implementors' and/or 'messages' to find some method which is doing whatever work is being studied. With each new call, the developer must place a new window somewhere on the screen. Often this means either making each window successively smaller, or loosing older windows under the pile. The CallBrowser was developed to replace most of these windows with a unified tool which keeps track of the expansion which has already been traversed, while making it possible to continue work with the least distraction.

A6.1.1 Description of Function

A CallBrowser has two panes, a list pane on top and a code pane on the bottom (see figure 53). The current list of calls being browsed is displayed in the list pane. The text of the selected call is displayed in the code pane. If the selected call is a message then its parent method is displayed, with the first instance of the message highlighted.

Performing 'senders', 'implementors' or 'messages' from another browser will use the selected method as the source call, with the chosen expansion displayed as its children. In other cases, where there is no preselected item, the result of the expansion becomes a collection of source calls.

Further expansion is tracked by displaying the collection of calls in a hierarchical structure, effectively a list of trees. The developer may select any entry and request that it expand on its senders, implementors and/or messages. If an expansion is made on a call which has no calls of that type, then 'Nobody' is printed to indicate this. Unwanted expansions may be discarded by the menu item 'contract' on the parent call. The protocol of each method may be observed, if desired.

To distinguish an item which has been expanded, new calls are inserted below the selected call with an increased indentation. The browser always places senders before implementors before messages. The type of expansion is also indicated by font style. Senders are displayed in italic, 'implementors in bold and messages are displayed without emphasis. Source calls are also displayed without emphasis.

A6.1.2 Implementation Notes

The CallBrowser is based on and extends the MethodListBrowser. The component views remain a SelectionInListView and a CodeView. The CallBrowser has replaced the MethodListBrowser in almost all of its uses. Only the degenerate case of a single method browser continues to use the older class.

Messages do not understand 'messages', as they lack sufficient information to perform this operation. Implementors do not understand 'implementors' as this would add no new calls to the list. Nobody cannot be expanded at all, as it is merely a visual reminder.

Each call in the browser is implemented as a distinct object, which keeps track of its children. Only the collection of root calls is maintained by the CallBrowser. The SelectionInListView is passed an OrderedCollection formed by a preOrder traversal of all calls. Each call knows how to print itself using printItems set to #printText: .

Figure 53 : A Sample CallGraph Window

A6.2 Hierarchical inspector

This is similar to the call browser in that it allows hierarchical inspecting of instance variables of objects. More details to be provided.

Appendix 7. Building CODE from Sources

This section outlines how to build a CODE4 image if you have access to the source code.

Source code is delivered as a series of files as follows:

• Several Smalltalk files in the pattern ‘CODE4-*.st’. These are categories of Smalltalk classes.

• A directory of Smalltalk files containing system-changes, including files containing special development tools.

• A ‘Make.st’ file.

• Possibly several 'patchnn' files.

In order to build a CODE4 image:

1) Start with a virgin Smalltalk image with the advanced programming toolkit filed in. (Call this the ‘base image’). To build 4.1B, you will also need to file in backwards-compatibility classes.

2) If you are filing CODE4 in on top of some other application, you may want to load this other application first. You may also want to set other Smalltalk defaults (e.g. site printing defaults). In our development environment we actually have a separate makefile to build our ‘base image’. This can be provided if desired.

3) Run the base image and open a file editor on ‘Make.st’.

4) It may be necessary to adjust path names in ‘Make.st’ to suit your environment.

5) Follow the instructions in ‘Make.st’. Most of these involve performing ‘doits’ in order to file in specific files.

6) Some file-ins may cause walkbacks. This is due to circular dependencies in the order in which things are to be filed in (i.e. some classes will not initialize properly). When a walkback is encountered, proceed anyway (three levels down in the walkback); the necessary reinitializations will be done again at the end of ‘Make.st’.

7) Save your new image and test it.

Contact us if you have any problems.

Appendix 8. Knowledge Preprocessor User's Manual

A8.1 Introduction

The knowledge preprocessor has been designed to work with the CODE4 knowledge management system to assist the user in finding the important knowledge in a document and adding it to a knowledge base. It automatically processes each sentence of the document and proposes phrases in the form of one or more sets consisting of a noun or verb with its left and right modifiers. These phrases are then assembled by the user into statements to add to the knowledge base. A complete statement contains a subject, a property and a value; a partial statement contains only one or two of these. The words in the sentence are analyzed using two dictionaries: an external dictionary, called the Kimmo dictionary, and a dictionary that the user builds up in the CODE4 knowledge base.

A8.2 Getting started

The knowledge preprocessor must be opened from CODE4 running on a Sun (4 or greater) and set at the intermediate, expert or developer user expertise level.

A8.2.1 Opening the knowledge preprocessor

The knowledge preprocessor is opened from the CODE4 KB control panel. In the loaded KBs subwindow select the knowledge base you want to work with and choose 'kb applications' and then 'open knowledge preprocessor on _ ' from the menu. You will then be prompted for the name of the text file to be processed. The first thing that is done to the file is to divide it into sentences. If this file has not been analyzed before you will be asked if you want the entire file to be changed to lower case or not. If the file has been processed before, you will be asked if you want to reuse the existing sentence file that was created the first time the file was processed. The name of the sentence file is the name of the original file with '.sentences' added to the end.

A8.2.2 Closing the knowledge preprocessor

To close the knowledge preprocessor, select close from the right mouse button menu. If you have opened a browser on the knowledge base from the preprocessor (see section A8.3.2.3), this browser will also be closed when you close the preprocessor.

A8.3 Using the knowledge preprocessor

The knowledge preprocessor consists of three main parts: a window showing the source file being processed, a window containing phrases selected from the sentence being processed, and a window where statements are composed so they can be added to the knowledge base.

A8.3.1 The source file window

The top window of the knowledge preprocessor displays the name of the file being processed and the text of the file separated into sentences with each sentence numbered for reference. The first letter of each sentence is also made lower case (unless the whole first word is capitalized) to make it easier for the preprocessor to analyze the sentence. It is also done so that words put in the knowledge base start with lower case letters (except when the whole word is capitalized) because CODE4 is case sensitive. Otherwise you could end up with two concepts for the same subject, one with an upper case first letter and one with a lower case first letter.

If the source file window is not big enough to display the entire file, the contents of the window can be scrolled up and down using the scroll bar at the left of the window.

A8.3.1.1 Menu items

The menu obtained by holding down the middle mouse button in the source file window presents the following options:

again find the next occurrence of the string that was last copied, cut or pasted over.

copy copies the highlighted text into the copy buffer. Copy can be used to copy items to and from other windows as well.

cut copies the highlighted text into the copy buffer and deletes the original text.

paste deletes the highlighted text and puts the contents of the copy buffer in its place.

save stores the text in the sentence file if it has been modified. The original text file is not changed.

cancel restores the text to the condition it was in before the file was last saved.

find sentences containing string prompts for a string consisting of a character, word or phrase and picks out each sentence in the file that contains that string. The selected sentences are displayed in a Smalltalk file editor and can be saved or printed if desired.

print file prints the sentence file

initialize CODE dictionary inserts into the built-in CODE dictionary a set of commonly used words such as *a, the, and* etc. This needs to be done only once when a new knowledge base is created.

is selection in CODE dictionary? searches for the selected word in the CODE dictionary. If the word is found its part of speech is indicated and whether it is singular or plural if it is a noun.

add selection to CODE dictionary inserts the selected word into the CODE dictionary. You will be prompted for the word's part of speech and, if it is a noun, whether the word is singular or plural. This can also be used to change the part of speech or singular/plural form of a word already in the dictionary.

look up selection in Kimmo dictionary searches for the selected word in the external Kimmo dictionary. If the word is found, the Kimmo code for the part of speech is shown.

The last menu item, **process sentence**, is described in the next section.

A8.3.2 Processing a sentence

To process a sentence, highlight both the sentence and the sentence number in the source file window by dragging the mouse with the left button held down or by double clicking at the beginning of the sentence. Then select *process sentence* in the middle button menu. The sentence will be analysed and the results will appear in the phrase window along with the sentence number. You can also process a part of a sentence but the sentence number will be shown with the sentence in the phrase window only if it was included as part of the selection.

A8.3.2.1 Unknown words

When a sentence is being processed, a part of speech must be found for each word. The word is first looked up in the CODE dictionary. If it is not found there, it is looked up in the Kimmo dictionary. If it is not in either dictionary, you will be asked to choose the part of speech from the following: *noun, verb, adjective, adverb, conjunction, preposition, qualifier, adverb or modal, and unknown.* The word will then be added to the CODE dictionary. If you do not want to continue processing the sentence at any point, click on *cancel* .

Some words found in the Kimmo dictionary may have more than one part of speech. For example, *dog* can be used both as a noun or a verb. The preprocessor uses rules to attempt to disambiguate such cases but if it is unable to decide the proper part of speech in this context you will be asked to make the choice.

A8.3.2.2 The phrase window

The phrase window consists of three columns labelled left modifiers, noun or verb and right modifiers which display the nouns and verbs with their associated left and right modifiers from each sentence processed. The preprocessor finds one or more complete or partial phrases for each sentence processed.The results for each sentence are indicated by a line containing the sentence number followed by one or more phrases.

If the phrase window is not big enough to display all the phrases that have been selected, the contents of the phrase window can be scrolled up and down using the scroll bar on the left of the window.

A8.3.2.3 Menu items in the phrase window

The middle button menu in the phrase window has the following items:

open browser on knowledge base opens a browser on the knowledge base that is linked to the knowledge preprocessor. The browser is the same kind as if you had opened it from the CODE4 control panel using an 'outline isa hierarchy from x with editable outline statements' template. You may have only one browser open at once. If you attempt to open a second browser, the existing browser will be brought to the front if it is not already there. You can also open a browser from the menu in the knowledge composing window.

edit selection in column allows you to edit the highlighted item in the subwindow (left modifiers, noun or verb, or right modifiers) where the cursor is positioned. It brings up a small dialog window where you can edit the item. When you are finished editing, press return. To cancel the editing while the dialog is still open, highlight the entire edited item, press the delete button to delete the entire item and press return.

copy selection in column copies the item that is highlighted in the subwindow (left modifiers, noun or verb, or right modifiers) where the cursor is positioned and places it in the copy buffer.

cut selection in column copies the highlighted item in the subwindow where the cursor is positioned, places it in the copy buffer and then deletes the original item.

paste over selection in column deletes the highlighted item in the subwindow where the cursor is positioned, then puts the contents of the copy buffer in its place.

remove "the", "a" and "an" from all selected items removes any occurrences of the words "the", "a" and "an" from the highlighted items (if any) in each of the three columns.

remove final s from all selected items looks at the highlighted items in all three columns and removes the last letter of any word that whose last letter is an s. This can be used to quickly change most plural words to singular.

delete all selected items deletes the highlighted item (if any) in each of the three columns.

delete all items on selected line deletes the items in all three columns on a single line that you have indicated by highlighting one or more items.

delete all statements for selected sentence deletes all the statements that were produced by processing one sentence. Indicate the sentence to be deleted by highlighting one or more items belonging to the statements for that sentence.

delete all statements deletes all the statements in the phrase window.

is selection in column in knowledge base? indicates if the item that is highlighted in the subwindow where the cursor is positioned is in the knowledge base and whether it appears as a subject or property or both.

