Hardy Frequently Asked Questions Version 3.0

Julian Smart
Artificial Intelligence Applications Institute
University of Edinburgh
EH1 1HN

July 1994

Contents

1. About this document and the SDK

This document is a list of frequently asked questions, or questions that might very well be asked, about developing custom Hardy applications, with answers and pointers to further information. It accompanies the Hardy Software Development Kit (SDK), which contains sample CLIPS and C++ files.

The first port of call for the Hardy developer is of course the user manual, which contains the Hardy and GUI CLIPS function reference, and a small introduction to programming in Hardy.

Other documents available are the CLIPS 6.0 manual (written by NASA), and a CLIPS 6.0 Windows Help file containing a terse syntax summary.

Note that some of the examples use function names that don't correspond to the Hardy CLIPS function reference. This is because they use an old naming scheme, which is still valid for backward compatibility. Please refer to the document *old2new.txt* (page 3) which lists old and new function names.

1.1. Using the SDK samples

Under Windows, you can partially automate the installation of CLIPS samples into the Program Manager, by inserting similar lines to the following into the `[Extensions]` section of WIN.INI:

```
clp=c:\hardy\hardy.exe -clips ^.clp
ind=c:\hardy\hardy.exe -f ^.ind
```

Change this to reflect where Hardy is installed on your system. Then, (re)start the File Manager, and drag .clp files (usually with 'load' in the filename) from the File Manager to the Program Manager.

If the sample includes an index file, you can add -f and a filename to the Program Manager item, or load the index manually from Hardy.

If you create new Program Manager items manually, fill out the fields as follows, changing the text to suit your setup:

```
Description:         Drag Test
Command line:        c:\hardy\hardy.exe -clips dragload.clp
Working directory:   c:\hardy\hardysdk\clips\drag
```

Setting the working directory is important.

Under UNIX, change your working directory to the sample directory, and invoke Hardy with appropriate command line arguments (see above).

1.2. Contents of the SDK

```
Welcome to the HARDY Software Development Kit (SDK) version 3.0
===============================================================

Julian Smart, February 8th 1996
===============================
```

This relatively informal gathering-together of examples and Frequently Asked Questions (FAQ) is intended to help HARDY programmers get started, particularly those using CLIPS. Since the C++ interface is a recent addition, there are very few examples available, but hopefully this will be remedied during the next few months.

Note that some of the examples use function names that don't correspond to the HARDY CLIPS function reference. This is because they use an old naming scheme, which is still valid for backward compatibility. Please refer to the document old2new.txt which lists old and new function names.

The contents of this SDK are as follows.

DOCS	
faq.hlp	Windows help version of FAQ
faq.ps	PostScript version of FAQ
faq.rtf	RTF version of FAQ
faq.xlp	wxHelp version of FAQ
clips6.hlp	CLIPS 6.0 help file
readme.txt	This file
CLIPS/CRITPATH	
*	Critical Path analysis demo. See readme.txt.
CLIPS/DDE	
ddetest.clp	Example of using DDE to talk to the Windows Program Manager
CLIPS/DRAG	
drag.clp	Example of intercepting right-drag events to allow dragging from or to 'empty space' on the canvas. Create a card and right-drag.
dragload.clp	Loader for the drag example
CLIPS/GRID	Grid canvas demo.
CLIPS/GUI	
hello.clp	Example GUI demo
guiload.clp	GUI demo loader
CLIPS/HTML	
loader.clp	Hardy to HTML converter (to be 'batched')
convert.clp	Main conversion routine
diagram.clp	Diagram conversion
dialog.clp	Dialogs
forward.clp	Forward declarations
globals.clp	Global variables
hypertxt.clp	Hypertext card conversion
text.clp	Text card conversion
utils.clp	Various utility functions
diagrams.def	Example diagrams.def file with menu for main window
CLIPS/HTML	HTML viewer demo using HTML canvas.

```
CLIPS/ODBC
    odbc.clp        ODBC browser demo.

CLIPS/POPUP         Popup menu on a card demo.

CLIPS/REPORTS
    demoload.clp    Example Windows report-writing demo (to be
'batched')
    ddeword.clp     Code to talk to Word for Windows and Tex2RTF via
DDE
    report.clp      Code to initiate writing a report from HARDY
    *.def           Example definition files
    *.dia           Example diagram file
    *.hyp           Example hypertext card contents
    *.ind           Example HARDY index

CLIPS/RULES
    *.clp           Examples from the CLIPS distribution, slightly
                    modified for HARDY input/output

CLIPS/RESOURCE
    *.clp           Resource-reading examples for wxCLIPS/wxCOOL
    dialog1.wxr     Dialog resource for examples

CLIPS/TOOLBAR
    toolbar.clp     Custom toolbar demo (Windows only).
    *.bmp           Bitmaps for the toolbar demo.

CLIPS/TREE
    tree.clp        Example tree-drawing application. Demonstrates
event
                    handling, creation of diagrams, diagram layout.
    treeload.clp    Loads the application
    tree.def        Tree diagram definition
    diagrams.def    Suitable definition list

CLIPS/UTILS
    utils1.clp      Filename and other utilities, used by report demo.

CLIPS/WXCOOL
    *.clp           wxCOOL, a wrapper around most of the wxCLIPS GUI
functionality.
```

1.3. old2new.txt

This file lists old and new function names.

```
create-card                                 card-create
delete-card                                 card-delete
find-card-by-title                          card-find-by-title
get-first-card                              hardy-get-first-card
get-next-card                               hardy-get-next-card
```

`get-card-string-attribute`	`card-get-string-attribute`
`get-card-x`	`card-get-x`
`get-card-y`	`card-get-y`
`get-card-with`	`card-get-width`
`get-card-height`	`card-get-height`
`get-first-card-item`	`card-get-first-item`
`get-next-card-item`	`card-get-next-item`
`get-first-item-link`	`item-get-first-link`
`get-next-item-link`	`item-get-next-link`
`get-card-special-item`	`card-get-special-item`
`get-link-from`	`link-get-item-from`
`get-link-to`	`link-get-item-to`
`get-link-card-from`	`link-get-card-from`
`get-link-card-to`	`link-get-card-to`
`get-link-type`	`link-get-type`
`get-link-kind`	`link-get-kind`
`get-item-type`	`item-get-type`
`get-item-kind`	`item-get-kind`
`get-top-card`	`hardy-get-top-card`
`goto-item`	`item-goto`
`is-card-shown`	`card-is-shown`
`is-card-modified`	`card-is-modified`
`load-text-file`	`text-card-load-file`
`load-index`	`hardy-load-index`
`save-index`	`hardy-save-index`
`move-card`	`card-move`
`quit-card`	`card-quit`
`set-card-status-text`	`card-set-status-text`
`set-card-string-attribute`	`card-set-string-attribute`
`show-card`	`card-show`
`iconize-card`	`card-iconize`
`add-object-attribute`	`diagram-object-add-attribute`
`delete-object-attribute`	`diagram-object-delete-attribute`
`clear-card-canvas`	`diagram-card-clear-canvas`
`copy-diagram`	`diagram-card-copy`
`create-diagram-card`	`diagram-card-create`
`create-expansion-card`	`diagram-card-create-expansion`
`create-node-image`	`node-image-create`
`create-arc-image`	`arc-image-create`
`delete-all-images`	`diagram-card-delete-all-images`
`delete-image`	`diagram-image-delete`
`deselect-all`	`card-deselect-all`
`draw-image`	`diagram-image-draw`
`draw-image-text`	`diagram-image-draw-text`
`duplicate-node-image`	`node-image-duplicate`
`erase-image`	`diagram-image-erase`
`find-diagram-root`	`diagram-card-find-root`
`format-object-text`	`diagram-object-format-text`
`diagram-layout-tree`	`diagram-card-layout-tree`
`diagram-layout-graph`	`diagram-card-layout-graph`
`diagram-save-bitmap`	`diagram-card-save-bitmap`
`diagram-save-metafile`	`diagram-card-save-metafile`
`diagram-set-layout-parameters`	`diagram-card-set-layout-`

