aiai/hardy/um/1.3

Artificial Intelligence Applications Institute

Hardy

User Guide

Prepared by
Julian Smart and Robert Rae

Artificial Intelligence Applications Institute
The University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, UK

21st February 1996
Version 1.3

Contents

1 Introduction

Hardy is a tool that has been designed and developed by the Artificial Intelligence Applications Institute at The University of Edinburgh, primarily for diagramming applications. It runs on Unix workstations under Motif or Open Look, and on PCs under Windows 3.1.

In this manual, the Windows version of Hardy is referred to as *Hardy for Windows*.

1.1 Diagramming

The idea behind Hardy is very simple. The *diagram* is a fundamental tool that is essential to many analysis and design activities. Diagrams provide an intuitive way of expressing relationships between concepts which most people can relate to, unlike most formal languages and notations. However, though diagrams are often easily drawn with pencil and paper, any subsequent modification normally means that the whole diagram has to be re-drawn, with all the usual problems of consistency checking, etc. Some support for diagramming can be provided by conventional computer-based drawing tools, but these suffer from two main draw-backs.

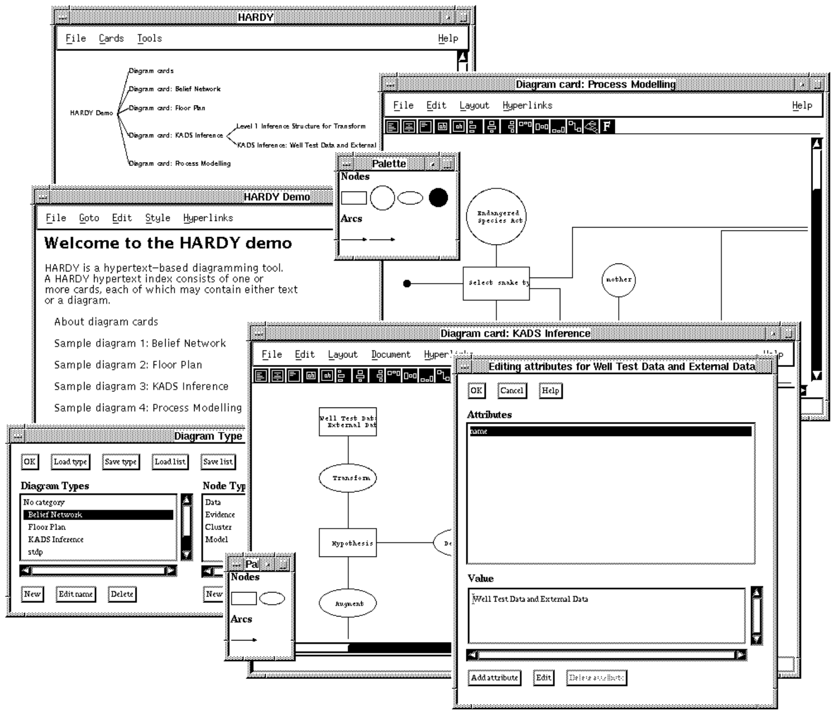
Firstly, tools are not normally specific to the type of diagram required; for example, when an image is erased or moved, there is no knowledge of what other images are related to it, so any links to and from the image remain where they were. Secondly, in most cases the diagram cannot be directly processed—the diagram must first be translated by hand to a different representation. In those tools that overcome these shortcomings, the types of diagram and means of customisation provided are limited; in addition, high costs are normally associated with them.

Hardy allows the user to build a *diagram type* (such as a dataflow diagram type) or to use a type already provided by someone else. The user may then select a type and rapidly produce diagrams, which consist essentially of a number of *nodes* linked by *arcs*. When constructing a diagram, arcs will follow nodes when they are moved, and values may be entered for node and arc *attributes* that are specified in the diagram type.

Once created, diagrams can be output in a variety of formats which allow the underlying system model to be processed by another program, so that Hardy can be used, for instance, as a knowledge capture tool feeding directly into a Knowledge Based System, or transformed into a document with mixed text and graphics, so that (for example) the documentation of organisational procedures becomes much less arduous. Hardy can also be used to display diagrams generated by other programs.

To achieve a greater degree of customisation, Hardy supports an *Application Programmer's Interface*. This allows the card type designer to intercept events such as selecting a menu item or clicking on a diagram node and implement specialised layout algorithms or animate particular scenarios.

1.2 Hardy and hypertext


HARDY
File Cards Tools Help
HARDY Demo
Diagram cards
Diagram card: Belief Network
Diagram card: Floor Plan
Diagram card: KADS Inference
Level 1 Inference Structure for Transform
KADS Inference: Well Test Data and External
Diagram card: Process Modelling
Diagram card: Process Modelling
File Edit Layout Hyperlinks Help
Endangered Species Act
Select snake ty
mother
Palette
Nodes
Arcs
HARDY Demo
File Goto Edit Style Hyperlinks
Diagram card: KADS Inference
File Edit Layout Document Hyperlinks
Well Test Data External Data
Transform
Hypothesis
Augment
Editing attributes for Well Test Data and External Data
OK Cancel Help
Attributes
name
Value
Well Test Data and External Data
Add attribute Edit Delete attribute
Diagram Type
OK Load type Save type Load list Save list
Diagram Types
No category
Belief Network
Floor Plan
KADS Inference
stdp
Node Types
Data
Evidence
Cluster
Model
New Edit name Delete New

Welcome to the HARDY demo

HARDY is a hypertext–based diagramming tool.
A HARDY hypertext index consists of one or
more cards, each of which may contain either text
or a diagram.

About diagram cards

Sample diagram 1: Belief Network

Sample diagram 2: Floor Plan

Sample diagram 3: KADS Inference

Sample diagram 4: Process Modelling

Figure 1.1: Example Hardy session under X

The diagramming capabilities of Hardy are built on top of a *hypertext* framework in which each diagram has its own *card* (window), and cards may be linked together to form a tree or network. The user may browse through this network, either by following hypertext links or by viewing the *index tree* and clicking on a card title.

Diagrams have a habit of being hierarchical, with a node on a high-level diagram representing an entire diagram at a lower level. Hardy supports this type of organisation though *expansion cards*. A diagram card with its expansion cards will all be treated as a single unit, being held in one file, for instance.

Figure 1.1 shows several elements of a Hardy session. Going clockwise, the top left window is the *control window*, the main window of the application. The control window can display a map of the cards, as shown here, and is also used for various administrative tasks, such as loading and saving index files. In front, to the right, is a *diagram card* with its symbol palette. On top of this is another diagram card and symbol palette of a different type, for which the user is editing the attributes of one of its nodes. To the left and at the bottom, is the *Diagram Type Manager*, a tool for defining and changing diagram types.

In addition to diagram cards, the user may create text cards, for example to document a diagram. Above and behind the Diagram Type Manager is a *hypertext card* using several different fonts, to which other cards have been linked.

1.3 Manual conventions

In this document, the following conventions will be used.

A term will be italicised when first used, as in *hypertext card*, above.

The names of particular files will be shown using a "teletype" face, as in `diagrams.def`. The same face will also be used for particular function names, as in `diagram-card-find-root`.

Button labels will be shown in bold face, for instance **OK**, and the notation **Menu: Entry** will refer to the entry **Entry** on the menu **Menu**.

2 Running Hardy

To run Hardy, you require a serial number, which may be obtained from AIAI if you are an academic user, or have otherwise arranged with AIAI to use Hardy.

This serial number should be entered into Hardy using the **Tools: Preference** menu. Click on the **Serial number** button, enter the number, and restart Hardy.

2.1 Starting a session

It is normally useful to tell Hardy the type of cards and symbols you expect to use during a session so that they are immediately available. This is done through a file which contains information on all the definitions needed. Its default name is `diagrams.def` so, before running Hardy, this file should be in your working directory or you should tell the system which alternative file you require.

This can be done either by specifying it on the command line through the **-def**flag followed by the full name of the file required (see Section 2.3), or by specifying it as a *resource* in `WIN.INI` under Windows or in `.hardyrc` under X (see Section 2.4).

To run Hardy under UNIX, just type `hardy`. Under Windows, install the program in the Program Manager by dragging `hardy.exe` from the File Manager.

If Hardy cannot find the diagram definition file, however it is specified, it will give you a warning. Under Open Look, other messages may also appear warning that it cannot load certain fonts. This is normal: Hardy is just searching for a font which is not available.

2.2 Ending a session

To exit from Hardy, use the **File: Exit Hardy** menu from the control window.

2.3 Command line options

Many system defaults can be over-ridden by giving options on the command line. For instance, to specify an initial hypertext index file, use the **-f** option followed by the file name, or to specify the directories in which to search for files, use the **-path** option followed by the full directory name.

These are Hardy's command line options for both UNIX and Windows, unless stated otherwise:

-block *filename*
Specify a block type definition file(see Section 6).

-clips *filename*
Specify a CLIPS filename to batch.

-def *filename*
Use the specified global diagram definition file, instead of the default `diagrams.def` in

the current directory. This file mentions all the diagram definition files which should be loaded on running.

-dir *directory*
Change to the given directory before loading CLIPS and other files. Useful to avoid specifying directories in CLIPS application code.

-h Print a screen summarising the command line options (*UNIX only*).

-help Print the list of recognised command line options (*UNIX only*).

-load *filename*
Specify a CLIPS filename to load (file must contain constructs only).

-mdi Run in MDI (Multiple Document Interface) mode (*Windows only*).
In this mode, child windows are constrained by the main window. This is the default under Windows. *NOTE:* this option has been withdrawn from version 1.76.

-nobanner Suppress the opening screen, regardless of initialisation file setting.

-path *path*
Add the given path to Hardy's path search list. This enables Hardy to find files which are not in the current directory or whose absolute path name is incorrect, perhaps due to transfer of files between UNIX and PC. This switch may be used repeatedly to add more than one path.

-P*printer*
Substitute *printer* with a printer name to use as the default printer.

-port *integer*
Specify a port number if Hardy is used as a DDE server.

-sdi Run in SDI (Single Document Interface) mode (*Windows only*).
In this mode, windows are not constrained by the main window. *NOTE:* this option has been withdrawn from version 1.76.

-server Use as a DDE server.

-version Display the current version number.

2.4 Hardy resources

Under Windows, Hardy resources should be held in the Hardy section of the file `WIN.INI`.in the form *name* = *value*.

Under X, they should be in the file `.hardyrc` in your home directory in the form hardy.*name* = *value*

An example of a `WIN.INI` entry is as follows:

```
[hardy]
definitionList=c:\diagrams\diagrams.def
```

For `.hardyrc`, the equivalent is the line:

```
hardy.definitionList=/user/11/jacs/diagrams/diagrams.def
```

Below are some of Hardy's resource names for use in both UNIX and Windows versions, unless otherwise stated.

definitionList = *filename* (`diagrams.def`)
: List of diagram definition files.

objectBitmapSize = *integer* (32)
: Size in pixels of node/arc images on diagram symbol palette.

annotationBitmapSize = *integer* (32)
: Size in pixels of annotation images on diagram symbol palette.

libraryBitmapSize = *integer* (32)
: Size in pixels of images on symbol library palette.

showLinkPanelOnCreate = *boolean* (0)
: If *1*, show the hyperlink panel when a diagram card is created.

showToolBarOnCreate = *boolean* (1)
: If *1*, show the toolbar when a diagram card is created.

showPaletteOnCreate = *boolean* (1)
: If *1*, show the symbol palette when a diagram card is created.

showErrors = *boolean* (1)
: If *1*, route error messages to the Development Window.

displayCategories = *boolean* (0)
: If *1*, card types will be requested by the system as category then type.

clickToSelect = *boolean* (0)
: If *0*, clicking on an object will follow any hyperlink present, shift-clicking will select it. If *1*, clicking will select, shift-clicking will follow the hyperlink.

mdi = *boolean* (1) *Windows only*
: If *1*, run Hardy in MDI (Multiple Document Interface) mode. In this mode, child windows are constrained by the main window. This is the default under Windows.

HARDYStart = *boolean* (1) *Windows only*
: If *1*, the audio file `hystart.wav` will be played when the system is started.

HARDYExit = *boolean* (1) *Windows only*
: If *1*, the audio file `hyexit.wav` will be played when the system is exited.

Standard defaults are shown in brackets.

2.5 Files used by Hardy

Apart from the executable file, `hardy`, the system will look for certain files in your working directory when it is started. The most important of these are:

1. `diagrams.def`– a list of diagram type files,
2. resources file – contains your preferred settings for various system values (`.hardyrc` under X or `WIN.INI` under Windows). See Section 2.4 .

Other information must also be available between Hardy sessions. Hardy uses files for this purpose, distinguishing several different types of file, each holding different but related types of information. Standard filename extensions are used to identify these files.

1. Definition list file (`diagrams.def`),
2. Symbol library files (`.slb`),
3. Card collection index files (`.ind`),
4. Diagram and hypertext definition files (`.def`),
5. Diagram card files (`.dia`),
6. Hypertext card files (`.hyp`),
7. Text card files (`.txt`).
8. Hardy package files (`.hpk`). These are explained in Section 2.7 .

2.6 Hardy application associations

Just as you can associate file extensions with programs in MS Windows, you can associate particular application-defined file extensions with Hardy command lines. The appropriate command line will be invoked when Hardy encounters a file which isn't an index file, and for which there is an entry in Hardy's association list.

This means that instead of invoking a specific Hardy application or document with a command line like this:

```
hardy -dir c:\hardy\tree -clips treeload.clp -f demo.tre
```

you could instead use:

```
hardy demo.tre
```

or, if no application main file needs to be specified,

```
hardy tre
```

which will put Hardy into the required application state.

What is the nature of demo.tre? Well, it could be a normal index file, or it could be an application-specific file, unrecognised by Hardy. In the latter case, the application code should register an OnLoadFile event handler which will be called when Hardy finds the application association and after it has executed the associated command line. If there is no OnLoadFile event handler, Hardy will assume the file is a normal index file.

You can edit the associations by selecting the Preferences dialog and clicking on the *Associations* button. The Hardy Application associations dialog will appear, with a list of applications (initially empty). To add an application association, click on **Add**, and fill in the **Extension**, **Name** and **Command** fields. Click on a listbox item or Ok to register the extension.

The values of the dialog fields should be filled in as follows:

- Extension: this is a short file extension unique to the application, such as 'tre' or 'btk'.
- Name: the name of the application, e.g. Tree Drawing Demo.
- Command: the Hardy command line that will be executed by Hardy to activate the application. It should not include an index-loading command (-f) since this index loading will be done automatically by Hardy if necessary. An example:

  ```
  -dir {HARDYDIR}\tree -clips treeload.clp
  ```

  Note the `{HARDYDIR}` keyword which will be substituted by the Hardy directory as determined by the HARDY environment variable, or hardy installation directory, or `hardy` directory under the user's UNIX home directory.

To delete an assocation, press the **Delete** button. Unfortunately the entry will not be deleted in win.ini (or other) initialisation file unless further items are added: edit the initialisation file by hand if necessary.

Under Windows, it's a good idea to use the File Manager to associate the .hpk extension (see next section) with the runhardy.exe program. You can also edit win.ini to do this. This will allow double clicking on a Hardy package file to run or reset Hardy and load the appropriate files. It will also allow World Wide Web browsers to do the right thing when you click on a .hpk file.

If when trying to use the associations, Hardy does not seem to be loading the application properly, check that the index filename has the correct extension. Hardy needs the extension to be correct for it to load the required definition files properly.

2.7 Packaging Hardy files

Because a single application or document may use several files, maintaining and distributing such files can become inconvenient. Hardy provides a 'composite' file type with extension `.hpk` which packages several Hardy or user files into one file. Hardy recognises the extension and unpacks the files into the standard Hardy area before executing the associated command line contained in the package file (if any). The standard Hardy area is determined by the HARDY environment variable, or if this is undefined, the Hardy installation directory under Windows or, under UNIX, the directory `hardy` under the user's home directory.

As mentioned above, under MS Windows you can associate the .hpk extension with the program runhardy.exe to allow invocation of Hardy when double-clicking on a .hpk file from the File Manager or Web browser. The reason why you need to associate the extension with runhardy.exe instead of hardy.exe is that only one copy of Hardy can run at a time under Windows, and runhardy.exe will communicate with Hardy by DDE if it is already running.

If the .hpk file is identified as residing in a temporary directory (such as TEMP or /tmp), it will be deleted after unpacking.

To create your own .hpk files, invoke the Package tool from the Hardy Tools menu. In the **HPK Filename** text box, enter the full pathname of the package file to be created, with .hpk extension.

In the **Current root directory** text box, enter the path to be subtracted from the real file path when storing in the package file. So if your application is stored in the directory `c:\hardy\apps\test`, you might enter the directory `c:\hardy\apps`. In this case, the package file will contain files such as `test\load.clp`. This allows unpacking into a directory relative to the user's standard Hardy data directory instead of replicating your original directory structure.

Use the **Add** button to add files to the list, and **Delete** to remove them. Check the **Load** checkbox for a file which is to be designated the application file to load immediately (if any).

Enter an optional comment into the **Comment** text box, and in **Association**, enter an association string of the same syntax used in the Association list as invoked from the Preferences dialog. For example:

```
btk,BITKit,-dir {HARDYDIR}\thing -clips load.clp
```

This consists of an extension, an application name, and a Hardy command line. You should not put a -f switch on this command line since an index or application main file will be invoked automatically if appropriate. You can use the keyword `{HARDYDIR}` to stand in for the user's current Hardy directory, where files are unpacked to.

When you have entered the details of the package file, you can save these details as a package file list (.pfl extension) for later loading. Press the **Generate** button to generate the Hardy package file.

3 Using Hardy

3.1 Hardy conventions

Hardy uses the keyboard, mouse and cursor in a regular way, so that you get similar effects from doing similar things anywhere in the system.

3.1.1 Mouse conventions

Generally, Hardy uses the left mouse button only, with the right button being used to provide short-cuts for common operations. We never use other buttons even if they exist.
With each mouse button, we can do three basic things:

1. Click on – move the cursor to where you want it, then depress and immediately release the button.
2. Double-click on – move the cursor to where you want it, then depress and immediately release the button twice in rapid succession.
3. Click-and-drag – move the cursor to where you want it, then depress the button and move the mouse, dragging the cursor to the new screen position, then release the button.

These operations may be modified by the control and the shift keys, so that control-click means: move the cursor to where you want it, then depress and immediately release the mouse button *while holding down the control key.*
We'll talk about "pressing" a button when we mean:

> *move the cursor over the button and click on it*

and "choosing" a menu entry will mean:

> *move the cursor over the menu, depress the button to open the menu and keep it depressed, then move the cursor down to the menu entry required, and then release the button.*

3.1.2 Cursor patterns

Six different cursors are used by Hardy, as follows:

pointer normal default pattern,

text pointer used when entering text,

hand used when you can move items around in a window,

cross-hairs used in a window when something has been selected and can be "dropped" onto the window,

bulls-eye when you move an arc from one point to another on a node,

hourglass/stopwatch used whenever a noticeable delay is expected, such as when loading a file.

3.2 Creating cards

Start Hardy without specifying any command line options, etc. The main control window will appear with the menus **File, Cards, Tools** and **Help**. Initially, there will be nothing on the canvas.

To create your first card, select the **Cards: Create top card** entry. A choice of card types is presented; try the **Text card** option since this is the simplest. Select it and press the **OK** button. A new window appears with a blank text subwindow.

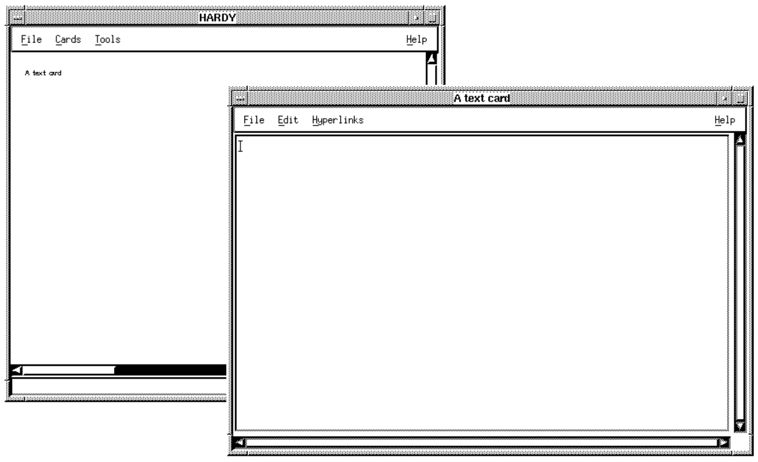


Figure 3.1: Hardy control window and text card

Now you can try making use of hypertext. There are four menus, **File**, **Edit**, **Hyperlinks** and **Help** on this new card. Goto the **Hyperlinks** menu of the new cardand select the **Link new card** option. Again, choose a text card. Another new card appears, linked to the original one. Link another card to the new one. Link a further card to the original card. The set of cards you've built up is shown as a treein the control window. The first card is called the Top Card since it's at the top or root of the hypertext 'tree'.

You can use the **File: Open file** menu entry to associate a text file with the last card you created. Any text file will do.

3.3 Browsing

There are two ways in which the term *Browsing* is used in Hardy.

- *Card browsing* gives the user an overview of cards in the current index.

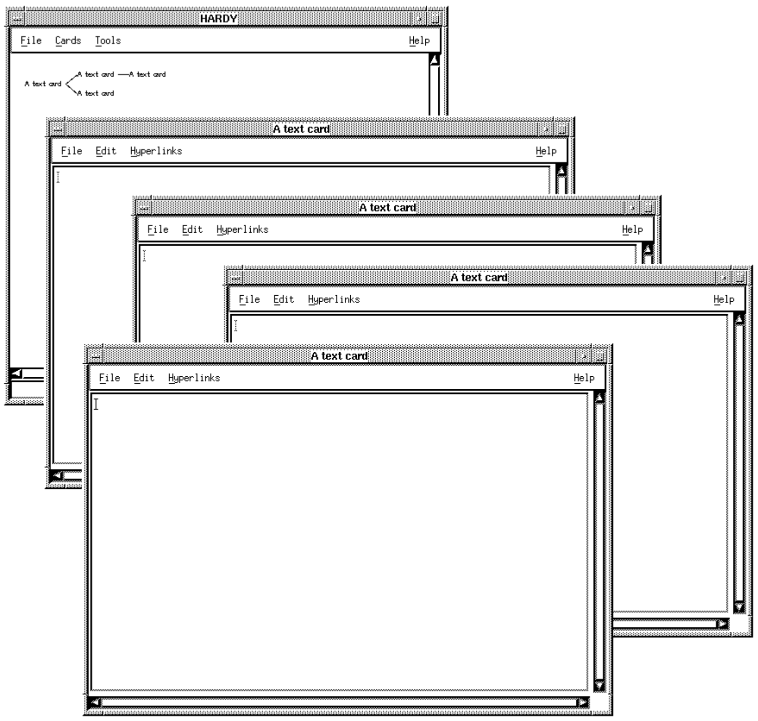


Figure 3.2: Linked cards

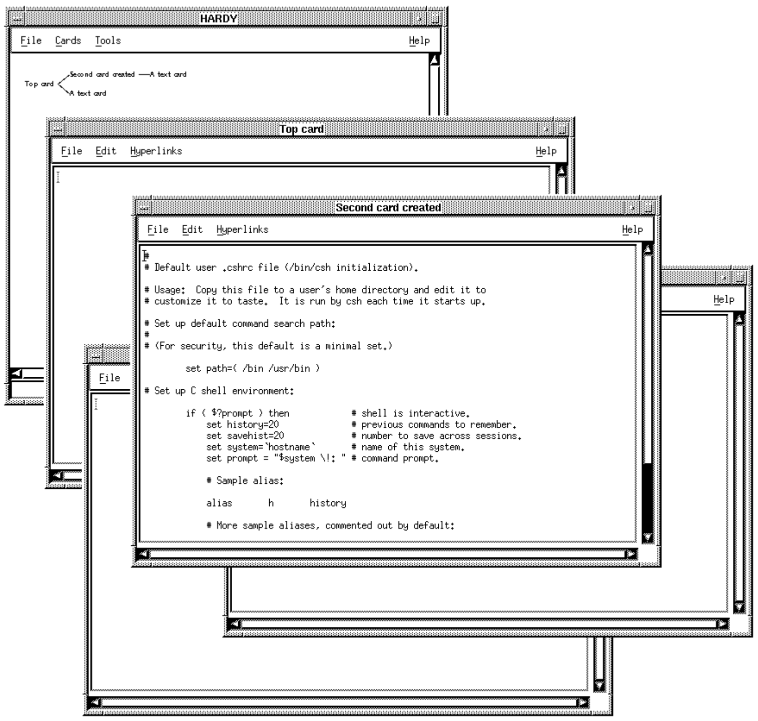
- *File browsing* gives detail on various kinds of Hardy file on disk, and allows loading these at will without having to know which tool to invoke first.

3.3.1 Card browsing

Although the tree of cards is shown in the control window, you may not know which card is which by now, because the cards all have the same title. The title of a card can be changed with the **File: Card title** menu entry. Change the titles of the Top Card and the second card you created, say. The changed titles are not shown immediately in the control window. You can ask for the tree to be redrawn by going back to the control window and selecting the **Cards: Draw tree** entry. Clicking on a title in this tree index gets you directly to the relevant card.

You can also search on card titles using the **Cards: Search** item. The Card Search dialog allows entry of a search string (or the "*" wildcard to find all cards), and pressing **Search** causes all matching titles to be displayed. Clicking on a title in the index tree brings the corresponding card to the front of the stack.

Press **OK** to quit from Card Search.



```
#
# Default user .cshrc file (/bin/csh initialization).

# Usage:  Copy this file to a user's home directory and edit it to
# customize it to taste.  It is run by csh each time it starts up.

# Set up default command search path:
#
# (For security, this default is a minimal set.)

        set path=( /bin /usr/bin )

# Set up C shell environment:

        if ( $?prompt ) then            # shell is interactive.
            set history=20              # previous commands to remember.
            set savehist=20             # number to save across sessions.
            set system=`hostname`       # name of this system.
            set prompt = "$system \!: " # command prompt.

            # Sample alias:

            alias       h       history

            # More sample aliases, commented out by default:
```

Figure 3.3: Renamed cards

3.3.2 File browsing

Instead of loading files individually from different tools and menus, it is possible to use the *File browser*. This is accessible from the *File: Browse files* menu on the control window. The file browser dialog shows a list of files, information about each file, and a list of directories so the user can navigate around the disk.

Single clicking on a file in the **Files** displays information about it in the **Description** area, and double clicking loads this file. Note that almost all Hardy file types are supported in the file browser, although it is not always possible to load displayed files since a specific diagram file, for example, might rely on a diagram type being already loaded. The user will be warned if an attempt is made to load a file whose type is not present.

There are checkboxes to allow selective browsing, and a *Show detail* checkbox to toggle between high detail mode (which can be slow) or lower detail mode (faster).

A restricted version of the file browser is available from other tools, such as the diagram card. The diagram card file browser allows browsing of files whose types match the type of that card.

3.4 Ordering your screen

The screen may be getting a bit cluttered by now. To hide cards, select the **File: Quit card** option on a card's **File** menu. This gets rid of the physical window, while keeping a record of the card in the hypertext index. (This is very different from the **File: Delete** option, which erases the card completely from Hardy's memory.) When you refer to a hidden card from another card, or from the control window, the card will spring into life again. This means that a hypertext index can consist of hundreds of cards without becoming totally unworkable on the screen. You need to keep visible only the cards you are working with at any given time.

If you delete a card, its links with other cards will disappear. Sometimes this means that cards are 'orphaned': they have no parents from which the user can get to the child. These can be linked up again by using the **Cards: Find orphans** option, choosing one of them, and selecting that card by using the **Hyperlinks: Select card** option. You can then go to another card and choose the **Hyperlinks: Link card to selection** option.

3.5 Hypertext links, cards and items

So far, links between information have always involved cards. However, linking a card with another card is only a special case of linking an *item* with an *item*. Hardy considers a card to contain a number of items, and these items may be linked with other items in the same or a different card.

The items in a diagram card are nodes and arcs. In a hypertext card, items are blocks of text.

3.6 Cards and files

Hardy uses files to save information from one session to another. Several different types of file are involved (see Section 2.5). For instance, every card has a file associated with it which will have to be saved individually if the card has been changed. (The exception is any diagram expansion card which is always a 'descendant' of a diagram card. Expansion card contents are saved with the ancestor diagram card, so that an entire hierarchy is stored in one file.) In addition, the main hypertext index which contains pointers to the card files and the links between cards will need to be updated. These files may be saved through the **File: Save** option on each card *and* on the main Hardy window.

Before saving a diagram to a file, Hardy will make a backup of any file with the same name by copying it to a file with a `.bak` extension. If for any reason Hardy crashes, leaving your diagram file in an unloadable state, or you need to get back to the previous version for some other reason, this backup file is available for editing, etc.

Index files can also be recovered if you forget to save the them. Each card is designed to be able to load a card file independently of the main index (assuming it's of an appropriate type). Therefore you can reconstruct a hypertext index manually, creating new cards and loading them with the appropriate files. This also means that you can import card contents from other sessions.

If you try to exit Hardy without saving the index file after changes have been made to it, or without saving changed cards, you will usually be asked if you want to save the changes. The same goes for individual cards which are being deleted.

3.7 Preferences

You can set some system values to suit yourself and save them between sessions to give the default behaviour that you want. You can, for instance, say whether or not you want to show the hyperlinks panel on every card, or what file you want to use to hold your list of diagram files.

These "preferences" are set through the Preferences dialog box which is opened from the "Preferences" entry on the "Tools" menu of the main control window.

To alter the default diagram definition list, select the text entry area labelled "Default definition list" by clicking on it, then type in the name of the file you want to use.

You can also set the sizes of the images used in library and diagram card palettes, again by selecting the text entry area associated with the bitmap in question and changing the contents to the value (in pixels) that you want.

The other preferences have boolean values which correspond to the state of the buttons associated with them: if the button is depressed the value is true, if the button is not depressed the value is false.

Once you have set all the values to be what you want, you press the **OK** button, whereupon the system will accept the values and dismiss the dialog box. The dialog box can be dismissed without changing any values by pressing **Cancel**.

These values are held between sessions in the files `.hardyrc` under Unix,or `WIN.INI` under Windows (see Section 2.4).

4 Diagram cards

4.1 Creating new diagrams

New diagrams are created through the **Cards: Create top card** menu item of the control window, or the **Hyperlinks: Link new card** menu option of an existing card. If you then select the particular diagram type you wish to use (see Section 9.1.1),a new diagram card will appear, together with a floating *symbol palette*.

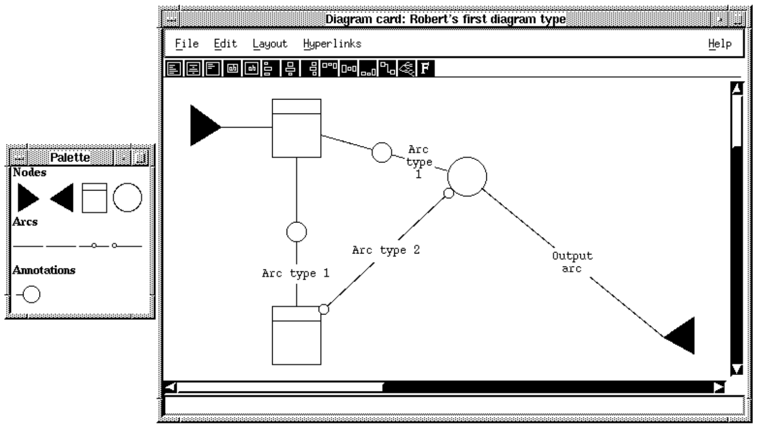


Figure 4.1: Example Hardy diagram card

The symbol palette is used for selecting the node and arc types that are available to you for this type of diagram card.