A8.3.3 Composing knowledge to add to the knowledge base

Statements to add to the knowledge base are composed in the bottom window using the items in the phrase window. Statements consist of three parts: a subject, a property and a value.

A8.3.3.1 Assembling knowledge in the knowledge composing window

Transferring words from the phrase window to the knowledge composing window is done using the three buttons at the left of the knowledge composing window labelled *subject*, *property* and *value*. To transfer a line from one of the columns in the phrase window to the knowledge composing window, decide whether it is to be a subject, property or value in the composed statement. Click on the appropriate button and then click on the line in the phrase window. The line will be transferred to one of the three spaces in the knowledge composing window depending on which button you clicked.

If you want to concatenate a word or phrase to the one already in a knowledge composing window, click first on the *concat.* button, then on the *subject, property* or *value* button, and then on the line to copy. It will be added after the existing phrase instead of overwriting it.

You can also copy text from the phrase window and paste it into the knowledge composing window and you can type directly in the knowledge composing window.

If you want to edit the statement you can do it in either the phrase window or in the knowledge composing window. See section A8.3.2.3 for editing commands in the phrase window and section A8.3.3.2 for editing commands in the knowledge composing window.

You do not always have to add a subject, a property and a value to the knowledge base. Sometimes you just want to add a subject or just a subject and a value. The following combinations are valid:

subject with no property and no value

subject and property with no value

subject, property and value

but anything else is invalid. For example, a property and value with no subject is invalid since the knowledge base has no way of knowing what subject to add the property and value to.

A8.3.3.2 Menu items in the knowledge composing window

The middle button menu in the knowledge composing menu has the following items:

open browser on knowledge base opens a browser on the knowledge base that is linked to the knowledge preprocessor. The browser is the same kind as if you had opened it from the CODE4 control panel using an 'outline isa hierarchy from x with editable outline statements' template. You may have only one browser open at once. If you attempt to open a second browser the existing browser will be brought to the front if it is not already there. You can also open a browser from the menu in the phrase window.

make subject a compound noun adds the phrase in the *subject* window to the CODE dictionary as a noun. You will be asked for the corresponding singular or plural form.

copy copies the highlighted text in the subwindow (subject, property or value) where the cursor is positioned into the copy buffer.

cut copies the highlighted text in the subwindow (subject, property or value) where the cursor is positioned into the copy buffer and deletes the original text.

paste deletes the highlighted text in the subwindow (subject, property or value) where the cursor is positioned and puts the contents of the copy buffer in its place.

remove "the", "a" and "an" removes any occurrences of the words "the", "a" and "an" from the items in each of the three subwindows.

remove final s removes the last letter of each word where the last letter is an s, in all three subwindows. This can be used to quickly change most plural words to singular.

make subject synonym of makes the subject a synonym of another word. You will be prompted to enter the second word in a dialog window. One of the two words (but not both) must be already in the knowledge base. After two words are made synonyms, the concept that they both represent in the knowledge base may be referred to by either of the synonyms.

A8.3.4 Adding to the knowledge base

Before you can add anything to the knowledge base there must be a browser open on it. To open a knowledge base browser, use the menu item **open browser on knowledge base** from the middle button menu in either the knowledge composing window or the phrase window.

A8.3.4.1 The 'add to kb' button

When your knowledge is correctly assembled in the knowledge composing window, click on the 'add to KB' button on the left side of the knowledge composing window to add it to the knowledge base. The added knowledge will immediately show up in the knowledge base browser. If you try to add an invalid combination nothing will happen.

A8.3.4.2 ISA relationships

If the statement you have composed consists of a subject, property and value and the property is 'is' or 'are', the preprocessor interprets this as an ISA relationship and assumes that you want to add the subject as a subconcept of the value. For example if the statement had 'rose' as the subject, 'is' as the property and 'flower' as the value it would assume that you want to add rose as a subconcept of flower rather than as a property of flower. If the property is 'are', the preprocessor also attempts to change the plural subject and value to singular by removing the final s (if present) from the subject and value. If this would make an incorrect word (for example 'foxes' would be changed to 'foxe') change the property to 'is' from 'are' and edit the subject and value to be correct before pressing the 'add to kb' button.

Some statements that have a subject, property and value and where the property is 'is' or 'are' do *not* describe a superconcept-concept relationship. For example, the statement 'rabbits are soft' would be proposed by the knowledge preprocessor as an ISA relationship. It is obvious, however, that 'rabbit' is not a subconcept of 'soft'. It is also incorrect to try and add the statement to the knowledge base with the subject 'rabbit', the property 'is' and the value 'soft' (and the preprocessor will not allow it). One way to enter this knowledge is with the subject 'rabbit', the property 'softness' and the value indicating the degree of softness, for example 'great'. This allows you to specify how soft different kinds of rabbit are. Another way to express the knowledge is by giving rabbit a property ‘attributes’ with a subproperty ‘soft’ that has no value. This indicates that all rabbits are soft without indicating how soft they are.

If the statement is an ISA relationship the preprocessor proceeds differently depending on whether the subject (concept) and/or value (superconcept) are already in the knowledge base. If neither is present, it adds the superconcept and then the subject. If the superconcept is already there but the concept is not, it adds the concept as a subconcept of the superconcept.

If the concept is there but the superconcept is not, you will be given a choice of options. You can:

add the superconcept as an additional parent of the concept

change the parent of the concept to be the superconcept

add new concepts for both the concept and superconcept

cancel the operation

If both the concept and superconcept are in the knowledge base, the options available depend on their relationship to each other. If the concept and superconcept are not related (neither is a descendant of the other), you can:

add the superconcept as an additional parent of the concept

change the parent of the concept to be the superconcept

add new concepts for both the concept and superconcept

add another concept with the same name as a subconcept of the superconcept

cancel the operation

If the concept is a descendant of the superconcept but not a child (for example, a grandchild), you can:

change the parent of the concept to be the superconcept

add new concepts for both the concept and superconcept

add another concept with the same name as a subconcept of the superconcept

cancel the operation

If the concept is a child of the superconcept or if the superconcept is a descendant of the concept, you can:

add new concepts for both the concept and superconcept

add another concept with the same name as a subconcept of the superconcept

cancel the operation

A8.3.4.3 Adding a statement that is not an ISA relationship

If the statement you have composed is not an ISA relationship, the preprocessor adds the subject as a concept, the property as its property and the value as its value.

If the subject is not already in the knowledge base, it will add a new concept for the subject. First you will be asked for the subject's superconcept. You can enter the name of a concept that is in the knowledge base or one that is not there. If you specify a superconcept that is not in the knowledge base you will be prompted again for the superconcept of the superconcept and so on until you enter the name of a concept that is present in the knowledge base. The subject and the chain of superconcepts will be added. Next, you will be prompted for the superproperty of the property (if the statement has a property) in a similar way to the superconcept. The property and its chain of superproperties will be added. Finally, the value (if any) will be added.

If the knowledge base already has a concept with the same name as the subject, you will be given the choice of adding another concept with the same name, using the concept already there or cancelling the operation. If you choose to add another concept, it proceeds the same as if it was adding a subject not in the knowledge base. If you choose to use the concept that is already there and you have also specified a property or a property and value to be added, the preprocessor checks if the concept already has a property with the same name. If not, it adds the property (prompting you for its superproperty or superproperty chain) and the value (if any). If the subject already has a property with the same name then you can either add another property with the same name, use the existing property or cancel the operation. If you choose to use the existing property and you also want to add a value the preprocessor then looks at the value of the existing property. If there is already a value present you can choose to replace the existing value, add the new value to the existing value or leave the existing value unchanged.

Whenever you add a statement with a property to the knowledge base, the name of the source file will be added to the knowledge reference facet of the statement in the knowledge base about the subject and property. The knowledge reference indicates the source of the knowledge. The knowledge reference facet may be seen by opening a properties subwindow from the property window of the knowledge base browser. For more information about facets see the CODE4 Reference Manual.

A8.4 What you need to run the knowledge preprocessor

To use the knowledge preprocessor you need the Kimmo dictionary database file called morph_english.db. The path to the dictionary database is set in the class method called initializeKimmoDictionaryPath in class SentenceAnalyzer. If the path to the database is changed, evaluate this method after changing it to the new path.

You must run your image on a Sun4 or greater. If another machine is used, you will not be able to make use of the Kimmo dictionary.