	parameters
set-image-pen-colour	diagram-image-set-pen-colour
set-image-brush-colour	diagram-image-set-brush-colour
set-image-text-colour	diagram-image-set-text-colour
get-arc-image-from	arc-image-get-image-from
get-arc-image-to	arc-image-get-image-to
get-arc-image-from-attachment	arc-image-get-attachment-from
get-arc-image-to-attachment	arc-image-get-attachment-to
get-card-for-image	diagram-image-get-card
get-card-print-width	diagram-card-get-print-width
get-card-print-height	diagram-card-get-print-height
get-first-object-image	diagram-object-get-first-image
get-next-object-image	diagram-object-get-next-image
get-first-card-arc-image	diagram-card-get-first-arc-image
get-next-card-arc-image	diagram-card-get-next-arc-image
get-first-card-node-image	diagram-card-get-first-node-image
get-next-card-node-image	diagram-card-get-next-node-image
get-first-card-arc	diagram-card-get-first-arc-object
get-next-card-arc	diagram-card-get-next-arc-object
get-first-card-node	diagram-card-get-first-node-object
get-next-card-node	diagram-card-get-next-node-object
get-object-from-image	diagram-image-get-object
get-object-string-attribute	diagram-object-get-string-attribute
get-first-object-attribute	diagram-object-get-first-attribute
get-next-object-attribute	diagram-object-get-next-attribute
get-first-expansion-descendant	diagram-card-get-first-descendant
get-next-expansion-descendant	diagram-card-get-next-descendant
get-first-node-image-arc	node-image-get-first-arc-image
get-next-node-image-arc	node-image-get-next-arc-image
get-first-node-object-arc	node-object-get-first-arc-object
get-next-node-object-arc	node-object-get-next-arc-object
get-first-image-expansion	diagram-image-get-first-

	`expansion`
`get-next-image-expansion`	`diagram-image-get-next-expansion`
`get-image-item`	`diagram-image-get-item`
`get-item-image`	`item-get-image`
`get-image-x`	`diagram-image-get-x`
`get-image-y`	`diagram-image-get-y`
`get-image-width`	`diagram-image-get-width`
`get-image-height`	`diagram-image-get-height`
`image-selected`	`diagram-image-selected`
`load-diagram`	`diagram-card-load-file`
`save-diagram`	`diagram-card-save-file`
`move-image`	`diagram-image-move`
`print-hierarchy-to-files`	`diagram-card-print-hierarchy`
`redraw-diagram`	`diagram-card-redraw`
`resize-image`	`diagram-image-resize`
`select-image`	`diagram-image-select`
`set-object-string-attribute`	`diagram-object-set-string-attribute`
`set-object-format-string`	`diagram-object-set-format-string`
`get-item-block`	`item-get-block`
`create-hypertext-card`	`hypertext-card-create`
`get-first-block-selection`	`hypertext-card-get-first-selection`
`get-next-block-selection`	`hypertext-card-get-next-selection`
`hypertext-get-current-char`	`hypertext-card-get-current-char`
`hypertext-get-current-line`	`hypertext-card-get-current-line`
`hypertext-get-line-length`	`hypertext-card-get-line-length`
`hypertext-get-offset-position`	`hypertext-card-get-offset-position`
`hypertext-get-no-lines`	`hypertext-card-get-no-lines`
`hypertext-get-span-text`	`hypertext-card-get-span-text`
`hypertext-insert-text`	`hypertext-card-insert-text`
`hypertext-string-search`	`hypertext-card-string-search`
`load-hypertext`	`hypertext-card-load-file`
`save-hypertext`	`hypertext-card-save-file`
`set-block-type`	`hypertext-block-set-type`
`hypertext-add-block`	`hypertext-block-add`
`hypertext-clear-block`	`hypertext-block-clear`
`get-block-text`	`hypertext-block-get-text`
`get-block-type`	`hypertext-block-get-type`
`get-block-item`	`hypertext-block-get-item`

2. General questions and answers

2.1. What are the hardware and software requirements for running Hardy?

Under X:

- About 10 MB of disk space (more if the static Motif executable is used).
- A Sun workstation with 16MB or more of RAM.
- Solaris 1.x or 2.x.
- If the static version is not being used, then the versions of the Motif, X11 and Xt libraries must agree with Hardy's expectations: can be hard to get right. Hint: symbolic linking from what Hardy expects to what you have installed sometimes work, as does setting LD_LIBRARY_PATH.

Under Windows:

- About 4MB of disk space.
- 486 or better processor.
- 8MB RAM minimum, preferably 16MB.
- Windows 3.x, Windows 95, Windows NT or (at a pinch) OS/2 with Windows emulation. Sun's WABI Windows emulator probably won't do.

Note that under Windows, there is a choice of executables: 16-bit, generic 32-bit (will work under NT, Windows 95, WIN32s) and Windows 95 (only works under Windows 95).

A Mac port is in progress and is expected around July 1996. For UNIX platforms other than Sun, please contact AIAI for a quote for porting Hardy.

2.2. Why is a diagrams.def file needed?

Hardy needs to know what diagram type definitions, hypertext type definitions, and symbol libraries to load before any indexes or diagrams can be loaded or created. This list of definition files enables you to switch between 'applications' by loading different sets of definition files.

2.3. Why is there no Undo facility?

Because it's very difficult! There have been many requests for such a facility, and I am thinking about how I would implement it.

2.4. How can I create new symbols for Hardy?

There are two main methods for create new symbols: use the node symbol editor and combine existing symbols using consraints; and converting Windows metafiles to Hardy symbols.

A convenient (at present, only) way of creating suitable metafiles is to use the shareware package TopDraw, available from our ftp site. This generates suitably simple metafiles for Hardy to recognise. There are heavy restrictions on the complexity of the metafile: no clipping areas are allowed, no text, no bitmaps, no arcs (except for those made up of lines).

If people have success with other metafile-generating packages, I would be interested to know about it.

2.5. What languages can I use to develop Hardy applications?

The main supported language is CLIPS, which is integrated with Hardy and supports an interactive style of development. C++ is supported inside AIAI, using the header file *hapi.h* and a library file. Unfortunately we cannot distribute the library file or source code outside AIAI. See *Using CLIPS* (page 9).

The *DDE* (page 28) facility can be used to call Hardy functions from other programs under Windows.

2.6. What have wxWindows and wxCLIPS got to do with Hardy?

wxWindows is the portable, C++ GUI toolkit used to implement Hardy. Many of the classes have been encapsulated in CLIPS functions (used in an object-oriented manner, with integer identifiers replacing object pointers). This package is known as wxCLIPS, and is both a stand-alone CLIPS development environment, and a library that has been linked with Hardy to provide the GUI functionality.

wxWindows and wxCLIPS were written at AIAI, and are both available free of charge by anonymous ftp from:

`ftp.aiai.ed.ac.uk`

in the directories /pub/packages/wxwin and /pub/packages/wxclips.

2.7. How can I write a customized installation procedure under Windows?

Use the wxWindows installation program, the latest version of which may be found in :

`ftp.aiai.ed.ac.uk:/pub/packages/wxwin/tools/install`

It can also be found in the SDK install directory, but may not be as up to date.

You edit a file called install.inf to specify destination directories, program manager icons, compression method and so on.