A panel showing all the hyperlinkages to and from the card may be displayed on the right side of the card. You can toggle this to be shown or hidden by using the **Hyperlinks: Toggle link panel display**menu option. Similarly, you can save space on the card by hiding the toolbar that appears at the top of the card below the menu bar through the **Hyperlinks: Toggle toolbar** option on the same menu. This toolbar gives easy access to some of the most used text formatting and diagram layout facilities that are on the **Layout** menu(see Section 4.9).

4.2 Creating nodes

You add a node to the diagram by choosing the one you want from the selection in the symbol palette. To help you tell which is which, the name of the symbol below the cursor is shown in the status line at the bottom of the diagram card as you move over the symbol palette. When you've found the one you're after, click on the image in the palette, then move your cursor back into the diagram card where it will change its shape to cross-hairs (a large addition sign. This means that you can now place a node on the diagram by clicking where you want it.

Move the node to a different place on the card by holding down the left button over the shape, dragging the mouse to the new position, and releasing the button.

You can add more of these nodes by continuing to click where you want them. You can always tell that clicking will drop something on to the window from the cross-hairs cursor pattern.

You can label nodes so as to tell them apart. See Section 4.6 .

4.3 Creating arcs

An arc can be drawn between one node and one or more other nodes. Basically, this is done using the right mouse button by clicking-and-dragging from the source node to the destination node where the button is released. (You can draw an arc from a node back to itself: in this case, you'll be asked to confirm that that was what you really meant.) Most of the time this is all that is necessary, however there are cases in which there is more than one type of arc that can join the nodes. There are two ways of telling Hardy which arc you want: either you can select the correct arc symbol from the diagram symbol palette before you join the nodes, or you can wait until after you've asked the system to join the nodes when it will pop-up a dialog box showing you which arcs might be suitable. You then select the node you want and press the **OK** button to dismiss the dialog box. The arc will follow the nodes correctly if you now move one of them.

4.4 Selecting nodes and arcs

Nodes and arcs may be *selected* by holding down the shift key and left-clicking over them. Selection handles are shown around the shape to indicate that it is selected. The shape may be deselected with the same operation. You can have more than one object selected at a time.

A selected node may be resized by clicking-and-dragging on one of its selection handles; if shift is also held down while a corner handle is moved, the shape will go back to its original proportions.

Various other operations may be done on selected objects using the card's **Edit** menu. Arcs may be divided into segments by selecting the arc (shift left click) and choosing the **Edit: Add control point** menu option. A line or spline arc's control points, except for the start and end points, may be dragged with the mouse to make the arc bend. Lines and splines always start off with no intermediate control points (giving a straight line).

4.5 Labelling nodes and arcs

Nodes and arcs can have more than one label if they have more than one *text region.* Nodes only have more than one text regionif they are *composite symbols* (made up from more than one basic shape(see Section 8.1.4). All arcs have three text regions: one at its start, one at its middle, and one at its end. There will be one label for each text region. Each label has its own specified appearance (its colour, font family and size, etc) and its own text string which is held in one of the node's attributes. To change the label, you need to change the value of the attribute holding its text (see Section 4.6).

4.6 Object attributes

An *attribute* is a defined component of the node which can be given a text string as its value by the user through the Attribute Editor.

You open the the Attribute Editor by control-left-clicking on the node so that a window pops-up with a list of all the node's attributes. The node label is usually called 'label' or 'name'. The value of the selected attribute name is shown in the text entry area below the list of attributes.

Under Unix only, you can now position the cursor in the text entry area by clicking, then type in characters directly from the keyboard. To delete the character before the cursor use the Back Space key; to delete the one after it use the Delete key.

Otherwise, you must press the **Edit** button below the text area whereupon a editor window will appear. Which editor is used depends on your value for the EDITOR environment variable. When you have made your changes and dismissed your editor, you must press the **OK** button on the dialog box that also popped up so as to tell Hardy that you have finished with the editor. After you have made all the changes you want, press the **OK** button and Hardy will accept them and dismiss the Attribute Editor. Or you can abandon all your changes and leave the attribute values unaltered by pressing **Cancel**.

You can add a further attribute to the node by pressing the **Add attribute** button. This will ask for the name of the attribute and add it to the list. You can then give it a value as before. And you can remove an attribute that isn't required by selecting it from the list and pressing the **Delete attribute** button. As before, no changes are made to your diagram until you press **OK**, and you can always throw all your changes away by pressing **Cancel**.

Arcs have attributes in exactly the same way as nodes, and exactly the same Attribute Editor is used to change them.

4.7 Multi-way arcs

As well as being able to join one node to another node, you can join one node to several others (of the same type) with a single *multi-way arc*. You can do this in two different ways. Either you select all of the destination nodes then right-drag from the source node to any one of the destination nodes, or you can connect an additional node to an existing multi-way set-up, by first selecting the *junction symbol*then right-dragging from the common source node to the new node, as usual, to create a new connection from the existing junction symbol to the new node.

If, for aesthetic purposes, you want to alter the way the multi-way arc is shown, you can move the junction symbol around by first selecting it, then left-dragging it as though it were a node symbol.

4.8 Deleting nodes and arcs

There are two ways of deleting images: you can either select the shape(s) and use the **Edit: Cut** menu entry, or you can right click on the image and select the pop-up menu's **Delete Image** item. You can clear the whole card by using the **Edit: Select all** menu item followed by **Edit: cut**.

The major advantage of using **Edit: Cut** is that the images deleted are actually copied into an internal buffer (and, under Windows, the Clipboard), so you can still change your mind and replace the image by selecting the **Edit: Paste** menu entry. **Edit: Paste** will add the contents of the clipboard back into the diagram. The same buffer is used by **Edit: Copy** which copies the selected objects into the buffer but doesn't delete their images from the diagram. This *cut-and-paste* mechanism will also work for copying images from one card to another of the same type. Again, you can copy the entire card easily by using **Edit: Select all**.

The entire card will be deleted if you use the **File: Delete card** menu entry, but this cannot be undone.

4.9 Layout

The **Layout** menu provides several options to help create neat diagrams. To use the **Align vertically** option, select several nodes and then choose the option. The first node selected is taken to be the one which the others should be aligned with, in the vertical direction. The **Align horizontally** option does the same thing in the horizontal direction.

The **Straighten lines** option acts on a selected *multiline*, i.e. a line which has had control points inserted. It will attempt to align the segments of the line horizontally and vertically, according to the direction each segment is already tending towards.

The **To front** option places the selected image at the front of the diagram, so that it will be displayed on top of any overlapping images. Similarly, the **To back** option places the selected image at the back of the diagram, so that it will be fully or partially obscured by any overlapping images.

Hardy can also automatically layout a group of nodes as a tree with the **Layout: Format tree** option. The selected node is taken as the root of the tree and the nodes connected to it will be arranged as a tree on its right hand side. The format of this tree can be altered by using the Diagram Card Options dialog box which is opened through the card's **File: Options** menu entry. See Section 4.14 .

The **Layout: Apply definition** menu option lets you update a displayed card if you have, in the meantime, changed its Diagram Type definition, and existing values of the various properties of the displayed objects will be updated as appropriate. The **Layout: Zoom** option allows you to change the scale of the entire diagram. Reducing the scale will allow you to view a larger diagram area and, hence, more objects for the same physical size of card on your screen.

If you have any doubts about whether or not your new layout has been properly displayed, use the **Edit: Refresh display** menu entry to carry out a full re-display of the card.

4.9.1 Arc attachment points

Some diagram types may be defined so that arc images stay attached to a particular side of a node image (or vertex in the case of triangles, diamonds and other polyline symbols), depending on where you place the start and end points of an arc when creating it. Nodes which impose this behaviour on arcs have their *Use attachments* toggle switched on from the Diagram Type Manager.

Circles, ellipses and rectangles have four attachment points, one at each point of the compass, triangles and diamonds have three and four respectively, one at each vertex. When you create a new arc by sweeping from one node to another, the nearest attachment point for each node is found and used. When another arc is drawn, all arcs on the same attachment point are spaced out evenly. Note that this spacing is *not* performed if the node's attachment mode is not switched on.

It may be that the arc spacing that Hardy chooses causes arcs to overlap untidily. You can order the arcs on a particular attachment point by selecting the arc and, while holding down the left mouse button, dragging the endpoint to a new preferred position by the attachment point. The cursor changes to a bullseye during this operation. If the attachment point itself is wrong, the right mouse button may be used to drag the endpoint to the correct attachment point on the same node. Again, the cursor changes to a bullseye.

4.9.2 The toolbar

Each diagram card can have a toolbar displayed at the top below the menu bar. You can save space on the card by hiding this toolbar through the card's **Hyperlinks: Toggle toolbar** menu entry. The toolbar is used to give easy access to some of the most used layout and formatting facilities which are mostly otherwise available through menu options and image properties.

1. Left justify text (**Edit: Format text**).
2. Centre text (**Edit: Format text**).
3. No centring or justification (**Edit: Format text**).
4. Fit images to contents.
5. Don't fit images to contents.
6. Vertically align selected images on left .
7. Vertically align selected images on centre (**Layout: Align vertically**).
8. Vertically align selected images on right.
9. Horizontally align selected images on top
10. Horizontally align selected images on centre (**Layout: Align horizontally**).
11. Horizontally align selected images on bottom
12. Straighten lines (**Layout: Straighten lines**).
13. Format tree (**Layout: Format tree**).
14. Choose font (**Edit: Change font**).

4.10 Hyperlinks

You can *hyperlink* individual nodes, arcs and cards to other cards. This helps you organise your diagram by allowing you to set up a hierarchy which can give you a top-down view of it, presenting only as much detail at any level as is appropriate.

4.10.1 Linking arcs and nodes to cards

You can link a node or arc to an existing card or to a new card.

To link the object to a new card, right click on the image and choose the **Hyperlink to new card** option. This will ask you to specify what type of card you want (see Section 4.1), and construct a new card of this type which will be linked to the object in question. You will see the new card reflected in the display of the index tree in the control window, and, if the object was a node, you'll probably see its boundary highlighted to tell you that it is linked to another card (though this property can be switched on or off).

An alternative way of linking an object to a new card is to select the object you want to link, then select the **Hyperlinks: Link new card** option and proceed as before.

Left clicking on the image will now take you to the new card in subsequent browsing, as would clicking on the appropriate item on the card's link panel.

If you want to link the object to an existing card, you should select the card by its **Hyperlinks: Select card** menu entry, then right-click on the object to pop-up the menu so you can chose the **Hyperlink to selection** entry. Alternatively, you can select the object then use the **Hyperlinks: Link card to selection** entry on the card.

4.10.2 Linking cards to cards

One card can be linked directly to another card by selecting the card itself through the **Hyperlinks: Select card** menu entry. This will work even when there are no nodes or arcs present on the card. You can then link it to a new card by using the **Hyperlinks: Link new card** option and proceeding as above, or link it to an existing card by using the **Hyperlinks: Link card to selection** entry on the card you want.

4.10.3 The hyperlinks panel

All links to and from a card or any items on the card can be shown in the Hyperlinks panel which may be displayed on the righthand side of the card. However, as it takes up quite a lot of space on the card, you can hide it through the card's **Hyperlinks: Toggle link panel display** menu entry. The same menu entry will show it if it is already hidden.

You can display any card listed in the panels by left-clicking on its entry.

The default order of links in the *Links* panel may not be appropriate, especially for applications such as on-line manuals. Use the **Hyperlinks: Order links** option, to bring up the Order Items dialog box. Press on the *Source* titles in the desired order. The *Destination* list shows the new order.

4.11 Diagram expansion cards

In some cases a complex diagram will need several cards. If you create separate diagram cards in the normal way, each diagram will be saved in a separate file. This may be acceptable if the diagrams are only conceptually related, but may not be good enough if you wish to display the same node or arc on more than one diagram, but only have to type in the attributes for one node or arc. The way this is achieved is explained in Section 4.11.1 , below.

First, how can we expand a top-level diagram so we can show more detail? If you have a node which you wish to expand, select it and use the **Edit: New expansion** option. This creates a new expansion card whose title is the name of the node, and which can be reached by clicking on the node. Alternatively, if there is no node you wish to expand, select nothing and again choose the **Edit: New expansion** option. This will link a new expansion card to the existing card so that, conceptually, the whole card, rather than an image, is linked to the expansion.

Now when you save the top-level diagram card, all its associated expansion cards are saved as well in the one file. For this reason, expansion cards don't have their own file saving option.

An expansion card is accessed in the same way as any other linked card, either via the index tree in the control window, by left-clicking on an item, or by selecting an entry in the hyperlinks panel.

4.11.1 Same object, different cards

Returning to the question of having the same image on multiple cards. You will require this when you need to ensure that changing the attributes of one image changes the attributes of the other(s).

You can achieve this for nodes by selecting the node on one card, going to another, and there selecting the **Edit: Duplicate image for same object** menu entry. This will *not* make a duplicate node—it merely creates a new image for the existing, selected node. There is an underlying concept of node and arc for which the visual representation is a 'handle'. When an entirely new node image is created, a node is also created. When a node image is deleted, the node is only deleted if there are no other images for this node still in existence.

You can copy an arc image in a similar way by selecting the arc image, then linking up two nodes on the destination card. Instead of selecting an arc type, use the bottom option in the pop-up menu **Use selected arc object**. This makes a new image for the same selected arc.

You can prove that these images refer to the same underlying object by changing the label text. All related images will have their labels changed to reflect the new text.

IMPORTANT NOTE: this will not work *across diagram files*, since all diagram files are stand-alone and cannot reference other files. Only cards in the same hierarchy of diagrams can be used.

4.12 Containers

Some nodes can be set-up to be *containers*. These are nodes which can contain other nodes (of specified types) so that, if you move the container node, its contents are moved with it. This can look the same as having nodes super-imposed on each other, but their behaviour is different.

There are two special things that you can do with containers.

1. You can move nodes into and out of the container .
2. You can split a container into sub-containers which you can then split further if you want.

You move nodes in and out of containers in the same way as you move them normally. However, if you move a node that was outside a container across the container's boundary, a dialog box will be popped-up to ask you whether or not you want it inside the container. If you prefer, you can leave a node sitting over the container: it will look the same as if it were inside but it won't move when the container moves. Similarly, when you move a node that is contained in a container across the container boundary, you will be queried to check whether or not you really want it to be moved outside the container.

To sub-divide a container (or a sub-container), you control-right-click on it and a menu will pop-up with entries that allow you to split the container into two: either horizontally (so the sub-containers lie beside each other), or vertically (so the sub-containers lie above and below each other). (Other menu entries let you change the appearance of the container's boundary. See Section 8.1 .)

4.13 Printing diagrams

Under the UNIX and X environment, Encapsulated PostScript (EPS) output is provided. An entire hierarchy of diagrams may be printed to EPS files in one shot using the **File: Print hierarchy to files** menu option. See also Section 10 on platform-specific features for recommended ways of printing out diagrams under Windows.

4.13.1 Printing under X

The diagram/expansion card **File: Print PostScript** option pops-up a Printer Settings dialog box which lets you change the default settings.

Printer Command: the printer command, e.g. `lpr`.

Printer Options: any command line options for your printer, e.g. `-PE17`.

Portrait: if on, will print in portrait mode. If off, will print to landscape mode.

Print to file: if on, will prompt for a filename to print to. If off, the printer command will be invoked with the printer options appended. Note that the preview toggle over-rides this option if on.

Preview only: if on (the default), the `ghostview` program is invoked to preview the PostScript output. Obviously **ghostview** needs to be installed and in your path. Previewing allows you to adjust the scaling and translation without wasting too many trees.

X/Y Scaling: scales the image.

X/Y Translation: translates the image.

To include a diagram in a LaTeX file, first print it to a PostScript file. Of the many possible ways of including the file in your LaTeX document, the preferred technique is to use a macro package such as `psbox` to scale and position the image based on its size. This is possible since Hardy outputs EPS files which contain size ('bounding box') information. To use *psbox*, put the following statement near the top of your document:

```
 \input psbox.tex
```

You may then uses commands such as the following:

```
\begin{figure}
  $$\psboxto(0.9\textwidth;0cm){screendump.ps}$$
  \caption{Example Hardy session under X}\label{screendump}
\end{figure}
```

The dollar signs centre the image. The "`0.9\textwidth;0cm`" indicates that the image is to be 90% of the normal width of the text on the page, but scaled proportionally in the vertical dimension. A non-zero value sets that dimension, possibly distorting the image. To omit dimensions, use the `psbox` macro instead (see documentation for the `psbox` package).

4.13.2 Printing under MS Windows

Hardy for Windows provides the same printing facilities as described above (see Section 4.13.1) for X through the **File: Print PostScript** and **File: Print hierarchy to file** options. If you have a word processor which can scale EPS images (such as LaTeX), you could include an EPS diagram in a document. A better alternative may be to use the **Edit: Copy** option.

Using the clipboard

Windows 3 has the concept of a *clipboard*, viewed using the *clipboard viewer*. Applications may exchange data using the clipboard. Types of data that may be exchanged include bitmaps and *metafiles*, which are 'recordings' of the drawing commands used to build up a picture, and are easily scaled without losing resolution, unlike a bitmap.

Hardy for Windows has an option in the diagram card **Edit** menu for copying a metafile version of the diagram to the clipboard. The user-defined scaling factor (set from the options dialog box called from **File: Options**) will affect the size of the diagram passed to the clipboard. You may then paste the image into another application for editing or printing out. For example, importing into the drawing program Corel Draw splits up the diagram into its component shapes, ready for fine-tuning. Importing into Windows Paintbrush makes it into a bitmap, and this may be the easiest and cheapest way of printing out a Hardy for Windows diagram.

4.14 Diagram card options

The **File: Options** menu pops-up a small window with some options that apply to that diagram card.

1. **Label Arcs with abbreviation**
   If on, will cause arcs to be labelled with the abbreviation stored in the arc definition. Useful on monochrome displays and non-colour printers.
2. **Snap to grid**
   If on, images will be positioned to the nearest grid line (conceptual, not visible) so that images are easier to align neatly. The default is on.
3. **Colour**
   If on, colour is used; if off, non-white colours will be painted in black. This applies to printing too.
4. **Quick edit mode**
   By default off, this inhibits diagram redraws when simple edit operations are performed. With quick edit mode off, the whole diagram will be redrawn if one node is moved, which can be slow for large diagrams.
5. **Scale**
   Enter a new number between 0 and 1 to scale the diagram. This currently has no effect on printing.
6. **Grid spacing**
   The grid spacing determines the coarseness of image positioning, if the snap to grid option is on. The default is 10 pixels.
7. **Print file**
   Allows the user to change the filename used for printing this diagram to PostScript.
8. **Auto-layout options**
   These are for use with the **Layout: Format tree** menu of a diagram card. They allow values to be set for the left and top margins, the width and height, and the X and Y spacing between nodes.

4.15 Diagram editing summary

The following is an alphabetical summary of common diagramming operations.

Bending an arc
Select the arc, add control point(s) with the **Edit: Add control point** option, drag control points. Turn the curve into a straight line by deleting all but two control points.

Copying images
Select images on a diagram card of the same type, go to the destination card and select the **Edit: Copy images(s)** option. The images will be copied (with new underlying node and arc objects). An alternative is to select the source card with **Hyperlinks: Select card** before the copy operation, in which case the whole card is copied.

Creating an arc
Holding down the right mouse button, drag from the start node to the end node. Choose an arc type from the pop-up menu.

Creating a diagram card
Either select **Cards: Create top card** on the main control window, or select **Hyperlinks: Link new card** on a particular card. Choose an appropriate diagram card.

Creating a node
Select a node symbol on the floating diagram symbol palette, move the cursor to where you want the node to be positioned, and click the left mouse button.

Deleting an image
Either select the image(s) and choose the **Edit: Cut** option, or right-click on the image and choose the **Delete image** option.

Editing attributes
Same as for *Labelling an image*. Press on the **Edit** button to use a text editor for editing large attribute values.

Expanding a node
Select the node, choose the **Edit: New expansion** option. A new expansion card is created. Expanding without selecting a node causes the new card to be linked to the original card, rather than to an image within it.

Labelling an image
Pop-up the attribute editing window by using control-left-click.

Linking an image to a new card
Right-click on the image and select the **Link new card** option. Left-click on the image to go to the card subsequently.

Loading a diagram
If you have saved an associated Hardy index file, use the -f option on the command line (see above) or use the **File: Open file** option on the control window. Otherwise, create a card of the appropriate diagram type, and select the **File: Open** option, entering the filename.

Maximising card space
Use the **Hyperlinks: Toggle link panel display** option and resize the card to fill the screen. Note that, unless you start in the middle of the canvas, (see *Scrolling around* below), you may have to move your diagram around node by node to extend up or left.

Moving node(s)
Select a node or nodes, and drag one to the desired position. If more than one image is selected, all selected images will be moved.

Printing/previewing
Choose the **File: Print PostScript** option, set **Preview only** for a preview or switch it off to print to the printer, set **Print to file** if that's wanted as well, then press **OK**. Enter any printer command options in the **Printer Options** text item. Change the scaling and/or translation to fit the picture to a page.

Resizing a node
Select image and drag control points. Dragging a corner point and holding down shift retains the proportions.

Saving a diagram
Select the **File: Save file** option, enter filename (not path). Also choose the control window's **File: Save file** option to save the Hardy index file (with a pointer to the diagram file).

Scaling diagrams
Either choose the **File: Options** menu item in the Control Window and enter a new number for the scaling factor, or choose the **Layout: Zoom** item for the card.

Scrolling around
Left-click click on the up and down arrows of the scroll bars, or drag the scroll bar. (Under Windows, dragging the scroll bar is slow since the diagram is repeatedly re-drawn. Under X, bitmaps are used for scrolling which makes it much faster.)

Selecting an image
Shift left click; same for deselecting. Select multiple images by left-dragging on the canvas and releasing when the rubber band encompasses all desired images. Deselect all selected images by left-clicking on the canvas.

Wrapping label text
Select image(s), choose **Edit: Format text** option.

4.16 Mouse functionality

4.16.1 Left button

1. Click on node – the name of the node is displayed in the card's status line. If a current annotation is selected that is legal for the node, it is dropped onto the node at the mouse position.

2. Click on arc – the name of the arc is displayed in the card's status line.
3. Click on a position in the canvas away from a node or arc – if a node is currently selected on the floating palette (indicated by the *cross-hairs* cursor), a new node is placed on the canvas at the mouse position.
4. Click-and-drag a node – the node moves to the new mouse position. If the final position of the node is over a container which can contain it, the user will be asked whether or not the node should be placed *inside* the container or simply left *on top* of it.
5. Click-and-drag a selected arc label – the label moves to the new mouse position.
6. Click-and-drag a node selection handle – the node image is rescaled, based on the direction of movement of the mouse.
7. Click-and-drag a divided node division control handle – the division is moved to the new mouse position.
8. Click-and-drag on the selection handle of an arc at the end where it meets a node for a diagram type that defines arc attachment points – the *bullseye* cursor appears and the handle may be moved to a different position at the *same* attachment point.
9. Click-and-drag on the canvas away from a node or arc – a rubberband box appears which the user can drag so that it surrounds card items, i.e. nodes and arcs. When the user releases the mouse button, the rubberband box disappears and the surrounded nodes and arcs are selected and their selection handles are displayed.
10. Shift-click on a node or arc – if the node or arc was not already selected, its selection handles are displayed, and the name of the node or arc is shown in the card's status line. If the node or arc was already selected, it becomes deselected and its selection handles disappear.
11. Control-click on a node or arc – if the node or arc type has defined attributes, an Object Attribute Editor dialog box appears which allows attribute values to be set or changed. See Section 4.6 .
12. Click on a card title in the *Links* or *Reverse Links* scrolling lists – the selected card is opened.

4.16.2 Right button

1. Click on a node or arc – a Node or Arc Action Menu containing a list of useful actions appears. Selecting an entry causes the appropriate action to be performed. An Action Menu provides a short cut for accessing popular actions as an alternative to first selecting the node or arc and then selecting an entry in one of the diagram card menus. The actions available are:
   (a) Edit attributes;
   (b) Hyperlink to selection;
   (c) Hyperlink to new card;
   (d) Unlink item;
   (e) Delete image.

2. Control-click on a divided node – the Divided Object Properties dialog box pops up. See Section 8.1.6 .
3. Control-click on a container – a menu pops up with four entries:
   (a) **Split horizontally** – the region of the container node image in which the cursor is positioned will be split into two sub-regions lying alongside each other.
   (b) **Split vertically** – the region of the container node image in which the cursor is positioned will be split into two sub-regions lying above and below each other.
   (c) **edit left edge** – this allows the image properties of the left edge of the sub-region of the container node image in which the cursor is positioned to be tailored. The Division Properties dialog box appears, see Section 8.1 .
   (d) **edit top edge** – this allows the image properties of the top edge of the sub-region of the container node image in which the cursor is positioned to be tailored. The Division Properties dialog box appears, see Section 8.1 .
4. Click-and-drag a node or node annotation – the outline of an arc is drawn from the node or node annotation following the mouse. If the node or node annotation has attachment points, the arc will come from the nearest attachment point to the initial mouse position.

   If the mouse button is released over a node or node annotation and there is a single legal link in the diagram type definition between the initial node and this one, then that arc is drawn.
   If more than one type of link is legal between the nodes, the currently selected arc in the floating palette will be used to resolve the ambiguity if possible, otherwise a dialogue is entered into with the user to specify which type of arc is intended.
   When a legal arc has been drawn, the status line of the diagram card says:

   "`Linked a` *start node type name* `to a end node type name with a` *arc type name*".

   If the mouse button is released over a node and there is not a legal link in the diagram type definition between the initial node and this one, then the status line of the diagram card says:

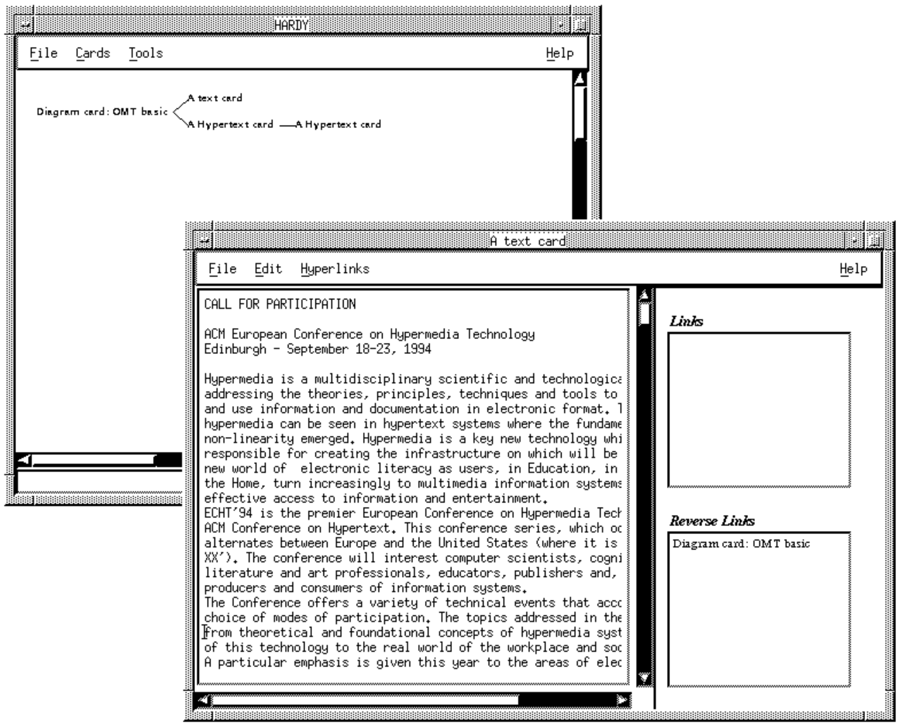
   "`No legal arcs from a` *start node type name* `to a` *end node type name*".

   If the mouse button is released over one of several selected nodes, and there is a legal multi-way arc between the nodes, a multi-way arc is drawn between the initial node and all the selected nodes. If *Auto dog-leg* is set in the Junction Editor, an extra control point will be inserted and the connections will be constrained to the horizontal and vertical.

   If the mouse button is released over the canvas away from a node, no arc is created.
5. Click-and-drag the selection handle of an arc that is attached to a node which has multiple attachment points defined for it – the handle will follow the mouse so it can be moved to a *different* attachment point of the node, where the button is released.

5 Text cards

The text card is the simplest form of card provided by Hardy. It consists of a text subwindow which displays the contents of a file.



```
CALL FOR PARTICIPATION

ACM European Conference on Hypermedia Technology
Edinburgh - September 18-23, 1994

Hypermedia is a multidisciplinary scientific and technologica
addressing the theories, principles, techniques and tools to
and use information and documentation in electronic format. T
hypermedia can be seen in hypertext systems where the fundame
non-linearity emerged. Hypermedia is a key new technology whi
responsible for creating the infrastructure on which will be
new world of  electronic literacy as users, in Education, in
the Home, turn increasingly to multimedia information systems
effective access to information and entertainment.
ECHT'94 is the premier European Conference on Hypermedia Tech
ACM Conference on Hypertext. This conference series, which oc
alternates between Europe and the United States (where it is
XX'). The conference will interest computer scientists, cogni
literature and art professionals, educators, publishers and,
producers and consumers of information systems.
The Conference offers a variety of technical events that acco
choice of modes of participation. The topics addressed in the
from theoretical and foundational concepts of hypermedia syst
of this technology to the real world of the workplace and soc
A particular emphasis is given this year to the areas of elec
```

Figure 5.1: Example Hardy text card

5.1 Editing text cards

When a text card is first created, there is no file associated with it. Open a file with the **File: Open file** option; if you edit the file later, you can use the **File: Save file** or **File: Save as...** options to save the file.

Text card files are edited through the **Edit: Run editor** option. This uses the EDITOR environment variable to allow you to specify your favourite editor. Under Unix, you must save the file and exit the editor to return control to Hardy; under Windows, a child process returns control to the parent immediately. Alternatively, under Unix only, the subwindow can be used to edit the text directly.

As with any card, the title may be changed using the **File: Card title** option.

To get back to the main Hardy control window, if it's buried under a pile of other windows, use the **File: Goto control window** option.

To delete the card, use the **File: Delete card** option. Remember that the difference between quitting (**File: Quit card**) and deleting a card is that quitting is merely getting rid of the physical display for the card, not deleting the actual card representation in the hypertext index. If you have many new cards, you may wish to quit some of them to avoid a build up of windows. Deleting, on the other hand, deletes the card from the hypertext index, although it does not in general delete any file which may be associated with that card.

5.2 Linking text cards

As a text card doesn't contain any items, you can only link the entire text card to other items or other items to the entire text card.

To link a new card to the current text card, choose **Hyperlinks: Link new card** and choose a card type to create when prompted. You will now get to the linked card by clicking on the relevant title in the link panel display.

The default order of links in the Links panel may not be correct, especially for applications such as on-line manuals. Use the **Hyperlinks: Order links** option, and press on the Source titles in the desired order. The Destination list shows the new order.

If the Links panel on the right-hand-side of the card is required, select the **Hyperlinks: Toggle link panel display** option. Use the same option again to hide it.

