Harlequin Dylan

DUIM Release Notes

Version 1.1

Release notes for the DUIM manuals *Building Applications Using DUIM* and *DUIM Reference*.

harlequin™

Europe:

Harlequin Limited
Barrington Hall
Barrington
Cambridge CB2 5RG
U.K.

telephone +44 1223 873 800
fax +44 1223 873 873

North America:

Harlequin Incorporated
One Cambridge Center
Cambridge, MA 02142
U.S.A.

telephone +1 617 374 2400
fax +1 617 252 6505

Electronic Access:

http://www.harlequin.co.uk/
http://www.harlequin.com/

Contents

Release Notes for DUIM Manuals

The following release notes refer to the documentation for DUIM that was distributed with Harlequin Dylan 1.0, which consisted of two manuals: *Building Applications Using DUIM*, and *DUIM Reference*. This document is divided into sections describing errors in, omissions from, and additions to both these manuals.

Throughout this document, any references given refer to a chapter or section in the relevant DUIM manual, unless it is explicitly stated otherwise.

1 Building Applications Using DUIM

This section describes errors in, omissions from, and additions to the 1.0 version of the manual *Building Applications Using DUIM*. Each of the following subsections refer to chapters and sections of the manual, and reflect the structure of the original manual. Only those chapters and sections to which changes or additions apply are included.

1.1 General notes

This section describes general changes to the *Building Applications Using DUIM*.

When experimenting with the Task List 2 example described in this manual, please ensure that you use the code available from the development environment, rather than the code described in the manual. To do this, choose **Help > Open Examples**, and in the Open Example Project dialog that appears, choose Task List 2 in the Documentation category. See Section 1.4 on page 7 and Section 1.6 on page 11 of this document for more details.

1.2 Chapter 3: Improving The Design

This section describes changes and additions to Chapter 3 of *Building Applications Using DUIM*.

1.2.1 Section 3.1: Defining a project

The last paragraph of Section 3.1 (page 12 of the printed manual), instructs you to create a file called **`frame.dylan`**, and add it to the project using **Project > Insert File**. Note that you should create the file using **File > New**, and then add the file to the project using **Project > Insert File**. The **Insert File** command *does not* create the file for you.

1.2.2 Section 3.5: Adding a tool bar

This section explains how you can add a tool bar to the Task List application, and adds two buttons (**`add-button`** and **`remove-button`**) to the tool bar that have already been used in the layout described in Section 3.4, "Defining a new frame class".

Before including **`add-button`** and **`remove-button`** in the definition of the tool bar described in this section, you should ensure that you remove them from the task-layout pane of the definition of **`<task-frame>`**. If you fail to do this, DUIM attempts to use the same buttons in two different parts of the interface, with undefined results.

1.2.3 Section 3.7: Gluing the new design together

Note that the definition for the new design of the frame class given at the end of this section does not incorporate the original task-text pane defined in Section 3.4, “Defining a new frame class”. In fact, this part of the original interface is handled differently in the final design, and is re-implemented in Section 3.8, “Creating a dialog for adding new items”.

1.2.4 Section 3.8: Creating a dialog for adding new items

The definition of the **`prompt-for-task`** method uses the **`<priority>`** type. Note that this type is defined in Section 5.1, “Defining the underlying data structures for tasks”. Until the relevant code in section 5.1 is added to your project, any attempt to build it will generate a serious warning.

1.3 Chapter 4: Adding Menus To The Application

This section describes changes and additions to Chapter 6 of *Building Applications Using DUIM*.

1.3.1 Section 4.2: Creating a menu hierarchy

Note that the **`make-keyboard-gesture`** function is defined in Section 4.2“.2, Keyboard accelerators”.

1.4 Chapter 6: Using Command Tables

This section describes changes and additions to Chapter 6 of *Building Applications Using DUIM*.

1.4.1 Problems with command table definitions

The command table definitions given in Section 6.2, “Implementing a command table”, and Section 6.3, “Re-implementing the menus of the task list manager” do not contain terminating `;` characters. Therefore, if you type the code exactly as described in the manual, the example will not compile correctly.