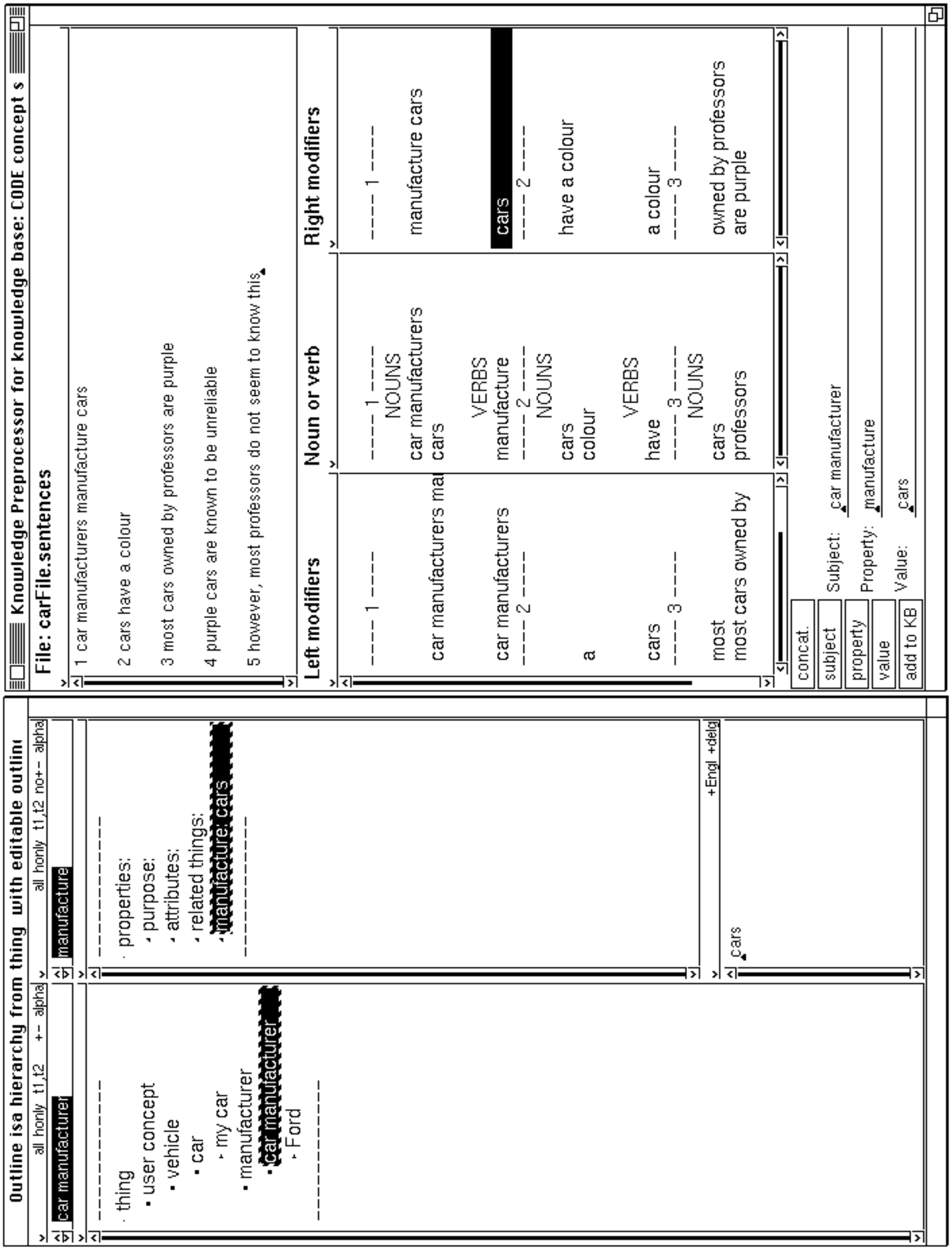


Figure 54 : A knowledge preprocessor being used with an outline browser

Appendix 9. The CODE4 Knowledge Server

The knowledge server is a TCP/IP capability that works in the following way:

• In the control panel, a user starts a knowledge server.

• A client (some special program written in any language with TCP/IP capabilities) connects to the server and initiates a dialogue. The dialogue in in the form of a series of commands.

Any number of clients can connect to a running knowledge server (subject to available resources). Client commands are grouped as follows:

1) Navigators: These pose simple queries to the knowledge base (e.g. connecting to a particular knowledge base, searching for a concept by name, finding the superconcept of a concept etc.)

2) Constructors: These allow the knowledge base to be edited (e.g. adding a subconcept, renaming a concept etc). Currently there is no capability to lock the knowledge base to prevent problems with concurrent access.

The mechanism for storing ckb files uses the same syntax and interpreter.

A9.1 General Information about the Server

The code4 knowledge server is started from the control panel. Using 'Start Knowledge Server'.

The user is prompted for a port number, for a default password and and for allowed hosts. The port number must be an integer. The allowed hosts must be a space-separated list of hostnames.

Once a server is started, any number of clients can connect. Also, any number of servers can be started in an image (using different port numbers, but sharing the password).

A9.1.1 How a client must behave

To connect and use the server, a client must do the following:

1. Set up a socket to the port on the server's host.
2. Read from the socket up to a period (periods enclosed in single-quoted strings do not count).
3. Alternately write a command to the server and read a command.
4. When the last read command starts with 'x', the client must gracefully terminate and close the connection.

The commands are described in from sections A9.2.1 to A9.2.4. The syntax of commands sent to the server is quite important:

• The commands must end in a period (with nothing extraneous following).

• The concept references and other strings must be correct

• Code4 could hang under some circumstances if syntax is incorrect (although attempts have been made to trap and report bad syntax).

When the client wants to quit, it should send an x command. It should then wait for an x command reply before quitting.The client should always quit upon an x command.

A9.1.2 Passwords:

There are two types of passwords:

1. A general server password.

This is the default password that gives read-only access to any knowledge base and read/write access to any non-protected knowledge base. This password is set up when a knowledge server is started. Note that all servers running in the same Smalltalk image share the same password.

2. A knowledge base password.

This is specified for each knowledge base after loading into the server. A user using such a password is granted read/write access to any knowledge base with a matching password or to any unprotected knowledge base, and read only access to others.

This password is specified using the control panel menu item 'kb applications/set server password on_'. The password can be changed at any time, with the possible result that users in the middle of a session may no longer be able to edit. If no password is specified for a knowledge base, then that knowledge base is said to be 'unprotected'. Any user can edit it (assuming they know the general server password).

The first thing a user must do after connecting is enter a password. This must either be the general server password or one of the passwords of the loaded knowledge bases.

The only command that may precede password entry is the navigator 'x', which requests disconnection (i.e. aborts the connection). The password must be preceded by 'w'. E.g. if the password is ‘mypass’, the proper password command would be ‘wmypass.’

A9.2 Commands: Constructors and navigators

A9.2.1 Syntax of references to concepts

Most arguments to commands are in the form of references to concepts. For most long term concepts this is a character string (a base-36 integer involving the digits 0-9 and the upper case letters A-Z). The server may accept lower case letters, but this may result in confusion with command codes or modifiers.

• ***Examples of concept references***: 1, R4, S, 5T, M3, S753

For metaconcepts, the concept reference is a lower-case 'm' followed by the concept reference of the submetaconcept.

• ***Examples of metaconcept references***: m1, mR4, mS, m5T, mS753

For statements, a references has a lower-case 's' followed by the concept reference of the predicate (which must be a property), a '/' and the the concept reference of the subject (which, of course may be a statement with an embedded /' -- the latter case specifies a facet).

• ***Examples of references to statements***: sR4/S, sR4/m5T, s21/sR4/m5T

A9.2.2 Alphabetical list of commands

This lists the legal constructors and navigators alphabetically. The first column is an alphabetic list of codes. The second column indicates whether the code is a constructor (c), navigator (n) or is sent by code4 to the client (s)

• A constructor (c) is used to edit the knowledge base.

• A navigator (n) merely queries the knowledge base

The Third column contains an 's' if the code is sent from the server rather than from the client. The Fourth column describes the code. See A9.2.3 and A9.2.4 for more details.

Command			**Description**
*****	n	s	Ignored - informational
-		s	Informational - failure
+		s	Informational - success
a	c		Create an additional subconcept relation
b	c		Create an additional subproperty
c		s	Single concept response to a command.
d		s	Value response to a command
e	c		Destroy a concept (*** new ***)
f	n		Search for all concepts with a matching term
g	n		List all subconcepts
h	n		List all superconcepts
i	c		Create a new instance concept (see also 'y' to create a type)
j			
k	n		Access a new knowledge base.
l	n		List kb names as text
m		(not a command since this is the code to indicate a metaconcept)	
n	n		Get the name of a concept
o	n		List all properties/subproperties (inherited by a concept)
p	c		Create a new property concept
q	c		Activates and positions a primitive property concept
r	n		Get primitive reference
s		(not a command since this is this is the code to indicate a statement)	
t	c		Create a term concept
u	c		Relate a concept to a term
v	c		Specify a value of a statement
w	n		Get a value given a concept and property ref
x	n	s	Log off, disconnect
y	c		Create a new type concept (see also 'i' to create an instance)
z	cn		Activate and position a primitive type concept or just query a primitive

A9.2.3 Exploring the knowledge base using navigators

This describes the legal navigators. Navigators are one-character commands that allow querying the knowledge base. Many have arguments.

The server always returns a response, which also follows the same syntax (although only a few combinations are possible). In general, the syntax is designed to be symmetric.

See also A9.2.4 for details of the constructors that can edit a knowledge base.

For an alphabetic list, see A9.2.2. The order of presentation below is thematic

? To: Allows the user to obtain help about any of the navigators or constructors.

Arguments:

1. A sequence of characters. Context dependent

   Specify a single navigator or constructor code for help.

Returns:

A string containing help text.

*** To: <u>Irrelevant comment</u> - ignored.**

*** From: <u>Informational message</u> (e.g. help)**

+ From: <u>Last command succeeded</u> (string argument is information only)

Returned upon first connection to the server, after verifying password, after connecting to a knowledge base, and after any command that needs to give no feedback.

- From: <u>Last command failed</u> (string argument explains why).

Example:

```
-'Second argument must refer to a property'.
```

l To: <u>List knowledge bases</u> active in memory

Arguments:

None (any arguments specified are ignored).

Returns:

* followed by a list of knowledge base names and numbers in the format of a string. There will always be at least one knowledge base.

Each knowledge base has (following its name) in brackets a number. Use this number as the argument to the k command to connect to a knowledge base.

Notes:

The numbers should not be counted on to be constant between sessions. Numbers are used because names can be duplicated.

Example:

```
l.
*'default (123), test (456)'.
```
- This server has two loaded knowledge bases

k **To: Access a knowledge base**

Argument:

1. A single integer with the knowledge base number to connect to.

Returns:

+ in response, or - on failure.

May fail if an incorrect knowledge base number was used.

The connection message may say 'read only' if the session password does not match the knowledge base password.

Notes:

Constructors and navigators listed below will not work unless the client is connected to a kb.

Example:

```
k456.
+'Connected to knowledge base <test>.'.
```
- The user may now proceed to navigate or edit the knowledge base.

r **To: Query the about a conceot or a primitive reference.**

Arguments:

1. A primitive reference or a concept reference to query about

Returns

If a valid primitive reference was specified:

c with the concept reference of the primitive

If a valid concept reference was specified d with a string containing general information about the concept. The following fields are separated by commas:

a) I for instance or T for type

b) If an instance:

Stmt for statement

Meta for metaconcept

Term for term

Prop for property

c) L if permanently stored in the knowledge base (Long term). Otherwise you can conclude the concept is computed.

d) The primitive reference if a primitive.

If an invalid primitive or concept is specified, expect - with an error message

Notes:

See A9.4 for more details about primitive references

This concept-lookup capability is largely for convenience, since much of the information can be gleaned from metaconcept properties, Using metaconcept properties is, in fact, the conceptually cleaner procedure.