5.3 Mouse functionality

5.3.1 Left button

1. Click on a position in the text subwindow – *under X only*, the text editor cursor will be positioned at the click.

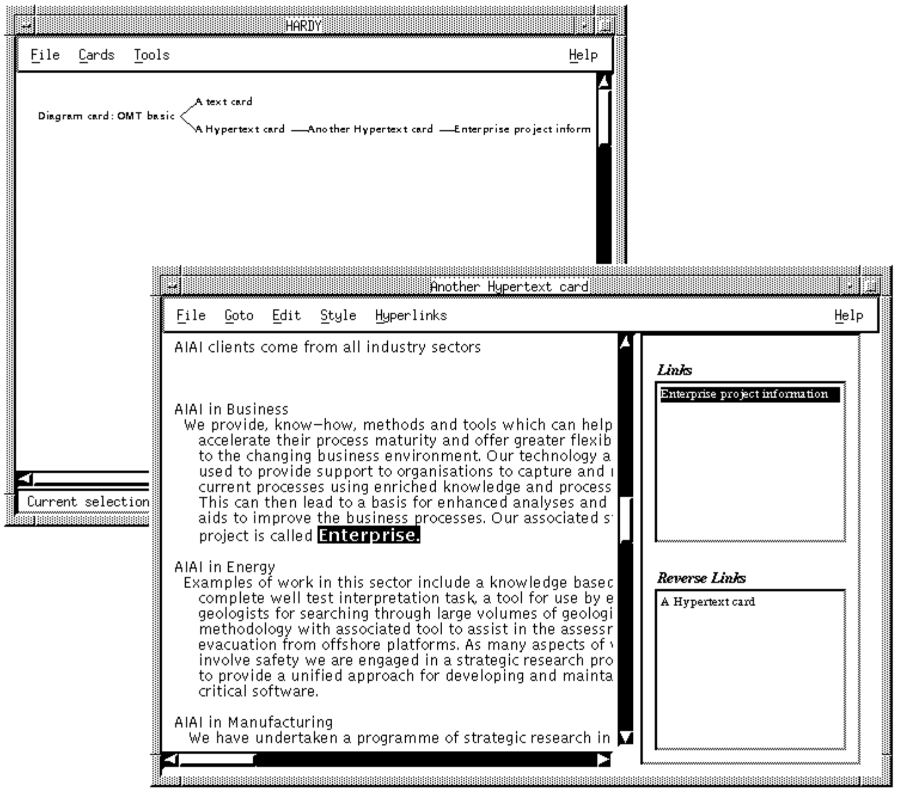
2. Click on a card title in the *Links* or *Reverse Links* scrolling lists – the selected card is opened.

5.3.2 Right button

No use is made of the right mouse button.

6 Hypertext cards

The hypertext card is similar to the text card, in that it may display plain text files and has some of the same menu options.



```
AIAI clients come from all industry sectors


AIAI in Business
 We provide, know-how, methods and tools which can help
   accelerate their process maturity and offer greater flexib
   to the changing business environment. Our technology a
   used to provide support to organisations to capture and
   current processes using enriched knowledge and process
   This can then lead to a basis for enhanced analyses and
   aids to improve the business processes. Our associated s
   project is called Enterprise.

AIAI in Energy
 Examples of work in this sector include a knowledge basec
   complete well test interpretation task, a tool for use by e
   geologists for searching through large volumes of geologi
   methodology with associated tool to assist in the assessr
   evacuation from offshore platforms. As many aspects of
   involve safety we are engaged in a strategic research pro
   to provide a unified approach for developing and mainta
   critical software.

AIAI in Manufacturing
  We have undertaken a programme of strategic research in
```

Figure 6.1: Example Hardy hypertext card

However, it has the following distinguishing features:

- Text blocks may be marked with the mouse, and given different font and colour attributes. A text block is a hypertext item, in the same sense that a diagram node or arc is an item, and may be linked with other cards and items.
- The text is not directly editable, unlike the text card under X, although a text editor may still be invoked.
- The hypertext card has the concept of *hypertext types* in the same sense as *diagram types* (see Section 9.1). Each hypertext card is an instance of a user-defined type, although you can access a default type of hypertext card by creating a card called simply “Hypertext card”.

- Hypertext cards have the concept of *hypertext sections*. You may define a new section by marking a block using a section block style. The **Goto** menu allows you to walk sequentially through the sections, and to go to the first section.

6.1 Hypertext blocks

Although the hypertext card may display ordinary text, it will most often be used to mark blocks of text and associate them with other cards and items. This causes codes to be stored in the text. These are interpreted specially by Hardy, and are in LaTeX compatible format.

To mark a new block,drag the mouse holding the left button down from the top left of the intended block to the bottom right (this may require some practice). Remember that you should drag a bounding box that doesn't extend beyond the text characters you wish to include: the box should be *just* inside the characters at the edge of the block. For instance, if selecting the word 'thing', the bounding box should start at the top left of the letter 't', and stop at the bottom right of the letter 'g'.

The selected blockwill go cyan (colour displays) or reverse video (monochrome displays). Any selected block, whether new or old, will use this highlighting. A block will only be remembered by Hardy if you choose a *style* for it with the **Style** menu. This allows you to set font and colour attributes for the block. Once the style has been set, the block is a Hardy hypertext item and can be linked with other cards and items in the usual way. You may remove a block by selecting it (shift-left click) and choosing the **Edit: Clear block** menu item. Blocks without a style set will simply be forgotten when deselected (shift-left click).

6.2 Editing hypertext cards

Apart from adding and clearing blocks, running the editor from the **Edit: Run editor** menu entry is the only way of editing a hypertext card's contents. You do this in exactly the same way as you do for a text card (see Section 5.1). Be sure that you don't disturb the embedded block codes. It will be more convenient to do as much editing as possible *before* marking blocks.

6.3 Mouse functionality

6.3.1 Left button

1. Click on a block within the text canvas – the corresponding card to which the block is hyperlinked will be selected and displayed, if one exists. If the block is not hyperlinked, no action results.

2. Click-and-drag within the canvas – will define and select a block specified by its top left corner being at the initial click position and its bottom right being at the release position. This block is 'temporary' until it has had a style set through the **Style** menu. A block will always contain a piece of continuous text.

3. Shift-click on a 'temporary' block within the canvas – the block is deselected and its highlight is removed.
4. Shift-click on a block with a set style – if the block was not already selected, it becomes selected; if the block was already selected, it becomes deselected.
5. Click on an entry in either the *Links* or *Reverse Links* list boxes – the corresponding card is selected and displayed.

6.3.2 Right button

1. Click on a block with a set style – a menu appears, presenting some of the most used actions associated with blocks for easy access.

7 Media cards

The media card is similar to the text card, in that it may display plain text files and has some of the same menu options.

However, it has the following distinguishing features:

- Text blocks may be marked with the mouse, and given different font and colour attributes. A text block is a hypertext item, in the same sense that a diagram node or arc is an item, and may be linked with other cards and items.
- The text is not directly editable, unlike the text card under X, although a text editor may still be invoked.
- The media card has the concept of *media types* in the same sense as *diagram types* (see Section 9.1). Each hypertext card is an instance of a user-defined type, although you can access a default type of media card by creating a card called simply “Media card”.

Please note that the media card is experimental and may not be present in public distributions of Hardy.

7.1 Media blocks

Although the media card may display ordinary text, it will most often be used to mark blocks of text and associate them with other cards and items.

To mark a new block,drag the mouse holding the left button down.

The selected blockwill go cyan (colour displays) or reverse video (monochrome displays). Any selected block, whether new or old, will use this highlighting. A block will only be remembered by Hardy if you choose a *style* for it with the **Style** menu. This allows you to set font and colour attributes for the block. Once the style has been set, the block is a Hardy hypertext item and can be linked with other cards and items in the usual way.

You may also mark text up using the default font attributes, accessed by via the items at the top of the **Style** menu. Marking up in this way is purely visual and does not create a hypertext block.

Currently, only one block may have a given start or end point, and blocks may not overlap. Eventually it is intended to remove at least the first restriction. Blocks may not currently be cleared once created, except by clearing all blocks, or by deleting the text and reinserting it. Bitmaps may not be used as blocks.

7.2 Editing media cards

You may edit the text directly. Blocks will move around with the text; but if you delete text at the start or end of the block, you may delete the block markers and cause the block to disappear.

Pictures may be inserted into the media card by selecting the **Edit: Insert Image** menu item. A pictures is stored as a reference to the bitmap filename, so this file should be present in the same place when you load the media file.

7.3 Mouse functionality

7.3.1 Left button

1. Left click in the media card to set the caret position, where text is inserted.
2. Control-left click on a block within the text canvas – the corresponding card to which the block is hyperlinked will be selected and displayed, if one exists. If the block is not hyperlinked, no action results.
3. Click-and-drag within the canvas – this will select an area of text, for later marking with a font or block style. A block will always contain a piece of continuous text.
4. Shift-left click on a block to select it.
5. Click on an entry in either the *Links* or *Reverse Links* list boxes – the corresponding card is selected and displayed.

7.3.2 Right button

1. Click on a block with a set style – a menu appears, presenting some of the most used actions associated with blocks for easy access.

8 Symbols

Symbols are used to construct the images that are displayed on a diagram card. There are two main types of symbol, node symbols and arc symbols, and a further type for *arc annotations* (such as arrowheads). All the symbols that are recognised for a particular type of diagram will be gathered together and presented as a floating palette. Within the palette the symbols are divided into Node symbols and Arc symbols. Annotation symbols may also be shown separately.

A basic range of symbols is provided by Hardy in the Standard Symbol Library for use when designing new diagram types. You can also construct new shapes when you require by using the Node Symbol Editor and the Arc Symbol Editor (see Section 8.4 and Section 8.5 respectively). New symbols may also be held in symbol libraries for later use in exactly the same way as for the standard symbols. Symbol libraries are organised by the Symbol Librarian (see Section 8.2 .

Generally, symbols are used in two different ways. The first, and more general, is the symbol that is available in a symbol library. At this level, the symbol has got a defined shape, colour and other general properties. It is then available for customising for use in the displayed image of a particular type of node or arc in a particular type of diagram. Its shape cannot be changed, but its colour, etc, properties may be over-ridden and other specialist properties can be added. This will be done through the Node Type Editor or the Arc Type Editor (see Section 9.1.2 and Section 9.1.9 , respectively).

8.1 Symbol properties

Symbols are made up of simple parts, largely lines and areas. Every line will have a width (in pixels), a style (solid, dashed, etc), and a colour.

Areas are more complicated. They can be *primitives*, normally provided through the Standard Symbol Library, or they can be *composites* made up from primitives or other composites. A primitive symbol has an outline with exactly the same properties as other lines (width, style, and colour), and an area with a colour property (termed the *fill colour*) bounded by the outline. A composite symbol will be made up from simpler symbols; its properties are the properties of its individual sub-symbols.

8.1.1 Metafiles

There are some shapes which would be (at best!) very difficult to construct from other standard symbols. In order to allow arbitrary shapes to be used, Hardy supports the use of *metafiles*. Metafiles let you add new primitive symbols to your repertoire.

Metafiles define drawings in terms of pens (for outlines), brushes (for “fill” areas), and shapes. As their contents are all relative to a starting position, the shape may be placed wherever it’s required and it can be scaled without loss of resolution. When a metafile is imported into Hardy from an external package for use as a node or arc symbol, selected pen and brush instructions can be intercepted in such a way that the metafile-defined symbol will have the same outline and fill properties as any other symbol.

This is done using the Metafile Colour Assignment dialog box which is opened from the Node Symbol Editor when editing a symbol which has been defined through a metafile. The different operations involved in defining the shape are shown in the list box labelled *Operations*. Each pen and brush operation is uniquely identified so that it may be distinguished and added to the lists of *Outline operations* (pens only) and *Fill operations* (brushes only) by selecting it and pressing the **Add** button. Only these selected pens and brushes will be altered if the symbol's outline or fill characteristics are changed later.

The **Clear** buttons will clear any entries in the relevant list boxes.

8.1.2 Arc symbols

If we look at arc symbols, we see that they are basically lines which may carry annotations (arrowheads, etc). If present, arc annotations are treated as part of the arc symbol and will share common properties, i.e. changing the colour of the arc will change the colour of the annotation. (Arc annotations have additional properties: see Section 8.1.3 .)

Every arc has a width (in pixels), a style (solid, dashed, etc), and a colour.

Arcs have three regions: their start, middle and end regions. Annotations and labels are usually placed in one or other of these. Each region will support only a single label, though several annotations can appear there.

8.1.3 Arc annotations

Arc annotations are symbols such as arrowheads that are used to decorate arcs. New symbols may be imported through metafiles(see Section 8.1.1). They can either be part of an arc symbol, or they can be added incrementally to a particular arc symbol as required when a diagram is being constructed. Annotations may be placed in one of the start, middle or end regions of the arc, and several annotations can appear in the same region.

Arc annotations can be customised by setting parameters which control their size and their position on the arc. Size is the length of the annotation image in pixels. Position has three separate aspects:

1. there is the gap, in pixels, allowed between one annotation and the next one in the same region;
2. there is the X offset of the annotation from the start of the arc. This is specified as a fraction: 0.0 means the start, 0.5 the middle, and 1.0 the end.
3. there is the Y offset between the mid-point of the annotation symbol and the arc: a positive offset moves the annotation above the arc, a negative one moves it below. This is specified in pixels.

8.1.4 Node symbols

Node symbols are primitive or composite shapes. A primitive shape has an outline and an enclosed area. The outline can be specified in terms of its width (in pixels), style (solid, dashed, etc), and colour. The enclosed area will have a colour: its *fill colour*. Two special types of primitive symbol are provided: divided nodes and polyline symbols. These have additional special properties (see Section 8.1.6 and Section 8.1.7).

Composite symbols are made up from other symbols, selected from existing Symbol Libraries and glued together using the Node Symbol Editor (see Section 8.4). A composite symbol will reflect the properties of its sub-symbols. These can be changed as a whole, i.e. the composite is treated as though it had a single outline and a single enclosed area like a primitive, or sub-symbols can be selected and treated individually.

As well as the general display properties, node symbols have additional properties reflecting their behaviour.

1. Use attachments – this controls where arcs will attach to the node symbol. If attachments are not used, arcs will appear as though they were connected to the centre of the node symbol; if attachments are used, arcs will appear as though they were connected to the nearest defined attachment point. See Section 8.1.5 for further details.

   If the attachment point chosen by the system is not the one you want, you can move the connection by selecting the arc and right-click-and-dragging the appropriate arc selection handle to a *different* attachment point of the same node.
2. Space attachments – if attachments are in use, they may be spaced or not. If attachments are not spaced, all arcs will appear as though they were connected directly to attachment points; if they are spaced, the arcs will automatically separate themselves at each attachment point where two or more join.

   If the ordering at any particular attachment point is not what you want, you can select the arc and left-click-and-drag the appropriate selection handle to a different position at the *same* attachment point.
3. Shadow – if shadowing is set, the symbol will have a "shadow" of the same shape in black and offset slightly.
4. Fixed width – if this is set, the width of the symbol cannot be changed once it is in use.
5. Fixed height – if this is set, the height of the symbol cannot be changed once it is in use.

Additional properties hold if the symbol has been defined through a metafile (see Section 8.1.1).

8.1.5 Attachment points

A node symbol may have attachment points defined (see Section 8.1.4) to which arcs can be connected. These rotate with the symbol when necessary.

The symbols provided in the Standard Symbol Library have pre-defined attachment points:

1. divided rectangles have a control point in the centre of their top and bottom edges, and a further control point in the centre of the vertical edges of each region,
2. polyline Symbols (triangles and diamonds) have attachment points at each of their vertices,

3. other primitive symbols have attachment points in the centre of each edge of their bounding box,
4. composite symbols have attachment points at each attachment point of their component sub-symbols, and additional attachment points in the centre of each edge of the overall bounding box.

The user may define the attachment points for symbols that are imported as metafiles, and may define additional attachment points for standard symbols. This is accomplished through the Attachment Point Editor which is invoked from the **Attachment points...** button of the Node Symbol Properties dialog box.

The dimensions of the symbol (its bounding box, in pixels) are shown in the message area, and the identifiers of all current user defined attachment points are displayed in the *Attachment points* list box. Pressing the **New** button generates a new unique identifier which is entered into the *Attachment points* list and selected. Alternatively you can select an existing identifier in the list. In either case, the currently selected identifier will be shown together with its position as X and Y offsets (in pixels) from the symbol's reference point. These values can now be changed: selecting another entry, or pressing **New** again or **OK**, will accept them.

You can remove an entry by selecting it in the list and then pressing **Delete**.

8.1.6 Divided nodes

Divided node symbols are supplied by the Standard Symbol Library for efficiency. They provide rectangular shapes with more than one text region so that more than one label can be displayed. This means there are two differences between these and normal primitive symbols.

You can alter the position of the divisions between the different regions by selecting the symbol. This will display its selection handles, and you will see an extra handle at the middle of each division. You can left-drag these to move the divisions to new positions within the existing rectangle. Note that you can't move one division past another.

You can tailor the appearance of these divisions through the Divided Object Properties dialog box which is invoked by control-right-clicking on a divided object in the Node Symbol Editor (see Section 8.4). This shows the line colour and style for each of the divisions (top one first), and these can be altered in the usual way.

8.1.7 Polyline symbols

A polyline symbol is a primitive used for constructing polygonal node symbols. Standard polyline symbols are supplied through the triangle and diamond symbols which can be modified to produce most other shapes needed.

Unlike the other standard node symbols, polyline symbols have control points at each of their vertices. These will be displayed when the symbol is selected, and can be moved around by left-dragging on them. Additional control points may be added to or deleted from polyline symbols in the Node Symbol Editor through the **Edit: Add control point** and **Edit: delete control point** menu items. Combining the number of control points with their positions allows you to define arbitrary polygonal shapes.

Every legal polyline symbol must have at least *one* control point.

8.2 Symbol librarian

8.2.1 Appearance and functionality

Symbols are organised in libraries which are managed by the Symbol Librarian. The Symbol Librarian may be invoked from the **Tools: Show symbol librarian** menu of the Control Window.

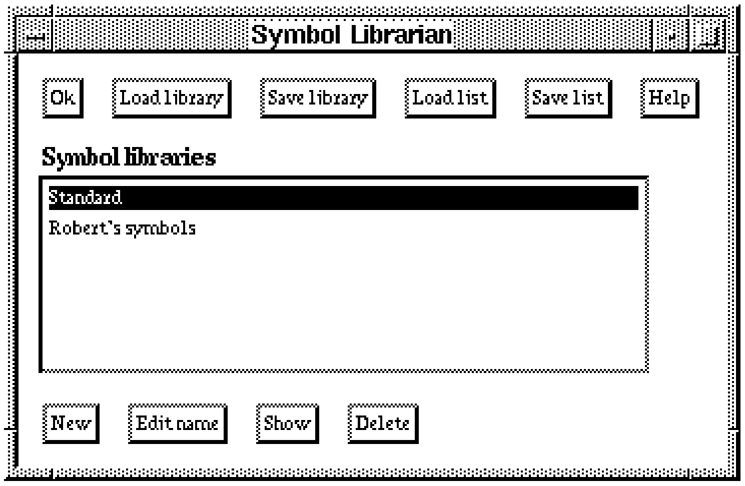


Figure 8.1: The Symbol Librarian

The *Symbol libraries* list box displays the names of all libraries which are currently open. The Standard library, containing the primitive symbols, is always available.

8.2.2 Buttons

The buttons are arranged in two groups, one providing general facilities, the other providing facilities for individual symbol libraries.

General

1. **OK** – the Symbol Librarian and any open Symbol Libraries are dismissed. If changes have been made to any Symbol Libraries and these have not been saved, the user is asked whether these changes should be saved.

2. **Load library** – open and load a Symbol Library from disk using a File Selector dialog box with the filter initialised to `*.slb`. Add its name to the end of the *Symbol Libraries* list.
3. **Save library** – save the contents of the currently selected Symbol Library to its associated disk file. If no disk file is defined for the library, the File Selector dialog box will appear.
4. **Load list** – open and load the Symbol Libraries whose names are specified in `diagrams.def`. Display their names in the *Symbol Libraries* list box.
5. **Save list** – save the names of the Symbol Libraries given in the *Symbol Libraries* list box into the definition list file (`diagrams.def`).
6. **Help** – for Hardy under X, the wxHelp program is started and information on the Symbol Librarian is displayed. Under Windows, the Windows Help system is started at the appropriate place in the Hardy manual.

Symbol libraries

1. **New** – create a new Symbol Library, add it to the list of loaded Symbol Libraries, and make it the current selection.
2. **Edit name** – change the name of the currently selected Symbol Library.
3. **Show** – display the currently selected Symbol Library.
4. **Delete** – delete the currently selected Symbol Library from the list.

8.2.3 Mouse and cursor functionality

Left button

1. Clicking on an entry in the *Symbol Libraries* list box selects the Symbol Library with that name.
2. Double-clicking on an entry in the *Symbol Libraries* list box selects that Symbol Library and proceeds as though the **Show** button had been pressed.

Right button

No use is made of the right mouse button.

Cursor

No special cursor pattern is used.

8.3 Symbol libraries

8.3.1 Appearance and functionality

A Symbol Library is opened from the **Load library** or **Show** buttons of the Symbol Librarian, or by Double-clicking on an entry in its *Symbol libraries* list box.

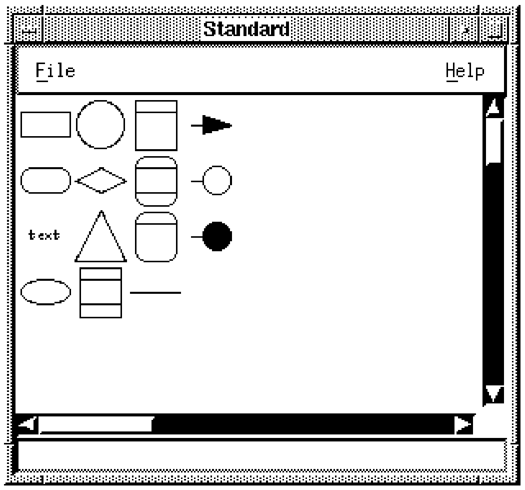


Figure 8.2: A Symbol Library palette

The name of the Symbol Library is displayed in the title bar. The status line is used to display the name of a symbol.

The Standard Library is provided with the system. The user cannot add further symbols to it, or delete symbols from it.

8.3.2 Menu options

The Symbol Library has two menus – **File** and **Help**.

File menu

1. **File: Edit selected symbol** – if a symbol is selected in the Symbol Library, the Arc or Node Symbol Editor is opened as appropriate, see Section 8.4 and Section 8.5 , and the symbol is read in so that it can be altered.
2. **File: Delete selected symbol** – if a symbol is selected in the Symbol Library and the user confirms that the symbol should be deleted, it is deleted.
3. **File: Load symbol from metafile** – a symbol definition is read into the Symbol Library from a file specified by the File Selector dialog box and named through the Symbol Name dialog box.

The name of the new symbol is entered into the text entry area labelled *Name of new symbol*. The checkboxes allow the properties of the symbol to be set: whether it is a node symbol or an arc annotation symbol, and whether the symbol should be rotated or not when it is used as an arc annotation.

4. **File: Exit symbol library** – the Symbol Library is dismissed.

Help menu

1. **Help: Help on symbol library** – for Hardy under X, the wxHelp program is started and information on the Symbol Library is displayed. Under Windows, the Windows Help system is started at the appropriate place in the Hardy manual.

8.3.3 Mouse and cursor functionality

Left button

1. Clicking on an unselected symbol in the palette selects that symbol. Clicking on a selected symbol deselects that symbol.

Right button

1. Clicking on a symbol in the library palette causes the same action as **File: Edit selected symbol**, opening the appropriate Symbol Editor for that symbol. See Section 8.4 and Section 8.5 . This has no effect for symbols held in the Standard Symbol Library.

Cursor

The cursor takes standard default patterns in the Symbol Library. However, when a node or arc symbol is selected in the library, the cursor in the Node or Arc Symbol Editor canvas will change to the *cross-hair* pattern. This indicates that selecting any position on the Symbol Editor canvas will add the selected symbol to the Symbol Editor (see Section 8.4 and Section 8.5).

As the cursor is moved over the palette, the name of the symbol under the cursor is shown in the status line.

8.4 Node symbol editor

8.4.1 Appearance and functionality

The Node Symbol Editor allows the diagram type designer to create new node symbols from existing symbols and to modify existing symbols. It is invoked from the **Tools: Show symbol editor** menu of the Control Window, or by right-clicking on a node symbol in a (non-Standard) symbol library. As well as shape, the Node Symbol Editor will support tailoring of node symbol properties.

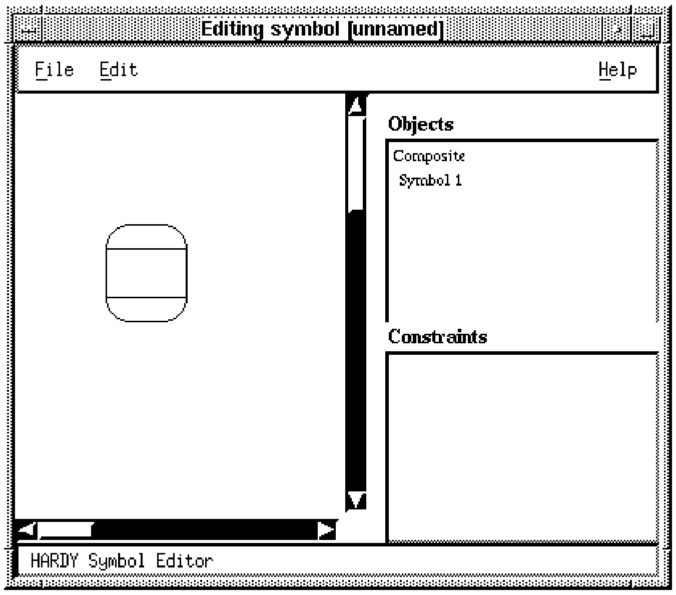


Figure 8.3: The Node Symbol Editor

The title bar shows the name of the symbol under construction, and the canvas displays the symbol currently being edited. The list box labelled *Objects* starts with the entry `Composite`, if a new symbol is being created, or the name of the symbol being edited. This is followed by the names of the different constituent sub-symbols in the same order as they are added to the canvas. Several names may be selected at once. The lower list box, labelled *Constraints*, details the different constraints in the order in which they were invoked.

8.4.2 Menu options

The Node Symbol Editor has three menus – **File, Edit** and **Help**.

File menu

1. **File: Add to library** – add the symbol under construction to the selected symbol Library. If no Symbol Library is selected, the Symbol Librarian is opened so that a selection can be made. The user is asked to supply a name for the symbol through a Text Entry dialog box.
2. **File: Update** – if the Node Symbol Editor was called from the **File: Edit selected symbol** menu of a Symbol Library, the symbol in the Symbol Library is updated and the Node Symbol Editor is dismissed.
3. **File: Exit** – the Node Symbol Editor is dismissed and, if a symbol was being constructed and has not been saved, the user is asked whether it should be saved or not.

Edit menu

1. **Edit: Add constraint** – pops up a Choice dialog box of available constraints from which one can be chosen. A Constraint Properties dialog box then allows the constraint to be tailored. See Section 8.4.4 .
2. **Edit: Edit symbol name** – a Text Entry dialog box is popped up to allow a new name to be specified for the symbol under construction.
3. **Edit: Edit selected object** – a dialog box is popped up which presents information about the current properties of the object selected in the *Objects* list box, and allows them to be changed. See Section 8.1.4 .
4. **Edit: Edit selected constraint** – the Constraint Properties dialog box is opened to allow values of the constraint selected in the *Constraints* list box to be changed. See Section 8.4.4 .
5. **Edit: Delete selected object** – the object selected in the *Objects* list box is removed.
6. **Edit: Delete selected constraint** – the constraint selected in the *Constraints* list box is removed.
7. **Edit: Add control point** – if the object selected in the *Objects* list box is a polyline object, a further control point is added to produce a shape with one vertex more than the previous shape.
8. **Edit: Delete control point** – if the object selected in the *Objects* list box is a polyline object, an arbitrary control point is removed to produce a shape with one vertex fewer than the previous shape. A polyline object cannot have fewer than one control point.
9. **Edit: Make symbol [not] a container** – if the symbol has been imported from a user-defined library (i.e. not the Standard library), it may be given container properties(see Section 9.1.7). If the symbol has container properties set, this menu item is changed to allow them to be unset.
10. **Edit: Deselect all** — all current selections are cleared.
11. **Edit: Refresh** – clears the canvas and redisplays the symbol under construction.

Help menu

1. **Help: Help on node symbol editor** – for Hardy under X, the wxHelp program is started and information on the Node Symbol Editor is displayed. Under Windows, the Windows Help system is started at the appropriate place in the Hardy manual.

8.4.3 Mouse and cursor functionality

Left button

1. Click on item in the *Objects* list box – if the entry is not already selected, it is added to the current selection; if it is already selected, it is now deselected. The composite object under construction may be selected/deselected in this manner. All selected items will be highlighted in the list box and have their selection handles displayed in

the canvas. Unselected items are not highlighted in the list box and do not have their selection handles displayed.

2. Click on a position in the canvas not on a node or arc when a symbol is selected in a Symbol Library (indicated by the *cross-hairs* cursor pattern) – a copy of the selected symbol is dropped onto the canvas at the selected point. The symbol in the Symbol Library is then deselected.
3. Click-and-drag an item in the canvas – the entire composite symbol under construction is moved as required. Note that all the current symbols are moved together. Relative movement is specified by defining suitable constraints.
4. shift-Click on an item in the canvas – if the item is not already selected, it is added to the current selection; if it was already selected it is now deselected. All selected items will be highlighted in the list box and have their selection handles displayed in the canvas. Unselected items are not highlighted in the list box and do not have their selection handles displayed.
5. Click-and-drag a selection handle in the canvas – the corresponding symbol is rescaled in the direction of the dragging operation.
6. Click-and-drag a divided node symbol division control point in the canvas – the division moves to the new position.
7. control-Click-and-drag a polyline control point in the canvas – the point is moved to the new position, altering the shape of the symbol.
8. Click on an entry in the *Constraints* list box – the appropriate constraint with that name is selected, the constrained objects are highlighted by displaying their selection handles in the canvas, and the constraint description is displayed in the status line.

Right button

This provides short-cuts for commonly used menu entries.

1. Click on an item in the canvas – provides the **Edit: Edit selected object** facility (see above).
2. control-Click on an item in the canvas – opens the Divided Object Properties dialog box for a composite symbol. See Section 8.1.6 .

Cursor

1. The *hand* pattern is used within the canvas to indicate that items may be moved.
2. The *cross-hairs* pattern is used when a node symbol is selected in a Symbol Library (see Section 8.3), to indicate that Clicking on a position on the canvas will place a copy of the symbol there. Once the selected symbol has been placed on the canvas, the symbol is deselected within the Symbol Library and the cursor reverts to the normal pattern.

8.4.4 Node symbol constraints

Node symbol constraints are used to specify the relative positioning of sub-symbols within composite symbols under two different circumstances:

1. when constructing new node symbols through the Node Symbol Editor;
2. when describing drop sites for a particular node type (see Section 9.1.6).