In addition, you should ensure that you place the command table definitions provided in Section 6.3, “Re-implementing the menus of the task list manager” *after* the callback definitions themselves, to avoid forward references.

1.4.2 Problems with callback definitions

For the Task List 2 project to function correctly, the following callbacks should be redefined so as to take an instance of **`<task-frame>`** as an argument, rather than an instance of **`<gadget>`**.

```
frame-add-task
frame-remove-task
open-file
save-file
save-as-file
about-task
exit-task
```

For complete definitions of these callbacks, you should refer to the source code available from the Open Example Project dialog in the environment.

1.4.3 Problems with button definitions

In order for the Task List 2 project to function correctly, the definition of each button in the definition of **`<task-frame>`** needs to be modified compared to their definition in the Task List 1 project, as described in Section 3.7, “Gluing the new design together”.

Broadly speaking, you need to update the **`command:`** keyword / argument pair for each button gadget, and you need to redefine the activate callback to allow for the fact that the callbacks now take frames as arguments.

Thus, for a button that is defined as:

```
pane add-button (frame)
  make (<push-button>, label: "Add task",
        activate-callback: frame-add-task);
```

the new definition is:

```
pane add-button (frame)
  make(<push-button>, label: "Add task",
       command: frame-add-task,
       activate-callback: method (gadget)
                            frame-add-task(frame) end);
```

For complete definitions, you should refer to the source code available from the Open Example Project dialog in the environment.

1.4.4 Problems with the note-task-selection-change callback

For the Task List 2 project to function correctly, two changes are required to the definition of the **`note-task-selection-change`** callback. The description of this callback, as required for the Task List 1 example project, is given in Section 5.3.3.3, "Enabling and disabling buttons in the interface". For the *Task List 1* project, this description remains correct. However, some modifications to this callback are necessary in order for it to work in the Task List 2 project. These are as follows:

- Only one definition of **`note-task-selection-change`** is required. The method that takes a gadget as an argument (the second method described in Section 5.3.3.3, on page 69 of the printed manual) is not required for Task List 2.
- The **`note-task-selection-change`** method that *is* required for the Task List 2 project needs a small modification compared to its description in Section 5.3.3.3. The last three lines of this callback should be replaced with a call to **`command-enabled?`**, as shown in the source code below.

The definition of **`note-task-selection-change`** in the Task List 2 project should therefore be as follows:

```
define method note-task-selection-change
    (frame :: <task-frame>) => ()
  let task = frame-selected-task(frame);
  if (task)
    frame.priority-box.gadget-value := task.task-priority;
  end;
  command-enabled?(frame-remove-task, frame) := task ~= #f;
end method note-task-selection-change;
```

Because of this change, a further change is required, this time to the definition of the **`task-list`** pane in the definition of **`<task-frame>`**. The value-changed callback for the **`task-list`** pane should be changed from

```
value-changed-callback: note-task-selection-change;
```

to

```
value-changed-callback: method (gadget)
                        note-task-selection-change(frame) end);
```

Note that the source code provided in the example project is correct, and will compile, link, and run successfully. This can be loaded from the Open Example Project dialog, and is listed as Task List 2 in the Documentation category.

1.4.5 Problems with the refresh-task-frame callback

The **`refresh-task-frame`** callback that is described in Section 5.3.3.4, “Refreshing the list of tasks”, requires a change in order for it to work correctly in the Task List 2 project. This change is similar to the change that needs to be made to **`note-task-selection-change`**, described in Section 1.4.4 on page 9 of this document.

For **`refresh-task-frame`** to work correctly in the Task List 2 project, replace the penultimate two lines of the definition given in Section 5.3.3.4 with a call to **`command-enabled?`**, so that the definition of **`refresh-task-frame`** is as follows:

```
define method refresh-task-frame
    (frame :: <task-frame>) => ()
  let list-box = frame.task-list;
  let task-list = frame.frame-task-list;
  let modified? = task-list.task-list-modified?;
  let tasks = task-list.task-list-tasks;
  if (gadget-items(list-box) == tasks)
    update-gadget(list-box)
  else
    gadget-items(list-box) := tasks
  end;
  command-enabled?(save-file, frame) := modified?;
  note-task-selection-change(frame);
end method refresh-task-frame;
```

1.5 Chapter 7: A Tour Of The DUIM Libraries

This section describes changes to Chapter 7 of *Building Applications Using DUIM*.