2.8. How can I customize the Hardy startup screen under Windows?

Put the files hystart.bmp and hystart.txt in a directory accessible by Hardy. hystart.bmp should be a 16-colour Windows bitmap 255 pixels or less in width and height; hystart.txt should contain a number of lines of text for the dialog text. If these files are not found, the Hardy defaults will be used.

Unfortunately, the Hardy icon cannot be customized at present.

3. Using CLIPS

For the sake of brevity, the following discussion assumes that CLIPS is being used to customize Hardy. If C++ is being used instead, most of this applies, except... when it doesn't (which should be fairly obvious).

3.1. Is CLIPS free?

This is a matter of continuing debate. It is free in the United States, but there are some claims by the official distributor COSMIC that outside the U.S., source code should be purchased for a nominal price. In fact, CLIPS is distributed with many CD-ROMS and at least one book, and is also available by anonymous ftp.

AIAI have written confirmation from COSMIC that there are no copyright problems associated with the Hardy version of CLIPS.

3.2. Is CLIPS any good? What about file-handling, and so on?

Built-in macro languages are sometimes viewed with suspicion, since they may not have all the flexibility of 'doing it in C'. Fortunately, CLIPS is a well-developed language with functions, objects, templates, rules, facts, I/O facilities, and a large set of mathematical functions. There are GUI extensions that allow fairly complex user interfaces and graphics to be constructed, which will need little or no change on other Hardy-supported platforms.

It's unlikely that CLIPS would prove too unwieldy or lack sufficient functionality for your application. If it did, you could fall back on C++, or with a little help from us, implement your own CLIPS functions to do intensive or specialist work.

3.3. But CLIPS is an A.I. language. How does that affect me?

It needn't affect you in the slightest, in that CLIPS was originally integrated with Hardy purely for the functions, as an extension language. The fact that it is also a rule-based shell was deemed irrelevant; after consideration of the alternatives, CLIPS was found to be the most portable, easily-tailorable language available.

The rule-based part of CLIPS is a bonus, and may be used solely or sparingly as the need arises. It's worth considering as a useful pattern matching facility.

Here is a rule-based example (see CLIPS/RULES on the SDK disk).