A constraint is always applied to at least two objects, one of which is taken as the reference relative to which the others are constrained. Normally the reference object is the first one that was selected selected. When defining drop sites, the reference object is the actual drop site, and the other object(s) will be the node annotation symbol(s) which will not be specified until a particular diagram card is constructed. We refer to this as a *partially satisfied constraint.*

A fixed repertoire of constraints is provided:

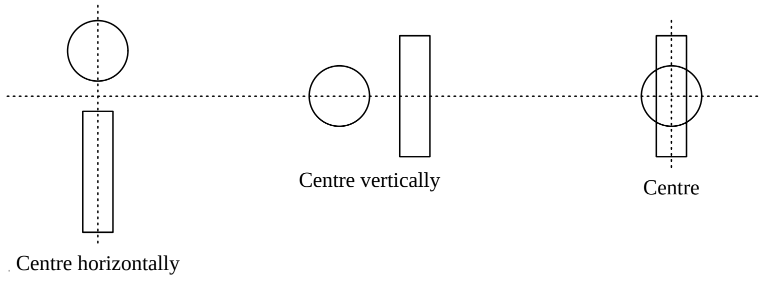
Above: The Y co-ordinates of the bottom horizontal edges of the bounding boxes of the constrained objects will be less than the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object.

Below: The Y co-ordinates of the top horizontal edges of the bounding boxes of the constrained objects will be greater than the X co-ordinate of the bottom horizontal edge of the bounding box of the constraining object.

Left of: The X co-ordinates of the right hand vertical edges of the bounding boxes of the constrained objects will be less than the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object.

Right of: The X co-ordinates of the left hand vertical edges of the bounding boxes of the constrained objects will be greater than the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object.

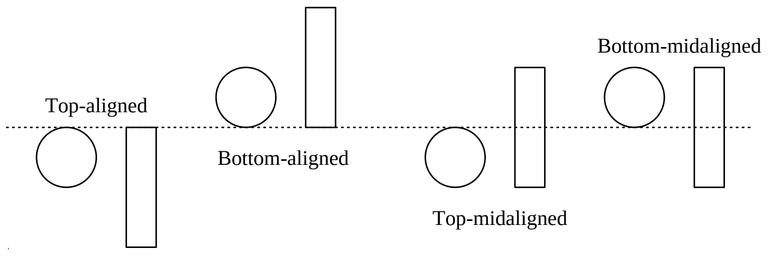
Centre horizontally: The X co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same.



Centre vertically: The Y co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same.

Centre: The co-ordinates of the centres of the bounding boxes of the constrained objects and the constraining object will be the same.

Top-aligned: The Y co-ordinates of the top horizontal edges of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object.

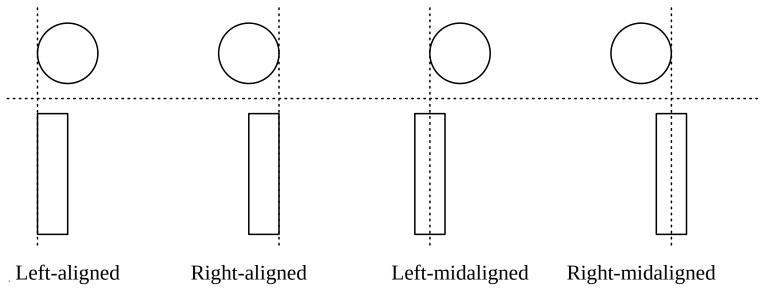


Bottom-aligned: The Y co-ordinates of the bottom horizontal edges of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the bottom horizontal edge of the bounding box of the constraining object.

Top-midaligned: The Y co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the top horizontal edge of the bounding box of the constraining object.

Bottom-midaligned: The Y co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the Y co-ordinate of the bottom horizontal edge of the bounding box of the constraining object.

Left-aligned: The X co-ordinates of the left hand vertical edges of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object.



Right-aligned: The X co-ordinates of the right hand vertical edges of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object.

Left-midaligned: The X co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the left hand vertical edge of the bounding box of the constraining object.

Right-midaligned: The X co-ordinates of the centres of the bounding boxes of the constrained objects will be the same as the X co-ordinate of the right hand vertical edge of the bounding box of the constraining object.

Once the required constraint has been chosen, an offset can be given between the reference points of the constraining and the constrained objects. This is specified as X and Y spacings, in pixels.

8.5 Arc symbol editor

8.5.1 Appearance and functionality

The Arc Symbol Editor allows the diagram type designer to define and modify the appearance of arc symbols held in Symbol Libraries. It is invoked from the **Tools: Show symbol editor** menu of the Control Window, or by right-clicking on an arc symbol in a (non-Standard) symbol library.

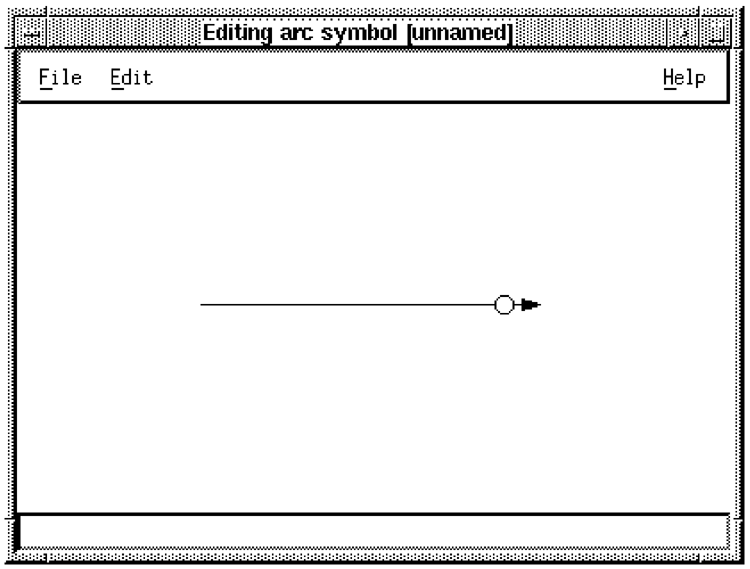


Figure 8.4: The Arc Symbol Editor

There are three menus – **File, Edit** and **Help**. The canvas displays the current properties of the arc image. It is initialised to a solid black line.

8.5.2 Menu options

File menu

1. **Edit: Add to library** – the arc symbol under construction is added to the currently selected Symbol Library. If no library is selected, a warning message is shown and the Symbol Library Manager is opened, if it is not already open, so that a selection can be made.

2. **File: Update** – if the Arc Symbol Editor was called from the **File: Edit selected symbol** menu of a Symbol Library, the symbol in the Symbol Library is updated and the Arc Symbol Editor is dismissed.
3. **File: Exit** – the Arc Symbol Editor is dismissed and, if a symbol was being constructed and has not been saved, the user is asked whether it should be saved or not.

Edit menu

1. **Edit: Edit symbol name** – a Text Entry dialog box is popped up to allow a new name to be specified for the symbol under construction.
2. **Edit: Edit arc properties** – a dialog box is popped up which presents information about the current properties of the selected arc symbol and allows them to be changed. See Section 8.1.2 .
3. **Edit: Edit annotation properties** – a dialog box is popped up which presents information about the current properties of the arc annotations and allows them to be changed. See Section 8.1.3 .
4. **Edit: Clear symbols** – any arc annotations that have been specified are cleared from the displayed arc symbol.
5. **Edit: Refresh** – clears the canvas and redisplays the symbol under construction.

Help menu

1. **Help: Help on Arc Symbol Editor** – for Hardy under X, the wxHelp program is started and information on the Arc Symbol Editor is displayed. Under Windows, the Windows Help system is started at the appropriate place in the Hardy manual.

8.5.3 Mouse and cursor functionality

Left button

1. Clicking on the canvas when an arc annotation symbol is selected in a Symbol Library will place that symbol at the right hand end of the displayed arc symbol. Arbitrary, multiple annotations may be constructed. When an annotation has been added to the current symbol, it is deselected in its Symbol Library.

Right button

No use is made of the right mouse button.

Cursor

1. The *hand* cursor is the default within the canvas.
2. The *cross-hairs* cursor within the canvas indicates that an arc annotation symbol is currently selected in a Symbol Library.

9 Card types

This section is concerned with setting up customised card types. This is done by the Card Type Designer, and we will reserve the term *user* throughout this section to refer to a person who makes use of a type of card that the Card Type Designer has defined. A user doesn't want to see the diagram type definition, and will be restricted in node and arc repertoire, node shape, etc. by the decisions you, the card type designer, have made in the card type definition.

Diagram and hypertext type definitions are held in files, usually with the file extension of `.def`, and it is necessary for the definition to be loaded before a diagram of that type can be created or edited (obviously!). When Hardy starts up, it looks for a definition list file (by default called `diagrams.def`) which contains a list of these filenames, each file containing a card type definition (see Section 2.5). The consequence to the user of adding a new type will be a new menu item in the dialogue requesting a card type selection when the user comes to create a new card (i.e. an instance of that card type).

Standard types of text card and hypertext card are provided. New types of diagram card (see Section 9.1) and hypertext card (see Section 9.2) can be created.

The different types of card available can be organised by the card type designer into *categories* if required. This allows different types which are in some way related to be presented as a group, rather than mixed up with other types.

9.1 Diagram card types

9.1.1 New diagram types

You create new diagram types or modify existing ones, through the Hardy Diagram Type Manager which is accessed from **Tools: Show diagram type manager** on the Hardy Control Window.

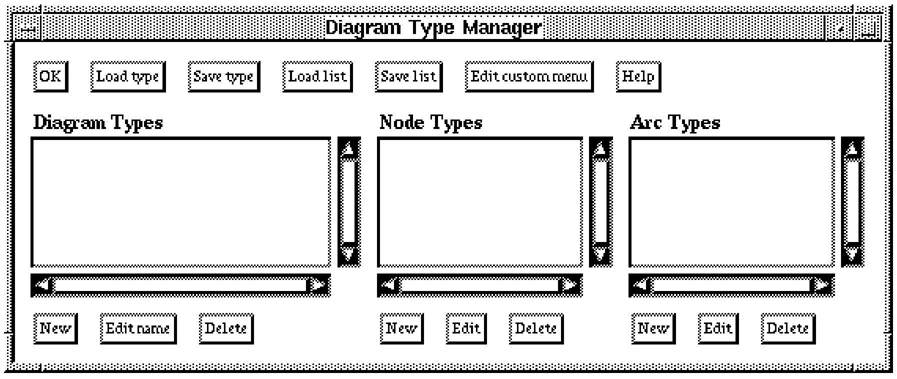


Figure 9.1: The Hardy Diagram Type Manager

You'll be presented with three lists. The first allows you to select a diagram type definition to work with, the second shows you all the node definitions for the selected diagram type, and the third shows you its arc definitions. As this implies, a diagram type definition consist of the diagram type name, a list of node definitions, and a list of arc definitions.

The operation of the Diagram Type Manager is controlled by various buttons. To create a new diagram type, press the **New** button underneath the *Diagram types* list, and you'll be asked to supply a new name for the card type. (You may supply a category for it as well, though this isn't necessary.) To edit a diagram type, select the diagram type name and press the **Edit** button.

Buttons

The buttons are organised into four groups: one for general operations, and one each for the three lists.

General

1. **OK** – the Diagram Type Manager is dismissed, and, if changes have been made to any diagram type definitions, you are asked whether these are to be saved or not.
2. **Load type** – the File Selector dialog box appears with the filter string set to `*.def`. You can select a particular diagram type definition file which will be opened, when its node and arc types will be listed in the *Node types* and *Arc types* list boxes.
3. **Save type** – the File Selector dialog box appears, and you can save the particular diagram type definition file.
4. **Load list** – the File Selector dialog box appears to allow you to specify the name of the diagram definition list file. By default, this is `diagrams.def`.
5. **Save list** – the File Selector dialog box appears with the filter initialised to `*.def` to allow a name for the definition file to be selected.
6. **Custom menu** – a dialog box appears, allowing you to specify, for the selected diagram type, the title of a custom menu, and to add or delete menu items from this custom menu. See Section 9.3 .
7. **Options** – a dialog box appears, allowing you to specify, for the selected diagram type, various properties of the diagram type that affect the appearance of a card of that type. These properties include whether the palette is displayed, the toolbar is displayed, and which menus are present.
8. **Help** – for X versions of Hardy, the wxHelp program is started and the Hardy manual is loaded and opened at the section concerning the Diagram Type Manager. For Hardy for Windows, the Windows Help system is started at the appropriate place in Hardy's manual.

Diagram types

1. **New** – a Diagram Type dialog box appears, allowing you to specify the new diagram type required in the text entry area, labelled *Type*. Types may be grouped together by

defining *categories* and then assigning each type to a particular category. If required, a category can be selected from the list of existing categories shown in the *Categories* choice box, or a new category name can be typed into the *Category* text input area. When finished, you should press the **OK** button to accept your specification, when your new name will appear in the *Diagram types* list, and Hardy will select it. Otherwise, you can abandon the operation by pressing **Cancel**.

2. **Edit name** – a Text Entry dialog box appears, allowing you to edit the name of the selected diagram type. The new name replaces the previous name in the existing entry in the *Diagram types* list.
3. **Delete** – the selected diagram type is deleted, and the topmost diagram type in the list becomes selected.

Node types

1. **New** – a Text Entry dialog box appears, allowing you to type in the name of the new node type. An entry with that name then appears in the *Node types* list and the newly created entry is selected. The Node Type Editor then appears (see Section 9.1.2), allowing you to edit the properties of the new node type (see Section 9.1.3).
2. **Edit** – the Node Type Editor appears, showing the information for the selected node type (see Section 9.1.2).
3. **Delete** – the selected node type is deleted, and the topmost entry in the *Node types* list becomes selected.

Arc types

1. **New** – a Text Entry dialog box appears, allowing you to specify the name of the new arc type. An entry with that name then appears in the *Arc types* list and the newly created entry is selected. The Arc Type Editor then appears(see Section 9.1.9), allowing you to edit the properties of the new arc type.
2. **Edit** – the Arc Type Editor appears, showing the information for the selected arc type (see Section 9.1.9).
3. **Delete** – the selected arc type is deleted and the topmost entry in the *Arc types* list becomes selected.

Mouse and cursor functionality

Left button

1. Clicking on an item in the *Diagram types* list selects that diagram type, and its node and arc types are displayed in the *Node types* and *Arc types* lists.
2. Clicking on an item in either the *Node types* or *Arc types* lists selects that item. Any previous selection in the list will be deselected.
3. Double-click an item in either the *Node types* or *Arc types* lists selects that item and proceeds as though the corresponding **Edit** button had been pressed. Any previous selection in the list will be deselected.

Right button
No use is made of the right mouse button.

Cursor
No special cursor pattern is used.

9.1.2 Node type editor

The Node Type Editor allows you to tailor the properties of a node type. This includes the displayed symbol shape, scale, colour, etc, as well as the user defined attributes. The Node Type Editor is invoked by selecting an entry in the *Node types* list of the Diagram Type Manager.

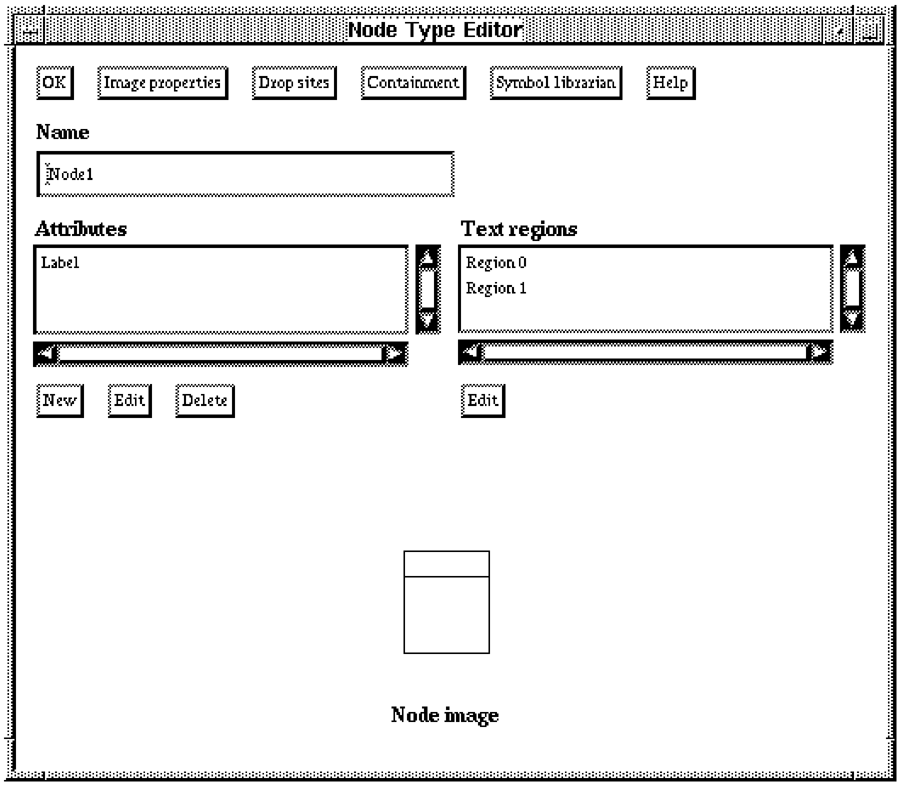


Figure 9.2: The Node Type Editor

A text entry area, labelled *Name*, allows the node type to be named. One list box, labelled *Attributes*, allows attributes of the node to be defined and edited. The other list, labelled *Text regions*, allows individual text regions to be selected so that the format and contents of the text string displayed in the region may be modified.

The preview canvas displays the current appearance of the node and is *not* directly editable. You can load a symbol into the editor either by selecting the symbol you require from a symbol library (see Section 8.3) and then clicking on the preview canvas. Alternatively, to modify an existing symbol, you can right-click on the symbol in a library (*not* the standard symbol library) to open the editor with the symbol loaded. Changing the properties of the node will change its appearance in the preview canvas.

Buttons

Buttons are organised into three groups: one for general operations and one each for the *Attributes* and *Text regions* lists.

General

1. **OK** – the Node Type Editor is dismissed and, if changes have been made to node definitions, you will be asked whether these should be saved or not.
2. **Image properties** – pops up a dialog box which allows the properties of the displayed image to be altered (see Section 9.1.3).
3. **Drop sites** – if the symbol in the preview canvas has partially satisfied constraints (see Section 8.4.4), the Drop Sites dialog box appears (see Section 9.1.4).
4. **Containment** – the Containment dialog box is opened, see Section 4.12 ,allowing the node to become a container.
5. **Symbol librarian** – invokes the Symbol Librarian (see Section 8.2).
6. **Help** – for X versions of Hardy, the wxHelp program is started and the Hardy manual is loaded and opened at the section concerning the Node Type Editor. For Hardy for Windows, the Windows Help system is started at the appropriate place in Hardy's manual.

Attributes

1. **New** – a Text Entry dialog box appears allowing a new attribute name to be specified.
2. **Edit** – a Text Entry dialog box appears, allowing you to edit the name of the selected attribute. The new name replaces the previous name in the existing entry in the *Attributes* list.
3. **Delete** – deletes the currently selected attribute.

Text regions

1. **Edit** – if an entry is selected in the *Text regions* area, the Region Properties dialog box will be opened, allowing the properties of the selected text region to be tailored. See Section 9.1.8 .

Mouse and cursor functionality

Left button

1. Click on any point in the preview canvas to replace the symbol currently being displayed with the node symbol currently selected in a Symbol Library. The Symbol Library symbol will then be deselected. If no Symbol Library has a currently selected node symbol, no action results.
2. Click-and-drag a symbol in the preview canvas to change the displayed position of the symbol.
3. Click on an item within the *Attributes* list box selects that item.
4. Double-click on an item within the *Attributes* list box selects that item and proceeds as though the **Edit** button had been pressed.
5. Click on an item within the *Text regions*list box selects that item. The name of the selected region is shown in the preview canvas.
6. Double-click on an item within the *Text regions* list box selects that item and proceeds as though the **Edit** button had been pressed.

Right button
No use is made of the right mouse button.

Cursor

1. The *hand* pattern is the default cursor in the preview canvas.
2. The *cross-hairs* pattern in the preview canvas indicates that a symbol is currently selected in a Symbol Library.

9.1.3 Node images

The node images specified for a particular diagram type are based on symbols held in some symbol library(see Section 8.3), but tailored specially for that diagram type through the Node Image Properties dialog box that is invoked by pressing the **Image properties** button of the Node Type Editor.

In addition to the basic properties that it already has, see Section 8.1.4 , when a node symbol is used in a diagram type it has additional features:

1. Abbreviation format string – determines how abbreviated references to the node (e.g. in the status line) should be written. See Section 9.1.8 .
2. Width – the default width of the node in pixels when it is first placed on to a diagram card. If the *fixed width* property is also set(see Section 8.1.4), the width will always be this value.

3. Height – the default height of the node in pixels when it is first placed on to a diagram card. If the *fixed height* property is also set(see Section 8.1.4), the width will always be this value.
4. whether or not attachments are used– see Section 8.1.5 .
5. whether or not connected arcs are equally spaced at attachment points – see Section 8.1.5 .
6. whether or not the Node Image should be highlighted if it is hyperlinked to another Item.
7. whether or not the symbol should be displayed on the Symbol Palette – this will prevent the user from creating particular symbols which you may need to control.

9.1.4 Drop sites

Drop sites may be defined for node symbols by establishing partially satisfied constraints(see Section 8.4.4) which will be fully satisfied by node annotation symbols (see Section 9.1.6) which can be "dropped" onto the node symbol when a particular diagram card is being constructed.

Drop sites are established or modified for a node symbol through the Drop Sites dialog box which is invoked from the **Drop sites** button of the Node Type Editor. This will show you the names of all existing drop sites for that node type in the *Drop sites* list.

To add a new drop site, press the **New** button to open the Drop Site Editor(see Section 9.1.5). Its name will be added to the list on exit. To modify an existing drop site, select its entry in the list, press the **Edit** button, and the Drop Site Editor will again be opened. If you want to throw a defined drop site away, select it in the list and press **Delete**.

9.1.5 Drop site editor

The Drop Site Editor allows you to name a drop site and its associated symbol, and to define its constraints. It is invoked from the **New** and **Edit** buttons of the Drop Sites dialog box (see Section 9.1.4).

The drop site name is specified in the *Drop site name* text entry area. You specify the node annotation symbol associated with the drop site by selecting a node symbol in a symbol library, then pressing the **Assign new symbol** button. The name of the symbol will be shown in the label of the *Annotation name* area. This is a logical name that you can specify by typing it in.

The range of partially satisfied constraints, previously defined for the node symbol through the Node Symbol Editor(see Section 8.4), is displayed in the *Available constraints* list box. You build up the particular combination of constraints that you want, shown in the *Drop sites constraints* list box. by using the **Add** –> and **Delete** buttons. The **Add** –> button adds the currently selected entry in the *Available constraints* list to the end of the*Drop sites constraints* list. The **Delete** button deletes the currently selected entry from the *Drop sites constraints* list.

The **Edit symbol properties** button will open the Node Annotation Symbol Properties dialog box to allow you to tailor the properties of the annotation symbol (see Section 9.1.6

).

9.1.6 Node annotation symbols

A node annotation is a node symbol which is associated with a particular node type. It may be "dropped" onto a node at a defined *drop site* when a particular diagram is being built. Arcs may be connected to this symbol, and it can have attachment points in the same way as to any other node symbol.

The Node Annotation Symbol Properties dialog box allows you to tailor a node annotation symbol for a particular node type. It is invoked from the **Edit symbol properties** button of the Drop Site Editor.

As well as the usual properties governing the appearance of the symbol (see Section 8.1.4), you can specify whether or not attachments are used, and whether or not arcs to attachment points should be equally spaced.

9.1.7 Containment

Containers are nodes which can *contain* other nodes of specified types (see Section 4.12). The Containment dialog box allows you to set the container properties for a particular node type. It is invoked from the **Containment** button of the Node Type Editor Editor.

The box labelled *Available nodes* allows you to choose from the recognised node types for the diagram or the "*" wild card, signifying any type. Once you have selected a type you want, press the **Add** button and it will be added to the end of the *Containable nodes* list which displays the different types of node that may be contained in this type of container node. If you want to change your mind and delete an entry, select it in the *Containable nodes* list, press **Delete**, and it will disappear from the list.

9.1.8 Regions

Labels are displayed in text regions, one label per region. Each label needs two parts: the text string to be displayed, and formatting information to determine how it looks. The text of an item's label is usually held as one of the item's attributes (see Section 4.6). The formatting information for the region is stored and altered through the Region Properties dialog box which is invoked by selecting the region from the *Text regions* list of the Node Type Editor and then pressing its **Edit** button.

The text entry area labelled *Format string* allows the text region format string to be specified (see below). The text entry area labelled *Point size* accepts an integer specifying the text size. Choice boxes labelled *Format mode*, *Text colour*, *Font family*, *Font style*, and *Font weight* allow choices to be made from the range of values supported by Hardy.

The format string

The format string is a simple way of specifying the label for a region of a node or arc. In particular, it allows you to display the values of an item's attributes in its image.

The format string may contain literal text and control characters (introduced by a "%" character) as follows:

1. `%%` – inserts the "%" character,
2. `%n` – inserts a new line,
3. `%1 - %9` – inserts a node or arc attribute, where the number corresponds to the position of the attribute in the attribute list as displayed in the node or arc type dialogue.

By convention, the first attribute is used to hold an item's label, so the default format string is `%1`. A more complex example might be `Name: %1%nValue: %2`. The `%n` control character may also be inserted in arc and node attributes, for example to prevent overlapping by adding some manual formatting.

9.1.9 Arc type editor

The Arc Type Editor allows you to tailor the properties of an arc type. This includes the displayed line shape, permitted annotations, scale, colour, etc, as well as user defined attributes. The Arc Type Editor is invoked through the **New** and **Edit** buttons of the *Arc types* scrolling list of the Diagram Type Manager.

Arc attributes and text regions (labels) are treated in the same way as for nodes in the Node Type Editor(see Section 9.1.2), with an arc always having three regions defined: *Start*, *Middle*, and *End*. These are displayed in the *Text regions* list. Two additional lists, *Arc constraints* and *Arc images*, allow arc constraints and arc images to be specified and selected.

The displayed image of the arc in the preview canvas, labelled *Arc image* followed by its name, will reflect its current image properties and the current choice of annotations. It is not directly editable, but changing the properties of the arc will change its appearance in the preview canvas.

If a Junction Symbol has been selected through the Junction Editor (see Section 9.1.13), it will be displayed separately in the preview canvas and labelled *Junction image*.

Buttons

Buttons are arranged in five groups, a general group for the editor and one each for the *Attributes*, *Text regions*, *Arc constraints*, and *Arc images* list boxes.

General

1. **OK** – the Arc Type Editor is dismissed, and, if changes have been made to arc definitions and these have not already been saved, you will be asked whether they are to be saved or not.
2. **Junction editor** – pops up the Junction Editor dialog box, see Section 9.1.13 , to allow a junction symbol to be chosen and given appropriate properties. If a Junction Symbol is chosen, it will be displayed in the preview canvas.

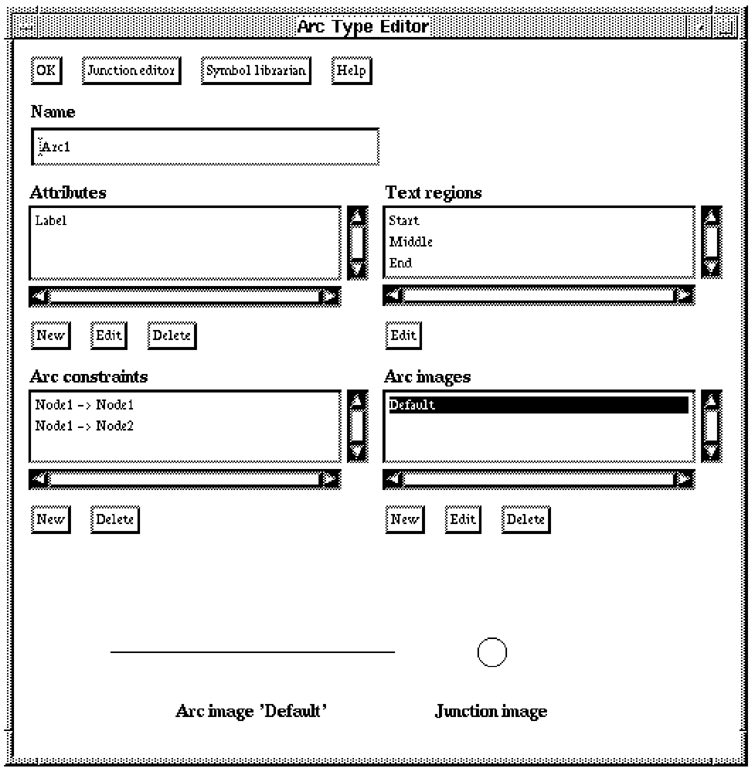


Figure 9.3: The Arc Type Editor

3. **Symbol librarian** – opens the Symbol Librarian.
4. **Help** – for X versions of Hardy, the wxHelp program is started and the Hardy manual is loaded and opened at the section concerning the Arc Type Editor. For Hardy for Windows, the Windows Help system is started at the appropriate place in Hardy's manual.

Attributes

1. **New** – a Text Entry dialog box appears, allowing a new attribute to be specified.
2. **Edit** – a Text Entry dialog box appears, allowing you to edit the name of the selected attribute. The new name replaces the previous name in the existing entry in the *Attributes* scrolling list.
3. **Delete** – deletes the currently selected attribute.

Text regions

1. **Edit** – if an entry is selected in the *Text regions* area, the Region Properties dialog box will be opened, allowing the properties of the selected text region to be tailored.

See Section 9.1.8 .

Arc constraints

1. **New** – an Arc Constraint dialog box appears allowing a new constraint to be specified (see Section 9.1.10). The choice boxes in this dialog box contain all the nodes types defined for this diagram type.
2. **Delete** – deletes the currently selected constraint.

Arc images

1. **New** – a Text Entry dialog box appears, allowing a name to be given for the new image. When a name has been specified, the Arc Image Properties dialog box appears (see Section 9.1.11).
2. **Edit** – the Arc Image Properties dialog box appears (see Section 9.1.11), allowing you to edit the properties of the arc symbol for the currently selected arc type.
3. **Delete** – deletes the currently selected arc symbol.

Mouse and cursor functionality

Left button

1. Click on a point in the preview canvas, if a node annotation symbol is currently selected in any Symbol Library, places that symbol on the nearest text region of the arc symbol currently being displayed. Any annotation symbol already at that site will be replaced. The Symbol Library symbol will then be deselected. If no Symbol Library has a currently selected arc annotation symbol, no action results.
2. Click on an item within the *Attributes* list box selects that item.
3. Double-click on an item within the *Attributes* list box selects that item and proceeds as though the **Edit** button had been pressed.
4. Click on an item within the *Text regions* list box selects thatitem. The name of the selected region is shown in the preview canvas.
5. Double-click on an item within the *Text regions* list box selects that item and proceeds as though the **Edit** button had been pressed.
6. Click on an item within the *Arc constraints* list box selects that item.
7. Click on an item within the *Arc images* list box selects that item.
8. Double-click on an item within the *Arc images* list box selects that item and proceeds as though the **Edit** button had been pressed.

Right button

No use is made of the right mouse button.

Cursor
No special cursor pattern is used.

9.1.10 Arc constraints

The Arc Constraints dialog box allows you to specify between which types of node the arc is legal. It is invoked from the **New** button of the *Arc constraints* area of the Arc Type Editor.

The two boxes allow you to choose from the types of node that are defined for the diagram type or the "*"wild card, signifying any type. They are labelled *Constrain from* and *Constrain to*, in the expected manner. When you have chosen a type for each end, press the **OK** button to dismiss the dialog box.