1.5.1 Section 7.1, Introduction

In the second paragraph, you are advised to use the **`duim-user`** module for running examples in this chapter. You should ignore this, and instead just use the Dylan Playground for running any examples.

1.5.2 Section 7.3.1: Row layouts and column layouts

The end of this section states that **`#"baseline"`** is an allowed value for the **`y-alignment:`** init-keyword of row layouts. This is, in fact, not the case.

1.6 Appendix A: Source Code For The Task List Manager

This section describes changes to Appendix A of *Building Applications Using DUIM*.

There are a number of problems with the source code provided in Section A.2, “A task list manager using command tables” as follows.

- Like the code examples in Chapter 6, the command table definitions do not contain terminating ; characters.
- The code contains a number of forward referencing problems.
- The definitions for a number of callback functions allowed the functions to be passed gadgets as arguments, whereas they should be passed frames to work correctly with command tables.
- The definition of the **`note-task-selection-change`** callback should use **`command-enabled?`** to test whether the appropriate gadgets in the frame are enabled. See Section 1.4.4 on page 9 of this document for details.

Note that the source code provided as an example project is correct, and will compile, link, and run successfully. This can be loaded from the Open Example Project dialog, and is listed as Task List 2 in the Documentation category,

2 DUIM Reference

This section describes errors in, omissions from, and additions to the 1.0 version of the *DUIM Reference* manual. Each of the following subsections refer to a library that forms part of DUIM. Only those libraries to which changes or additions apply are included.

2.1 DUIM-Geometry library

No changes.

2.2 DUIM-Extended-Geometry library

No changes.

2.3 DUIM-DCs library

No changes.

2.4 DUIM-Sheets library

2.4.1 The base classes in the DUIM-Sheets library

The following is a replacement for Table 5.1, Overall class hierarchy for the DUIM-Sheets library, in Section 5.2.1 of the *DUIM Reference* manual.

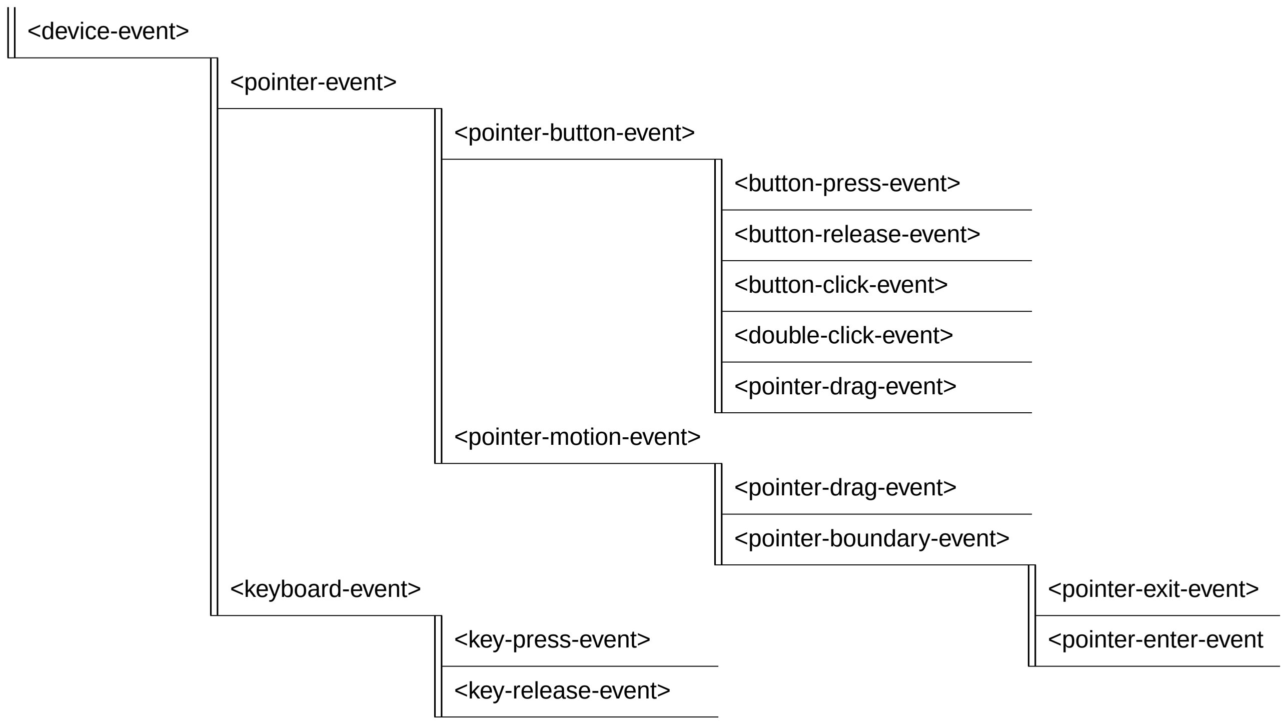
Table 2.1 Overall class hierarchy for the DUIM-Sheets library