The string returned describing a concept may be extended with other comma-separated fields in the future.

Examples:

```
r#thingConcept.
c1.
```

- Above looks up the concept reference to primitive #thingConcept

```
r1.
d{'T,,L,thingConcept'}.
```

- Above looks up general information about concept 1

c From: <u>A single concept</u> sent in response to r or any constructor

Following the c is a concept reference (base 36 -- with possible m and s modifiers).

Examples

```
c1.
cm34.
cA2.
cs6/1.
```

d From: <u>A statement value</u> sent in response to w.

The data that follows is surrounded by braces and has the same syntax as passed to the v constructor. May be a list of concepts, a string or some other syntax (described in A9.5)

The contents between the braces may be empty, signifying a null set of concepss.

Examples:

```
d{1}.
d{14,15,I}.
d{'the cat in the hat'}.
```

w To: <u>Given a concept and a property, get a value</u>

Arguments:

1. The subject concept reference.
2. The predicate concept reference (i.e. a property).

Result:

d in response with a string or set of concepts.

\- on failure (e.g. bad concept specified, predicate does not inherit to subject).

Example:

```
w1:6..
d{'a set of thing'}.
```

f To: <u>Search for all concepts which have a term corresponding to a string</u>

Argument:

1. The string to search for in single-quotes.

Result

d followed by a list of concepts

\- if none found.

In many cases there is just one concept in the result, but there can be many.

Example:

```
f'thing'.
d{1}.
```

n To: <u>Get the name of a concept</u>.

Argument:

1. A concept reference.

Returns:

d with a string.

- on failureonly fails when the argument is not a concept)

Notes:

Largely the inverse of f, but will return a string for a concept that has no term.

Will return the string of the main (first listed) term if there are many.

The returned value is purely a string. To get an actual term, get the value of #valueTerms of the metaconcept.

Example:

```
n1.
d{'thing'}.
```

g To: <u>List all subconcepts</u> of a type concept

Argument:

1. A concept reference.

Returns:

d with a list of superconcepts

- on failure

Only fails when the argument is not a concept

Notes:

This is a shortcut. An alternative is to get the value of #valueSubconcepts of the metaconcept (ilustrated below).

Examples:

a) Simply look up the subconcepts

```
g2U.
d{2V,2W,2X}.
```

b) The alternate approach, via the metaconcept property valueSubconcepts

```
r#valueSubconcepts.
c8.
wm2U:8.
d{m2V,m2W,m2X}.
```

- It is up to the client to strip off the m, but the result is the same as a)

c) To find out which of the above are instances, do the following

```
r#valueInstances.
c9.
wm2U:9.
d{m2V}.
```

- By definition the answer was a subset of b)

h To: <u>List all superconcepts</u> of a concept

Argument:

1. A concept reference.

Returns:

d with a list of superconcepts

- on failure

Only fails when the argument is not a concept

Notes:

This is a shortcut. An alternative is to get the value of #valueSuperconcepts of the metaconcept (illustrated below).

Examples:

a) Simply look up the single superconcept

```
h16.
d{3}.
```

b) The alternate approach, via the metaconcept property valueSuperconcepts

```
r#valueSuperconcepts.
cB.
wm16:B.
d{m3}.
```

- It is up to the client to strip off the m, but the result is the same as a)

o To: <u>List subproperties</u> (inherited by a concept)

Argument:

1. A concept reference. This can be empty. If argument 1 is present, only properties inherited by that concept will be returned, otherwise all relevant subproperties are returned.

2. A property, which may be empty. If the property is present, its subproperties are listed. Otherwise the subproperties of the top property (#valueProperties) are listed

Returns:

d with a list of subproperties (possibly empty)

- on failure (e.g. if argument 2 is not a property)

Notes:

This is used to navigate the property hierarchy to get all the properties of a concept, or to get all the properties regardless of concept.

Examples:

a) Simple examples

```
o1:6.
d{CN}.
```
- Above there is just one subproperty of 6 inherited by concept 1

```
o1:CN.
d{}
```
- Above there are no subproperties

```
o::CM.
d{Q,R,S,T,U,V,W,X,Y}.
```
- Above we get all the subproperties of CM, regardless of subject.

```
o1:.
d{2P,2R}.
```
-Above we get all the subproperties of the top property that are introduced at or inherited by concept 1.

b) Here is how we can get related information using metaconcept properties

```
r#valueSourceProperties.
cD.
```
-This property can tell us those properties that are introduced, but it ignores inherited ones.

```
wm1:D.
d{m6,m2P,m2R,m2T}.
```
- Notice that all levels of the property hierarchy are shown, flattened.

x **To: <u>Terminates session</u>.**

Arguments:

None (all are ignored).

Returns:

x in response to get client to clean up and quit.

Notes:

Sent to code4 to ask it to gracefully clean up.

x From: Terminate session. - sent in response to x, end-of-file, bad password or fatal error.

Examples:

```
x.
x.'User requested termination'.
```

A9.2.4 Using constructors to edit the knowledge base

This describes the legal constructors. These are also listed in method 'initializeConstructors' inside the smalltalk software

General description of constructor syntax.

• Each constructor starts with a constructor code. This defines the operation and is a single character. The constructor code is case sensitive (only lower case codes are currently defined).

• Each constructor has one or more arguments. The first argument follows the constructor code immediately. Subsequent arguments are separated by colons or > signs. An exception to this is a 'primitive symbol' argument -- the argument separator preceding one of these must be a # sign.

Examples (semantically meaningless -- illustrating syntax only)

`z:A:B:C.`

- constructor code z
- first argument empty.
- second argument A; third argument B, fourth argument C.

`xR4:#cat.`

- constructor code x.
- first argument R4.
- second argument empty.
- Third argument (a primitive symbol rather than a concept reference since it follows a #) cat.

Many constructors have, as their first argument, a 'resultSpecifier'. This is always optional. Result specifiers are the first argument of constructors that create or activate a new concept. Result specifiers should only be used in environments where no navigators will be used. The result specifier is a concept reference that will act as an alias for the concept created by the constructor. All subsequent constructors should use this alias. Result specifiers are provided so that certain tools can 'precompute' a long series of constructors. They are necessary for example when a list of constructors is read from a file (as in ckb format).

Without result specifiers, subsequent commands would not know which concept specifier to use -- as generally concept specifiers can only be known after prerequisite constructors are executed.

The following constructor codes are sufficient to build a knowledge base.

i **Creates a new instance concept**

Arguments:

1. Possibly empty resultSpecifier (see above)

2. A reference to one of the new concept's superconcepts; a previously constructed type concept.

Returns:

New instance concept (may be referred to using the result specifier). The new instance concept is a subconcept of argument 2.

Example:

```
i:1.
c2U.
```

- Add an instance of concept 1. Concept 2U is created.

See also example e under constructor 'v' below for a shortcut.

Notes:

To add any other superconcepts to the new concept, use constructor code 'a'.

To add a subtype instead of an instance, use constructor 'y'.

It is never possible to have an instance without a superconcept (its type).

An error occurs if argument 2 is an instance concept.

y **Creates a new type concept**

Arguments:

1. Possibly empty result specifier (see above)

2. A reference to one of the new concept's superconcepts; a previously constructed type concept

Returns:

New type concept (may be referred to using the result specifier). The new type concept is a subconcept of argument 2.

Examples:

```
y:1.
c2Q.
```

- add a new concept as a subconcept of concept 1. The new concept is called 2Q.

```
yZZZ>1.
c2Q.
```

- add a new concept as a subconcept of concept 1. We will refer tothis as concept ZZZ in any future constructors because we have used the result specifier.

***Notes*::**

To add any other superconcepts to the new concept, use constructor code 'a'.

To add an instance instead of a subtype, use constructor 'i'.

An error occurs if argument 2 is an instance concept.

z <u>Positions a primitive type concept in the inheritance hierarchy</u>, or merely allows one to query a primitive concept to get its concept reference.

Arguments:

1. Possibly empty resultSpecifier (see above)

2. A reference to one of the primitive concept's superconcepts; a previously constructed type concept (May be omitted in order to simply perform a query).

3. The primitive identifier

Returns:

The primitive concept coresponding to argument 3 (may be referred to using the result specifier). If argument 2 is specified, the primitive concept must be a type concept and is made a subconcept of argument 2

Examples:

```
z:2D#selfTermConcept.
c2.
```

- positions the primitive selfTermConcept (the superconcept of all terms) to be a subconcept of 2D. Does not actually create the concept, since all primitives are created at knowledge base creation time. However this constructor is said to 'activate' the primitive. We are told that the primitive is concept 2.

```
z1:#thingConcept.
c1.
```

- queries which concept is the primitive thingConcept (the top concept in the knowledge base). We are told the answer is c1.

```
z:#selfMetaconceptConcept.
c5.
```

- queries which concept is the common superconcept of all metaconcepts.

Notes:

To add any other superconcepts to the primitive, use constructor code 'a'.

The constructor form of this command is not expected to be used much.

Before a ckb file is executed, a rudimentary knowledge base is constructed, containing the essential primitives. The ckb file need not contain all the essential primitives; those it does not contain will be present in their default position following execution. Those contained in the ckb file will be repositioned.

See A9.4 for further information about available primitives

a Creates an additional subconcept relation

Arguments:

1. A reference to the previously constructed concept that is to have a parent added

2. A reference to the previously constructed concept that is to be the additional parent; circularity is not allowed, i.e. this argument cannot be a subconcept of the receiver

Returns:

Nothing (a + symbol)

Example:

```
y:1
c35.
```
- A subconcept of 1 is created ...

```
y:35.
c36.
```
- A second level subconcept is created

```
y:1.
c37.
```
- Another subconcept if 1 is created

```
a36:37.
+.
```
- 37 is made the second parent of 36

```
g37.
d{36}.
```
- This proves that 36 is now the child of 37

```
g35.
d{36}.
```
- This proves that 36 is still the child of 35

```
h36.
d{35,37}.
```
- This shows that 35 and 37 are both parents of 36.

```
a37:36.
```

```
-'Failed: You are attempting to reparent a concept such
that its new parent is a descendant...'
```
- This shows that circularity is prevented from occurring.

t **<u>Creates a new term concept</u>**

Arguments:

1. Possibly empty resultSpecifier (see above)

2. A reference to one of the new term concept's superconcepts; a previously constructed type concept, that must be a direct or indirect subconcept of primitive type concept #selfTermConcept (or #selfTermConcept itself).