3.3.1. Animal demo

```
;;;======================================================
;;;   Animal Identification Expert System
;;;
;;;     A simple expert system which attempts to identify
;;;     an animal based on its characteristics.
;;;     The knowledge base in this example is a
;;;     collection of facts which represent backward
;;;     chaining rules. CLIPS forward chaining rules are
;;;     then used to simulate a backward chaining inference
;;;     engine.
```

```
;;;
;;;     CLIPS Version 5.1 Example
;;;
;;;     To execute, merely load, reset, and run.
;;;     Answer questions yes or no.
;;;======================================================

;;;**************************
;;;* INFERENCE ENGINE RULES *
;;;**************************

(defrule propogate-goal ""
   (goal is ?goal)
   (if ?variable $? then ?goal ? ?value)
   =>
   (assert (goal is ?variable)))

(defrule goal-satified ""
   (declare (salience 30))
   ?f <- (goal is ?goal)
   (variable ?goal ?value)
   (answer ? ?text ?goal)
   =>
   (retract ?f)
   (format t "%s%s%n" ?text ?value))

(defrule remove-rule-no-match ""
   (declare (salience 20))
   (variable ?variable ?value)
   ?f <- (if ?variable ? ~?value ? ? ? $?)
   =>
   (retract ?f))

(defrule modify-rule-match ""
   (declare (salience 20))
   (variable ?variable ?value)
   ?f <- (if ?variable ? ?value and $?rest then ?goal ?holder ?goal-
value)
   =>
   (retract ?f)
   (assert (if $?rest then ?goal ?holder ?goal-value)))

(defrule rule-satisfied ""
   (declare (salience 20))
   (variable ?variable ?value)
   ?f <- (if ?variable ? ?value then ?goal ? ?goal-value)
   =>
   (retract ?f)
   (assert (variable ?goal ?goal-value)))

(defrule ask-question-no-legalvalues ""
   (declare (salience 10))
   (not (legalanswers $?))
   ?f1 <- (goal is ?variable)
   ?f2 <- (question ?variable ? ?text)
   =>
   (retract ?f1 ?f2)
   (assert (variable ?variable =(string-to-symbol (get-text-from-user
?text)))))

(defrule ask-question-legalvalues ""
   (declare (salience 10))
   (legalanswers ? $?answers)
   ?f1 <- (goal is ?variable)
   ?f2 <- (question ?variable ? ?text)
   =>
   (retract ?f1)
;   (format t "%s " ?text)
   (printout t ?answers)
   (format t "%n")
   (bind ?reply (string-to-symbol (get-text-from-user ?text)))
   (if (member (lowcase ?reply) ?answers)
     then (assert (variable ?variable ?reply))
          (retract ?f2)
     else (assert (goal is ?variable))))

;;;***************************
;;;* DEFFACTS KNOWLEDGE BASE *
;;;***************************

(deffacts knowledge-base
   (goal is type.animal)
   (legalanswers are yes no)
   (if backbone is yes
    then superphylum is backbone)
   (if backbone is no
    then superphylum is jellyback)
   (question backbone is "Does your animal have a backbone?")
   (if superphylum is backbone and
       warm.blooded is yes
    then phylum is warm)
   (if superphylum is backbone and
       warm.blooded is no
    then phylum is cold)
   (question warm.blooded is "Is the animal warm blooded?")
   (if superphylum is jellyback and
       live.prime.in.soil is yes
    then phylum is soil)
   (if superphylum is jellyback and
       live.prime.in.soil is no
    then phylum is elsewhere)
   (question live.prime.in.soil is "Does your animal live primarily in
soil?")
   (if phylum is warm and
       has.breasts is yes
    then class is breasts)
   (if phylum is warm and
       has.breasts is no
    then type.animal is bird/penguin)
   (question has.breasts is "Normally, does the female of your animal
nurse its young with milk?")
   (if phylum is cold and
       always.in.water is yes
    then class is water)
   (if phylum is cold and
       always.in.water is no
    then class is dry)
   (question always.in.water is "Is your animal always in water?")
   (if phylum is soil and
       flat.bodied is yes
    then type.animal is flatworm)
   (if phylum is soil and
       flat.bodied is no
    then type.animal is worm/leech)
   (question flat.bodied is "Does your animal have a flat body?")
   (if phylum is elsewhere and
       body.in.segments is yes
    then class is segments)
   (if phylum is elsewhere and
       body.in.segments is no
    then class is unified)
   (question body.in.segments is "Is the animals body in segments?")
   (if class is breasts and
       can.eat.meat is yes
    then order is meat)
   (if class is breasts and
       can.eat.meat is no
    then order is vegy)
   (question can.eat.meat is "Does your animal eat red meat?")
   (if class is water and
       boney is yes
    then type.animal is fish)
   (if class is water and
       boney is no
    then type.animal is shark/ray)
   (question boney is "Does your animal have a boney skeleton?")
   (if class is dry and
       scally is yes
    then order is scales)
   (if class is dry and
       scally is no
    then order is soft)
   (question scally is "Is your animal covered with scaled skin?")
   (if class is segments and
       shell is yes
    then order is shell)
   (if class is segments and
       shell is no
    then type.animal is centipede/millipede/insect)
   (question shell is "Does your animal have a shell?")
   (if class is unified and
       digest.cells is yes
    then order is cells)
   (if class is unified and
       digest.cells is no
    then order is stomach)
   (question digest.cells is "Does your animal use many cells to digest
it's food instead of a stomach?")
   (if order is meat and
       fly is yes
    then type.animal is bat)
   (if order is meat and
       fly is no
    then family is nowings)
   (question fly is "Can your animal fly?")
   (if order is vegy and
       hooves is yes
    then family is hooves)
   (if order is vegy and
       hooves is no
    then family is feet)
   (question hooves is "Does your animal have hooves?")
   (if order is scales and
       rounded.shell is yes
    then type.animal is turtle)
   (if order is scales and
       rounded.shell is no
    then family is noshell)
   (question rounded.shell is "Does the animal have a rounded shell?")
   (if order is soft and
       jump is yes
    then type.animal is frog)
   (if order is soft and
       jump is no
    then type.animal is salamander)
   (question jump is "Does your animal jump?")
   (if order is shell and
       tail is yes
    then type.animal is lobster)
   (if order is shell and
       tail is no
    then type.animal is crab)
   (question tail is "Does your animal have a tail?")
   (if order is cells and
       stationary is yes
    then family is stationary)
   (if order is cells and
       stationary is no
    then type.animal is jellyfish)
   (question stationary is "Is your animal attached permanently to an
object?")
   (if order is stomach and
       multicelled is yes
    then family is multicelled)
   (if order is stomach and
       multicelled is no
    then type.animal is protozoa)
   (question multicelled is "Is your animal made up of more than one
cell?")
   (if family is nowings and
       opposing.thumb is yes
    then genus is thumb)
   (if family is nowings and
       opposing.thumb is no
    then genus is nothumb)
   (question opposing.thumb is "Does your animal have an opposing
thumb?")
   (if family is hooves and
       two.toes is yes
    then genus is twotoes)
   (if family is hooves and
       two.toes is no
    then genus is onetoe)
   (question two.toes is "Does your animal stand on two toes/hooves per
foot?")
   (if family is feet and
       live.in.water is yes
    then genus is water)
   (if family is feet and
       live.in.water is no
    then genus is dry)
   (question live.in.water is "Does your animal live in water?")
   (if family is noshell and
       limbs is yes
    then type.animal is crocodile/alligator)
   (if family is noshell and
       limbs is no
    then type.animal is snake)
   (question limbs is "Does your animal have limbs?")
   (if family is stationary and
       spikes is yes
    then type.animal is sea.anemone)
   (if family is stationary and
       spikes is no
    then type.animal is coral/sponge)
   (question spikes is "Does your animal normally have spikes radiating
from it's body?")
   (if family is multicelled and
       spiral.shell is yes
    then type.animal is snail)
   (if family is multicelled and
       spiral.shell is no
    then genus is noshell)
   (question spiral.shell is "Does your animal have a spiral-shaped
shell?")
   (if genus is thumb and
       prehensile.tail is yes
    then type.animal is monkey)
   (if genus is thumb and
       prehensile.tail is no
    then species is notail)
   (question prehensile.tail is "Does your animal have a prehensile
tail?")
   (if genus is nothumb and
       over.400 is yes
    then species is 400)
   (if genus is nothumb and
       over.400 is no
    then species is under400)
   (question over.400 is "Does an adult normally weigh over 400
pounds?")
   (if genus is twotoes and
       horns is yes
    then species is horns)
   (if genus is twotoes and
       horns is no
    then species is nohorns)
   (question horns is "Does your animal have horns?")
   (if genus is onetoe and
       plating is yes
    then type.animal is rhinoceros)
   (if genus is onetoe and
       plating is no
    then type.animal is horse/zebra)
   (question plating is "Is your animal covered with a protective
plating?")
   (if genus is water and
       hunted is yes
    then type.animal is whale)
   (if genus is water and
       hunted is no
    then type.animal is dolphin/porpoise)
   (question hunted is "Is your animal, unfortunately, commercially
hunted?")
   (if genus is dry and
       front.teeth is yes
    then species is teeth)
   (if genus is dry and
       front.teeth is no
    then species is noteeth)
   (question front.teeth is "Does your animal have large front teeth?")
   (if genus is noshell and
       bivalve is yes
    then type.animal is clam/oyster)
   (if genus is noshell and
       bivalve is no
    then type.animal is squid/octopus)
   (question bivalve is "Is your animal protected by two half-shells?")
   (if species is notail and
       nearly.hairless is yes
    then type.animal is man)
   (if species is notail and
       nearly.hairless is no
    then subspecies is hair)
   (question nearly.hairless is "Is your animal nearly hairless?")
   (if species is 400 and
       land.based is yes
    then type.animal is bear/tiger/lion)
   (if species is 400 and
       land.based is no
    then type.animal is walrus)
   (question land.based is "Is your animal land based?")
   (if species is under400 and
       thintail is yes
    then type.animal is cat)
   (if species is under400 and
       thintail is no
    then type.animal is coyote/wolf/fox/dog)
   (question thintail is "Does your animal have a thin tail?")
   (if species is horns and
       one.horn is yes
    then type.animal is hippopotamus)
   (if species is horns and
       one.horn is no
    then subspecies is nohorn)
   (question one.horn is "Does your animal have one horn?")
   (if species is nohorns and
       lives.in.desert is yes
    then type.animal is camel)
   (if species is nohorns and
       lives.in.desert is no
    then type.animal is giraffe)
   (question lives.in.desert is "Does your animal normally live in the
desert?")
   (if species is teeth and
       large.ears is yes
    then type.animal is rabbit)
   (if species is teeth and
       large.ears is no the type.animal is
rat/mouse/squirrel/beaver/porcupine)
   (question large.ears is "Does your animal have large ears?")
   (if species is noteeth and
       pouch is yes
    then type.animal is "kangaroo/koala bear")
   (if species is noteeth and
       pouch is no
    then type.animal is mole/shrew/elephant)
   (question pouch is "Does your animal have a pouch?")
   (if subspecies is hair and
       long.powerful.arms is yes
    then type.animal is orangutan/gorilla/chimpanzie)
   (if subspecies is hair and
       long.powerful.arms is no
    then type.animal is baboon)
   (question long.powerful.arms is "Does your animal have long,
powerful arms?")
   (if subspecies is nohorn and
       fleece is yes
    then type.animal is sheep/goat)
   (if subspecies is nohorn and
       fleece is no
    then subsubspecies is nofleece)
   (question fleece is "Does your animal have fleece?")
   (if subsubspecies is nofleece and
       domesticated is yes
    then type.animal is cow)
   (if subsubspecies is nofleece and
       domesticated is no
    then type.animal is deer/moose/antelope)
   (question domesticated is "Is your animal domesticated?")
   (answer is "I think your animal is a " type.animal)
)

;;; Auto-boot function for wxCLIPS (Windows only)
(deffunction app-on-init ()
 (unwatch all)
 (show-ide-window)
 (reset)
 (run)
 (return 0)
)
```

3.4. How do Hardy and my CLIPS code communicate?

Hardy and CLIPS are essentially two big chunks of code linked together, in the same process space. In normal use, CLIPS code only executes in two situations: when Hardy is started up, using the -clips command, and when certain *events* call your code.

At startup, the CLIPS custom code is given a chance to load files, and initialize before returning control back to Hardy. The C++ equivalent of this is defining the function *gyInitializeCustomCode* for Hardy to call at startup.

The user will also want to register *event handlers* (either CLIPS or C++ functions) which will be called by Hardy when the specified events happen. The CLIPS function *register-event-handler* is used to specify a context (for example "Toplevel" or a user-defined card type), an event name, and the function name (CLIPS) or address (C++).

For example, the following registers interest in the CustomMenu event for diagrams of type "bsdm", so Hardy will call the function *test-event-handler* when the user selects an item on the custom menu.

```
; We are interested in CustomMenu events for BSDM demo diagram type
(register-event-handler CustomMenu "bsdm" test-event-handler)
```

During a callback, custom code will probably call back into Hardy by means of the Hardy CLIPS functions.