9.1.11 Arc images

The Arc Image Properties dialog box allows you to tailor the image properties of an arc type symbol, such as its line width, style, colour, etc. The Arc Image Properties dialog box is invoked by selecting an entry in the *Arc images* scrolling list in the Arc Type Editor and pressing the **Edit** button. This dialog box will also appear when a new Arc Image is being defined, after specifying the Arc Image's name.

In addition to the basic properties that it already has, see Section 8.1.2 , when an arc symbol is used in a diagram type, it has additional features:

1. Abbreviation format string,
2. whether the arc is drawn as a straight line or a spline curve and, if it is a straight line, whether it should be drawn as a spline if it connects one node to the same node.
3. whether annotations are *divisible* or not, i.e. presented in a separate section on the diagram symbol palette.
4. whether or not the symbol should be displayed on the Symbol Palette.

Pressing the **Annotation properties...** button opens the Arc Type Annotation Properties dialog box, see Section 9.1.12 ,allowing high-level properties of any arc annotations to be set.

9.1.12 Arc type annotations

The Arc Type Annotation Properties dialog box is invoked from the **Annotation properties...** button of the Arc Image Properties dialog box. It allows allows properties of arc annotations to be set for a particular type of arc.

Each entry corresponds to an annotation specified for the region, and consists of:

1. its symbol name,
2. a checkbox indicating whether the annotation will always be present on the arc or whether it will be added incrementally by the user as desired, and
3. a text entry area allowing a logical name to be given to the annotation.

9.1.13 Multi-way arcs and junction symbols

Multi-way arcs allow you to connect one node to several others with the same arc type and have the diagram display the result as a single leg emerging from the source node, joining it to a multi-way junction symbol. The destination nodes are then all joined to this junction symbol. This can result in tidier diagrams, particularly if some grid constraints are applied to the arc segments (see Figure 9.4, below).

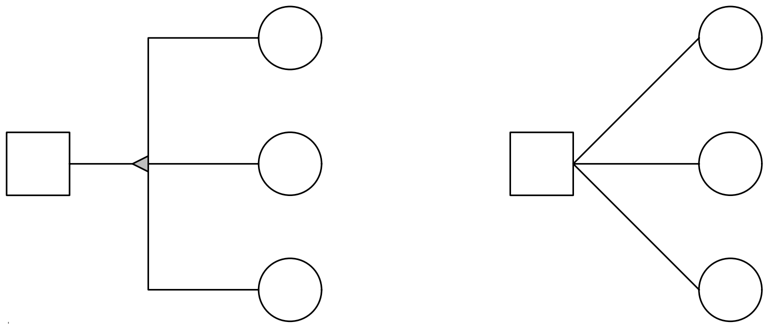


Figure 9.4: Using a multi-way arc Normal one-to-one arcs

A junction symbol is a node symbol which has been specially selected for the role using the Junction Editor dialog box, invoked from the **Junction editor** button of the Arc Type Editor.

This allows you to specify several important properties:

1. selecting the junction symbol – there is a message at the foot of the dialog box, asking you to select the node symbol that you want from some symbol library. (You do this in the usual way, opening the Symbol Librarian if necessary from the Arc Type Editor or the Control Window's **Tools** menu.) Once the symbol is selected, you will see the *cross-hairs* cursor pattern when you move into the preview window of the Arc Type Editor. Clicking here will "drop" the selected symbol onto the preview window and the label *Junction symbol* followed by the name of the symbol will be shown below it.
2. changing the junction symbol – press the **Clear Junction** button, and the current junction symbol image displayed in the Arc Type Editor preview window is cleared.
3. size – you can specify the height and width of the displayed symbol, in pixels.
4. two-way arcs – you can decide whether or not the junction symbol should appear when the source node is connected to a single destination node.
5. grid geometry – if *auto dog-leg* is set, an extra control point will be added to each leg connecting the junction symbol to the destination nodes. Hardy will then alter these arcs so that they are horizontally and vertically aligned. (as though **Layout: Straighten lines** had been selected.)
6. specifying attachment points – this uses the numbered identifiers of the junction symbol's attachment points to specify finer detail for tidier diagrams:

   Input: the attachment point id used for connecting the input (source) node,

One output: the attachment point id used for connecting the output node when there is only one (a two-way arc),

Two outputs (1): the attachment point id to be used for connecting the first of more than one output arcs (a multi-way arc),

Two outputs (2): the attachment point id to be used for connecting the second of more than one output arcs,

Three outputs: the attachment point id to be used for connecting the third and subsequent output arcs.

Pressing the **OK** button accepts the current values and dismisses the dialog box.

9.2 Hypertext card types

As for diagrams, it is possible to create *hypertext card types* which modify the default capabilities of the standard hypertext card. Differences between one type and another all relate to alternative styles for *marking-up* text blocks and the use of custom menus. The text is marked-up by selecting blocks of text—phrases, sentences, paragraphs, etc—and associating *block types* with them. Each block type is mapped onto a particular *style* which allows the block to be distinguished from the surrounding text in terms of its font, colour, etc. To understand the process of building a new card type, please read the chapter on creating a new diagram type first (Section 4.1).

9.2.1 New hypertext types

The Hardy Hypertext Type Manager, accessed from the **Tools: Show hypertext type manager** menu item of the Hardy Control Window, allows the default capabilities of the standard hypertext card to be modified so that the user can create different hypertext card types.

The Hypertext Type Manager presents you with a list of currently defined hypertext types and a selection of buttons, many of which are identical to those of the Diagram Type Manager (see Section 9.1.1).

To create a new type, press the **New** button, and enter a name for the hypertext type. To alter the name of an existing type, select the one you want from the *Hypertext types* list and press the **Edit name** button and you'll be asked for the new name. You can also change its category(see Section 9.1.1) if you want to, though this isn't necessary. You can delete a type by selecting the entry you want to remove from the list, then pressing the **Delete** button.

The actual work of defining the hypertext type is done through altering hypertext block mappings. You open the Block Mappings dialog box by pressing the **Edit block mappings** button(see Section 9.2.2).

Buttons

The buttons are arranged in two groups, one group providing general facilities, the other dealing with facilities for the type currently selected in the *Hypertext types* list box.

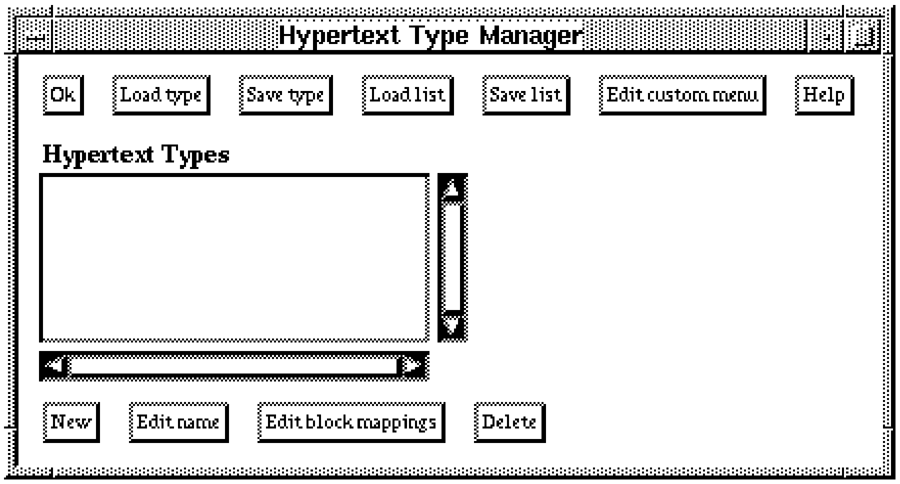


Figure 9.5: The Hardy hypertext type manager

General

1. **OK** – the Hypertext Type Manager is dismissed and, if changes have been made to any hypertext type definitions and these have not been saved, the user is asked whether these should be saved or not.
2. **Load type** – the File Selector dialog box appears with the filter string set to `*.def`. The user can select the hypertext type definition file to open, and this is then loaded and displayed in the *Hypertext types* list box.
3. **Save type** – the File Selector dialog box appears, and the user can specify a file name for saving the hypertext type definition file.
4. **Load list** – the File Selector dialog box appears to allow the user to specify the name of the hypertext definition list file. By default, this is `diagrams.def`.
5. **Save list** – the File Selector dialog box appears with the filter initialised to `*.def` to allow a name for the definition file to be selected.
6. **Custom menu** – if a hypertext type is currently selected in the *Hypertext types* list box, a dialog box appears, allowing the user to specify for that hypertext type the title of a custom menu, and to add or delete menu items from this custom menu. See Section 9.3 .
7. **Help** – for X versions of Hardy, the wxHelp program is started and the Hardy manual is loaded and opened at the section concerning the Hypertext Type Manager. For Hardy for Windows, the Windows Help system is started at the appropriate place in Hardy's manual.

Hypertext types

1. **New** – a Text Entry dialog box appears, allowing the user to type in the name of a new hypertext type. An entry with that name then appears in the *Hypertext types* scrolling list, and the newly created hypertext type is selected.
2. **Edit name** – a Text Entry dialog box appears, allowing the user to change the name of the selected hypertext type. An entry of that name then appears in the *Hypertext types* scrolling list, replacing the previous entry.
3. **Edit block mappings** – if a hypertext type is selected in the *Hypertext types* scrolling list, a Block Mapping dialog box appears (see Section 9.2.2 below), allowing the user to alter the mappings between block types and their displayed styles.
4. **Delete** – the selected hypertext type is deleted from the hypertext type definition list and the corresponding entry is removed from the *Hypertext types* list box.

9.2.2 Hypertext block mappings

The block type mapping defines how the type identifier of a text block is interpreted in terms of text colour and style. Using different block type mappings on the same hypertext files results in the same text being displayed differently. When a hypertext type is first created, the default block styles are given. You may wish to modify these for your own application. Note that the block type identifier must be unique for each block type mapping.

A block type can have the following characteristics:

font family (Swiss, Roman, Modern or Default),

point size (such as 10, 12, 24),

style (Normal, Italic or Default),

weight (Normal, Bold, Light or Default),

colour (such as BLACK, RED, FOREST GREEN, BLUE, CYAN or Default).

A block may, in fact, have some of the attributes specified as *Default*, which means "use whatever was specified before this block". The point size value representing the default is -1. When a block ends, the attributes revert to their previous values. This means that a block need only change one or two attributes, such as colour or weight. Blocks may be nested: for example, within an Italic block, a word could be highlighted in RED. If the RED block type specified "Default" for text style, the block would be RED *and* Italic.

An additional characteristic is whether or not it supports *sections*. If sections are supported by a particular hypertext type, cards of this type will be able to move backwards and forwards using the **Goto: Top**, **Goto: Next section**, and **Goto: Previous section** menu options.

The Block Mapping dialog box allows the user to alter the mappings between hypertext block types and their displayed styles. It is invoked from the **Edit block mappings** button of the Hypertext Type Manager,see Section 9.2.1 .

The names of the different block styles are listed in the *Blocks* area. The type id corresponding to the currently selected block is shown in the *Block type* area. Details of the text font and other characteristics are shown and may be altered in the other areas.

To add a new block type name, press the **Add** button. A Text Entry dialog box will appear, allowing you to type in the name required. An entry with that name will appear in the *Blocks* list and that entry will be selected. You can alter the name of a type by selecting it in the *Blocks* list, then pressing the **Change name** button. The Text Entry dialog box will appear, allowing you to edit the selected name. The new name will replace the old name in the existing entry in the list. To delete a type, select the one you want to remove from the *Blocks* list, press **Delete**, and the entry will disappear from the list.

9.3 Custom menus

Diagram types and hypertext types may have a extra menu added to the card's menu bar. This supports an interface to foreign code which intercepts particular mouse and menu events for the card type.

You specify your menu requirements using the Custom Menu dialog box which is invoked by pressing the **Custom menu** button of either the Diagram Type Manager or the Hypertext Type Manager. This lets you specify the menu name and add or delete menu entries.

The menu title is specified in the area labelled *Menu title* and the selectable names of the menu entries are shown in the *Menu items* list. To add a new entry, type its name into the *Menu item name* area and press the **Add** button. What you typed will now be added to the end of the *Menu items* list and *Menu item name* will be cleared. To change the name of an entry, select the name you want to change in the *Menu items* list, type the new name into *Menu item name*, and press the **Save name** button. What you typed will now replace the selected entry in the list. To remove an entry, select the name you want removed from the *Menu items* list and press the **Delete** button. The selected entry will be removed from the list.

Pressing **OK** commits to the current settings and dismisses the dialog box.

9.4 Saving card type definitions

Once you have finished defining or changing a card type, go to the type manager (the Diagram Type Manager or the Hypertext Type Manager as appropriate) and press the **Save type** button to save the currently selected type. You will be prompted for file names if any files are needed that you haven't referred to before.

To update the definitions list file, use the **Save list** button of the type manager.

9.5 Editing previously created card type definitions

Assuming the type definition is available from the type manager (the Diagram Type Manager or the Hypertext Type Manager as appropriate), select it in the *Diagram types* list (by clicking on its name), then press the **Edit name** button to change its name, or the **Delete** button to delete the whole type completely from the type index. (The file will remain on the disk.)

To edit an existing node or arc type in the Diagram Type Manager, make selections in the same way as above, but using the *Node types* or *Arc types* lists instead.

NOTE that it can be dangerous to change diagram type settings when a diagram of that type is being edited, and it is possible to get Hardy to crash this way. For many settings, however, old diagram files will continue to work correctly using the new definitions. However, this should be done with caution.

The **Layout: Apply definitions** menu option of a diagram card does let you update a displayed card if you have, in the meantime, changed its Diagram Type definition; existing values of various properties of the displayed items will be updated where possible.

10 Differences between the X and Windows versions

Since the X and Windows windowing systems have differences in philosophy and design, some specific features are necessary for each platform. These have been kept to a minimum to avoid inconsistency.

10.1 Printing

Under X, the only form of printing is to PostScript files or printers. Hardy for Windows provides the same options, but will also support copying diagrams to the Windows clipboard though the **Edit: Copy** option.

10.2 The clipboard

Hardy for Windows has an option in the diagram card **Edit: Copy** menu for copying a metafile version of the diagram to the clipboard. See Section 4.13.2 .

10.3 MDI mode

Under Windows, MDI (Multiple Document Interface) is a style used by most modern applications where child windows are constrained by the top level window. For applications which allow many documents to be open at once, it is more convenient to hide all windows when the top level window is iconised than to have to close many windows individually. The application effectively has its own desktop, on which document windows may be placed and iconised. The menu bar for each window is always placed on the top level window, and changes according to which child window is currently activated.

In addition, MDI applications have an extra menu called **Window**, with standard **Cascade**, **Tile Arrange icons** and **Next** options, and a list of MDI child windows which can be activated.

Hardy supports MDI mode under Windows (in fact it's the default) using the `-mdi` command line switch. The switch `-sdi` selects SDI (Single Document Interface) mode, for those who dislike the MDI style. Hardy is probably unique amongst Windows applications to offer this choice! In MDI, the hypertext tree browser is displayed in a separate child window; this is now synonymous with the control window, taking on the top level window menu bar (main menu). You can get to the main menu by activating the browser window or using the **Goto control window** menu item from a card's **File** menu.

10.4 Text editing

Plain text cards cannot be edited directly under Windows. A separate text editor must be invoked, and the file read back into the card explicitly.

11 Programming Hardy

Hardy has a built-in language based on NASA's CLIPS 6.0. For detailed information on CLIPS, please refer to the CLIPS user and reference manuals. This chapter describes the simple CLIPS development environment included with Hardy, and lists the Hardy-specific CLIPS functions; the function reference may be viewed on-line from the CLIPS help menu by selecting.

After the Hardy specific functions, a separate section lists CLIPS functions relevant to lower-level construction and manipulation of windows and other interface components. This reference is also accessible from the CLIPS help menu as *Interface functions reference.*

11.1 The Hardy CLIPS environment

Hardy comes with a cut-down version of the embeddable expert system shell, CLIPS. There are new Hardy-specific functions which may be called from CLIPS, allowing Hardy to be tailored to a greater degree than by using the diagram type manager alone.

The **Tools** menu on the control window has a **Show Development window** option. Selecting this displays the Hardy CLIPS development window consisting of a menu bar, a command prompt, and a text output window. Some CLIPS operations may be achieved using the menu such as loading a CLIPS definition file, and all may be accessed from the command prompt. To execute an arbitrary CLIPS command, type in the command and press the **Do** button. The command is echoed on the text output window, and any results of the executed command are also displayed.

The end user will normally not use the Hardy CLIPS window. When your CLIPS code has been debugged, it can be loaded at runtime using the **-clips** *filename* command line option. Any function calls in the file will also be executed (for example to register Hardy event handlers, or load further definitions).

To load functions, you may use the **File: Batch** or **File: Load** menu options. The Load option checks constructs such as functions, printing out error messages; however, *only* construct definitions are allowed, so functions cannot be executed from a file. The Batch option allows both construct definitions and the use of these constructs (e.g. to register interest in a Hardy event); however, construct error messages are not given. The **-clips** command line switch uses the batch method.

It is strongly recommended that you use at least two CLIPS files: a small loader and one or more constructs file. The loader loads the main file or files, and then calls the appropriate event handler registration functions.

For example:

```
(load "constructs.clp")
(register-event-handler NodeLeftClick "KADS Inference" node-left-click)
```

This is very much quicker than having all the code in the top-level batch file.

Also, you may wish to execute the command `(unwatch all)`, or put it early on in your program. This cuts down on the amount of information CLIPS displays on the window, which can be time consuming.

11.2 Debugging CLIPS code

At present, there are few facilities for debugging CLIPS code. For trying out small code fragments, you can type in one command at a time in the text input panel. Once one or more functions have been written, print statements at important points in the code are probably the best way to proceed. A single-stepping facility may be incorporated in later versions of Hardy. This might be emulated for now by placing dummy **read-string** statements in the code to prevent the code proceeding until the user allows it.

Also, typing `(watch all)` makes CLIPS show a trace of function calls and execution of other CLIPS constructs. The `(dribble-on file)` command writes CLIPS error messages and other output that would normally be written to the development window, into a file. `(dribble-off)` flushes and closes the dribble file.

Type errors in early versions of Hardy tended to be fatal, since Hardy could not check that an identifier referred to an existing object of a different type (such as a node object instead of a node image). Type checking is now performed on all objects, so such mistakes should be more readily identifiable.

A common error is forgetting a closing bracket. This may not cause any error message when a file is loaded into CLIPS, but the definitions or function calls after the error will not be made, and so code will not appear to be working. It may be convenient to put a print statement at the end of a file you are debugging: if all is well, a message will be printed; otherwise, there may be a problem with brackets.

Another thing to watch out for is non-reentrant loops. All the CLIPS for Hardy functions whose names contain **get-first-** cannot be used within a loop which already makes use of this function. To get around this, first build a list of identifiers using the **get-first-** and **get-next-** functions and the CLIPS **mv-append** function, and iterate through this list instead. For example,

```
(bind ?id (diagram-card-get-first-node ?card))
(bind ?list (mv-append))
(while (> ?id -1) do
  (bind ?list (mv-append ?list ?id))
  (bind ?id (diagram-card-get-next-node))
)
; The following loop can now call other functions which use
; get-first-card-node
(bind ?counter 1)
(while (> ?counter (length ?list)) do
  (bind ?element (nth ?counter ?list))
  ...
  (bind ?counter (+ ?counter 1))
)
```

11.3 Diagram and hypertext structures

The specialized Hardy CLIPS functions manipulate various structures essential to Hardy, which must be understood before any code can be written.

Most accessible structures are referred to in CLIPS code by integer identifiers, except for named node, arc and other *types* (as opposed to instances of structures of that type) which are referred to by name. For example, a node image of type "Knowledge Role" may have identifer 187.

A Hardy diagram card has an integer identifier, retrievable via the arguments of an event handler function or by other means. There are two sorts of diagram card: *top-level*, and *expansion*. The top-level card is the one first created, and has menu options for file saving and loading. For simple diagrams, this type may be all that is required. An expansion card is a diagram card which 'hangs off' the top-level diagram card or another expansion card, and may be used to build up a hierarchical diagram. Only one file is used to save a hierarchy of diagrams. Most operations may be done on diagram cards without worrying whether it is an expansion or the top-level diagram card.

A Hardy diagram consists of an underlying network of *nodes* and *arcs* (referred to generically as *objects*). They are visually represented by node and arc *images*. The distinction is necessary to accommodate multiple images for one object (for example, the same node appearing on different cards). Usually, there will be only one image for each node or arc. At present there are no functions to allow creating additional images for existing objects. When an image is created, an underlying object is automatically created, the id of which can be retrieved from the image if required. Objects are associated with the top-level card, whereas images are associated with the card on which they are displayed.

Node and arc objects have string *attributes*, some of which are hard-wired (such as "type") and some of which are defined by the user in the diagram type manager. One or more of the user-definable attributes may be used in the image label, determined by a user-supplied format string. The user-definable attributes may be set and retrieved via CLIPS or by the user.

The hypertext structure is based on the concept of the hypertext *item*. Each type of card has its own idea of what corresponds to an item — for the diagram card, each image is conceptually an item and therefore contains an item structure. Items may be linked to other items by hypertext *links* (or *hyperlinks*). A card always has at least one item, called the *special item*, so that a card which either does not support the concept of items (e.g. the text card), or has no appropriate items, may still be linked to another card or item.

Using the appropriate functions, items may be retrieved from images, and any links attached to the items may be traversed, to access other connected items, cards or images. For convenience, there are functions to manipulate expansion cards (which are cards linked to image items via special links) without needing to access the items and traverse the links explicitly.

For further information and some simple examples, please refer the Hardy Software Development Kit and the accompanying Frequently Asked Questions document.

12 Hardy Functions Reference

This section specifies the functions that provide the functionality of Hardy at a card and item level. Display functionality is specified in Section ?? .

This section is presented in five parts based on the card index and the different available card types, with miscellaneous functionality being gathered together at the end.

In the definitions below, function names and parameter names are shown in bold face, with types being shown in italics. The types used are as follows:

1. *double* is a double-precision floating point number.
2. *long* is a long integer.
3. *string* is a double-quoted ASCII string.
4. *word* is an unquoted string.

Functions involving diagram images, objects and hypertext items will use the diagram card identifier, an integer, to ensure uniqueness.

Parameters can be *optional*, in which case the defaults are specified.

Function names are constructed by appending an 'action' to an 'object', for instance `card-get-string-attribute`and `diagram-image-get-width`.

There is an implicit type hierarchy which allows some functions to be general purpose,so `card-get-special-item` can refer to all card types, whereas `diagram-card-find-root` operates on diagram cardsonly. Similarly, a `diagram-object` can be used for`node-objects` and `arc-objects`, and `diagram-image` can be used for`node-images` and `arc-images`.

Note: In Windows NT or WIN32s versions of Hardy, integer identifiers can be negative. So when validating integer identifiers, test for values of zero or -1, rather than for values less than zero.

12.1 Card index functions

hardy-clear-index

long (**hardy-clear-index**)

Clears the hypertext index, with no user confirmation. Returns 1 if successful, 0 otherwise.

hardy-get-first-card

long (**hardy-get-first-card**)

Gets the first card in the index. Returns -1 if there are no cards, or a card id otherwise. Use `hardy-get-next-card` for retrieving further cards.

hardy-get-next-card

long (**hardy-get-next-card**)

Gets the next card in the index. Returns -1 if there are no more cards, or a card id otherwise. Use `hardy-get-first-card` to start iterating through cards.

Note that if you perform an operation that deletes a card during an iteration through the index, this function could give an error. A possible solution is to put all card ids in a list, iterate through this list, and use `card-is-valid` to check if the card still exists.

hardy-get-top-card

long (**hardy-get-top-card**)

Returns the id of the top card, or -1 if none.

hardy-load-index

long (**hardy-load-index** *string* **file**)

Loads the hypertext index from the specified file, returning 1 if successful, 0 otherwise.

hardy-save-index

long (**hardy-save-index** *string* **file**)

Saves the hypertext index in the specified file, returning 1 if successful, 0 otherwise.

12.2 Card functions

The following functions apply to any card.

card-create

long (**card-create** *long* **parent_id**, *string* **card_type**, *optional long* **iconic = 0**, *optional long* **x = -1**, *optional long* **y = -1**, *optional long* **width = -1**, *optional long* **height = -1**, *optional long* **window = 0**)

Creates a new card and returns the id, or -1 if the call failed. *parent_id* may be zero (no parent) or a valid parent card id. *card_type* should be a string: the only valid value at present is "Text card" (diagram cards are created using `diagram-card-create`).

If *iconic* is 1, the card will be created in iconic (minimized) form.

The position and size arguments are optional; if they are omitted or take the value -1, their values will be given defaults.

window may contain the identifier of the frame to display the card in. If *window* is present and non-zero, the card is not already displayed in a window, and the card that is already displayed in *window* is of the same type, then the card will be displayed in this window.

card-delete

long (**card-delete** *long* **card_id**, *optional long* **warn = 1**)

Deletes the given card; returns 1 for success and 0 for failure.

If *warn* is 1 (the default), the user will be asked for confirmation before deleting, otherwise the card will be deleted silently.

card-deselect-all

long (**card-deselect-all** *long* **card_id**)

Deselects all images on the given card; returns 1 for success and 0 for failure.

card-find-by-title

long (**card-find-by-title** *string* **name**, *optional long* **substring=1**)

Finds card for first matching title. *name* may be a substring (if *substring* is 1). This function is useful for testing against cards which have been created manually and whose id is, therefore, not known.

card-get-canvas

long (**card-get-canvas** *long* **card_id**)

Returns the id for the canvas of a card, if the card currently has a physical window. The canvas is the main subwindow of a card, such as the diagram editing canvas of a diagram card.

card-get-frame

long (**card-get-frame** *long* **card_id**)

Returns the frame identifier associated with the card, returning 0 if there is no frame. This allows, for example, a card frame to be used as a parent for a dialog box.

card-get-first-item

long (**card-get-first-item** *long* **card_id**)

Gets the first hypertext item associated with the given card. Returns -1 for end of list. Further items are returned by calls of `card-get-next-item`.

card-get-height

long (**card-get-height** *long* **card_id**)

Returns the height of the card's window.

card-get-next-item

long (**card-get-next-item**)

Following a call of `card-get-first-item` which returns the first hypertext item associated with a specified card, this gets the next hypertext item for that card. Returns -1 for end of list.

card-get-special-item

long (**card-get-special-item** *long* **card_id**)

Returns the "special" hypertext item for the given card. The "special" item always exists, even for empty cards, for the purpose of linking one card to another.

card-get-string-attribute

string (**card-get-string-attribute** *long* **card_id**, *string* **attribute_name**)

Get the value of the given string attribute associated with the card. *attribute_name* may be one of:

1. diagram-type: for diagram or expansion cards only, returns user-defined diagram type;
2. filename;
3. print-file (diagram or expansion cards only);
4. title;
5. type: returns "Text card", "Diagram card", "Hypertext card" or "Diagram expansion".

card-get-toolbar

long (**card-get-toolbar** *long* **card_id**)

Returns the toolbar id for the card, if the card currently has a physical window and if there is a toolbar associated with the card. Returns 0 otherwise.

Note that you should only perform window or toolbar operations on this id if it has been created by a wxCLIPS operation (i.e., is not a Hardy-created toolbar).

card-get-width

long (**card-get-width** *long* **card_id**)

Returns the width of the card's window.

card-get-x

long (**card-get-x** *long* **card_id**)

Returns the x coordinate of the top left corner of the card's window.

card-get-y

long (**card-get-y** *long* **card_id**)

Returns the y coordinate of the top left corner of the card's window.

card-iconize

long (**card-iconize** *long* **card_id**, *optional long* **iconic = 1**)

Iconizes or restores the given card, if it has a physical window associated with it. If *iconic* is 1, the card will be iconized. If 0, it will be restored.

card-is-modified

long (**card-is-modified** *long* **card_id**)

Returns 1 if the given card has been modified and needs to be saved, 0 otherwise.

card-is-shown

long (**card-is-shown** *long* **card_id**)

Returns 1 if the given card is displayed on the screen (i.e. has a physical window associated with it), 0 otherwise. See also `card-show`.

card-is-valid

long (**card-is-valid** *long* **card_id**)

Returns 1 if the card exists, 0 otherwise.

card-move

long (**card-move** *long* **card_id**, *long* **x**, *long* **y**)

Moves the given card to the specified position on the screen. Returns 1 if successful, 0 otherwise.

card-quit

long (**card-quit** *long* **card_id**, *optional long***quit_level=0**)

Quits the given card, i.e. deletes the physical window associated with the card, but does not delete the card from the hypertext index. Returns 1 if successful, 0 otherwise.

Action depends on value of quit_level:

0: full user prompting,

1: if the card has a filename, save and quit without prompting,

2: don't save anything and quit without prompting.

card-select-all

long (**card-select-all** *long* **card_id**)

Selects all images on the given card. Returns 1 if successful, 0 otherwise.

card-send-command

long (**card-send-command** *long* **card_id**, *long* **command_id**)

Sends a menu command identifier to the card, which must be displayed. *command_id* is an internal identifier that can be obtained from an equivalent string form using Section 12.21 .

This function can be used in custom code to provide features that the default user interface normally provides.

See Section 12.22 for a list of identifiers you can use in conjunction with this function.

card-set-icon

long (**card-set-icon** *long* **card_id**, *long* **icon_id**)

Sets the icon for a card, where *icon_id* is a valid icon created with a wxCLIPS function.

card-set-modified

long (**card-set-modified** *long* **card_id**, *optional long* **modified = 1**)

Sets the card 'modified' flag, to 1 by default.

card-set-status-text

long (**card-set-status-text** *long* **card_id**, *string* **text**)

Sets the status line of the given card to display the given text (only if the card has a status line—all diagram cards do, text cards currently do not). Use the empty string ("") to clear the status line. Returns 1 if successful, 0 otherwise.

card-set-string-attribute

long (**card-set-string-attribute** *long* **card_id**, *string* **attribute**, *string* **value**)

Sets the attribute of the given card to the given value. Returns 1 if successful, 0 otherwise. The **attribute** parameter may be one of the following:

1. filename,
2. print-file (diagram card only),
3. title.

card-show

long (**card-show** *long* **card_id**, *optional long* **iconic = 0**, *optional long* **window=0**)

Shows the given card. If the card is being displayed, it is brought to the fore. If it is not being displayed, it is given a new physical window, its contents loaded, and it is brought to the fore (or iconised if there is a second argument which is non-zero).

window may contain the identifier of the frame to display the card in. If *window* is present and non-zero, the card is not already displayed in a window, and the card that is already displayed in *window* is of the same type, then the card will be displayed in this window.

12.3 Item functions

The following functions apply to hypertext items.

item-get-first-link

long (**item-get-first-link** *long* **card_id**, *long* **item_id**)
Gets the first link associated with the item. Returns -1 for end of list. Further links are returned by calls of `item-get-next-link`.

item-get-kind

string (**item-get-kind** *long* **card_id**, *long* **item_id**)
Gets the kind of the item. The 'kind' is a way of labelling an item without deriving a new C++ class, and is currently unused.

item-get-next-link

long (**item-get-next-link**)
Following a call of `item-get-first-link` which returns the first link associated with a specified item, this gets the next link for that item. Returns -1 for end of list.

item-get-type

string (**item-get-type** *long* **card_id**, *long* **item_id**)
Gets the type of the item. A type refers to the item's C++ class, and may currently be "Default item" or "Diagram item".

item-goto

long (**item-goto** *long* **card_id**, *long* **item_id**, *optional long* **iconic = 0**)
Highlight the item on the given card, optionally iconising the card if a new physical window needs to be created for the card. *item_id* can be -1 to use the special item.