3. A string. Strictly speaking, this argument could have been dispensed with: The string could have been added with the v constructor, however a term without a string is considered of little use, so it was decided to specify the string upon term creation. The string can be a word or phrase in any language, or an encoding of some other symbolic entity such as a picture.

Returns:

A new term concept, to be related to the result specifier

Note:

A term concept is a special case of an instance concept

Examples:

```
r#selfTermConcept.
c2.
```
- The superconcept of most terms.

```
t:2'object'.
c3I.
```
- New term has been created

u **<u>Relates a concept to a term that can be used to refer</u> (possibly ambiguously) to the concept**

Arguments:

1. A reference to a previously constructed concept to which an additional term concept is to be related.

2. A reference to the previously constructed term concept which is to be related to the receiver

Returns:

+ if successful.

Examples:

```
u1:3I..
+.
```
- The term is associated with the top concept.

```
r#valueTerms.
cE.
wm1:E.
d{'thing, object'}.
```
- This shows the concept has synonyms.

p **Creates a new property concept**

Arguments

1. Possibly empty resultSpecifier (see above)

2 A reference to one of the new property concept's superconcepts; a previously constructed type concept, that must be a direct or indirect subconcept of primitive type concept #selfOrdinaryPropertyConcept (or #selfOrdinaryPropertyConcept itself)

3. A reference to one of the new property concept's superproperties, a previously constructed property concept.

4. A reference to the new property concept's most general subject, a previously constructed or derivable concept that inherits the concept referred to in argument 1

Returns:

c with a property concept

Notes:

To add any other superconcepts, use constructor code 'a'.

To add any other superproperties, use constructor code 'b'.

Example:

```
r#selfOrdinaryPropertyConcept
c3.
```
- Above is the most usual superconcept for a property

```
r#valueProperties.
c6.
```
- Above is the most usual superproperty for a property

```
y:1.
c34.
```
- This will be our most general subject

```
p:3:6:34.
```

```
c35.
```
- The new property is created under the top property.

```
p:3:35:34.
c36.
```
- Creates a second level property under the first.

```
p:3:35:1.
    -'Failed ...'
```
- Because the most general subject must inherit the superproperty

See also example b) of constructor e below.

q **Ensures the activation of, and positions, a primitive property concept**

Arguments

1. Possibly empty resultSpecifier (see above)

2. A reference to one of the new property concept's superconcepts; a previously constructed type concept, that must be a direct or indirect subconcept of primitive type concept #selfPropertyConcept

3. A reference to one of the new property concept's suprproperties, a previously constructed property concept

4. A reference to the new property concept's most general subject, a previously constructed or derivable concept that inherits the concept referred to in argument 1

5. The primitive identifier

Returns:

c with primitive property concept reference

Notes:

To add any other superconcepts, use constructor code 'a'.

To add any other superproperties, use constructor code 'b'.

This is relatively rarely used

Use r to just look up a primitive property reference

Example:

```
q:3:M:5#valueLastChangeReason.
```
- Activate primitive property valueLastChangeReason; make it a subconcept of concept 2; make it a subproperty of property M; make concept 5 its most general subject.

b **Creates an additional subproperty relation**

Arguments:

1. A reference to the previously constructed property that is to have a superproperty added

2. A reference to the previously constructed property that is to be the additional superproperty; circularity is not allowed, i.e. this argument cannot be a subproperty of the receiver

Returns:

+ on success.

Example:

```
p:3:35:34.
c38.
p:3:35:34.
c39.
p:3:38:34.
c3A.
b3A:39.
+.
```

- A diamond is successfully formed.

v **Specifies the Value of a statement**

Arguments:

1. A reference to a previously constructed or derivable concept that is to be the Subject of the statement of which the value is to be specified

2. A reference to a previously constructed property that is to be the Predicate of the statement of which the value is to be specified

3. A User language argument, representing the value

Returns:

+ to indicate success. (can only fail on a syntax error).

Notes:

Values of statements inherit to the same property in subconcepts unless overridden.

Examples:

a) Create a subconcept, specify a value at the superconcept, verify inheritance

```
y:1.
c2Y.
v2Y:6{1}.
+.
```

```
v1:6{1}.
+.
```
- the value is specified at the top concept

```
w2Y:6.
d{1}.
```
- the value indeed inherits

b) Specify a more complex value composed of a list of concepts.

```
v2Y:6{1,2,3}.
+.
w2Y:6.
d{1,2,3}.
```

c) Specify an informal value and look at the recursive nature of the value facet

```
v2Y:6{'this is arbitrary text'}.
+.
w2Y:6.
d{1,2,3}.
```
- This is what we see by directly looking at the value

```
r#value.
cR.
```
- We can also get the value by looking at the value property of the statement

```
ws6/2Y:R.
d{1,2,3}.
```
- It is the same thing as expected

```
wsR/s6/2Y:R.
d{1,2,3}
```
- This can be done recursively forever.

d) Specify the value of a primitive facet (the knowledge reference)

```
r#valueKnowledgeReference.
cV.
vs6/1:V{'Encyc. Britannica 1990, Vol 6 Page 206'}.
+.
```

e) Specify the value of a metaconcept property

- We will use the metaconcept property 'instances'. Specifying a string as a value of this automatically adds a subconcept (an instance since the concept only has instances).

- First we will look up the metaconcpet primitive property

```
r#valueInstances.
```

```
c9.
```
- Next we look at the instances of a concept

```
wm2Y:9.
d{m32,m35,m37}.
```
- There are already three. We could add another using i:2Y. and then give the new instance a term. However the following shortcut can also be used.

```
vm2Y:9{'cow'}.
+.
```
- We just modified the value. The results are shown below:

```
wm2Y:9.
d{m32,m35,m37,m39}.
n39..
d{'cow'}.
n3A.
d{'term ''cow'''}.
```

e **Destroys (Eliminates) a concept.**

Arguments:

1. A reference to a concept that can be eliminated (one that does not require the destruction of another concept first).

Returns:

+. if successful.

- with an error message if unsuccessful

Notes:

You must eliminate a cocept's introduced properties before you eliminate the concept.

Eliminating a statement eliminates the value of the statement; to undo this you can use the v constructor.

Examples:

a) Create a subtype of concept 1; eliminate it and verify its elimination

```
y:1.
c2R.
e2R.
+.
r2R.
```
-'Concept 2R was not found in the knowledge base'.

b) create a subject, a predicate and then a value. Then try eliminating them.

```
y:1.
```

```
c2S.
```
- Concept 2S is created

```
p:3:6:2S.
c2T.
```
- Property 2T is created.

```
v2S:2T{'cat'}.
+.
```
- The value is specified for this statement

```
e2S.
-'Failed .....'.
```
- Elimination can only be done after properties are removed from the subject.

```
e2T
+.
```
- The property is eliminated

```
e2S.
+.
```
- The subject can now be eliminated.

c) Look at the value of a statement, eliminate it, then look again

```
w1:6.
d{'a set of thing '}.
es6/1.
+.
w1:6.
d{' '}.
```

A9.3 Examples using commands to work with terms

These examples follow one after the other and build on what waslearned before

Start by finding needed primitives

```
r#selfTermConcept.
c2.
r#valueTerms.
cE.
r#valueMeanings.
c14.
r#valueString.
c13.
```

EXAMPLE 1: Assignment of terms to a concept.

- Add a new concept:

```
y:1.
c4O.
```

- Add a new term and assign the term to the concept:

```
t:2'car'.
c4P.
u4O:4P.
+.
```

- Next we prove that the term is correctly assigned:

```
f'car'.
d{4O}.
n4O.
d{'car'}.
```

- We can also look in the valueTerms property:

```
wm4O:E.
d{4P}.
```

- Or in the valueMeanings property:

```
w4P:14.
d{4O}.
```

- Or in the valueString property.

```
w4P:13.
d{'car'}.
```

EXAMPLE 2: Working with synonyms

- Now let us add a synonym term to the original concept

```
t:2'automobile'.
c4Q.
u4O:4Q.
+.
```

- And let us look at several results:

```
f'automobile'.
d{4O}.
f'car'.
d{4O}.
wm4O:E.
d{4P,4Q}.
n4O.
d{'automobile'}.
```

- Notice that the last term entered and listed becomes the 'main term'. Now let us rearrange the order of terms by using valueTerms. Unfortunately there is a quirk in the system that reverses the order so we have to specify the terms backwards. (main term first).

- Note that we can add and remove terms from a concept using this mechanism too - see below.

```
vm4O:E{4P,4Q}.
+.
```

- Let us look at the results ('car' has become the main term):

```
wm4O:E.
d{4Q,4P}.
n4O.
d{'car'}.
```

EXAMPLE 3: Removing terms from a concept

- We can remove one of the terms (4Q).

```
vm4O:E{4P}.
+.
wm4O:E.
d{4P}.
```

- If the term did not have any properties of its own, nor was it the term of another concept, then the term is deleted.

```
n4Q.
-'Concept 4Q was not found in the knowledge base'.
```

EXAMPLE 4: Changing a term's string.

- We can change the string of a term. This changes the the term itself (and possibly the concept names of several concepts). This should only be done if there is a spelling mistake because assigning a new term to a concept is normally preferable since this does not affect other concepts.

```
r#valueString.
c13.
w4P:13.
d{'car'}.
v4P:13{'kar'}.
+.
f'car'.
-'No concept found with term: car'.
f'kar'.
d4O.
```

EXAMPLE 5: Creating multiple concepts with the same term.

- This can be done in several ways:

- The first way is simply to assign a term to two concepts:

```
y:1.
c4R.
u4R:4P.
+.
```

- We can then see that the term has several meanings:

```
w4P:14.
d{4O,4R}.
f'kar'.
d{4O,4R}.
```

- A second way to edit the list of concepts (to add or delete) is to edit the valueMeanings property.

```
y:1.
c4S.
v4P:14{4O,4R,4S}.
+.
f'kar'.
d{4O,4R,4S}.
```

- A third way is to edit the valueTerms property

```
y:1.
c4T.
vm4T:E{4P}.
+.
f'kar'.
d{4O,4R,4S,4T}.
```

EXAMPLE 6: Getting rid of all terms from a concept.

- The resultis that the system generates a name.

```
vm4T:E{}.
+.
f'kar'.
d{4O,4R,4S}.
n4T.
d{'specialized thing'}."! !
```

A9.4 Symbols used to refer to primitive concepts

This section describes codes to access the primitive concepts in CODE4 then connected to a knowledge server. For more information on primitives see chapter 6.