Apart from the simple developer's window, there is no other way in which Hardy and CLIPS are related; no object-oriented association between diagrams and chunks of code. Everything is essentially global, and code must explicitly register interest in particular events in particular contexts.

Here's an example of an application that intercepts menu and left-click events to allow very quick construction of tree diagrams.

3.4.1. Tree demo code

```
;;; Tree-drawer demo
;;; Use -clips treeload.clp on HARDY command line.
;;; Create a tree card, create a root with the custom menu,
;;; then keep clicking on nodes to create children.

(defglobal ?*tree-root* = 0)

;;; Asks for a name and creates the tree root
(deffunction tree-add-root-image (?card-id)
  (bind ?msg (get-text-from-user "Enter name for new root node"))
  (if (eq ?msg "") then 0 else
   ; Create a new image
   (bind ?image1 (create-node-image ?card-id "Node"))

   ; Find it's underlying node object
   (bind ?object1 (get-object-from-image ?card-id ?image1))
   ; Set the name attribute
   (set-object-string-attribute ?card-id ?object1 "name" ?msg)

   ; Format the text on the image
   (format-object-text ?card-id ?object1)

   (bind ?*tree-root* ?image1)
   ; Layout the tree
   (diagram-layout-tree ?card-id ?*tree-root*))
 )

;;; Callback for custom menu
(deffunction tree-menu-handler (?card-id ?option)
 (if (eq ?option "Add root") then
     (tree-add-root-image ?card-id) else
  (if (eq ?option "Layout tree") then
     (diagram-layout-tree ?card-id ?*tree-root*)))
)

;;; Left-click handler to create children
(deffunction tree-node-handler (?card-id ?image-id ?x ?y ?shift
?control)
  (declare ?x float)
  (declare ?y float)
  (if (and (neq ?shift 1) (neq ?control 1)) then
    (bind ?msg (get-text-from-user "Enter name for new node"))
    (if (eq ?msg "") then 0 else

     ; Create a node image
     (bind ?image1 (create-node-image ?card-id "Node"))

     ; Get the underlying node object
     (bind ?object1 (get-object-from-image ?card-id ?image1))

     ; Set the name attribute
     (set-object-string-attribute ?card-id ?object1 "name" ?msg)

     ; Format the text on the image
     (format-object-text ?card-id ?object1)

     ; Add an arc
     (bind ?image3 (create-arc-image ?card-id "Arc" ?image-id ?image1 1
3))

     ; Layout the tree
     (diagram-layout-tree ?card-id ?*tree-root*)
     0)
  else 1)
)

;;; Handler for the user deleting images
(deffunction tree-delete-image-handler (?card-id ?image-id ?type)
 (declare ?type string)
 (if (eq ?image-id ?*tree-root*) then
  (bind ?*tree-root* 0) else
  (diagram-layout-tree ?card-id ?*tree-root*)
  )
)

;;; Handler for initialisation
(deffunction tree-init ()
  (message-box "Welcome to the tree demo: create a new Tree card and
see Options menu");
  (return 1)
)
```

3.4.2. Tree demo loader

```
;;; Tree-drawing demo
;;; Use -clips treeload.clp on HARDY command line.
;;; Create a tree card, create a root with the custom menu,
;;; then keep clicking on nodes to create children.

(load "tree.clp")

; Event handlers for Tree Demo
(register-event-handler CustomMenu "Tree" tree-menu-handler)
(register-event-handler NodeLeftClick "Tree" tree-node-handler)
(register-event-handler DeleteNodeImage "Tree" tree-delete-image-
handler)
(register-event-handler DeleteArcImage "Tree" tree-delete-image-
handler)
(register-event-handler OnHardyInit "Toplevel" tree-init)
```

3.5. Why does it take so long to load CLIPS code into Hardy?

There are two ways of loading CLIPS code: *batch* and *load*, each with a entry on the developer window File menu, and a function equivalent.

batch executes *all* possible CLIPS commands and construct definitions, but doesn't do much checking of constructs. It is slow.

load only recognises CLIPS constructs such as *deffunction*, as opposed to function calls. It is relatively fast.

Therefore, the recommended practise is *not* to batch all CLIPS code: instead, have a small file contains several further *load* commands, and a few function calls (such as *register-event-handler*). This 'loader' file can be batched or used as the command line with `-clips`, so the minimum possible amount is subject to the slow batch reading.

Another reason for a slow-down is because tracing is on. See *Slow execution* (page 20).

Here's a demo loader file that should be *batched* but calls *load* to load most of the program.

3.5.1. Loader example

```
;;; File:     demoload.clp
;;; Purpose:  Report-writing demo loader (Windows only).
;;;           For use with the demo BSDM diagram type
;;; Author:   Julian Smart
;;; Created:  16/7/94

(load "utils1.clp")
(load "globals.clp")
(load "dialog.clp")
(load "latex.clp")
(load "ddeword.clp")
(load "report.clp")

; We are interested in CustomMenu events for the Hardy window
(register-event-handler CustomMenu Toplevel report-event-handler)

; Turn off excessive output to the developer window
(unwatch all)
```

3.6. Is there an Integrated Development Environment for Hardy?

No, but under Windows there's the next best thing: the ability to execute some of the Developer Window functionality from a DDE program. The program editor/ddesend.exe allows the user to load and batch the current buffer, by defining a macro which calls ddesend.exe with an appropriate command line.

The DDE client (of which ddesend.exe is an example) must connect to the server using the service name HARDY, and establish a conversation on the HARDY topic.

The file editor/macros.rc is a sample MicroEMACS for Windows macro file, that implements commnds for loading files into Hardy, and executing CLIPS commands, from within MicroEMACS, using ddesend.exe.

Here are some examples of Hardy DDE commands. They consist of a letter, a space, and then some data.

```
L c:\example.clp
B c:\example.clp
E (app-on-init)
Q
C
```

Here is a list of commands that Hardy recognizes:

- L: load the given file of constructs into CLIPS
- B: batch the given file of constructs into CLIPS
- E: execute the given command
- C: clear the Hardy development window
- Q: quit Hardy

3.7. Why is execution sometimes slow?

It could be because tracing is enabled, and lots of text is being output on the development window. Try calling

```
(unwatch all)
```

3.8. Sometimes CLIPS callbacks fail to execute. Why?

There is a condition that can occur when a CLIPS command has failed; subsequent attempts to run the code cause FALSE to be displayed, then nothing further happens. You may have quit Hardy and rerun occasionally during your development. *Note:* this has now been cured (as of late 1995).

3.9. How can I have a custom menu on the Hardy control window?

There isn't a convenient user interface for doing it, but you can create such a menu by inserting lines like these in the `diagrams.def` definition list:

```
custom(custom_menu_name = "&Custom options",
  custom_menu_strings = ["&First item", "&Second item"]).
```

Then use the "CustomMenu" event for the "Toplevel" context. See the manual entry for *register-event-handler* for further details on events.

3.10. What facilities are available for generating reports?

The one-word answer to this is, 'none'. A better answer is that the Hardy allows total report-generating flexibility via CLIPS. There are no *built-in* facilities that allow instant document generation from a hypertext index or diagram hierarchy.

The reason for this is that it is unlikely that a generic facility would be sufficient for all possible uses of Hardy. Instead, the Hardy application programmer can write report generators using CLIPS or C++ code. There are several Hardy functions that help you in this task, such as *diagram-card-save-metafile*.

The easiest approach is probably to generate a LaTeX file. Why LaTeX? Because it is a relatively simple format to generate, and can be converted into various other formats using *Tex2RTF* (page 22) or similar utility. Sample code is available in the Hardy SDK that generates a report with mixed text and diagrams.