Returns the id of the top card, or -1 if none.

item-set-kind

long (**item-set-kind** *long* **card_id**, *long* **item_id**, *string* **kind**)
Sets the *kind* of the item. This string value is not used by Hardy but might be used by an application for some purpose.

12.4 Link functions

The following functions apply to hyperlinks.

link-get-card-from

long (**link-get-card-from** *long* **link_id**)
Returns the card owning the item pointed from by the link, or -1 if unsuccessful.

link-get-card-to

long (**link-get-card-to** *long* **link_id**)
Returns the card owning the item pointed to by the link, or -1 if unsuccessful.

link-get-item-from

long (**link-get-item-from** *long* **link_id**)
Returns the item pointed *from* by the link, or -1 if unsuccessful.

link-get-item-to

long (**link-get-item-to** *long* **link_id**)
Returns the item pointed *to* by the link, or -1 if unsuccessful.

link-get-kind

string (**link-get-kind** *long* **link_id**)
Gets the kind of the link. The 'kind' is a way of labelling a link without deriving a new C++ class, and may currently be "Expansion", for an expansion link, or the empty string.

link-get-type

string (**link-get-type** *long* **link_id**)
Gets the type of the link. A type refers to the link's C++ class, and may currently be one of "Default link" and "Expansion link".

link-cards

long (**link-cards** *long* **from_card**, *long* **to_card**)
Creates a default hyperlink between the two cards, returning the link id if successful or -1 if unsuccessful.

link-items

long (**link-items** *long* **from_card**, *long* **from_item**, *long* **to_card**, *long* **to_item**)
Creates a hyperlink between the two items, returning the link id if successful or -1 if unsuccessful.

link-set-kind

long (**link-set-kind** *long* **link_id**, *string* **kind**)
Sets the *kind* of the link. This string value is not used by Hardy but might be used by an application for some purpose (such as providing 'typed' links).

12.5 Arc image functions

The following functions apply to arc images.

arc-image-change-attachment

long (**arc-image-change-attachment** *long* **card_id**, *long* **image_id**, *long* **end**, *long* **attachment**, *long* **position=-1**)

This function allows the programmer to change the attachment point at which the arc enters or leaves the node. Attachment points start from zero and are either hard-wired or defined in the symbol editor. For example, rectangles, circles and ellipses have four attachment points, starting from the top and going clockwise from zero to three.

If the final *position* argument is given (and more than -1), the function can change the position of the arc relative to the arcs connected at the same attachment point. If zero, the arc will be drawn at the first position. If a very large number, it will be drawn at the last position.

Note that these attachment points are only valid if the node image has attachments switched on (from the symbol editor or node type editor).

If *end* is 1, the 'to' end of the arc is acted on. If *end* is 0, the 'from' end is acted on.

attachment should be a valid attachment point. After the function is called, the arc will be redrawn at the given attachment point.

position, if supplied, specifies the position of the arc relative to other arcs connected to the node at this attachment point.

arc-image-control-point-add

long (**arc-image-control-point-add** *long* **card_id**, *long* **image_id**)

Adds a control point to the arc image (between the middle two points). If the arc image is selected, the application must deselect it before calling this function (and select it again if necessary).

arc-image-control-point-count

long (**arc-image-control-point-count** *long* **card_id**, *long* **image_id**)

Counts the number of control points in the arc image. This number will be at least two (one for each end).

arc-image-control-point-move

long (**arc-image-control-point-move** *long* **card_id**, *long* **image_id**, *long* **point_id**, *double* **x**, *double* **y**)

Moves the given control point to an absolute position on the canvas. If the arc image is selected, the application must deselect it before calling this function (and select it again if necessary). The application must redraw the arc image after this function is called.

The point id must be between 1 and the number of control points in the line. The coordinate system has an origin at the top left of the canvas.

arc-image-control-point-remove

long (**arc-image-control-point-remove** *long* **card_id**, *long* **image_id**)

Removes an abitrary control point from the arc image. If the arc image is selected, the application must deselect it before calling this function (and select it again if necessary).

arc-image-control-point-x

double (**arc-image-control-point-x** *double* **card_id**, *long* **image_id**, *long* **point_id**)

Gets the X position of the control point.

The point id must be between 1 and the number of control points in the line. The coordinate system has an origin at the top left of the canvas.

arc-image-control-point-y

double (**arc-image-control-point-y** *double* **card_id**, *long* **image_id**, *long* **point_id**)

Gets the Y position of the control point.

The point id must be between 1 and the number of control points in the line. The coordinate system has an origin at the top left of the canvas.

arc-image-create

long (**arc-image-create** *long* **card_id**, *string* **arc_type**,
long **from_node**, *long* **to_node**,
optional long **from_attachment**, *optional long* **to_attachment**, *optional string* **alternative_image**)

Creates a new arc and arc image between the given node images. The new arc is stored at the root of the diagram hierarchy; the arc image is associated with the displaying card and its id is returned if the operation is successful. The *arc_type* parameter must be a valid arc type for this diagram type, as defined interactively using the Diagram Type Manager. The optional attachment points refer to where the arc should be attached at the node image end-points; usually the attachment option will not have been activated in the diagram type manager so these will have no effect.

The optional parameter *alternative_image* can specify an alternative image name if the arc definition has more than one image definition defined.

Returns -1 if unsuccessful.

Unlike with `node-image-create`, the arc image is drawn immediately and no further operation is required to make it visible.

arc-image-get-alignment-type

string (**arc-image-get-alignment-type** *long* **card_id**, *long* **image_id**, *long* **end**)

Gets the alignment type for a particular end of the line.

end should be 1 for the arc image end, 0 for the arc image start.

The returned type will be ALIGN_TO_NEXT_HANDLE or ALIGN_NONE.

See also Section 12.5 .

arc-image-get-attachment-from

long (**arc-image-get-attachment-from** *long* **card_id**, *long* **arc_image_id**)

Given a diagram card id and the id of an arc image on that card, retrieves the attachment point of the node image at the 'from' end of the arc. Returns -1 on failure.

The value of the attachment point depends on the type of node, whether attachment mode is on for this node, and where the arc has been drawn from. By default, the value is zero, which if attachment mode is off means that the arc is drawn from the centre of the node. For a rectangle, circle or ellipse, there are four attachment points numbered zero to three clockwise, at the main points of the compass. A polyline's attachment points are at its vertices.

arc-image-get-attachment-to

long (**arc-image-get-attachment-to** *long* **card_id**, *long* **arc_image_id**)

Given a diagram card id and the id of an arc image on that card, retrieves the attachment point of the node image at the 'to' end of the arc. Returns -1 on failure.

The value of the attachment point depends on the type of node, whether attachment mode is on for this node, and where the arc has been drawn to. By default, the value is zero, which if attachment mode is off means that the arc is drawn from the centre of the node. For a rectangle, circle or ellipse, there are four attachment points numbered zero to three clockwise, at the main points of the compass. A polyline's attachment points are at its vertices.

arc-image-get-image-from

long (**arc-image-get-image-from** *long* **card_id**, *long* **arc_image_id**)

Given a diagram card id and the id of an arc image on that card, retrieves the id of the node image at the 'from' end of the arc. Returns -1 on failure.

arc-image-get-image-to

long (**arc-image-get-image-to** *long* **card_id**, *long* **arc_image_id**)

Given a diagram card id and the id of an arc image on that card, retrieves the id of the node image at the 'to' end of the arc. Returns -1 on failure.

arc-image-is-leg

long (**arc-image-is-leg** *long* **card_id**, *long* **image_id**)

Returns 1 if the arc image is a leg joining a junction image *to* a node image, otherwise 0.

arc-image-is-spline

long (**arc-image-is-spline** *long* **card_id**, *long* **arc_image_id**)

Returns 1 if the image is a spline, or 0 if the image is a line.

arc-image-is-stem

long (**arc-image-is-stem** *long* **card_id**, *long* **image_id**)

Returns 1 if the arc image is a stem joining a *from* node image to a junction image, otherwise 0.

The junction image can be found from a stem or leg by following the *from* and *to* pointers in the normal way. Similarly, if the junction image is known, the arc images can be determined.

arc-image-set-alignment-type

long (**arc-image-set-alignment-type** *long* **card_id**, *long* **image_id**, *long* **end**, *string* **type**)

Sets the alignment type for a particular end of the line.

end should be 1 for the arc image end, 0 for the arc image start.

type can be ALIGN_TO_NEXT_HANDLE or ALIGN_NONE. If the former, the point at which the arc image hits the node image is calculated by looking at the next handle (control point) along. Depending on the attachment point, the x or y coordinate is set to the same position as the next handle.

This only applies if attachment mode is on, and the attached node image is rectangular. The alignment position is bounded by the size of node image.

See also Section 12.5 .

arc-image-set-spline

long (**arc-image-set-spline** *long* **card_id**, *long* **arc_image_id**, *long* **is_spline**)

Sets the arc image to be a spline (*is_spline* is 1) or a line (*is-spline* is 0). Returns 1 if successful, 0 otherwise.

12.6 Diagram card functions

The following functions apply to diagram cards.

diagram-card-clear-canvas

long (**diagram-card-clear-canvas** *long* **card_id**)

Clears the canvas associated with the given diagram card. This does not delete any images, it merely blanks the canvas: calling `diagram-card-redraw` will bring back the diagram. Use `diagram-card-delete-all-images` to actually destroy all diagram images.

diagram-card-copy

long (**diagram-card-copy** *long* **card_id**)

Copies the selected images of the card *card_id* into the Hardy clipboard buffer (and onto the Windows clipboard if running under Windows), if the cards are of the same diagram type. Returns 1 if successful, 0 otherwise.

The function `diagram-card-paste` may be used to paste the clipboard buffer contents onto another card. The function `diagram-card-cut` copies and then deletes the selected images.

diagram-card-cut

long (**diagram-card-cut** *long* **card_id**)

Copies the selected images of the card *card_id* into the Hardy clipboard buffer (and onto the Windows clipboard if running under Windows), and then deletes the selected images from the card if the cards are of the same diagram type. Returns 1 if successful, 0 otherwise.

The function `diagram-card-paste` may be used to paste the clipboard buffer contents onto another card. The function `diagram-card-copy` will copy without deleting the selected images.

diagram-card-create

long (**diagram-card-create** *long* **parent_id**, *string* **diagram_type**, *optional long* **iconic=0**, *optional long* **x = -1**, *optional long* **y = -1**, *optional long* **width = -1**, *optional long* **height = -1**, *optional long* **window=0**)

Creates a new diagram card and returns the id, or -1 if the call failed. *parent_id* may be zero (no parent) or a valid parent card id. *diagram_type* should be a valid diagram type, as defined using the interactive Diagram Type Manager.

If *iconic* is 1, the card will initially be shown in the iconic state.

The position and size arguments are optional; if they are omitted or take the value -1, their values will be given defaults.

window may contain the identifier of the frame to display the card in. If *window* is present and non-zero, the card is not already displayed in a window, and the card that is already displayed in *window* is of the same type, then the card will be displayed in this window.

diagram-card-create-expansion

long (**diagram-card-create-expansion** *long* **parent_id**, *long* **image_id**, *optional* *long***window=0**)

Creates a new diagram expansion card and returns the id, or -1 if the call failed. *parent_id* must be a valid parent card id. *image_id* should be a valid node or arc image id to be expanded, or -1 to signify the card should be linked to the parent card itself and not an image within it.

window may contain the identifier of the frame to display the card in. If *window* is present and non-zero, the card is not already displayed in a window, and the card that is already displayed in *window* is of the same type, then the card will be displayed in this window.

diagram-card-delete-all-images

long (**diagram-card-delete-all-images** *long* **card_id**)

Attempts to delete all the images on the canvas of the given diagram card. It may fail if images are connected to expansion cards. Returns 1 if successful, 0 otherwise.

diagram-card-find-root

long (**diagram-card-find-root** *long* **card_id**)

Given a diagram card id, finds the id of the diagram card at the base of the diagram hierarchy (not necessarily of the whole hypertext hierarchy). This may or may not be the same as *card_id*. Returns -1 for failure.

For example, if a callback provides a diagram id which is that of an expansion card somewhere in the hierarchy, supplying the id to this function will return the root of the hierarchy (*not* the 'top card', which is something else entirely!).

diagram-card-get-first-arc-first-image

long (**diagram-card-get-first-arc-first-image** *long* **card_id** *optional long* **arc_id** *optional* *long* **selected**)

Given a diagram card id, retrieves the id of the first arc image on the card (or another card on the same hierarchy). Further arc images are accessed by calls to `diagram-card-get-next-arcImage`. Returns -1 on failure.

arc_id may be omitted or zero to all return arc images for this card regardless of arc object; or a valid arc object id to restrict the images to those belonging to the arc object.

If *selected* is 1, the images returned will be those currently selected, in the order in which they were selected. If zero, all arc images will be returned.

diagram-card-get-first-arc-first-object

long (**diagram-card-get-first-arc-first-object** *long* **card_id**)

Given a diagram card id, retrieves the id of the first arc (*not* arc image) on the card (or root of this card). Further arcs are accessed by calls to `diagram-card-get-next-arc-next-object`. Returns -1 on failure.

diagram-card-get-first-descendant

long (**diagram-card-get-first-descendant** *long* **card_id**)

Get the first expansion card (always a diagram card) of a card. Returns -1 if no expansion cards.

Used with `diagram-card-get-next-descendant`, allows iteration on all expansion cards which are descendants from a given root card, without having to follow the hypertext links attached to diagram images.

diagram-card-get-first-node-first-image

long (**diagram-card-get-first-node-first-image** *long* **card_id** *optional long* **node_id** *optional long* **selected**)

Given a diagram card id, retrieves the id of the first node image on the card. Further node images are accessed by calls to `diagram-card-get-next-node-next-image`. Returns -1 on failure.

node_id may be omitted or zero to all return node images for this card regardless of node object; or a valid node object id to restrict the images to those belonging to the node object.

If *selected* is 1, the images returned will be those currently selected, in the order in which they were selected. If zero, all node images will be returned.

diagram-card-get-first-node-first-object

long (**diagram-card-get-first-node-first-object** *long* **card_id**)

Given a diagram card id, retrieves the id of the first node (*not* node image) on the card (or root of this card). Further cards are obtained by calling `diagram-card-get-next-node-next-object`. Returns -1 on failure.

diagram-card-get-grid-spacing

long (**diagram-card-get-grid-spacing** *long* **card_id**)

Returns the current grid spacing for the card; a value of zero means that snap-to-grid is switched off.

diagram-card-get-parent-card

long (**diagram-card-get-parent-card** *long* **card_id**)

If the card is an expansion card, returns the id of the parent diagram card. Otherwise returns -1.

diagram-card-get-parent-image

long (**diagram-card-get-parent-image** *long* **card_id**)

If the card is an expansion card, returns the id of the connected node or arc image on the parent diagram card. Otherwise returns -1.

diagram-card-get-next-arc-next-image

long (**diagram-card-get-next-arc-next-image**)

Following a call of `diagram-card-get-first-arc-first-image` for a specified card, retrieves the id of the next node image on the card. Returns -1 on failure or to signify no more images.

diagram-card-get-next-arc-next-object

long (**diagram-card-get-next-arc-next-object**)

Following a call of `diagram-card-get-first-arc-first-object` for a specified diagram card id, this retrieves the id of the next arc (*not* arc image) on the card (or root of this card). Returns -1 on failure or to signify no more arcs.

diagram-card-get-next-descendant

long (**diagram-card-get-next-descendant**)

Following a call of `diagram-card-get-first-descendant` for a specified node, get the next expansion card for that node Returns -1 if no more expansion cards.

diagram-card-get-next-node-next-image

long (**diagram-card-get-next-node-next-image**)

Following a call of `diagram-card-get-first-node-first-image` for a specified card, retrieves the id of the next node image on the card. Returns -1 on failure or to signify no more images.

diagram-card-get-next-node-next-object

long (**diagram-card-get-next-node-next-object**)

Following a call of `diagram-card-get-first-card-first-nodeObject` for a specified card, retrieves the id of the next node on the card (or root of this card). Returns -1 on failure or to signify no more nodes.

diagram-card-get-print-height

long (**diagram-card-get-print-height** *long* **card_id**)

Returns the height of the diagram card's Postscript image in points. This call must be made *after* a call to `diagram-card-print-hierarchy`, which calculates the print size.

diagram-card-get-print-width

long (**diagram-card-get-print-width** *long* **card_id**)

Returns the width of the diagram card's Postscript image in points. This call must be made *after* a call to `diagram-card-print-hierarchy`, which calculates the print size.

diagram-card-get-scale

double (**diagram-card-get-scale** *long* **card_id**)

Returns the scaling factor for the card.

diagram-card-layout-graph

long (**diagram-card-layout-graph** *long* **card_id**)

Lay out the given diagram using a simple graph layout algorithm. The current layout parameters, set through `diagram-card-set-layout-parameters`, are used.

diagram-card-layout-tree

long (**diagram-card-layout-tree** *long* **card_id**, *long* **image_id**, *long* **orientation**)

Lay out the given diagram as a tree, using the given image as root. The current layout parameters, set through `diagram-card-set-layout-parameters`, are used.

If *orientation* is 1, the layout is top to bottom, otherwise it is left to right.

diagram-card-load-file

long (**diagram-card-load-file** *long* **card_id**, *string* **filename**)

Loads the given diagram file onto the given card. Returns 1 for success, 0 for failure.

diagram-card-paste

long (**diagram-card-paste** *long* **card_id**)

Copies images from the Hardy clipboard buffer onto the card *card_id*, if the diagram types of buffer and card match. The clipboard buffer should have been filled with `diagram-card-copy` or `diagram-card-cut` prior to this operation. Returns 1 if successful, 0 otherwise.

The function `diagram-card-copy` will copy without deleting the selected images. The function `diagram-card-cut` copies and then deletes the selected images.

diagram-card-popup-menu

long (**diagram-card-popup-menu** *long* **card_id**, *long* **menu_id**, *double* **x**, *double* **y**)
Popups up a menu previously created using menu-create in wxCLIPS.

diagram-card-print-hierarchy

long (**diagram-card-print-hierarchy** *long* **card_id**)
Prints the diagram card hierarchy to separate PostScript files, prompting for filenames if necessary.

diagram-card-redraw

long (**diagram-card-redraw** *long* **card_id**)
Redraws the entire diagram, returning 1 for success, 0 otherwise.

diagram-card-save-bitmap

long (**diagram-card-save-bitmap** *long* **card_id**, *string* **filename**)
For Windows version only: saves the diagram in a Windows RGB-encoded bitmap (usual extension .BMP). Returns 1 if successful, 0 otherwise.

diagram-card-save-file

long (**diagram-card-save-file** *long* **card_id**, *string* **file**)
Saves the diagram on the given card in the specified file, returning 1 if successful, 0 otherwise.

diagram-card-save-metafile

long (**diagram-card-save-metafile** *long* **card_id**, *string* **filename**, *optional double* **scale = 1.0**)
For Windows version only: saves the diagram in a Windows metafile (usual extension .wmf). Returns 1 if successful, 0 otherwise.

The optional *scale* parameter defaults to 1.0, and is used to reduce or enlarge the metafile.

A metafile is a 'recording' of the graphics functions used to draw a picture. Its chief merits are its scaleability, and its economy of disk space for many types of picture. Metafiles may be included in RTF (Rich Text Format) files to allow programmatic construction of word processor documents containing text and pictures.

diagram-card-set-grid-spacing

long (**diagram-card-set-grid-spacing** *long* **card_id** *long* **grid_spacing**)
Sets the grid spacing for the card; a value of zero switches snap-to-grid off.

diagram-card-set-layout-parameters

long (**diagram-card-set-layout-parameters** *long* **card_id**,
double **left_margin**, *double* **right_margin**, *double* **width**, *double* **height**,
double **spacing_x**, *double* **spacing_y**)

Sets layout parameters used by auto-layout functions.

diagram-card-set-scale

long (**diagram-card-set-scale** *long* **card_id**, *float* **scale**)

Sets the scaling factor for the card. 1.0 is 100 per cent. Note that factors above 1 can cause scrolling problems under MS Windows (vertical and horizontal lines get left behind).

12.7 Diagram object functions

The following functions apply to diagram objects (diagram nodes or arcs).

diagram-object-add-attribute

void (**diagram-object-add-attribute** *long* **card_id**, *long* **object_id**, *string* **attribute**)

Adds a new attribute name to the user-defined attributes section of an object (node or arc), and initialises it with the empty string("").

This should *not* be called whilst the user is still editing attributes.

diagram-object-delete-attribute

void (**diagram-object-delete-attribute** *long* **card_id**, *long* **object_id**, *string* **attribute**)

Deletes the named user-defined attribute from an object (node or arc). This does not just delete an attribute value value, it deletes the attribute itself.

This should *not* be called whilst the user is still editing attributes.

diagram-object-format-text

long (**diagram-object-format-text** *long* **card_id**, *long* **object_id**)

Formats the visible text of the node or arc object, for all images associated with this object. The format string specified in the diagram definition is used, and all images redrawn. Returns 1 if successful, 0 otherwise.

diagram-object-get-first-attribute

string (**diagram-object-get-first-attribute** *long* **card_id**, *long* **object_id**)

Get the first attribute name from the object on the given card. Further attribute names are obtained by calling `diagram-object-get-next-attribute`. Returns the empty string if no attribute.

diagram-object-get-first-image

long (**diagram-object-get-first-image** *long* **card_id**, *long* **object_id**)

Given a diagram card id and a node or arc object id, retrieves the id of the first image belonging to the object. (Node and arc objects may have more than one image associated with them.) The card id may be any card in the hierarchy since all nodes and arcs are associated with the top card in the hierarchy. Further image ids are obtained by calling `diagram-object-get-next-image`. Returns -1 on failure.

diagram-object-get-next-attribute

string (**diagram-object-get-next-attribute**)

Following a call of `diagram-object-get-first-attribute` for a specified object, get the next attribute name from the object. Returns the empty string if no further attributes.

diagram-object-get-next-image

long (**diagram-object-get-next-image**)

Following a call of `diagram-object-get-first-image` for a specified object, retrieves the id of the next image belonging to the object. (Node and arc objects may have more than one image associated with them.) Returns -1 on failure or to signify no more images.

diagram-object-get-string-attribute

string (**diagram-object-get-string-attribute** *long* **card_id**, *long* **object_id**,
string **attribute**)

Given a diagram card id, node or arc object id and string attribute name, returns the value of the attribute if found, the empty string if not found. The only attribute you may rely on is *type*; any others depend on the attributes defined in the particular diagram definition.

diagram-object-set-format-string

long (**diagram-object-set-format-string** *long* **card_id**, *long* **object_id**,
string **format_string**)

Sets the format string for the object. Normally, the format string for a node or arc type is set in the diagram type manager; this function allows the programmer to dynamically change the format string on a per-object basis. You may wish to show more or less information on an image depending on the context.

If the local format string is the same as the object type format string, it will not be written to file for that object, to save space and time.

diagram-object-set-string-attribute

long (**diagram-object-set-string-attribute** *long* **card_id**, *long* **object_id**,
string **attribute**, *string* **value**)

Sets the object (node or arc) attribute to the given value. *attribute* may be one of the attributes named in the node or arc type definition. Do *not* try to set an image attribute directly; you may obtain the object for an image using `diagram-image-get-object`.

Returns 1 for success, 0 for failure.

12.8 Diagram palette functions

The following functions apply to a diagram palette.

diagram-palette-get-arc-selection

string (**diagram-palette-get-arc-selection** *long* **card_id**)
Returns the type of the arc symbol selected on the diagram card palette, or the empty string if no arc symbol was selected or the palette was not displayed.

diagram-palette-get-arc-selection-arc-image

string (**diagram-palette-get-arc-selection-arc-image** *long* **card_id**)
Returns the image definition type of the arc symbol selected on the diagram card palette, or the empty string if no arc symbol was selected or the palette was not displayed.

For each arc type, there are one or more arc image definitions: usually there is only one, with the name 'Default". If there are several arc image definitions for an arc type, each will be displayed on the palette, and this function will distinguish between them.

diagram-palette-get-first-annotation-first-selection

string (**diagram-palette-get-first-annotation-first-selection** *long* **card_id**, *string* **type**)
Gets the first selected annotation symbol on the diagram palette. *type* may be one of "Both", "Node" or "Arc", to get different kinds of annotation selection. The function returns the first annotation name or the empty string.

`diagram-palette-get-next-annotation-next-selection` generates any further ones, as several annotation images may be selected at once.

diagram-palette-get-next-annotation-next-selection

string (**diagram-palette-get-next-annotation-next-selection** *long* **card_id**)
Returns the name of the next selected annotation symbol for the diagram type from the palette, following a call of `diagram-palette-get-next-annotation-next-selection`, or the empty string.

diagram-palette-get-node-selection

string (**diagram-palette-get-node-selection** *long* **card_id**)
Returns the type of the node symbol selected on the diagram card palette, or the empty string if no node symbol was selected or the palette was not displayed.

diagram-palette-show

long (**diagram-palette-show** *long* **card_id** *long* **show=1**)

If the card is currently visible, shows or hides the palette.

diagram-palette-set-annotation-selection

long (**diagram-palette-set-annotation-selection** *long* **card_id**,
string **annotation_name**, *long* **select**)

Toggles the named annotation symbol on the diagram palette on or off, depending on the value of *select*: 1 for on, 0 for off. Returns 1 if the function succeeds, 0 otherwise.

diagram-palette-set-arc-selection

long (**diagram-palette-set-arc-selection** *long* **card_id**,
string **type_name**, *string* **image_def**, *long* **flag**)

Toggles the given arc symbol on the diagram palette on or off, depending on the value of *select*: 1 for on, 0 for off. Returns 1 if the function succeeds, 0 otherwise.

type_name is the arc type name, and *image_def* is the image definition for the arc. Normally *image_def* would be "Default" since most arc definitions only have one image definition.

diagram-palette-set-node-selection

long (**diagram-palette-set-node-selection** *long* **card_id**,
string **type_name**, *long* **select**)

Toggles the given node symbol on the diagram palette on or off, depending on the value of *select*: 1 for on, 0 for off. Returns 1 if the function succeeds, 0 otherwise.

12.9 Diagram image functions

The following functions apply to a diagram image (a node or arc image).

diagram-image-add-annotation

long (**diagram-image-add-annotation** *long* **card_id**, *long* **image_id**,
string **annotation_name**, *string* **dropsite_name**)

Adds an annotation to the given node or arc image, returning an id if successful or -1 if unsuccessful.

For node images, annotations are additional node-like children of a composite node image. Legal annotation symbols are defined in the Drop Site Editor, and their names and drop sites can be used in this function.

For arc images, annotations are usually arrow-heads that can be added at three drop sites, "Start", "Middle" and "End", in the order defined in the Arc Type Editor. The annotation name in this case is the *logical* or *displayed* name for the annotation, which is the same as the *physical* name (such as "Normal arrowhead") unless overridden in the Arc Type Editor.

See also `diagram-image-delete-annotation`.

diagram-image-annotation-get-drop-get-site

string (**diagram-image-annotation-get-drop-get-site** *long* **card_id**, *long* **image_id**,
long **annotation_id**)

Gets the drop site name for the given annotation, or the empty string if the call fails.

diagram-image-annotation-get-logical-get-name

string (**diagram-image-annotation-get-logical-get-name** *long* **card_id**, *long* **image_id**,
long **annotation_id**)

Gets the *logical name* of the given annotation, or the empty string if the call fails.

The *logical name* is the same as the *name* for a node annotation. For an arc annotation, the logical name is the same as the name unless the logical name for the annotation has been changed in the Annotation Properties dialog in the Arc Type Editor. All logical names for a physical arc annotation are listed in the status line when the cursor is moved over the annotation in the diagram card symbol palette.

The logical name is used to reflect a notational convention for a particular arc, even though the underlying arc annotation symbol may be used several times in different contexts.

See also `diagram-image-annotation-get-name`.

diagram-image-annotation-get-name

string (**diagram-image-annotation-get-name** *long* **card_id**, *long* **image_id**,
long **annotation_id**)

Gets the name of the given annotation, or the empty string if the call fails.

See also `diagram-image-annotation-get-logical-get-name`.

diagram-image-delete

long (**diagram-image-delete** *long* **card_id**, *long* **image_id**)

Erases and deletes the given image. Note that if quick edit mode is on, damaged areas will not be redrawn automatically. Returns 1 if successful, 0 otherwise.

diagram-image-delete-annotation

long (**diagram-image-delete-annotation** *long* **card_id**, *long* **image_id**, *long* **annotation_id**)

Deletes an annotation from a node or arc image.

diagram-image-draw

long (**diagram-image-draw** *long* **card_id**, *long* **image_id**)

Draws the image, returning 1 if successful, 0 otherwise.

diagram-image-draw-text

long (**diagram-image-draw-text** *long* **card_id**, *long* **image_id**, *string* **text**, *optional string* **region_name**)

Draws text in the image. This is a lower-level operation than `diagram-object-format-text` since it is on a per-image basis and does not use the format string as defined in the Diagram Type Manager. This may be useful for displaying text that the format string will not allow, such as user-defined attribute values.

region_name (default value "0") names the text region of the image for images that have multiple text regions, such as composites, divided rectangle images, and arcs.

Simple images, such as ellipses and rectangles, have one region called "0".

Divided rectangles have as many regions as the number of divisions and, for a divided rectangle that is not part of a composite, the naming is "0", "1", "2" and so on.

Arc images always have three regions called "Start", "Middle" and "End".

Composite node images have a region "0", but `diagram-image-draw-text` can be used for the components: the text regions of the components are named automatically. For a given node type, see the node type editor for a list of text regions and a visual indication of where these regions are on the composite.

diagram-image-erase

long (**diagram-image-erase** *long* **card_id**, *long* **image_id**)

Erases the given image. Note that if quick edit mode is on, damaged areas will not be redrawn automatically. Returns 1 if successful, 0 otherwise.

diagram-image-get-brush-colour

string (**diagram-image-get-brush-colour** *long* **diagram_id**, *long* **image_id**)

Gets the brush (fill) colour for the given image. The colour is a wxWindows colour string such as "BLACK" (see wxWindows documentation). Returns a colour string if successful, the empty string otherwise.

diagram-image-get-card

long (**diagram-image-get-card** *long* **card_id**, *long* **image_id**)

Given the id of a card somewhere in the hierarchy, and the id of a node or arc image on one of the cards in the hierarchy, get the id of the card on which the image appears. Returns -1 if unsuccessful.