Primitives are accessed through the q and z constructors and the r navigator. All primitives whose code ends in in 'Concept' are primitive types. All primitives whose code starts with 'Value' are primitive properties

Essential primitives that should exist in each ckb file

```
#thingConcept
#valueProperties
```

Essential primitives that must exist in a loaded knowledge base, but which will be activated by default if not present

```
#selfMetaconceptConcept
#selfTermConcept.
#selfStatementConcept.
#selfOrdinaryPropertyConcept
```

Computed primitives

Optimized primitives accessed by reasoning mechanisms

```
#value
#valueSubproperties               #valueSuperproperties
#valueSubconcepts                 #valueSuperconcepts
#valueInstances
#valueSubject                     #valuePredicate
#valueTerms                       #valueMeanings
#valuePartOfSpeech                #valuePlural
#valueString
#valueMostGeneralSubject
#valueMostGeneralSubjectOfPredicate
#valueSourcesOfValue
#valueKnowledgeBase
#valueSourceProperties
#valueDimensions
#valueKinds                       #valueDisjointConcepts
#valueModality                    #valueLayout
```

Primitives present purely for optimized storage of knowledge

```
#valueKnowledgeReference
#valueComment
#valueStatementComment
```

Bibliography

Note: Some of these can be found at URL http://www.csi.uottawa.ca/~tcl

Bradshaw, J., J. Boose, D Skuce, and T. Lethbridge. (1992). Steps Toward Sharable Ontologies for Design Rationale. AAAI-92 Design Rationale Capture and Use Workshop. San Jose, CA, 9.

Lethbridge, T. C. (1991). "A model for informality in knowledge representation and acquisition.". *Workshop on Informal Computing,*, Santa Cruz, Incremental Systems.

Lethbridge, T. C. (1991). "Creative Knowledge Acquisition: An Analysis". *6th Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff

Lethbridge, T. C. and D. Skuce (1992). "Informality in Knowledge Exchange". *AAAI Workshop on Knowledge Representation Aspects of Knowledge Acquisition*, San Jose

Lethbridge, T. C. and D. Skuce (1992). "Integrating Techniques for Conceptual Modeling". *7th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff,

Lethbridge, T. C. and D. Skuce (1992). "Beyond hypertext: Knowledge management for the technical documenter". *SIGDOC 92*, Ottawa, ACM.

Lethbridge, T.C (1994) “Practical Techniques for Organizing and Measuring Knowledge”, PhD Thesis, University of Ottawa

Meyer, I., D. Skuce, et al. (1992). "Towards a New Generation of Terminological Resources: An Experiment in Building a Terminological KB". *13th International Conference on Computational Linguistics (COLING).*, Nantes,

Skuce, D. and I. Meyer (1991). "Terminology and Knowledge Acquisition: Exploring a Symbiotic Relationship.". *Proc. 6th Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, 29.1-29.21

Skuce, D. (1992). “A Wide Spectrum Knowledge Management System.” *To appear in: Knowledge Acquisition* : 49 pp.

Skuce, D. (1992). “Managing Software Design Knowledge: A Tool and an Experiment.” *Resubmitted with reviewers changes to: IEEE Transactions on Knowledge and Data Engineering* : 36.

Glossary

More complete information about most of these terms can be found in the main text. Use the index to find appropriate page references.

action button: A graphical symbol in the menu bar at the top of most browser subwindows. The user can mouse on this in order to issue the command represented by the symbol. At the current time most action button windows are just short text strings.

arc: see *link*.

artificial intelligence (AI): The study of systems that perform functions that, if done by humans, people feel require intelligence (in the literature there are many other definitions, but this one is useful because it allows us to define AI in terms of whatever people think intelligence is). CODE4 is can be considered an AI system because it represents and reasons with knowledge, and most people think that such functions require intelligence. CODE4 research falls in the AI subdisciplines of 'knowledge acquisition' and 'knowledge representation' (q.v.).

associated concept: A concept that comes into existence by virtue of a reference to its superconcept in a property value. This feature is planned for a future release. For example, if we have the statement 'suburbanites have grass-cutting machines which are lawnmowers', we have made a reference to lawnmowers through the value of the statement. But now, what if we want to say something special about those lawnmowers that are the grass-cutting machines of suburbanites? Currently we must make an explicit subconcept of lawnmowers and then add the statement, but with associated concepts we would not have to explicitly add the subconcept. An associated concept will be removed when the statement from which it is derived ceases to exist.

browser: A window containing one or more subwindows that are linked together. A selection in one subwindow causes an update of what is displayed in another subwindow. Each subwindow operates on a knowledge map.

browser subwindow: see subwindow.

browser template: A specification of how to build a browser describing its subwindows and how they are interconnected. Each subwindow description includes the type of knowledge map, the interaction paradigm, etc. There is a set of default browser templates. In the future it will be possible for the user to add new templates

browser type: see browser template

child: A node that is immediately below another in a hierarchy.

ckb format: The language used to represent CODE4 knowledge in files. Each ckb format file contains one knowledge base. Ckb format files can be loaded rapidly into a running CODE4 system; they contain instructions (constructors) for building a knowledge base from scratch. Ckb format files can also be translated into formats suitable for exchange with non-CODE4 knowledge representation systems such as Ontolingua (q.v.). Ckb format was completely redesigned for the 4.1' release of CODE4, but CODE4 will remain fully backward compatible with the old format until the 4.2 release.

Cleartalk: A language used to represent knowledge that closely resembles English but is a very restricted subset of it. The advantage of using Cleartalk is that it allows the user to write in a way close to what he or she is used to (i.e,. English), but to nevertheless be precise (being precise is very difficult in full English). In CODE4, Cleartalk noun phrases can be used to specify the values of statements. Cleartalk was developed by Doug Skuce, and is derived from his PhD research. It continues to undergo evolution.

collapsing: The same as *minimizing*.

command: An operation that can be performed in a browser subwindow, the control panel or elsewhere in the code system. Commands can be issued by using a hot key, selecting from menus or mousing on an action button. The feedback panel lists all the commands that have been issued.

concept: A unit of knowledge. Specialized types of concepts include properties, statements, instance concepts, type concepts, metaconcepts, temporary concepts, etc. Concepts can become the *subjects* about which things can be said. To say something about a concept, one combines it with a *property* making a *statement*. Concepts can be arranged in many types of relationships. In other AI applications, the words 'unit', or 'frame' are often used in place of 'concept'. Also, the word 'concept' sometimes refers only to our notion of 'type'.

control panel: A window through which the user controls aspects of the CODE4 session. The control panel allows loading and saving of files and the setting of various types of default parameters.

coordinate: A sibling concept within one particular dimension. For example, while the concept compact disc has the siblings videodisc, read-only disc, WORM disc and erasable disc (under the parent optical disc), its only only coordinate concept is videodisc. This is because both concepts are found within the dimension of "physical form" under optical disc. Read-only disc, WORM disc and erasable disc are coordinate concepts within the dimension "writability."

copying: The process of preparing a concept for use in a subsequent command. The command is put in a copy buffer. Text can also be 'copied' in a textual window.

deferring: The process of requesting that when the knowledge base is updated, a subwindow is not to automatically refresh. This is done to increase performance on a slow machine and/or when many subwindows are open on a large knowledge base. Deferral can be requested globally from the control panel, or in each subwindow. A more radical radical type of deferral is also available that prevents even dependent windows from updating when the selection of their driving windows changes.

delegation: The process whereby a statement gets its value from the value of another statement about the same subject. For example, if we have the statement 'a car has an engine', we could add the statement 'a car has a heaviest component which is the engine'. Here, 'the engine' is a delegated reference to engine, and the value of the latter statement will be the value of the former (both in 'car' and in any lower-level concepts where we may talk about specific engines). We may also add the statement 'a car has a hottest area which is the hottest area of the engine'. Here 'the hottest area of the engine' indicates that the value should be obtained by looking at the statement of 'hottest region' about 'engine'. In CODE4, there are also certain special forms of delegation that use special symbols – please see the main text for these. Delegation is a kind of *inference mechanism*..

dependent browser subwindow: A browser subwindow that updates automatically when the selection in its driving subwindow changes. For example, in a '... isa hierarchy with editable ... statements' the subwindow in the left drives the two dependent subwindows on the right (e.g. the property hierarchy updates whenever a new subject is selected in the isa hierarchy).

detached browser subwindow: A subwindow that is in a separate window from the subwindow from which it was opened.

dimension: An informal label placed on a subconcept link to specify the criteria by which a subconcept is related to a particular parent. The 'dimensions' property can be edited in the metaconcept, the inverse 'kinds' property and by directly changing the subconcept label. For example, if 'person' is the parent concept, then 'by age' might be the dimension for the concepts 'child' and 'adult', and 'by sex' might be the dimension for the concepts 'man' and 'woman'.

dragging: Selecting several concepts by pressing the mouse button, and moving the pointer over a set of concepts before releasing the mouse button.

driving browser subwindow: A browser subwindow which is linked to one or more other (dependent) browser subwindows. A selection in the driving subwindow causes an update of what is displayed in the dependent subwindow. Closing the window of the driver causes the window of the dependent to close as well..

dynamic browser subwindow: The same as *dependent browser subwindow*.

edge: see *link*.

editing area: An area at the top of the subwindows of the graphical and outline interaction paradigms where the name of a concept can be changed by standard Smalltalk editing methods.

environment: A collection of information that in future functionality will be loadable and saveable as a whole. Included in a user's environment would be: (1) a set of browser types, (2) some default parameters, (3) some masks, (4) knowledge bases that the user wants loaded on startup. Currently, only the default parameters component is manipulable through the control panel.

facet: A property of statements. Examples are 'value', 'modality', 'subject', 'predicate'.

feedback panel: A window associated with a knowledge base that indicates to the user what commands have been executed and what problems have occurred. In the future it will make active suggestions about future actions.

formal knowledge: Knowledge that is represented in a language with a well-defined syntax and semantics, and which can be manipulated by inference mechanisms. Knowledge representation systems vary in their formal capabilities; some are capable of representing anything in first order logic, with the addition of many useful extensions such as set theory. CODE4's semantics is somewhat more limited, because its strengths lie elsewhere. One of CODE4's strengths is its ability to represent knowledge informally (see also *informal knowledge*). Work aimed at translating CODE4 to and from Ontolingua (q.v.) have proved that CODE4 can be given a valid formal semantics. Many of the formal semantics decisions that in other systems would be fixed, remain flexible in CODE4 — the user can declare the semantics of a certain representational scheme he or she uses *within* CODE4.