On the UNIX platform, the diagrams can be saved in PostScript, LaTeX used to convert to a .dvi file, and then `dvips` can be used to produce a PostScript file. The LaTeX macro package **PSBOX** may be used for incorporating the PostScript files.

Under Windows, Tex2RTF can be used to generate Rich Text Format (RTF) files, which can be read by most popular word processors. Diagrams are saved as metafiles by the report writer, as they can be scaled in the word processor with no adverse effect.

The example CLIPS code demonstrates using DDE to automate the conversion to RTF and loading into MS Word.

Under Windows there is the additional possibility of generating a Windows Help file, to produce a read-only, browseable hypertext version of the Hardy document. This requires the Windows Help compiler (hc.exe or hc505.exe), available free from various anonymous ftp sites.

Under both Windows and UNIX, Tex2RTF can be used to produce HTML files for viewing on the World Wide Web. However, at present the conversion of graphics files to GIF must be done manually, although it could be automated. Use *diagram-card-save-bitmap* to save a bitmap, and try to find a command-line utility to convert from bitmap to GIF format (but see *HTML generation* (page 22)).

3.10.1. HTML generation

The HTML directory contains a Hardy application to generate HTML files for a whole Hardy index (unlike the report generator mentioned above which only works for diagram hierarchies).

Currently it works under Windows only, because only the Windows version can save suitable colour bitmaps. To convert the saved BMP files to GIF, the utility calls a shareware DOS program called **ColorView** which must be in the PATH. Any other suitable BMP to GIF conversion program could also be used with suitable changes to the code.

3.10.2. If you don't want to generate LaTeX

If you really want to generate a native format such as PostScript and RTF, then that's fine, but be prepared to work long hours! There are a couple of Hardy functions to help RTF writers: *convert-bitmap-to-rtf*, and *convert-metafile-to-rtf*. These functions convert image files into the hexadecimal RTF data required by some RTF readers. Note that different RTF readers have different requirements (Word for Windows can cope with an *INCLUDEPICTURE field* that simply references a metafile on disk, whereas others may require the hexadecimal data to be embedded in the file).

An RTF specification is available from the Microsoft ftp site, or from the Microsoft Developer's Network CD-ROM. You can also look at the source code of *Tex2RTF* (page 22) for some hints.

3.10.3. Where can I get Tex2RTF?

The latest version of Tex2RTF can be accessed by anonymous ftp from:

`ftp.aiai.ed.ac.uk/pub/packages/tex2rtf`

It is available in SPARC Open Look and Windows 3.1 versions.

Tex2RTF was developed using the free Open Look, Motif and Windows 3.1 C++ class library *wxWindows*, also available from the above FTP site in the /pub/wxwin/beta directory.

3.11. How can I build customized user interfaces in Hardy?

For many applications, using simple functions such as *message-box*, *get-choice*, *file-selector* and *get-text-from-user* may be adequate, combined with intercepting custom menu and mouse click events. But you may find after a while that you need something a little more complex.

This is where wxCLIPS comes in (see also *wxWindows and wxCLIPS* (page 8)). It provides a range of building blocks for building your own interfaces, including frames, panels, buttons, list boxes, text windows, canvases, pens, brushes, and fonts. The demo supplied with the SDK in `CLIPS/GUI` shows off some of these features.

Here's the simple GUI example.