This may be needed, for example, when retrieving arc images from arc objects, in order to find the level at which the connection is taking place.

diagram-image-get-first-annotation

long (**diagram-image-get-first-annotation** *long* **card_id**, *long* **image_id**)

Returns the id of the first annotation for a node image (another node image) or an arc image (an annotation image) or -1 if no annotation is present. Together with `diagram-image-get-next-annotation`, this allows iteration through all annotations of an image.

diagram-image-get-first-expansion

long (**diagram-image-get-first-expansion** *long* **card_id**, *long* **image_id**)

Get the first expansion card (always a diagram card) from the given image on the given card. Further expansion cards are obtained by calling `diagram-image-get-next-expansion`. Returns -1 if no expansion cards.

diagram-image-get-height

double (**diagram-image-get-height** *long* **card_id**, *long* **image_id**)

Returns the floating-point value of the image height, or rather the height of the bounding box enclosing the image.

diagram-image-get-item

long (**diagram-image-get-item** *long* **card_id**, *long* **image_id**)

Returns the hypertext item corresponding to the given image on the given card, or -1 if unsuccessful.

diagram-image-get-next-annotation

long (**diagram-image-get-next-annotation**)

Returns the id of the next annotation for a node (another node image) or an arc (an annotation image) image, or -1 if no further annotations exist. Together with `diagram-image-get-first-annotation`, this allows iteration through all annotations of an image.

diagram-image-get-next-expansion

long (**diagram-image-get-next-expansion**)

Following a call of `diagram-image-get-first-expansion` for an image on a specified card, get the next expansion card. Returns -1 on failure or if no more expansion cards.

diagram-image-get-object

long (**diagram-image-get-object** *long* **card_id**, *long* **image_id**)

Gets the id of the node or arc object corresponding to the given node or arc image id. Returns -1 on failure.

diagram-image-get-pen-colour

string (**diagram-image-get-pen-colour** *long* **diagram_id**, *long* **image_id**)

Gets the pen (outline) colour for the given image. The colour is a wxWindows colour string such as "BLACK" (see wxWindows documentation). Returns a colour string if successful, the empty string otherwise.

diagram-image-get-text-colour

string (**diagram-image-get-text-colour** *long* **diagram_id**, *long* **image_id**,
optional string **region_name = "0"**)

Gets the text colour for the given image. The colour is a wxWindows colour string such as "BLACK" (see wxWindows documentation).

The optional parameter *region_name* identifies the text region, for composite images or images (such as divided rectangles) that have multiple text regions.

Returns a colour string if successful, an empty string otherwise.

diagram-image-get-width

double (**diagram-image-get-width** *long* **card_id**, *long* **image_id**)

Returns the floating-point value of the image width, or rather the width of the bounding box enclosing the image.

diagram-image-get-x

double (**diagram-image-get-x** *long* **card_id**, *long* **image_id**)
Returns the floating-point value of the image x coordinate (at the centre of the image).

diagram-image-get-y

double (**diagram-image-get-y** *long* **card_id**, *long* **image_id**)
Returns the floating-point value of the image y coordinate (at the centre of the image).

diagram-image-is-shown

long (**diagram-image-is-shown** *long* **card_id**, *long* **image_id**)
Returns 1 if the image is visible, 0 otherwise.

diagram-image-move

long (**diagram-image-move** *long* **card_id**, *long* **image_id**, *double* **x**, *double* **y**)
Moves the centre of the image to the given position and redraws it. Note that if quick edit mode is on, damaged areas will not be redrawn automatically. Returns 1 if successful, 0 otherwise.

diagram-image-pending-delete

long (**diagram-image-pending-delete** *long* **card_id**, *long* **image_id**)
Returns 1 if the diagram image is about to be deleted by Hardy.

This function is occasionally necessary when you need to determine, from within an arc deletion event, whether a node attached to that arc can be safely deleted by the custom code. If the current arc image is being deleted automatically because a node is being deleted, then calling this function will determine that it is not safe to delete the node image, because the node image would be deleted twice.

Important note: if the node image to be tested is potentially part of a composite, you should check if there is a parent node image, and if so, whether there is a deletion pending on that, and so on.

diagram-image-put-to-front

long (**diagram-image-put-to-front** *long* **card_id**, *long* **image_id**, *optional long* **front = 1**)
Puts the image to the front (if front = 1) or back (if front = 0) of the canvas.

diagram-image-resize

long (**diagram-image-resize** *long* **card_id**, *long* **image_id**,
double **width**, *double* **height**)

Resizes the image to the given width and height, and redraws it. Note that if quick edit mode is on, damaged areas will not be redrawn automatically. Returns 1 if successful, 0 otherwise.

diagram-image-select

long (**diagram-image-select** *long* **card_id**, *long* **image_id**, *long* **flag**)

Selects and redraws the image if *flag* is 1, deselects if *flag* is 0. Note that other parts of the diagram may be damaged if an image is deselected, since control points are erased. If quick edit mode is on, the application must call `diagram-card-redraw` to refresh the diagram.

diagram-image-selected

long (**diagram-image-selected** *long* **card_id**, *long* **image_id**)

Returns 1 if the node or arc image is selected, 0 otherwise.

diagram-image-set-brush-colour

long (**diagram-image-set-brush-colour** *long* **diagram_id**, *long* **image_id**,
string **colour**)

Sets the brush (fill) colour for the given image, and redraws the image. The colour is a wxWindows colour string such as "BLACK" (see wxWindows documentation). Returns 1 if successful, 0 otherwise.

diagram-image-set-pen-colour

long (**diagram-image-set-pen-colour** *long* **diagram_id**, *long* **image_id**,
string **colour**)

Sets the pen (outline) colour for the given image, and redraws the image. The colour is a wxWindows colour string such as "BLACK" (see wxWindows documentation). Returns 1 if successful, 0 otherwise.

diagram-image-set-shadow-mode

long (**diagram-image-set-shadow-mode** *long* **card_id**, *long* **image_id**, *long* **shadow =**
1, *optional long* **offset_x = 0**, *optional long* **offset_y = 0**)

Sets the shadow mode (1 for shadow, 0 for no shadow) for the given image. Optional shadow offsets can be given; 0 for an offset means assume the default.

diagram-image-set-text-colour

long (**diagram-image-set-text-colour** *long* **diagram_id**, *long* **image_id**,
string **colour**, *optional string* **region_name = "0"**)

Sets the text colour for the given image, and redraws the image. The colour is a wxWindows colour string such as "BLACK" (see wxWindows documentation).

The optional parameter *region_name* identifies the text region, for composite images or images (such as divided rectangles) that have multiple text regions.

Returns 1 if successful, 0 otherwise.

diagram-image-show

long (**diagram-image-show** *long* **card_id**, *long* **image_id**, *long* **show**)

If *show* is TRUE, the image will be in the visible state (the default). If *show* is FALSE, the image will be in the invisible state, and not drawn or sensitive to mouse events. The function works recursively for composite images.

diagram-item-get-image

long (**diagram-item-get-image** *long* **card_id**, *long* **image_id**)

Returns the diagram node or arc image corresponding to the given hypertext item on the given card, or -1 if unsuccessful (for example if the item is not on a diagram card).

12.10 Node image functions

The following functions apply to a diagram node image.

node-image-create

long (**node-image-create** *long* **card_id**, *string* **node_type**)

Creates a new node and node image. The new node is stored at the root of the diagram hierarchy; the node image is associated with the displaying card and its id is returned if the operation is successful. The *node_type* parameter must be a valid node type for this diagram type, as defined interactively using the Diagram Type Manager. Returns -1 if unsuccessful. Note that the image is not drawn automatically immediately after creation, to give your application a chance to move it somewhere appropriate using `diagram-image-move`.

Use `node-image-duplicate` if you wish to create a new image for an *existing* node object.

node-image-duplicate

long (**node-image-duplicate** *long* **card_id**, *long* **node_id**)

Creates a new node image for the existing node object. Give the destination card and node object id. Returns -1 if unsuccessful.

Note that the image is not drawn automatically immediately after creation, to give your application a chance to move it somewhere appropriate using `diagram-image-move`.

node-image-get-container

long (**node-image-get-container** *long* **card_id**, *long* **image_id**)

Returns the id of the container this node is in, otherwise -1.

node-image-get-first-arc-first-image

long (**node-image-get-first-arc-first-image** *long* **card_id**, *long* **image_id**)

Given a diagram card id and node image id, retrieves the id of the first arc image associated with the node. Calling node-image-get-next-arcImage will generate any further arc images. Returns -1 on failure.

node-image-get-first-child

long (**node-image-get-first-child** *long* **card_id**, *long* **image_id**)

Given a diagram card id and composite node image id, retrieves the id of the first child of the image. Returns -1 on failure or to signify no more images. Further child images are generated by calls of `node-image-get-next-child`.

node-image-get-first-container-first-region

long (**node-image-get-first-container-first-region** *long* **card_id**, *long* **image_id**)

Returns the id of the first container region belonging to the given container node image, or -1 if the image is not a container. Further container regions are generated by `node-image-get-next-container-next-region`.

A container image has one or more *container regions*, each of which can contain other node images (subject to constraints defined in the Node Type Editor). The user may split existing container regions into further regions (using control-right click to bring up the container region menu).

A container region is a node image in its own right, with a single corresponding node object of type *Container region.* This means that a container region may be linked by arcs to other nodes.

node-image-get-parent

long (**node-image-get-parent** *long* **card_id**, *long* **image_id**)

Returns the id of the parent node image of this node image, or zero if there is none, or if the parent is a container region of a different node.

node-image-get-container-parent

long (**node-image-get-container-parent** *long* **card_id**, *long* **image_id**)

Returns the id of the container region which is a parent of this node image, or zero if there is none. By implication, the node image and the parent belong to separate node objects.

node-image-get-next-arc-next-image

long (**node-image-get-next-arc-next-image**)

Following a call of `node-image-get-first-arc-first-image` for a specified node image on a card, retrieves the id of the next arc image associated with that node image. Returns -1 on failure or to signify no more images.

node-image-get-next-child

long (**node-image-get-next-child**)

Returns the ID of the next child of the composite, or -1 if no further child images are present. Together with `node-image-get-first-child`, this allows iteration through all child images of a composite.

node-image-get-next-container-next-region

long (**node-image-get-next-container-next-region**)

Returns the id of the next container region belonging to the given container node image or -1 if there are no further regions. The node image is specified by the last call of `node-image-get-first-container-first-region`.

node-image-is-composite

long (**node-image-is-composite** *long* **card_id**, *long* **image_id**)

Returns 1 if node is a composite, otherwise 0.

A composite image is an image that has, or is capable of having, one or more child images. This includes container nodes.

node-image-is-container

long (**node-image-is-container** *long* **card_id**, *long* **image_id**)

Returns 1 if node is a container, otherwise 0.

A container is a node image that has been defined in the Node Type Editor or Node Symbol Editor to accept certain types of contained node images, which form a composite relationship with the container.

node-image-is-junction

long (**node-image-is-junction** *long* **card_id**, *long* **image_id**)

Returns 1 if the image is a junction image, otherwise 0. This test must be used when traversing node images since junction images have fewer properties than normal.

A junction is an image used in multiway arcs, with some properties similar to ordinary node images, but with no corresponding node object, and it is never a composite. It is usually represented by a 'metafile' symbol that can be rotated according to the direction of the multiway arc.

node-image-order-arcs

long (**node-image-order-arcs** *long* **card_id**, *long* **image_id**, *long* **attachment**, *multi-field* **arc_images**)

Reorders the arc images linked to this node image at this specific attachment point, according to the ordering of the list of arc images. Any arc images not explicitly mentioned in the list will be appended.

12.11 Node object functions

The following functions apply to a diagram node object.

node-object-get-first-arc-first-object

long (**node-object-get-first-arc-first-object** *long* **card_id**, *long* **image_id**)

Given a diagram card id and node object id, retrieves the id of the first arc object associated with the node. Calling `node-object-get-next-arc-next-object` generates any further arc objects. Returns -1 on failure.

Note that there are no functions to retrieve the 'to' node and 'from' node from an arc object, because there may be several connections between nodes for the same arc object (for instance, an arc may be represented at several levels of a diagram, between different nodes). To retrieve the nodes at either end of an arc, get the arc *image(s)*, then the 'to' and 'from' node *images*, and then the node *objects* from these.

node-object-get-next-arc-next-object

long (**node-object-get-next-arc-next-object**)

Following a call of `node-object-get-first-arc-first-object` for a specified diagram card and node, retrieves the id of the next arc object associated with the node. Returns -1 on failure or to signify no more objects.

Note that there are no functions to retrieve the 'to' node and 'from' node from an arc object, because there may be several connections between nodes for the same arc object (for instance, an arc may be represented at several levels of a diagram, between different nodes). To retrieve the nodes at either end of an arc, get the arc *image(s)*, then the 'to' and 'from' node *images*, and then the node *objects* from these.

12.12 Arc annotation functions

The following functions apply to a diagram arc image annotation.

arc-annotation-get-name

string (**arc-annotation-get-name** *long* **card_id**, *long* **annotation_id**)
Returns the symbol name as used in the Arc Symbol Editor.

12.13 Container region functions

The following functions apply to a diagram container image.

container-region-add-node-image

long (**container-region-add-node-image** *long* **card_id**,
long **container_id**,*long* **contained_image_id**, *double* **x**, *double* **y**)

Moves *contained_image_id* into *container_id* if legal, moving the contained node to the given coordinates.

See `node-image-get-first-container-first-region` for an explanation of container regions.

container-region-remove-node-image

long (**container-region-remove-node-image** *long* **card_id**,
long **container_id**, *long* **contained_image_id**, *double* **x**, *double* **y**)

Moves *contained_image_id* out of *container_id*, without deleting *contained_image_id*. The contained node is moved to the given coordinates.

See `node-image-get-first-container-first-region` for an explanation of container regions.

12.14 Hypertext card functions

The following functions are relevant to hypertext cards.

hypertext-card-create

long (**hypertext-card-create** *long* **parent_id**, *string* **hypertext_type**, *optional long* **iconic = 0**)

Creates a new hypertext card and returns the id, or -1 if the call failed. *parent_id* may be zero (no parent) or a valid parent card id. *hypertext_type* should be a valid hypertext type, as defined using the interactive Hypertext Type Manager.

If *iconic* is 1, the card will be shown in the iconic state.

hypertext-card-get-current-char

int (**hypertext-card-get-current-char** *long* **card_id**)

Gets the current character position for a successful search operation, or the character position calculated by `hypertext-card-get-offset-position`.

hypertext-card-get-current-line

int (**hypertext-card-get-current-line** *long* **card_id**)

Gets the current line number for a successful search operation, or the line number calculated by `hypertext-card-get-offset-position`.

hypertext-card-get-first-selection

long (**hypertext-card-get-first-selection** *long* **card_id**)

Get the first selected block for a given a hypertext card id. Returns -1 if no more selected blocks.

`hypertext-card-get-next-selection` can be used to get the next selection.

hypertext-card-get-line-length

int (**hypertext-card-get-line-length** *long* **card_id**, *long* **line_no**)

Gets the number of characters in the given line, or -1 if the line was not found.

hypertext-card-get-next-selection

long (**hypertext-card-get-next-selection**)

Given a hypertext card id, get the next selected block. Returns -1 if no more selected blocks.

Use `hypertext-card-get-first-selection` to get the first selection.

hypertext-card-get-no-lines

int (**hypertext-card-get-no-lines** *long* **card_id**)

Gets the number of lines currently displayed in the hypertext card.

hypertext-card-get-offset-position

long (**hypertext-card-get-offset-position** *long* **card_id**,
  long **line_pos**, *long* **char_pos**, *long* **offset**)

Given a position in the text and an offset from it, calculates the position in terms of line number and character position and returns 1 if successful.

`hypertext-card-get-current-line` and `hypertext-card-get-current-char` can be used to find the position.

hypertext-card-get-span-text

string (**hypertext-card-get-span-text** *long* **card_id**,
  long **line1**, *long* **char1**, *long* **line2**, *long* **char2**, *optional long* **convert_new_lines**)

Gets the text between the two positions, optionally converting newlines to spaces (the default if the final parameter is omitted).

hypertext-card-insert-text

int (**hypertext-card-insert-text** *long* **card_id**, *long* **line**, *long* **char**, *string* **text**)

Inserts the given text at the given line and character position.

Warning: This function has not been tested extensively and probably contain bugs.

hypertext-card-load-file

long (**hypertext-card-load-file** *long* **card_id**, *string* **filename**)

Loads the given hypertext (or plain) file onto the given hypertext card. Returns 1 for success, 0 for failure.

hypertext-card-save-file

long (**hypertext-card-save-file** *long* **card_id**, *string* **file**)

Saves the hypertext file on the given hypertext card in the specified file, returning 1 if successful, 0 otherwise.

hypertext-card-string-search

long (**hypertext-card-string-search** *long* **card_id**,
  string **search_string**, *optional long* **line_pos**, *optional long* **char_pos**)

Search for the given string from the given position, returning 1 if successful.

`hypertext-card-get-current-line` and `hypertext-card-get-current-char` can be used to retrieve the position of the matching text.

The search start position may be omitted, in which case the start position is taken to be the position of the previous match plus one.

The search is case-independent.

hypertext-card-translate

long (**hypertext-card-translate** *long* **card_id**, *word* **func**)

Starts the translation process for a hypertext card. *func* must be a function that takes four integer arguments: the card id, the event type, the current block type (if appropriate) and the current block id (if appropriate).

The event type is one of:

1. Start of block
2. End of block
3. Start of file
4. End of file
5. Double newline (which often means a paragraph break)

The callback function is responsible for opening and closing the file at the start and end of file, and outputting appropriate codes (such as HTML codes) at the start and end of blocks. Note that the block type passed is always -1 at the end of a block, so the programmer must maintain a stack of block types if he or she wishes to make use of the block type at the end of the block.

Use the function Section 12.14 to output text, Section 12.14 to open a file, and Section 12.14 to close a file. Up to two output streams may be opened.

hypertext-card-translator-close-file

long (**hypertext-card-translator-close-file** *long* **card_id**, *long* **which_file**)

Closes the translation output stream, identified by the number *which_file*.

hypertext-card-translator-open-file

long (**hypertext-card-translator-open-file** *long* **card_id**, *long* **which_file**, *string* **filename**)

Opens the translation output stream (identified by the number *which_file*).

hypertext-card-translator-output

long (**hypertext-card-translator-output** *long* **card_id**, *long* **which_file**, *string* **text**)
Outputs text on the translation output stream identified by the number *which_file*.
If *which_file* is -1, all open streams will be used.

12.14.1 Items

12.15 Hypertext card block functions

The following functions are relevant to hypertext blocks.

hypertext-block-add

long (**hypertext-block-add** *long* **card_id**,
long **line1**, *long* **char1**, *long* **line2**, *long* **char2**, *long* **block_type**)
Marks the given span of text as a block of the given type.
Note that if *block_type* has the value of 9999, the block will be a selection with no hypertext block or item. If the user deselects this selection (for example with shift-left click), the block will disappear without a trace. Subsequently setting a selection block type to a valid type identifier will turn the selection into a proper hypertext block.
The following values of *block_type* are recognised as standard:

1. hyBLOCK_NORMAL
2. hyBLOCK_RED
3. hyBLOCK_BLUE
4. hyBLOCK_GREEN
5. hyBLOCK_LARGE_HEADING
6. hyBLOCK_SMALL_HEADING
7. hyBLOCK_ITALIC
8. hyBLOCK_BOLD
9. hyBLOCK_INVISIBLE_SECTION
10. hyBLOCK_LARGE_VISIBLE_SECTION
11. hyBLOCK_SMALL_VISIBLE_SECTION
12. hyBLOCK_SMALL_TEXT
13. hyBLOCK_RED_ITALIC
14. hyBLOCK_TELETYPE

hypertext-block-clear

long (**hypertext-block-clear** *long* **card_id**, *long* **block_id**)
Clears the current block, returning 1 if successful. This deletes the hypertext item and links.

hypertext-block-get-item

long (**hypertext-block-get-item** *long* **card_id**, *long* **block_id**)
Given a hypertext card id and block id (*not* hyperitem id), get the Hardy hyperitem associated with the block. There may not be a hyperitem associated with the block if the user has made an initial, temporary selection. A Hardy hyperitem is not created until the block type has been set.

hypertext-block-get-text

string (**hypertext-block-get-text** *long* **card_id**, *long* **block_id**)
Given a hypertext card id and block id (*not* hyperitem id), get the plain text within the block (up to a limit of 1000 characters).

hypertext-block-get-type

long (**hypertext-block-get-type** *long* **card_id**, *long* **block_id**)
Given a hypertext card id and block id (*not* hyperitem id), get the block's type (the number used to identify the mapping to text colours and styles).

hypertext-block-selected

long (**hypertext-block-selected** *long* **card_id**, *long* **block_id**)
Given a hypertext card id and block id, return 1 if the block is selected or 0 if it is not.

hypertext-block-set-type

long (**hypertext-block-set-type** *long* **card_id**, *long* **block_id**, *long* **type_id**)
Given a hypertext card id and block id (*not* hyperitem id), sets the block's type (the number used to identify the mapping to text colours and styles), deselects the block if selected, and 'recompiles' and displays the file. Recompilation involves scanning the entire file in order to resolve block scope and compute actual font and colour information, and is necessary if the text attributes change in any way (including selection/deselection). The display position may change as a side effect of this call, and any other call involving recompilation.

12.16 Hypertext card item functions

The following functions are relevant to hypertext card items.

hypertext-item-get-block

long (**hypertext-item-get-block** *long* **card_id**, *long* **item_id**)

Given a hypertext card id and hyperitem id, get the block associated with the item. There may not be a block associated with the item if the item is the special item (used for card linking without using an explicit item).

12.17 Media card functions

The following functions are relevant to media cards. The media card is an experimental card which will eventually replace the hypertext card: it is editable and has more features than the hypertext card. It is based on a set of media classes written by Matthew Flatt of Rice University.

Note that this facility will not be included in distributions of Hardy outside AIAI until early 1996.

Media cards allow mark up using either standard attributes such as weight, family, style and underlining, or attributes that are combined into *font mappings* in the same way as the hypertext card. Blocks are associated with the latter but not the former.

Some media card functions accept a *position*, a single integer representing a character index in the buffer. A position can be converted into a line number and character position within that line.

12.17.1 Events

These are the media card events you can intercept.

Event	Description
BlockLeftClick	Called when a block is left-clicked. Takes card, block id, position, shift (1 or 0), control (1 or 0). Return 0 to veto default event processing, 1 otherwise.
BlockRightClick	Called when a block is right-clicked. Takes card, block id, position, shift (1 or 0), control (1 or 0). Return 0 to veto default event processing, 1 otherwise.
CustomMenu	Called when a custom menu item is invoked. Takes card and menu item name.

media-block-create

long (**media-block-create** *long* **card_id**, *long* **block_type**, *optional long* **start_position=-1**, *optional long* **end_position=-1**)

Creates a block of the given type, returning the new block id. start_position and end_position specify the span of the block; if they are both -1 or absent, the current selection will be used.

media-block-get-item

long (**media-block-get-item** *long* **card_id**, *long* **block_id**)

Returns the hypertext item for the given block id.

media-block-get-position

long (**media-block-get-position** *long* **card_id**, *long* **block_id**)
Returns the start position of the block.

media-block-get-type

long (**media-block-get-type** *long* **card_id**, *long* **block_id**)
Returns the type id of the block.

media-block-set-type

long (**media-block-set-type** *long* **card_id**, *long* **block_id**, *long* **block_type**)
Sets the type of the block. Currently does *not* redraw the block in the new style.

media-item-get-block

long (**media-item-get-block** *long* **card_id**, *long* **item_id**)
Gets the block corresponding to the hypertext item.

media-card-append-text

long (**media-card-append-text** *long* **card_id**, *string* **text**)
Appends the given text at the end of the card's contents.

media-card-apply-family

long (**media-card-apply-family** *long* **card_id**, *string* **family**, *long* **from=-1**, *long* **to=-1**)
Applies the given family to the current selection (if the *from* and *to* parameters are omitted or are -1) or to the given span of text.

family may be one of wxSWISS, wxROMAN, wxDECORATIVE and wxMODERN.

media-card-apply-foreground-colour

long (**media-card-apply-foreground-colour** *long* **card_id**, *string* **colour**, *long* **from=-1**, *long* **to=-1**)
Applies the given colour to the current selection (if the *from* and *to* parameters are omitted or are -1) or to the given span of text.

media-card-apply-point-size

long (**media-card-apply-point-size** *long* **card_id**, *long* **size**, *long* **from=-1**, *long* **to=-1**)
Applies the given point size to the current selection (if the *from* and *to* parameters are omitted or are -1) or to the given span of text.

media-card-apply-style

long (**media-card-apply-style** *long* **card_id**, *string* **style**, *long* **from=-1**, *long* **to=-1**)

Applies the given font style to the current selection (if the *from* and *to* parameters are omitted or are -1) or to the given span of text.

style may be one of wxNORMAL, wxITALIC.

media-card-apply-underline

long (**media-card-apply-underline** *long* **card_id**, *long* **underline**, *long* **from=-1**, *long* **to=-1**)

Applies underlining to the current selection (if the *from* and *to* parameters are omitted or are -1) or to the given span of text.

underline may be 1 for underlining, 0 for no underlining.

media-card-apply-weight

long (**media-card-apply-weight** *long* **card_id**, *string* **weight**, *long* **from=-1**, *long* **to=-1**)

Applies normal or bold weight to the current selection (if the *from* and *to* parameters are omitted or are -1) or to the given span of text.

weight may be wxNORMAL or wxBOLD.

media-card-clear

long (**media-card-clear** *long* **card_id**)

Clears the contents of the card.

media-card-clear-all-blocks

long (**media-card-clear-all-blocks** *long* **card_id**)

Clears the blocks from the card, leaving a plain text file with no styles or graphic images.

media-card-create

long (**media-card-create** *long* **parent_id**, *string* **media_type**, *optional long* **iconic = 0**)

Creates a new media card and returns the id, or -1 if the call failed. *parent_id* may be zero (no parent) or a valid parent card id. *hypertext_type* should be a valid media type, as defined using the interactive Media Type Manager.

If *iconic* is 1, the card will be shown in the iconic state.

media-card-copy

long (**media-card-copy** *long* **card_id**)

Copies the selected text to the clipboard.

media-card-cut

long (**media-card-cut** *long* **card_id**)

Copies the selected text to the clipboard, and then clears the selection.

media-card-delete

long (**media-card-delete** *long* **card_id**, *long* **from=-1**, *long* **to=-1**)

Deletes the current selection (if the *from* and *to* parameters are omitted or are -1), or given span of text if the parameters are specified.

media-card-find-string

long (**media-card-find-string** *long* **card_id**, *string* **text**, *long* **from=-1**, *long* **to=-1**, *long* **case_sensitive=1**, *long* **direction=1**)

Finds the given *text*, starting the search from the beginning if *from* is absent or -1, and continuing until the end if *to* if is absent or -1.

case_sensitive should be 1 to be case sensitive, 0 to be case insensitive. *direction* should be 1 to search forward, -1 to search backward.

The return value is the text start position if the text was found, or -1 if the text was not found.

media-card-get-character

long (**media-card-get-character** *long* **card_id**, *long* **position**)

Returns the ASCII code of the character at the given position.

media-card-get-selection-start

long (**media-card-get-selection-start** *long* **card_id**)

Returns the position of the start of the selection, or the cursor position if there is no selection.

media-card-get-selection-end

long (**media-card-get-selection-end** *long* **card_id**)

Returns the position of the end of the selection, or -1 if there is no selection.

media-card-get-first-block

long (**media-card-get-first-block** *long* **card_id**)

Returns the first block id (not necessarily the first in the card, but first from the point of view of getting all blocks).

media-card-get-last-position

long (**media-card-get-last-position** *long* **card_id**)
Returns the position of the last element in the card. This will never be less than zero.

media-card-get-line-length

long (**media-card-get-line-length** *long* **card_id**, *long* **line**)
Returns the length of the given line (starting from zero).

media-card-get-line-for-line-position

long (**media-card-get-line-for-line-position** *long* **card_id**, *long* **position**)
Returns the number of the line on which *position* appears.

media-card-get-next-block

long (**media-card-get-next-block** *long* **card_id**)
Returns the next block id (not necessarily the next in the card, but next from the point of view of getting all blocks).

media-card-get-position-for-position-line

long (**media-card-get-position-for-position-line** *long* **card_id**, *long* **line**)
Returns the start position for the given line.

media-card-get-number-of-number-lines

long (**media-card-get-number-of-number-lines** *long* **card_id**)
Returns the total number of lines in the text card.

media-card-get-text

string (**media-card-get-text** *long* **card_id**, *long***start**, *long***end**)
Returns the text between the given positions.

media-card-insert-text

long (**media-card-insert-text** *long* **card_id**, *string***text**, *long***pos=-1**)
Inserts text at the given position, or at the cursor if *pos* is -1.

media-card-insert-image

long (**media-card-insert-image** *long* **card_id**, *string***filename**, *long***pos=-1**)
Inserts a Windows bitmap at the given position, or at the cursor if *pos* is -1.

media-card-load-file

long (**media-card-load-file** *long* **card_id**, *string***filename**)
Loads a file into the media card. Can be a plain text file or a media file (usual extension .med).

media-card-paste

long (**media-card-paste** *long* **card_id**)
Pastes the contents of the clipboard (if there is any and it is textual) into the media card.

media-card-redo

long (**media-card-redo** *long* **card_id**)
Redoes the last media card command (except for block operations).

media-card-save-file

long (**media-card-save-file** *long* **card_id**, *string***filename**)
Saves the media file.

media-card-scroll-to-position

long (**media-card-scroll-to-position** *long* **card_id**, *long***position**)
Scrolls to the given position.

media-card-select-block

long (**media-card-select-block** *long* **card_id**, *long***block**, *long***select=1**)
Selects the given block if *select* is 1, deslects if *select* is 0.

media-card-set-selection

long (**media-card-set-selection** *long* **card_id**, *long***from**, *long***to**)
Sets the text between the given positions.

media-card-undo

long (**media-card-undo** *long* **card_id**)
Undoes the last media card command (except for block operations).

12.18 Text card functions

These functions may be used with text cards.

text-card-load-file

long (**text-card-load-file** *long* **card_id**, *string* **file**)
Loads the specified text file onto the given text card. Returns 1 if successful, 0 otherwise.

12.19 Diagram Definition functions

The following functions access diagram definition information.

hardy-diagram-definition-get-first-get-arc-type

string (**hardy-diagram-definition-get-first-get-arc-type** *string* **name**)
For the given diagram type, gets the first arc name in the diagram definition's list of arc types.
Returns the empty string if none are found.

hardy-diagram-definition-get-next-get-arc-type

string (**hardy-diagram-definition-get-next-get-arc-type**)
Gets the next arc name in the diagram definition's list of arc types.
Returns the empty string if no more are found.

hardy-diagram-definition-get-first-get-node-type

string (**hardy-diagram-definition-get-first-get-node-type** *string* **name**)
For the given diagram type, gets the first node name in the diagram definition's list of node types.
Returns the empty string if none are found.

hardy-diagram-definition-get-next-get-node-type

string (**hardy-diagram-definition-get-next-get-node-type**)
Gets the next node name in the diagram definition's list of node types.
Returns the empty string if no more are found.

hardy-get-first-diagram-definition

string (**hardy-get-first-diagram-definition**)
Gets the first name in the list of diagram definitions currently loaded.
Returns the empty string if there are none loaded.

hardy-get-next-diagram-definition

string (**hardy-get-next-diagram-definition**)
Gets the next name in the list of diagram definitions currently loaded.
Returns the empty string when there are no more.

object-type-get-first-attribute-first-name

string (**object-type-get-first-attribute-first-name** *string* **diagram_def_name** *string* **object_type_name** *optional string* **node_or_arc**)

Gets the first attribute name of an object (node or arc) type definition. If *node_or_arc* is “any”, this function will search for either a node or arc of the given name. Otherwise, you can specify “node” or “arc” to be more specific.

object-type-get-next-attribute-next-name

string (**object-type-get-next-attribute-next-name**)

Gets the next attribute name of an object (node or arc) type definition.