format: Information associated with each subwindow that refines its appearance. It includes such attributes as font, alphabetical *vs.* hierarchical mode, graph layout algorithm, etc.

graphical interaction paradigm: An interaction paradigm where node concepts can be positioned anywhere within a virtual screen. The display has the visual appearance of a traditional directed graph.

hiding: The process of using the mask to exclude a concept or set of concepts from view. [see also *minimizing].*

hot key: A way of issuing a command by holding down 'control' and pressing an alphanumeric key.

icon: A small picture contained in a term representing a concept, that can be displayed in the graphical interaction paradigm. Also can refer to a picture in an action button that can be displayed in the menu bar at the top of a subwindow.

image: A Smalltalk file that contains a compiled version of the CODE4 software, along with loaded knowledge bases and layed-out windows.

inference mechanism: A process where new knowledge is derived, beyond what the user directly entered. Examples of inference mechanisms in CODE4 are inheritance, delegation and aspects of Cleartalk parsing. Unlike many knowledge representation systems, automatic inference is of secondary importance to CODE4. Of primary importance is the ability to represent knowledge effectively and rapidly, and to help the user to perform his or her own mental inference. Additional automatic inference mechanisms could readily be added to CODE4.

informal knowledge: Knowledge that cannot be processed fully and unambiguously by an inference mechanism, and whose meaning is grounded primarily in the understanding of a human or a computer program. For example, the strings or images that form terms are informal, as are CODE4 statement values that are not parsed as Cleartalk. Several facets and other properties such as 'comment', 'graphical layout' and 'dimension' contain only informal knowledge — useful primarily to people.

inheritance: The process whereby when one adds a property at a more general concept in an isa hierarchy, all more specific concepts also automatically have that property. Furthermore, when a value is added to a statement about a higher-level concept, all statements with the same predicate at lower levels have the same value, unless this is overridden. CODE4 is highly optimized to provide efficient inheritance. Inheritance is a kind of *inference mechanism..*

instance concept: A concept representing a *specific* thing (real or imagined). For example, if you owned a particular dog called 'Lassie', you might describe him in a knowledge base using an instance concept. On the other hand, the concept 'dog' is a type concept because it abstractly describes dogs in general.

interaction paradigm: A specification of look and feel of a subwindow. Examples include 'outline', 'graphical', 'user language' and 'matrix'.

inverse property: A property that is automatically calculated as the inverse of another. This functionality is not yet available.

isa hierarchy: An arrangement of concepts so that lower-level concepts are specialized kinds of higher level concepts. Between any lower-level concept, X and its parent Y, the statement 'an X *is a* Y' holds true. In CODE4, all concepts in a knowledge base are arranged in an isa hierarchy whose topmost element is a primitive concept usually called 'thing'. Normally, only a subset of the entire isa hierarchy is shown at a time.

item: Either a node or a link. Anything that is selectable in a browser subwindow.

knowledge acquisition: The process of gathering knowledge from various sources and representing it in a knowledge base. CODE4 assists in this process by providing representational capabilities and visual feedback about what is being entered (so as to stimulate ideas). The main aspect of knowledge acquisition for which CODE4 is useful is 'knowledge modelling' which considers that much of the knowledge being entered may not exist prior to the knowledge acquisition effort.

knowledge base: A group of concepts that is loaded from a file into the on-board memory of a computer (RAM). Multiple knowledge bases may be loaded at a time. In the future, knowledge bases will be linkable in dependency relationships, but for now they are all independent.

knowledge map: A specification of a network of related concepts. Types of knowledge maps include isa hierarchies (taxonomies), property hierarchies, partonomies, other relation graphs, property tables etc. Knowledge maps are treated as directed graphs and are displayed (textually or graphically) in subwindows of browsers (q.v.). Typically, the concepts in knowledge maps form hierarchies, although this is not necessary. [see also *traversal*].

knowledge mask: A filter that determines whether a concept will be included in a knowledge map (and thus displayed to the user). Concepts that are excluded by the mask are either completely hidden or minimized (see *hiding* and *minimizing*). A knowledge mask contains a logical expression (in future this will be arbitrary CNF – conjunctive normal form – but for now it is just a list of conditions that must all be true for a concept to be included) relating a set of boolean conditions that are applied to each concept. Masks control the *visibility* of concepts; they are used for hiding and showing specific sets of concepts, as well as more detailed patterns of knowledge. Each knowledge map contains a knowledge mask. The conditions that are related by a mask's logic expression are Smalltalk functions called *mask predicates*, that return true or false when given a particular concept as an argument

knowledge preprocessor: A prototypical mechanism within CODE4 for extracting knowledge from a text file. The preprocessor splits the file into a sentences, and then scans the sentences for patterns that it recognizes. Then it proposes statements that the user can add to the knowledge base.

knowledge representation (KR): 1) The process of structuring concepts so they correspond to knowledge represented elsewhere (typically in the mind), 2) The results of the process, and 3) The schema used for the process. CODE4 contains a conceptual KR schema (in which knowledge is composed of concepts, properties etc.), a physical KR schema (implemented in terms of Smalltalk objects), various user interface KR schemas (the interaction paradigms), and an interchange file format KR schema (ckb format). Users create particular representations of knowledge using CODE4.

launcher: a Smalltalk window that contains a menu for opening other windows. Most of the time, CODE4 users will use the control panel instead of the launcher, because the control panel contains many more options.

layout: A set of instructions about how to position nodes in the graphical interaction paradigm. A layout contains a name, and a set of coordinates for each concept. A layout may only cover a few concepts, or all the concepts in the knowledge base. In future it is intended to enhance layout to include font and node shape information. All the layouts in which a concept is involved are available in a metaconcept property.

leaf: A concept that has no children during traversal of a knowledge map. For example, a concept that has no subconcepts when traversing an isa hierarchy. It is possible to restrict an outline or graphical display to ‘leaves only’.

link: A line drawn between concepts (nodes) in the graphical interaction paradigm. A link represents a statement whose subject is pointed-to by the arrow at the end of the link. A link is also sometimes called an *edge* or an *arc*..

long term concept: a concept that is explicitly stored in the knowledge base, as opposed to a temporary concept which is computed when needed.

marquee: A rectangular area that can be dragged-out by the mouse in the graphical interaction paradigm in order to select concepts. The control key must be held down, and the mouse button must be pressed when the mouse is pointing to the background of the subwindow.

mask: See knowledge mask

mask predicate: A Smalltalk function that, when applied to a concept, returns true or false. Mask predicates are used in masks of the mask view and the selection criteria. (do not confuse with ‘predicate’ – q.v.)

mask view: A window that displays the contents of a knowledge mask and allows editing of that mask, and hence querying of a knowledge base. Mask predicates can be added or deleted and their arguments can be edited. Once editing is complete the mask can be ‘applied’ to the knowledge map that contains the mask, resulting in a change to the set of concepts that is being displayed. An alternate way to edit the mask is to use the ‘visibility’ browser submenu. A ‘selection criteria’ view looks and works identically to a mask view, except that concepts are *highlighted* rather than being selectively displayed.

master selection: One of the concepts that form the selection in a browser subwindow. The master selection is surrounded by a crosshatch pattern. The user can change the master selection by clicking with the mouse. Some commands, such as ‘rename’ operate only on the master selection.

matrix interaction paradigm: An interaction paradigm where knowledge is presented in a rectangular array with row and column headers. Currently this paradigm only works with the property table knowledge map, and is at the beta test stage.

meanings: The concepts a term refers to or the concepts that are pointed to by the value of a statement. It is possible to open a subwindow to see the meanings of a term or statement.

merging: The process of adding two knowledge bases to create a larger one.

metaconcept: A concept of a concept. Anything can be represented by a concept, including concepts themselves which are represented by metaconcepts. In CODE4, metaconcepts are instance concepts that are subconcepts of the type 'metaconcept within self'. Most of the time metaconcepts are hidden in the isa hierarchy. To look at the properties of a metaconcept, select a concept whose metaconcept's properties are to be viewed and open a 'metaconcept properties' subwindow.

minimizing: The process of using the mask to make a node or set of nodes in the graphical or outline interaction paradigms appear as a tiny *place holder*. By default, nodes are hidden rather than minimized, but the an action button command '+/-' changes this default. [see also *hiding*].

modality: a facet that describes whether a statement is typical, necessary or optional. For example, birds *typically* fly. It isn't *necessary* to fly to be called a bird. The property 'modality' is a primitive property.

most general subject: A property of a second property that specifies the *domain* of the second property. Only the most general subject and its subconcepts have (inherit) that second property.

multi-selection: see *selection*.

name: A string used to identify a concept. Typically the name comes from the main term for the concept, but the name can also be computed. Most system concepts have computed names, as do most newly-added concepts.

navigation area: The bottom area of a browser subwindow that can be scrolled in order to navigate around the concepts on display.

node: A shape in the graphical interaction paradigm representing a concept. Nodes are linked by edges. The indented elements in the outline interaction paradigm are also sometimes called nodes.