3.11.1. GUI example

```
;;; hello.clp
;;; Shows how a frame may be created, with a menu bar and
;;; panel, using low-level windows functions.
;;; Load using -clips <file> on the command line or using the Batch
;;; or Load commands from the CLIPS development window; type
;;; (app-on-init) to start.

(defglobal ?*main-frame* = 0)
(defglobal ?*subframe* = 0)
(defglobal ?*panel* = 0)
(defglobal ?*canvas* = 0)
(defglobal ?*text-win* = 0)
(defglobal ?*hand-cursor* = 0)

(defglobal ?*small_font* = 0)
(defglobal ?*green_pen* = 0)
(defglobal ?*black_pen* = 0)
(defglobal ?*red_pen* = 0)
(defglobal ?*cyan_brush* = 0)

(defglobal ?*xpos* = -1.0)
(defglobal ?*ypos* = -1.0)

(defglobal ?*bitmap* = 0)
(defglobal ?*button-bitmap* = 0)
(defglobal ?*icon* = 0)

;;; Sizing callback
(deffunction on-size (?id ?w ?h)
 (if (and (> ?id 0) (> ?*panel* 0) (> ?*text-win* 0)) then
  (bind ?client-width (window-get-client-width ?id))
  (bind ?client-height (window-get-client-height ?id))
  (window-set-size ?*panel* 0 0 ?client-width (* ?client-height 0.666))
  (window-set-size ?*text-win* 0 (* ?client-height 0.666) ?client-width
(/ ?client-height 3))
 )
)

;;; Utility function for drawing a bitmap
(deffunction draw-bitmap (?dc ?bitmap ?x ?y)
 (bind ?mem-dc (memory-dc-create))
 (memory-dc-select-object ?mem-dc ?bitmap)
 ; Blit the memory device context onto the destination device context
 (dc-blit ?dc ?x ?y (bitmap-get-width ?bitmap) (bitmap-get-height
?bitmap)
   ?mem-dc 0.0 0.0)
 (dc-delete ?mem-dc)
)

(deffunction draw-graphics (?dc)
  (if (> ?*bitmap* 0) then
   (draw-bitmap ?dc ?*bitmap* 0.0 250.0))

  (dc-set-font ?dc ?*small_font*)
  (dc-set-pen ?dc ?*green_pen*)
  (dc-draw-line ?dc 0.0 0.0 200.0 200.0)
  (dc-draw-line ?dc 200.0 0.0 0.0 200.0)

  (dc-set-pen ?dc ?*red_pen*)
  (dc-set-brush ?dc ?*cyan_brush*)
  (dc-draw-rectangle ?dc 100.0 100.0 100.0 50.0)
  (dc-draw-rounded-rectangle ?dc 150.0 150.0 100.0 50.0)

  (dc-set-clipping-region ?dc 150.0 150.0 100.0 50.0)
  (dc-draw-text ?dc "This text should be clipped within the rectangle"
150.0 170.0)
  (dc-destroy-clipping-region ?dc)

  (dc-draw-ellipse ?dc 250.0 250.0 100.0 50.0)
  (dc-draw-spline ?dc (mv-append 50.0 200.0 50.0 100.0 200.0 10.0))
  (dc-draw-line ?dc 50.0 230.0 200.0 230.0)
  (dc-draw-text ?dc "This is a test string" 50.0 230.0)
 )

;;; Painting callback
(deffunction on-paint (?id)
 (if (> ?id 0) then
  (bind ?dc (canvas-get-dc ?id))
  (draw-graphics ?dc)
 )
)

(deffunction on-event (?canvas ?event)
  (bind ?dc (canvas-get-dc ?canvas))
  (dc-set-pen ?dc ?*black_pen*)
  (bind ?x (mouse-event-position-x ?event))
  (bind ?y (mouse-event-position-y ?event))
  (bind ?dragging (mouse-event-dragging ?event))
  (if (and (> ?*xpos* -1) (> ?*ypos* -1) (> ?dragging 0)) then
   (dc-draw-line ?dc ?*xpos* ?*ypos* ?x ?y)
  )
  (bind ?*xpos* ?x)
  (bind ?*ypos* ?y)
)

(deffunction on-close (?frame)
 (format t "Closing frame.%n")
 (window-delete ?*subframe*)
 (bind ?*panel* 0)
 (bind ?*text-win* 0)
 1)

(deffunction on-menu-command (?frame ?id)
 (switch ?id
  (case 200 then (message-box "CLIPS for wxWindows Demo
by Julian Smart (c) 1993" wxOK 1 0 "About wxWindows CLIPS Demo"))
  (case 3 then (if (on-close ?frame) then (window-delete ?frame)))
  (case 1 then
    (bind ?file (file-selector "Choose a text file to load"))
    (if (neq ?file "") then
     (text-window-load-file ?*text-win* ?file)))
  (case 4 then
    (bind ?dc (postscript-dc-create "" 1))
    (if (and (> ?dc 0) (= (dc-ok ?dc) 1)) then
     (if (= (dc-start-doc ?dc "Printing") 1) then
       (dc-start-page ?dc)
       (draw-graphics ?dc)
       (dc-end-page ?dc)
       (dc-end-doc ?dc)
     )
    )
    (if (> ?dc 0) then (dc-delete ?dc))
   )
  )
)

;;; Button callback
(deffunction frame-button-proc (?id)
 (bind ?parent (window-get-parent ?id))
 (bind ?grandparent (window-get-parent ?parent))
 (format t "Pressed button %d%n" ?id)
 (message-box "Hello")
)

;;; Text callback
(deffunction text-callback (?id)
 (bind ?event-id (panel-item-get-command-event))
 (if (eq "wxEVENT_TYPE_TEXT_ENTER_COMMAND" (event-get-event-type
?event-id)) then
  (format t "The text was %s%n" (text-get-value ?id))
 )
)

;;; Test program to create a frame
(deffunction app-on-init ()
  (unwatch all)
  (if (= ?*small_font* 0) then
    (bind ?*small_font* (font-create 10 wxSWISS wxNORMAL wxNORMAL 0))
    (bind ?*green_pen* (pen-create GREEN 1 wxSOLID))
    (bind ?*black_pen* (pen-create BLACK 1 wxSOLID))
    (bind ?*red_pen* (pen-create RED 3 wxSOLID))
    (bind ?*cyan_brush* (brush-create CYAN wxSOLID))
    (bind ?*hand-cursor* (cursor-create "wxCURSOR_HAND"))
    (if (eq "Windows 3.1" (get-platform)) then
     (bind ?*bitmap* (bitmap-load-from-file "wxwin.bmp"))
     (bind ?*button-bitmap* (bitmap-load-from-file "aiai.bmp"))
     (bind ?*icon* (icon-load-from-file "aiai.ico"))
    )
  )

  (bind ?*main-frame* (frame-create 0 "Hello wxCLIPS!" -1 -1 500 460))
  (frame-create-status-line ?*main-frame*)
  (frame-set-status-text ?*main-frame* "Welcome to wxCLIPS")
  (if (> ?*icon* 0) then
   (frame-set-icon ?*main-frame* ?*icon*)
  )

  (window-add-callback ?*main-frame* OnSize on-size)
  (window-add-callback ?*main-frame* OnClose on-close)
  (window-add-callback ?*main-frame* OnMenuCommand on-menu-command)
  ;;; Make a menu bar
  (bind ?file-menu (menu-create))
  (menu-append ?file-menu 1 "&Load file")
  (menu-append ?file-menu 4 "&Print to PostScript")

  (bind ?pull-right (menu-create))
  (menu-append ?pull-right 100 "&Twips")
  (menu-append ?pull-right 101 "&10th mm")

  (menu-append ?file-menu 2 "&Scale picture" ?pull-right)
  (menu-append-separator ?file-menu)
  (menu-append ?file-menu 3 "&Quit")

  (bind ?help-menu (menu-create))
  (menu-append ?help-menu 200 "&About")

  (bind ?menu-bar (menu-bar-create))
  (menu-bar-append ?menu-bar ?file-menu "&File")
  (menu-bar-append ?menu-bar ?help-menu "&Help")

  (frame-set-menu-bar ?*main-frame* ?menu-bar)

  ;;; Make a panel and panel items
  (bind ?*panel* (panel-create ?*main-frame* 0 0 500 250))
  (panel-set-label-position ?*panel* wxVERTICAL)

  (bind ?*text-win* (text-window-create ?*main-frame* 0 250 500 250))

  (bind ?button (button-create ?*panel* frame-button-proc "A button"))
  (if (> ?*button-bitmap* 0) then
   (bind ?bitmap-button (button-create-from-bitmap ?*panel* frame-
button-proc ?*button-bitmap*))
  )
  (bind ?text (text-create ?*panel* "text-callback" "A text item"
"Initial value" -1 -1 200 -1 "wxPROCESS_ENTER"))
  (bind ?check (check-box-create ?*panel* "" "A check box"))

  (panel-new-line ?*panel*)

  (bind ?choice (choice-create ?*panel* "" "A choice item" -1 -1 -1 -1
(mv-append
   "One" "Two" "Three" "Four")))
  (choice-set-selection ?choice 0)

  (message-create ?*panel* "Hello! A simple message")

  (bind ?list (list-box-create ?*panel* "" "A list" 0 -1 -1 100 80))
  (list-box-append ?list "Apple")
  (list-box-append ?list "Pear")
  (list-box-append ?list "Orange")
  (list-box-append ?list "Banana")
  (list-box-append ?list "Fruit")

  (panel-new-line ?*panel*)

  (bind ?slider (slider-create ?*panel* "" "A slider" 40 22 101 200))
  (bind ?multi (multi-text-create ?*panel* "" "Multiline text" "Some
text" -1 -1 200 100))

;  (window-fit ?*panel*)
;  (window-fit ?*main-frame*)

  (text-window-load-file ?*text-win* "hello.clp")
  (bind ?*subframe* (frame-create 0 "Canvas Frame" 300 300 400 400))
  (bind ?*canvas* (canvas-create ?*subframe* 0 0 400 400))
  (window-set-cursor ?*canvas* ?*hand-cursor*)
  (window-add-callback ?*canvas* OnPaint on-paint)
  (window-add-callback ?*canvas* OnEvent on-event)
  (canvas-set-scrollbars ?*canvas* 20 20 50 50 4 4)

  (window-centre ?*main-frame* wxBOTH)

  (window-show ?*main-frame* 1)
  (window-show ?*subframe* 1)

  ?*main-frame*)
```

3.12. My popup windows make the main window pop up.

If you use frames, dialog boxes convenience dialog functions with no parent window, Hardy will use the main window. This may then pop up (on some platforms) and obscure other windows.

To avoid this, pass a valid frame or dialog identifier to the window you're creating. If the window you want to be the parent is a card, use **card-get-frame** to retrieve a valid frame identifier. *Do not* use the card identifier itself, this will most likely cause a crash.

3.13. How can I keep my code portable between platforms?

- When writing GUI code, use relative positioning where possible:
    - don't use absolute values for positions, use -1 for x and y values
    - use **panel-new-line** to space items.
    - use **window-fit** on a panel/dialog box, then on the frame if there is one, to shrink a window around its contents
- Use **get-platform** to test what platform Hardy is running on, and execute slightly different code where necessary.
- When copying diagrams.def and index files between platforms, you may need to edit the files to eliminate absolute pathnames. Alternatively, use the **-path** command line option to specify where files will be found; Hardy will search such paths.
- Pass valid parent windows to all frames and dialogs (see *Window problems* (page 27).
- Use short, DOS-compatible filenames (8+3 characters).
- Use end of line converters where necessary when copying between UNIX and DOS platforms.

3.14. What interprocess-communication facilities can I use?

Hardy supports synchronous Dynamic Data Exchange (DDE) under Windows, and a simulation of DDE using sockets under UNIX.

An external client program can use Hardy as a server, making a connection and using the *execute* DDE command to call CLIPS code, followed by a *request* DDE command to get a return value. Note that this kind of connection is only available on the CLIPS-enabled version of Hardy, since an interactive language interpreter is required.