12.20 Windows printing functions

The following functions and event handlers enable Windows printing to be manipulated.

12.20.1 Windows printing event handlers

- OnPreparePrinting: called before any pages are printed. The function takes no arguments. The function should return 1 (processed) or 0 (default processing should be done).
- OnPrintPage: called when each page should be printed. Takes page number (integer), device context (integer), page width in pixels (integer), page height in pixels (integer), page width in mm (integer), page height in mm (integer). The function should return 1 (processed) or 0 (default processing should be done).
- OnGetPageInfo: called to get information from the application. Takes a name and an integer value. Return -1 to allow default processing; otherwise: if the name is HASPAGE, return 1 if the document has this page (given by the second argument), or 0 to finish printing. If the name is MINPAGE, MAXPAGE, PAGEFROM, PAGETO, return an appropriate value. See the printing demo in the HARDY SDK for more details.

hardy-preview-all

long (**hardy-preview-all** *long* **Page_From**, *long* **Page_To**)

Invokes Windows previewing for the given page range, for all diagram cards if no custom printing functionality is defined, or for whatever pages the application defines by intercepting the OnPrintPage event handler.

If the page range values are both -1, the entire document will be previewed. Unlike with printing, flow of program control continues immediately after the preview window appears, so be careful not to call the function again.

hardy-preview-diagram-card

long (**hardy-preview-diagram-card** *long* **card**)

Invokes Windows previewing for the given diagram card. Unlike with printing, flow of program control continues immediately after the preview window appears, so be careful not to call the function again.

hardy-print-all

long (**hardy-print-all** *long* **prompt**, *long* **page_from**, *long* **page_to**)

Invokes Windows printing for the given page range, for all diagram cards if no custom printing functionality is defined, or for whatever pages the application defines by intercepting the OnPrintPage event handler.

If *prompt* is 1, the standard print dialog box is shown; otherwise, the document will be printed immediately. If the page range values are both -1, the entire document will be printed.

Unlike with previewing, flow of program control stops until printing has finished (or the user cancels the dialog box).

hardy-print-diagram-card

long (**hardy-print-diagram-card** *long* **card**, *long* **prompt**)

Invokes Windows printing for the given diagram card.

If *prompt* is 1, the standard print dialog box is shown; otherwise, the diagram will be printed immediately.

hardy-print-diagram-in-box

long (**hardy-print-diagram-in-box** *long* **card**, *double* **x**, *double* **y**, *double* **width**, *double* **height**)

This function should be used from within an OnPrintPage event handler to scale and position the given diagram on the page. The *x, y* coordinate represents the top left of the bounding box to contain the diagram. The units are in device units (pixels). It should not be called from outside OnPrintPage since it implicitly references the current print or preview device context.

hardy-print-diagram-page

long (**hardy-print-diagram-page** *long* **card**, *long* **page_num**)

This function should be used from within an OnPrintPage event handler to call the default diagram printing code for this page. It should not be called from outside OnPrintPage since it implicitly references the current print or preview device context.

hardy-print-get-header-footer

string (**hardy-print-get-header-footer** *long* **field**)

This function should be used to get the value of a header or footer field, used when printing a standard diagram page or printing the headers and footers with Section 12.20.1 .

field should be an integer between 1 and 6, referencing the left, middle and right fields of the header and footer respectively.

hardy-print-get-info

double (**hardy-print-get-info** *string* **name**)

Returns information according to the *name* argument passed. The value of *name* can be:

- TEXTSCALE: returns an appropriate scaling factor for printing text. It sets the scaling for the printer or preview device context, and returns the scaling factor. Note that the factor *returned* does not include the adjustment made for scaling for a preview device context.

- LEFTMARGIN: returns the value of the left margin setting, in millimetres.
- RIGHTMARGIN: returns the value of the right margin setting, in millimetres.
- TOPMARGIN: returns the value of the top margin setting, in millimetres.
- BOTTOMMARGIN: returns the value of the bottom margin setting, in millimetres.
- HEADERRULE: returns 1 or 0 for header rule on or off.
- FOOTERRULE: returns 1 or 0 for footer rule on or off.

hardy-print-header-footer

long (**hardy-print-header-footer** *long* **page_num**)
This function should be used from within an OnPrintPage event handler to call the default header and footer printing code for this page. It should not be called from outside OnPrintPage since it implicitly references the current print or preview device context.

You can use Section 12.20.1 and Section 12.20.1 to change the look of headers and footers for this page.

hardy-print-set-header-footer

long (**hardy-print-set-header-footer** *string* **text**, *long* **field**)
This function should be used to set a header or footer field, used when printing a standard diagram page or printing the headers and footers with Section 12.20.1 .

field should be an integer between 1 and 6, referencing the left, middle and right fields of the header and footer respectively.

hardy-print-set-info

long (**hardy-print-set-info** *string* **name**, *float* **value**)
Sets printing information according to the *name* argument passed. The value of *name* can be:

- LEFTMARGIN: sets the value of the left margin setting, in millimetres.
- RIGHTMARGIN: sets the value of the right margin setting, in millimetres.
- TOPMARGIN: sets the value of the top margin setting, in millimetres.
- BOTTOMMARGIN: sets the value of the bottom margin setting, in millimetres.
- HEADERRULE: 1 or 0 for header rule on or off.
- FOOTERRULE: 1 or 0 for footer rule on or off.

Note that margin settings are only used automatically when printing a diagram page or headers and footers. For custom pages, these margins must be taken into account by the custom code.

hardy-print-set-title

long (**hardy-print-set-title** *string* **title**)

This function should be used to set the current page title (used when printing a standard diagram page).

hardy-print-text-in-box

long (**hardy-print-text-in-box** *string* **text**, *double* **x**, *double* **y**, *double* **width**, *double* **height**, *string* **how**)

This function should be used from within an OnPrintPage event handler to format text within the given bounding box using the current device context scaling and font. The *x, y* coordinate represents the centre of the bounding box to contain the text.

The *how* parameter should be one of CENTREHORIZ, CENTREVERT, CENTREBOTH and NONE to determine how the text should be formatted.

The units are in device units (pixels). This function should not be called from outside OnPrintPage since it implicitly references the current print or preview device context.

12.21 Miscellaneous functions

The following are miscellaneous Hardy functions.

convert-bitmap-to-rtf

long (**convert-bitmap-to-rtf** *string* **bitmap-file**, *string* **output-file**)
Converts an existing RGB-encoded Windows bitmap file to RTF format for inclusion in an RTF document.

convert-metafile-to-rtf

long (**convert-metafile-to-rtf** *string* **metafile-file**, *string* **output-file**)
Converts an existing placeable Windows metafile file to RTF format for inclusion in an RTF document.

dde-advise-global

long (**dde-advise-global** *char ** **item**, *char ** **data**)
Sends a DDE ADVISE message to all connections currently using Hardy as a server. The client can process these messages (or ignore them). If Hardy were to be used as a user interface to some other client package, the client could call Hardy functions through the DDE interface (or via a program called *DDEPIPE* which allows non-DDE aware UNIX applications to access DDE programs using simple commands). The client could wait for ADVISE messages back (for example when a custom menu item was selected), and then do further processing or call additional Hardy functions.

hardy-command-int-to-string

long (**hardy-command-int-to-string** *long* **command_id**)
Converts an integer command identifier into a command name, such as HardyExit or DiagramCut.

hardy-command-string-to-int

long (**hardy-command-string-to-int** *string* **command_name**)
Converts a command name, such as HardyExit or DiagramCut, into the integer identifier form.

hardy-get-browser-frame

long (**hardy-get-browser-frame**)
Returns the integer id of the Hardy card browser frame, which can then be passed to GUI functions such as `window-show`.

This is only different from the top level frame if Hardy is running under Windows MDI mode, where the top level frame encloses other frames and the browser window is a separate child frame.

hardy-get-top-level-frame

long (**hardy-get-top-level-frame**)

Returns the integer id of the top level Hardy frame, which can then be passed to GUI functions such as `window-show`.

hardy-get-version

double (**hardy-get-version**)

Returns a floating-point number representing the version of Hardy that the application code is currently running under.

hardy-path-search

string (**hardy-path-search** *string* **filename**)

Searchs the current Hardy path list for the given file, and if it exists, returns the full pathname. Hardy builds up a list of paths as files become known to it; so sometimes Hardy will load files that do not have absolute paths, which CLIPS programs would not find without this function.

Returns the empty string if the file is not found.

hardy-help-display-block

long (**hardy-help-display-block** *long* **block_id**)

The given block is displayed. It is best to call `hardy-help-load-file` before this call. It is probably better to use section numbers than block numbers, unless a block other than a section must be displayed.

The value 1 is returned if successful, otherwise 0.

hardy-help-display-contents

long (**hardy-help-display-contents**)

The contents (first section) is displayed. It is best to call `hardy-help-load-file` before this call.

The value 1 is returned if successful, otherwise 0.

hardy-help-display-section

long (**hardy-help-display-section** *long* **section**)

The given section (numbered 1 upwards) is displayed. It is best to call `hardy-help-load-file` before this call.

The value 1 is returned if successful, otherwise 0.

hardy-help-keyword-search

long (**hardy-help-keyword-search** *string* **keyword**)

Performs a keyword search on section titles. If more than one matching title is found, the search dialog is displayed; otherwise, that section is displayed.

hardy-help-load-file

long (**hardy-help-load-file** *string* **file**)

If wxHelp is not currently running, it is executed. The named file is then loaded if it is an absolute path, or found in the current directory, or found in a directory mentioned in the WXHELPFILES or PATH directories.

If the file is already loaded into wxHelp, it is not reloaded, and therefore this function can (and should) always be called before attempting to display a section or block, since the user may have loaded another file.

Note that there is no function to quit the help system programmatically; wxHelp will be closed when Hardy closes, except under Windows where there is only one copy of wxHelp active at a time.

The value 1 is returned if successful, otherwise 0.

hardy-send-command

long (**hardy-send-command** *long* **command_id**)

Sends a menu command identifier to the Hardy main window. *command_id* is an internal identifier that can be obtained from an equivalent string form using Section 12.21 .

This function can be used in custom code to provide features that the default user interface normally provides.

See Section 12.22 for a list of identifiers you can use in conjunction with this function.

hardy-set-about-string

long (**hardy-set-about-string** *string* **about_string**)

Sets the text for the 'About box' invoked from the main window's Help menu.

hardy-set-author

long (**hardy-set-author** *string* **author**)

Sets the custom author name.

hardy-set-custom-help-file

long (**hardy-set-custom-help-file** *string* **file**)

Sets the filename for the custom help file. The name should have no extension (so an appropriate format will be used for the platform). This is the file used in the main window's Help menu.

hardy-set-help-file

long (**hardy-set-help-file** *string* **file**)

Sets the filename for the normal Hardy help file. The name should have no extension (so an appropriate format will be used for the platform). This is the file used in the Hardy-specific menus and dialog boxes; it might be overidden for a heavily customised version of the tool.

hardy-set-name

long (**hardy-set-name** *string* **name**)

Sets the custom tool name (default is "Hardy").

hardy-set-title

long (**hardy-set-title** *string* **title**)

Sets the custom tool title (default is "Hardy"). Used in the main window title bar.

object-is-valid

long (**object-is-valid** *long* **card_id**, *long* **object_id**)

Given a card id and an object id (a diagram node, arc or image id), returns 1 if the object exists and 0 otherwise.

quit

long (**quit** *int* **quit_level**)

Quits from Hardy, return 1 if successful, 0 otherwise.

Action depends on value of quit_level:

0: full user prompting,

1: save everything and quit without prompting,

2: don't save anything and quit without prompting.

register-event-handler

long (**register-event-handler** *string* **event_type**, *string* **context**,
word **function_name**)

Registers interest in a given Hardy event for the given card type. *context* may be one of:

1. "Toplevel",
2. "Text card",
3. any valid diagram type,
4. any valid hypertext type.

The function name specifies a valid function whose arguments, when called by Hardy, will vary according to the event type, but which will usually start with the card id.
Top level events recognised:

`Exit` Called when Hardy is about to exit, after all cards and the index have been saved (assuming the user did not veto these saves). The function has no arguments but returns an integer, which is 0 to veto the exit command or 1 to confirm the exit.

`CustomMenu` When the user selects a custom menu item on the top level control window, the named function will be called with one argument: the *menu item string* that the user selected. At present there is no user interface to edit the top-level custom menu; edit `diagrams.def` with a text editor and insert (for example):

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

`OnCreateMenuBar` Register this event to create a custom main menu. The function is called with no arguments, and should create and return a wxCLIPS menu bar, or zero to allow the default menu bar to be created.

`OnCreateToolBar` Register this event to create a custom main window toolbar (Windows only). The function is called with the frame identifier, and should create and return a panel or canvas, or zero to allow the default toolbar to be created. The initial height of the returned window will be used to determine sizing, and the width will be made to fit the main window.

`OnHardyInit` Called after Hardy initialisation has taken place. It is called with no arguments, and must return 1 for Hardy to continue running. Returning 0 terminates the session. Depending on the underlying window system, it may or may not be possible to minimize or hide the main window at this point. If your custom code needs to start running on startup, use this event to start it, rather than at CLIPS loading time which will not allow the initialisation to terminate.

`OnMenuCommand` Called when the user selects an option on the main window. It is called with an integer identifier representing the command, and the function should return 0 to veto normal processing, or 1 to perform the default action.

`OnPreparePrinting` See Section 12.20.1 .

`OnPrintPage` See Section 12.20.1 .

`OnGetPageInfo` See Section 12.20.1 .

For diagram cards, the event type may be:

`AddArcAnnotation` Called when the user adds an annotation to an arc, with arguments *card id, node image id, annotation id, annotation name, drop site name.* If the user function returns 0, the annotation is vetoed. Returning 1 lets the annotation addition take place.

`AddNodeAnnotation` Called when the user adds an annotation to a node, with arguments *card id, node image id, annotation id, annotation name, drop site name.* If the user function returns 0, the annotation is vetoed. Returning 1 lets the annotation addition take place.

`ArcLeftClick` Called when the arc is left-clicked, with arguments *card id, arc image id, x, y, shift pressed (1 or 0), control pressed (1 or 0).* If the user function returns 0, the default left click action is not performed. Returning 1 lets the default behaviour take place.

`ArcMoveControlPoint` Called when a control point is moved on this arc, or an attached node is moved. The callback is invoked with the arguments *card id, arc type, arc image id, control point, x, y.* The control point id is an integer greater than or equal to zero.

`ArcRightClick` Called when the arc is right-clicked, with arguments *card id, arc image id, x, y, shift pressed (1 or 0), control pressed (1 or 0).* If the user function returns 0, the default right click action is not performed. Returning 1 lets the default behaviour take place.

`AttributesUpdated` When the user has finished editing attributes for a node or arc, this is called with arguments *card id*, *object id* and *object type*. Note that this is only called, once, when the user closes the standard attribute editor, and not at any time.

`AttributeModifiedPre` This is called as a 'daemon' just before an attribute is changed, either by the user or programmatically. If the value is the same as the old, the function will not be called. The function is called with arguments *card id*, *object id*, *attribute name*, *old value*, *new value*. If the function returns 0, the modification will be vetoed: the function must return 1 to allow the update.

`AttributeModifiedPost` This is called as a 'daemon' just after an attribute is changed, either by the user or programmatically. If the value is the same as the old, the function will not be called. The function is called with arguments *card id*, *object id*, *attribute name*, *old value*, *new value*. The function should return no value.

`ContainerAddPost` This is called after a node image has been added to a division. The function is called with arguments *card id*, *parent image id*, *child image id*, *division image id*.

`ContainerAddPre` This is called when a node image is about to be added to a division. The function should return 0 to veto, or 1 to allow the containment operation. The function is called with arguments *card id*, *parent image id*, *child image id*, *division image id*.

`ContainerRemovePost` This is called after a node image has been removed from a division. The function is called with arguments *card id*, *parent image id*, *child image id*, *division image id*.

`ContainerRemovePre` This is called when a node image is about to be removed from a division. The function should return 0 to veto, or 1 to allow the containment operation.

The function is called with arguments *card id*, *parent image id*, *child image id*, *division image id*.

`CreateCard` After diagram card creation, the named function is called with one argument: the *card id*. Note that this does not get called when a card is loaded, only when the card is created interactively.

`CustomMenu` When the user selects a custom menu item, the named function will be called with two arguments: the *card id* and the *menu item string* that the user selected. Use the diagram type manager to specify a custom menu for a particular diagram type.

`DeleteArcAnnotation` Called when the user deletes an arc annotation, with arguments *card id, arc image id, annotation id.* If the user function returns 0, the annotation deletion is vetoed. Returning 1 lets the annotation deletion take place.

`DeleteCard` Just before diagram card deletion, the named function is called with one argument: the *card id*. The function should return an integer, 0 to veto the delete (may be overriden by Hardy) and 1 to continue.

`DeleteNodeAnnotation` Called when the user deletes a node annotation, with arguments *card id, arc image id, annotation id.* If the user function returns 0, the annotation deletion is vetoed. Returning 1 lets the annotation deletion take place.

`CanvasLeftClick` Called when the mouse is left-clicked on the card canvas. The function's arguments are *card id, x, y, shift pressed (1 or 0), control pressed (1 or 0).* Return 0 to veto normal processing, 1 otherwise.

`CanvasRightClick` Called when the mouse is mouse-clicked on the card canvas. The function's arguments are *card id, x, y, shift pressed (1 or 0), control pressed (1 or 0).* Return 0 to veto normal processing, 1 otherwise.

`CreateNodeImage` After node image creation, the named function is called with three arguments: *card id, image id, node type.* Note that this does not get called when a diagram is loaded, only when images are created interactively.

`CreateNodeImagePre` After node image creation, the named function is called with three arguments: *card id, image id, node type.* The difference between this function and CreateNodeImage is that the function must return 0 or 1. If 0 is returned, Hardy deletes the image, otherwise normal processing continues. Note that this does not get called when a diagram is loaded, only when images are created interactively.

`CreateArcImage` After arc image creation, the named function is called with three arguments: *card id, image id, arc type.* Note that this does not get called when a diagram is loaded, only when images are created interactively.

`CreateArcImagePre` After arc image creation, the named function is called with three arguments: *card id, image id, arc type.* The difference between this function and CreateArcImage is that the function must return 0 or 1. If 0 is returned, Hardy deletes the image, otherwise normal processing continues. Note that this does not get called when a diagram is loaded, only when images are created interactively.

`DeleteNodeImage` Just before node image deletion, the function is called with arguments *card id, image id, node type.* Arcs are still accessible at this point, although DeleteArcImage events may be generated as a result.

`DeleteNodeImagePost` Just after node image deletion, the function is called with arguments *card id, image id, node type. image id* is invalid at this point.

`DeleteArcImage` Just before arc image deletion, the function is called with arguments *card id, image id, arc type.*

`DeleteArcImagePost` Just after arc image deletion, the function is called with arguments *card id, image id, arc type. image id* is invalid at this point.

`LoadDiagram` When a diagram has just been loaded, the function is called with the card id as argument.

`NodeMovePre` Called when the node is moved but before it is redrawn, with arguments *card id, node image id, x, y, old x, old y.* Returning 1 lets the default behaviour take place; returning 0 vetoes the move.

`NodeMovePost` Called when the node is moved, after it is redrawn, with arguments *card id, node image id, x, y, old x, old y.* Return 1 from this function.

`NodeLeftClick` Called when the node is left-clicked, with arguments *card id, node image id, x, y, shift pressed (1 or 0), control pressed (1 or 0).* If the user function returns 0, the default left click action is not performed. Returning 1 lets the default behaviour take place.

`NodeRightClick` Called when the node is right-clicked, with arguments *card id, node image id, x, y, shift pressed (1 or 0), control pressed (1 or 0).* If the user function returns 0, the default right click action is not performed. Returning 1 lets the default behaviour take place.

`OnCreateMenuBar` Register this event to create a custom card menu. The function is called with the card identifier, and should create and return a wxCLIPS menu bar, or zero to allow the default menu bar to be created.

`OnCreateToolBar` Register this event to create a custom card toolbar (Windows only). The function is called with the Hardy card identifer and wxCLIPS frame identifier, and should create and return a panel or canvas, or zero to allow the default toolbar to be created. The initial height of the returned window will be used to determine sizing, and the width will be made to fit the card window.

`OnMenuCommand` Called when the user selects an option on the card. It is called with a card id, and an integer identifier representing the command. The function should return 0 to veto normal processing, or 1 to perform the default action.

`RightDragCanvasToCanvas` Called when the mouse is right-dragged from somewhere on the canvas, and released on another part of the canvas. The function's arguments are *card id, initial x, initial y, final x, final y, shift pressed (1 or 0), control pressed (1 or 0).*

`RightDragCanvasToNode` Called when the mouse is right-dragged from somewhere on the canvas, and released on a node image. The function's arguments are *card id, node image id, node attachment, x, y, shift pressed (1 or 0), control pressed (1 or 0).*

`RightDragNodeToCanvas` Called when the mouse is right-dragged from a node image, and released on the canvas. The function's arguments are *card id, node image id, node attachment, x, y, shift pressed (1 or 0), control pressed (1 or 0). x* and *y* represent the position of the mouse when released.

`RightDragNodeToNode` Called when the mouse is right-dragged from a node image and released on another node image. The function's arguments are *card id, first node image id, first node attachment, second node image id, second node image attachment, x, y, shift pressed (1 or 0), control pressed (1 or 0)*. *x* and *y* represent the position of the mouse when released. This function must return 1 to let processing continue, or 0 to override normal Hardy behaviour.

`SaveDiagram` When a diagram is about to be saved, the function is called with the card id as argument. If the function returns 1, saving continues; if 0 is returned, saving is aborted.

`SelectNodeImage` Called after a node image has been selected or deselected, either by the user or programmatically. The function is called with three arguments: *card id, image id, selection flag (0 or 1)*. No value need be returned.

`SelectArcImage` Called after an arc image has been selected or deselected, either by the user or programmatically. The function is called with three arguments: *card id, image id, selection flag (0 or 1)*. No value need be returned.

For hypertext cards, the event types may be:

`CreateCard` After hypertext card creation, the named function is called with one argument: the *card id*. Note that this does not get called when a card is loaded, only when the card is created interactively.

`DeleteCard` Just before hypertext card deletion, the named function is called with one argument: the *card id*. The function should return an integer, 0 to veto the delete (may be overriden by Hardy) and 1 to continue.

`BlockLeftClick` Called when a block is left-clicked, with arguments *card id, block id, character position, line number, shift pressed (1 or 0)* and *control pressed (1 or 0)*. If the user function returns 0, the default left click action is not performed. Returning 1 lets the default behaviour take place. A block id of -1 indicates a click on unmarked text.

`BlockRightClick` Called when a block is right-clicked, with arguments *card id, block id, character position, line number, shift pressed (1 or 0), control pressed (1 or 0)*. If the user function returns 0, the default left click action is not performed. Returning 1 lets the default behaviour take place. A block id of -1 indicates a click on unmarked text.

`CustomMenu` When the user selects a custom menu item, the named function will be called with two arguments: the *card id* and the *menu item string* that the user selected. Use the hypertext type manager to specify a custom menu for a particular hypertext type.

`OnCreateMenuBar` Register this event to create a custom card menu. The function is called with the card identifier, and should create and return a wxCLIPS menu bar, or zero to allow the default menu bar to be created.

`OnMenuCommand` Called when the user selects an option on the card. It is called with the card id, and an integer identifier representing the command. The function should return 0 to veto normal processing, or 1 to perform the default action.

`OnCreateToolBar` Register this event to create a custom card toolbar (Windows only). The function is called with the Hardy card identifer and wxCLIPS frame identifier, and should create and return a panel or canvas, or zero to allow the default toolbar to be created. The initial height of the returned window will be used to determine sizing, and the width will be made to fit the card window.

For media cards, the event type may be:

`CustomMenu` When the user selects a custom menu item, the named function will be called with two arguments: the *card id* and the *menu item string* that the user selected. Use the diagram type manager to specify a custom menu for a particular diagram type.

`OnCreateMenuBar` Register this event to create a custom card menu. The function is called with the card identifier, and should create and return a wxCLIPS menu bar, or zero to allow the default menu bar to be created.

`OnCreateToolBar` Register this event to create a custom card toolbar (Windows only). The function is called with the Hardy card identifer and wxCLIPS frame identifier, and should create and return a panel or canvas, or zero to allow the default toolbar to be created. The initial height of the returned window will be used to determine sizing, and the width will be made to fit the card window.

`OnMenuCommand` Called when the user selects an option on the card. It is called with a card id, and an integer identifier representing the command. The function should return 0 to veto normal processing, or 1 to perform the default action.

`BlockLeftClick` Called when a block is left-clicked. Takes card, block id, position, shift (1 or 0), control (1 or 0). Return 0 to veto default event processing, 1 otherwise.

`BlockRightClick` Called when a block is right-clicked. Takes card, block id, position, shift (1 or 0), control (1 or 0). Return 0 to veto default event processing, 1 otherwise.

`CustomMenu` Called when a custom menu item is invoked. Takes card and menu item name.

12.22 Menu command identifiers

Most of the menu commands that the user can issue have names which can be used by custom code which replaces the default menu bars. When responding to the OnMenuCommand event for the main window or cards, custom code can call Section 12.21 or Section 12.2 to invoke standard functionality. These functions, and the OnMenuCommand event handler, take integer command arguments, so you will need to use Section 12.21 and Section 12.21 to send or test commands.

In order to avoid confusion with Hardy integer identifiers, please note that replacement main window or card menu bar integer identifiers should start from at least 800.

The following sections list the menu command names.

12.22.1 Hardy main window commands

- HardyBrowseFiles
- HardyClearIndex
- HardyConfigure
- HardyDeselectAllItems
- HardyDrawTree
- HardyExit
- HardyFindOrphans
- HardyHelpAbout
- HardyHelpContents
- HardyHelpSearch
- HardyLoadApplication
- HardyLoadFile
- HardyPrint
- HardyPrintPreview
- HardyPrintSetup
- HardySaveFile
- HardySaveFileAs
- HardySearchCards
- HardyShowArcSymbolEditor
- HardyShowDevelopmentWindow
- HardyShowDiagramManager
- HardyShowHypertextManager
- HardyShowNodeSymbolEditor
- HardyShowPackageTool
- HardyShowSymbolLibrarian
- HardyViewTopCard

12.22.2 Generic card commands

- CardDeleteAllLinks
- CardGotoControlWindow
- CardDelete
- CardDeleteLink
- CardEditTitle
- CardEditFilename
- CardLinkNewCard
- CardLinkToSelection
- CardOpenFile
- CardOrderLinks
- CardSaveFile
- CardSaveFileAs
- CardSelectItem
- CardToggleLinkPanel
- CardQuit

12.22.3 Diagram card commands

- DiagramAddAnnotation
- DiagramAddControl
- DiagramApplyDefinition
- DiagramBrowse
- DiagramChangeFont
- DiagramClearAll
- DiagramCopy
- DiagramCopyDiagram
- DiagramCopySelection
- DiagramCopyToClipboard
- DiagramCut
- DiagramDeleteAnnotation
- DiagramDeleteControl
- DiagramDeselectAll
- DiagramDuplicateSelection
- DiagramEditOptions
- DiagramFormatGraph

- DiagramFormatText
- DiagramFormatTree
- DiagramGotoRoot
- DiagramHelp
- DiagramHorizontalAlign
- DiagramHorizontalAlignTop
- DiagramHorizontalAlignBottom
- DiagramNewExpansion
- DiagramPaste
- DiagramPrint
- DiagramPrintAll
- DiagramPrintEPS
- DiagramPrintPreview
- DiagramRefresh
- DiagramSaveBitmap
- DiagramSaveMetafile
- DiagramSelectAll
- DiagramStraighten
- DiagramToBack
- DiagramToFront
- DiagramTogglePalette
- DiagramToggleToolbar
- DiagramVerticalAlign
- DiagramVerticalAlignLeft
- DiagramVerticalAlignRight
- DiagramZoom30
- DiagramZoom40
- DiagramZoom50
- DiagramZoom60
- DiagramZoom70
- DiagramZoom80
- DiagramZoom90
- DiagramZoom100

12.22.4 Hypertext card commands

- HypertextClearAllBlocks
- HypertextClearBlock
- HypertextClearSelection
- HypertextDeleteLinks
- HypertextEditOptions
- HypertextHelp
- HypertextNextSection
- HypertextPreviousSection
- HypertextRunEditor
- HypertextTop

12.22.5 Text card commands

- TextCopy
- TextCut
- TextHelp
- TextPaste
- TextRunEditor

Glossary

API Application Programmer's Interface - a set of calls and classes defining how a library can be used.

Bit list A bit list in wxCLIPS is a way of specifying several window styles. It derives from C and C++ syntax, where by defining identifiers with carefully chosen binary numbers, it is possible to combine several values in one integer. In wxCLIPS, you use similar syntax to C, but enclose the list in quotes:

```
"wxCAPTION | wxMINIMIZE_BOX | wxMINIMIZE_BOX | wxTHICK_FRAME"
```

Callback Callbacks are application-defined functions which receive events from the GUI. You normally add a callback for a particular window (such as a canvas) and event (such as OnPaint) using window-add-callback, or pass the callback in a panel item creation function, such as button-create.

Canvas A canvas is a subwindow on which graphics (but not panel items) can be drawn. It may be scrollable. A canvas has a Section **??** associated with it.

DDE Dynamic Data Exchange - Microsoft's interprocess communication protocol. wx-CLIPS provides a subset of DDE under both Windows and UNIX.

Device context A device context is an abstraction away from devices such as windows, printers and files. Code that draws to a device context is generic since that device context could be associated with a number of different real device. A canvas has a device context, although duplicate graphics calls are provided for the canvas, so the beginner doesn't have to think in terms of device contexts when starting out. See Section **??** .

Dialog box In wxCLIPS a dialog box is a convenient way of popping up a window with panel items, without having to explicitly create a frame and a panel. A dialog box may be modal or modeless. A modal dialog does not return control back to the calling program until the user has dismissed it, and all other windows in the application are disabled until the dialog is dismissed. A modeless dialog is just like a normal window in that the user can access other windows while the dialog is displayed.

Frame A visible window usually consists of a frame which contains zero or more subwindows, such as text subwindow, canvas, and panel.

GUI Graphical User Interface, such as MS Windows or Motif.

Menu bar A menu bar is a series of labelled menus, usually placed near the top of a window.

Metafile MS 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. wxCLIPS implements enough metafile functionality to use it to pass graphics to other applications via the clipboard or files.

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 wxCLIPS sits on top of XView (among other toolkits).

Panel A panel is a subwindow on which a limited range of panel items (widgets or controls for user input) can be placed. wxCLIPS allows panel items to be placed explicitly, or laid out from left to right, top to bottom, which is a more platform independent method since spacing is calculated automatically at run time. Panel items cannot be placed on a canvas, which is specifically for drawing graphics. However, you can draw on a panel.

Resource Resource takes several meanings in wxCLIPS. The functions get-resource, write-resource deal with MS Windows `.ini` and X `.Xdefaults` resource entries. The wxWindows/wxCLIPS 'resource system', on the other hand, is a facility for loading dialog specifications from `.wxr` files (which may be created by hand or using the wxWindows Dialog Editor).

Status line A status line is often found at the base of a window, to keep the user informed (for instance, giving a line of description to menu items, as in the **hello** demo).

XView An X toolkit supplied by Sun Microsystems for implementing the Open Look 'look and feel'. Freely available, but virtually obsolete.

Index

Index