Ontolingua: A knowledge representation schema that is being promoted as a standard for the interchange of knowledge between systems. CODE4 currently contains a prototype capability for translating from Ontolingua to Ckb format and hence to the CODE4 internal format. In future this capability will be extended.

outline interaction paradigm: An interaction paradigm where knowledge is displayed as an indented textual list. The user may display the knowledge hierarchically or alphabetically, and may perform many of the types of operations available in outline processor software.

parent: A concept that is found immediately above another concept in a hierarchy. It is found before its children during knowledge map traversal. Arrows in the graphical interaction paradigm point from children to parents.

partonomy: A type of relation graph showing the decomposition of something into its parts. Typically a partonomy is in the form of a hierarchy with major parts (e.g. 'engine') high in the hierarchy, and increasingly smaller parts-of-parts (e.g. 'piston') lower in the hierarchy. In CODE4, it is typical to have a property that represents the part-whole relation, and then to have subproperties relating lower-level parts to the whole. If the knowledge is entered correctly, and the values of properties are are parsed as Cleartalk, then a partonomy can be displayed by opening a 'relation' subwindow. A partonomy is the most useful type of relation subwindow to draw, but any other property can also form the basis for a relation subwindow. A partonomy is also sometimes known as a 'meronymy'.

preprocessor: see knowledge preprocessor.

predicate: A property considered in its role as the part of a statement that describes the particular relation the subject has with the value. In the statement 'a table has parts which are (legs, surface)', the predicate is 'has parts' (typically the property name would be 'parts'). The property 'predicate' is one of the primitive facets of every statement. (Do not confuse 'predicate' with 'mask predicate' – q.v.).

primitive concept: A concept that is referred to by some of CODE4's computational machinery. Some primitive concepts are essential, and some are optional, but if present allow useful computations to be made. A primitive concept can be a type (e.g. the top of the isa hierarchy, 'thing'; or the types 'metaconcept within self' etc.) or a property (e.g. the properties 'subconcepts' and 'value' are primitive because the ideas of subconcepts and statement value are so fundamental to CODE4; without the subconcept relation, there could be no inheritance).

property: A kind of concept representing a relation between concepts. An example of a property that applies to humans is 'friends'. The fact that such a property is associated with the concept 'person' indicates that it makes sense to talk about 'the friends of a person'. A property describes a relation abstractly only; a statement contains a property as its predicate in order to describe a *particular* occurrence or tuple of the relation (e.g. '*John* has friends'). In CODE4, properties are instance concepts that are subconcepts of the type 'property within self'. Most of the time properties are hidden in the isa hierarchy. To look at the properties of a property, open a '... property hierarchy' from the control panel and from the newly-opened property hierarchy open a properties subwindow. A property is an instance concept, because it represents a particular relation. A property may have subproperties that represent more restrictive relations between concepts. In some other AI applications, the word 'slot' often refers to what we call a property, although 'slot' sometimes also refers to our idea of 'statement'.

property hierarchy: The arrangement of properties into a hierarchy such that the most general properties are higher. The top of the property hierarchy is a property usually called 'properties'. In CODE4, a property hierarchy is displayed in the right hand subwindow of any '...isa hierarchy with editable ... statements' browser. All properties are arranged in one big property hierarchy, but each concept only inherits a subhierarchy of this, with many branches pruned. A property hierarchy that is restricted to displaying the properties of a particular subject becomes a 'statement hierarchy'.

property table: A knowledge map that is displayed using the matrix interaction paradigm, and shows concepts on one axis, properties on the other axis, and statement values in cells of the matrix. A property matrix allows 1) comparison of concepts (differences can easily be seen), 2) knowledge acquisition to fill in cells and thus distinguish concepts, and 3) construction of definitions (by highlighting the most important differences between concepts).

relation: Mathematically, a set of tuples with a common pattern of connection between elements. In CODE4, a relation is represented using a property. Statements using the property as predicate form the tuples.

relation graph: A knowledge map where the edges represent arbitrary statements. Examples of relation graphs are partonomies (q.v.), state diagrams, network interconnection diagrams etc.

refreshing: The process of redrawing a subwindow. This is done automatically by the system whenever a knowledge base is updated, in case the updated knowledge base results in the need to display different knowledge. It is possible to defer, the automatic refresh process, in which case the user may have to explicitly select the refresh command. If multiple knowledge bases have had updates deferred, it may be necessary to request a full update of all related knowledge maps.

reparenting: The process of changing the parent of a concept. For example, to change the superconcept of a concept, or the superproperty of a property. Reparenting is done by issuing a command to a knowledge map. A similar operation, *adding* a parent, is also available.

selection: The set of concepts that is highlighted in a graphical or outline subwindow. The selection can be changed by the user clicking or dragging the mouse, or by opening a selection criteria window or using the 'goto' functionality. There are three reasons to select concepts: 1) to highlight them so they stand out, 2) to cause a dependent browser subwindow to change what it displays, and 3) To prepare for the issuing of some command that will operate on the selection. One element of the selection is called the 'master selection'. The selection is also often called the 'multi-selection' to distinguish it from the master selection.

selection criteria: A facility that uses a knowledge mask to select (i.e. highlight) a set of concepts. It uses a mask view like a knowledge mask, but whereas a normal knowledge mask causes all nodes that to *not* pass to be minimized or hidden, the selection criteria causes all nodes that *do* pass to be highlighted.

set: Mathematically, a collection of members considered as a whole. In CODE4, a set is represented as any other concept, typically by creating a concept called 'set' which has a property 'members'. In future, primitive set concepts will be introduced so that sets of things can be referred to without manually adding a new concept, and so that sets can be described both intentionally (by describing potential members) and extensionally (by specifying individual members).

sibling: A concept that shares a parent with another in a knowledge map. All the immediate subconcepts of a concept are siblings.

Smalltalk: The object oriented programming language in which the CODE4 software is written.

static browser subwindow: A subwindow that is not driven by another, and hence will only changes its contents when the knowledge base is edited, not when a selection is made somewhere. The opposite of ‘dynamic browser subwindow’

statement: A kind of concept representing the occurrence of a particular property (the predicate) at a particular concept (the subject). A statement can be expressed as a sentence. In CODE4, statements are instance concepts that are subconcepts of the type ‘statement within self’. Most of the time statements are hidden in the isa hierarchy. To look at the properties of a statement (which are called facets), 1) open an ‘... isa hierarchy with editable ... statements’ browser, 2) select a statement by selecting a concept (the subject) and a property (the predicate), and then 3) open a properties subwindow. In some AI applications, the word ‘triple’ corresponds to our notion of ‘slot’.

statement hierarchy: A hierarchy of the statements of one or more subjects. The predicates of the statements form a subhierarchy of the knowledge bases property hierarchy.

subject: A concept considered in its role as part of a statement answering the question, ‘what is a given statement *about*?’. Any concept can be treated as a subject because every concept has properties (even of all of the properties are inherited). In the statement ‘a table has parts which are (legs, surface)’, the subject is the type ‘table’. The property ‘subject’ is one of the primitive facets of every statement.[see also ‘most general subject’].

substitution: The process that goes on when a value is being computed using delegation. The concept found is substituted for original delegation specification.

subtree: A set of concepts in a knowledge map that contains a particular concept and all its children, and their children etc. There are three ways to manipulate subtrees in CODE4: 1) Using the control key when selecting a concept to select the whole subtree, 2) Opening a subwindow displaying just one or more subtrees, and 3) Using the visibility menu to hide one or more subtrees.

subwindow: A bordered rectangular area within a window, containing a set of concepts. Associated with each subwindow is a 1) knowledge map that controls which concepts are displayed and how they are connected; 2) a knowledge mask that can cause some concepts to be hidden, and 3) an interaction paradigm that describes the graphical appearance of the subwindow and how the concepts can be manipulated. A subwindow can be ‘driving’ or ‘dependent’ (q.v.).

system concept: A primitive concept that most users would prefer not to see displayed in the main isa hierarchy. The system concepts include metaconcepts, properties, statements and terms.

system control panel: see *control panel*..

taxonomy: In CODE4, this is a synonym for ‘isa hierarchy’ (q.v.), although in common English usage, the term may have a more general meaning.

template: see browser template.

term: a kind of concept that represents a symbol for a concept. Typically a term represents a word or phrase, but may represent a picture (an icon). There can be a many-to many relationship between terms and other concepts, although in general the relationship is closer to one-to-one. In CODE4, terms are instance concepts that are subconcepts of the type ‘term within self’. Most of the time terms are hidden in the isa hierarchy. To look at the properties of terms, select a concept at whose terms you want to look, and open a ‘terms’ subwindow. From the new terms subwindow, then open a ‘properties’ subwindow. Note: We have received some criticism from terminologists for our semantics of ‘terms’. It has been proposed that what we called terms should be called ‘words’, although ‘compound words’ would be allowed. Terminologists feel that terms never have a one-to-many relationship with concepts. [see also name].

temporary concept: A concept that is not stored permanently, but is computed on request. Its existence is only implicit in the knowledge base. Any statement or metaconcept is temporary if the user has not explicitly edited any of its properties (e.g. a statement where the value is purely inherited). In almost all respects temporary concepts behave just like other concepts (long term concepts), so their existence should not be of concern to users.The number of potential temporary concepts is infinite (e.g. one can have a ‘the metaconcept of the metaconcept of the metaconcept of the... of X’ or ‘the value of the value of the value of ... of statement X’) and the entire set of concepts, both temporary and long term, is called the ‘virtual knowledge base’.

textual interaction paradigm: An interaction paradigm under development. It will operate in a similar manner to the outline paradigm, except that editing will take place directly in the navigation area, and concept names will wrap over several lines.

topic: One of the main concepts that the user is interested in describing. The word ‘topic’ is not used in this manual, because topics are treated in the same way as other concepts. However, some of the literature discusses topics, and in CODE2 the only things referred-to as concepts were topics (code4 has introduced many generalities, including the idea of treating properties, statements, terms etc. as full-fledged concepts). Topics are sometimes also referred to as ‘primary subjects’.

traversal: The process of enumerating the concepts of a knowledge map so they can be displayed. A knowledge map does not store the actual network of concepts, it only stores instructions about how traversal is to take place (it stores the top concepts, the set of links to follow and a mask to allow hiding of concepts). Traversal takes place in a depth-first manner; when the traversal reaches a leaf, it backs up and traverses another branch. It is possible that in some knowledge maps traversal could get into a loop: to prevent the system from hanging, a ‘restrict traversal depth’ capability is provided.

type: A kind of concept that represents an abstraction, not a particular thing. A type is holding-place for a partial description: a set of properties called the intension. A type can also be considered to represent a set: all those things that fit the description.

user language: Text that a user types when describing a property of a concept (i.e. a statement). A user may restrict the language to a specialized syntax called Cleartalk that the system can parse and process.

update: The process of changing a knowledge base, or the contents of a window to reflect a knowledge base update. [See also *refresh].*

user language interaction paradigm: An interaction paradigm only displaying text containing the value of a single statement.

value: A facet of a statement that describes what the subject is related to by the predicate. In the statement ‘a table has parts which are (legs, surface)’, the value is ‘(legs, surface)’. A value may be informal, in which case it is merely a string that is subject to human interpretation. Or the value may be a formal pointer to a concept or a set of concepts, or a *delegation* rule for such a concept or set of concepts. The value the only facet that inherits *automatically*, and the property ‘value’ is primitive and highly optimized. A formal value may point to an instance concept, indicating that the statement is ‘ground’ (that the subject is related to a specific thing), or the value may point to a type, indicating a constraint on the *range* of the property (only instances of the type should be specified as the values of statements of instances of the subject).

virtual knowledge base: The infinite set of all concepts that can be accessed in a knowledge base. It includes both the long term concepts and the temporary concepts that can be computed. The term was first discussed in the PhD thesis of Liane Acker (U. Texas, Austin).

window: A rectangular region on the screen that can be opened, resized, iconized etc. A window may be divided into one or more subwindows. In CODE4, major types of windows are the control panel, browsers, feedback panels and mask views and selection criteria views.

Index