In addition, CLIPS DDE functions are available within Hardy, so a CLIPS program can make Hardy the client, requesting information from or sending information to an external application, such as a spreadsheet or word processor. The report-generating example in the Hardy SDK shows DDE being used to invoke Tex2RTF and Word for Windows.

Under UNIX, there is an undocumented program called *DDEPIPE* that sites between a normal pipe-aware UNIX application and Hardy. This offers a simple command interface that translates messages sent to the pipe into a subset of the simulated DDE protocol required by Hardy.

The DDE facilities available in CLIPS are documented in the Hardy User Guide. They are a *functional* encapsulation of the *object-oriented* equivalent in *wxWindows* (page 8), and so further information can be obtained from the wxWindows manual. Another place to look is Microsoft documentation and technical notes, plus Charles Petzold's *Programming Windows 3.1*.

Here's an example of talking to the Windows Program Manager using DDE. As you can see, DDE needn't be at all frightening: the example is only a page or so long.

3.14.1. ProgMan example

```
;;; Demo of DDE functions: chatting to PROGMAN
;;;

(defglobal ?*progman-server* = 0)
(defglobal ?*progman-server-name* = "PROGMAN")
(defglobal ?*progman-host-name* = "none")
(defglobal ?*progman-topic-name* = "PROGMAN")
(defglobal ?*progman-client* = 0)
(defglobal ?*progman-connection* = 0)

;;; Convert a multi-value list of strings to one string
(deffunction many-strings-to-one ($?strings)
  (bind ?counter 1)
  (bind ?string "")
  (while (<= ?counter (length $?strings)) do
    (bind ?string (str-cat ?string (nth ?counter $?strings)))
    (bind ?counter (+ ?counter 1))
  )
  (return ?string)
)

(deffunction progman-demo ()
 ;; Get a group name from the user
 (bind ?new-group-name (get-text-from-user "New PROGMAN group name"))
 (if (neq ?new-group-name "") then
  ;; Form create group command
  (bind ?command (many-strings-to-one (mv-append "[CreateGroup(" ?new-
group-name ")]")))

  ;; Construct a client object
  (bind ?*progman-client* (client-create))

  ;; Construct a connection object
  (bind ?*progman-connection* (client-make-connection
              ?*progman-client* ?*progman-host-name*
              ?*progman-server-name* ?*progman-topic-name*))

  ;; Execute a command to create a group
  (bind ?exe (connection-execute ?*progman-connection* ?command))

  ;; Request a list of groups
  (bind ?req (connection-request ?*progman-connection* "PROGMAN"))
  (format t "%nProgram Manager Groups:%n")
  (format t "%s%n%n" ?req)

  ;; Disconnect
  (connection-disconnect ?*progman-connection*)
 )
)

;;; Automatically called when running application from command line
;;; e.g. wxclips -start -clips ddetest.clp
;;; Also runnable from the Application: Run application.
(deffunction app-on-init ()
  (progman-demo)
)
```

4. Digging into Hardy diagrams

4.1. Why the difference between node and arc *objects*, and *images*? I'm confused.

Yes, it can seem bewildering. But there is a reason for this difference. Node and arc *objects* are intended as an internal, not displayed, representation of a diagram: they contain *attributes*, and are typed. Node and arc *images*, on the other hand, merely reflect these objects visually: there may even be two or more images for the same object, perhaps at different levels of a hierarchical diagram. Images can be hyperlinked to other cards, objects can't be.

4.2. What is the relationship between item and image?

An *item* in Hardy-land is a generic concept for 'something on a card'. Therefore an image on a diagram card is an item, and so is a block of text on a hypertext card. You'd expect to be able to do similar things with items of different kinds, and so you can -- any item can form a hyperlink with any other item. In addition, there are non-visual items: the *special item* that exists on each card, to form hyperlink relationships in the absence of a visual item such as a text block or diagram image.

Fine, as far as the user is concerned. An item is an item is a block, or an image. *But* for the programmer, things are a little different: items are separate entities, with different identifiers. For any image or block, there is one item. It is *this* identifier you must use to form or follow hyperlinks. There's a good reason for this separation, to do with being able to examine items and links when card contents are not loaded. The distinction means explicitly getting an image's item id, or an item's image id, using the appropriate Hardy functions.

4.3. Can I access the diagram type definition from CLIPS?

No, at present there aren't any functions for accessing or changing the type definitions. The only way to create new ones programmatically would be to write them to file and have the user load them in (or write to the diagrams.def list of definitions).

However, when examining diagram structures, you can query the type of node and arc objects (using *diagram-object-get-string-attribute* with a "type" as the attribute name) and the diagram type (using *card-get-string-attribute*, again with "type" as the attribute name).

4.4. Can I have attributes of different types?

No, only string attributes are allowed, but you can convert different CLIPS types to and from strings.

5. Hardy bug list

The following is a (probably incomplete) list of the known Hardy bugs and problems. Please contact us if you find any more.

5.1. All platforms

Duplicate images Duplicate images can occasionally cause problems with crashes, and when deleted can cause removal of arcs connected to images for which this is duplicate. Please use this facility with caution.

Crash when deleting cards If you delete a card hierarchy, and have event handlers that might (implicitly) expose a card whilst it was being deleted, a crash can result. Try to disable as many event handlers as you can whilst a card is being deleted (e.g. by setting a global variable from within a DeleteCard event handler). A card can get exposed (a physical window created for it) by calling a function such as diagram-card-clear-canvas when the card is not shown.

5.2. X only

Asynchonicity problems Due probably to the asynchonous nature of X, it is dangerous to do a lot of programmatic creation and deletion of windows in sequence. For example, highlighting a window to load a diagram from disk, then immediately deleting the window, seems to cause a crash (the canvas refresh occurs even though there's no canvas any more). Try to split these actions up if you are getting a crash, such that the user must initiate more actions manually (using a timer might help, also).

5.3. Windows only

Modal lockout It has been reported that it is possible to push a dialog box behind the Hardy window and not get it back again, so Hardy is stuck in a modal loop until Windows is terminated. Using parent windows for all dialog boxes should alleviate this problem.

Glossary

GUI

Graphical User Interface, such as Windows 3 or X.

HTML

Hypertext Markup Language; an SGML document type, used for providing hypertext information on the World Wide Web, a distributed hypertext system on the Internet.

LaTeX

A typesetting language implemented as a set of TeX macros. It is distinguished for allowing specification of the document structure, whilst taking care of most layout concerns. It represents the opposite end of the spectrum from WYSIWYG word processors.

Metafile

Microsoft Windows-specific object which may contain a restricted set of GDI primitives. It is device independent, since it may be scaled without losing precision, unlike a bitmap. A metafile may exist in a file or in memory. wxWindows implements enough metafile functionality to use it to pass graphics to other applications via the clipboard. A placeable metafile is a metafile with a 22-byte header which can be imported into several Windows applications, and is a format used by the Microsoft help compiler.

Open Look

A specification for a GUI 'look and feel', initiated by Sun Microsystems. XView is one toolkit for writing Open Look applications under X, and wxWindows sits on top of XView.

RTF

Rich Text Format: an interchange format for word processor files, used for importing and exporting formatted documents, and as the input to the Windows Help compiler.

wxHelp

wxHelp is the hypertext help facility used to provide on-line documentation for UNIX-based wxWindows applications. Under Windows 3.1, Windows Help is used instead.

wxWindows

wxWindows is a free C++ toolkit for writing applications that are portable across several platforms. Currently these are Motif, Open Look, Windows 3.1 and Windows NT.

XView

An X toolkit supplied by Sun Microsystems, initially just for porting SunView applications to X, but which has become a popular toolkit in its own right due to its simplicity of use. XView implements Sun's Open Look 'look and feel' for X, but is not the only toolkit to do so.

Index