<object>		
	<sheet>	
		<display>
	<port>	
	<clipboard>	
	<caret>	
	<pointer>	
	<medium•	
	<frame-manager•	
	<event>	

2.4.2 Subclasses of <device-event>

The following is a replacement for Table 5.3, Subclasses of the **`<device-event>`** class, in Section 5.2.3 of the *DUIM Reference* manual.

Table 2.2 Subclasses of the **`<device-event>`** class



Note: The **`<pointer-drag-event>`** class is a subclass of both **`<pointer-button-event>`** and **`<pointer-motion-event>`**.

2.4.3 Correction to keyword descriptions for choose-* functions

In the entries for **`choose-color`**, **`choose-directory`**, and **`choose-file`**, in the Harlequin Dylan 1.0 documentation, the *frame* and *owner* keywords were incorrectly described. A correct description is given below.

The *frame* argument is an instance of **`<frame>`**. If specified, the top-level sheet of *frame* becomes the owner of the dialog.

Alternatively, you can specify the owner directly using the *owner* argument, which takes an instance of **`<sheet>`** as its value.

By default, both *frame* and *owner* are **`#f`**, meaning the dialog has no owner. You should not specify both of these values.

In general,

2.4.4 Documentation for choose-from-dialog

The following replaces the entry for **`choose-from-dialog`** in Chapter 5 of the *DUIM Reference* manual.

choose-from-dialog *Generic function*

`choose-from-dialog` ***items***
  `#key` ***frame owner title value default-item label-key value-key***
    selection-mode gadget-class gadget-options width height foreground
background text-style
`=>` ***value success?***

Prompt the user to choose from a collection of *items*, using a dialog box.This generic function is similar to **`choose-from-menu`**. The value given to *items* is an instance of **`type-union(<sequence>, <menu>)`**. This generic function returns the values chosen by the user, and a boolean value; **`#t`** if a value was chosen, **`#f`** if nothing was chosen. Unlike **`choose-from-menu`**, the user can choose several values if desired, depending on the value of *selection-mode*, described below.

At its most basic, **`choose-from-dialog`** can be passed a simple sequence of items, as follows:

```
choose-from-dialog(range(from: 1, to: 10));
```

However, any of a large number of keywords can be supplied to specify more clearly the dialog that is created. A range of typical options can be chosen: The *frame* keyword specifies a frame whose top level sheet becomes the owner of the menu. Alternatively, you can specify this top level sheet explicitly using *owner*. The *title* keyword lets you choose a title for the dialog. By default, each of these values is **`#f`**.

In addition, **`choose-from-dialog`** offers options similar to collection gadgets, that can act upon the items specified. The *default-item* keyword lets you specify an item that is returned by default if no value is chosen explicitly (thereby ensuring that *success?* will always be `#t`). You can also specify a *value-key* or *label-key* for the items in the menu. The *selection-mode* keyword is used to make the dialog box single-selection (the user can only choose one value) or multiple-selection (the user can return any number of values). The default value of *selection-mode* is **`#"single"`**. By specifying **`selection-mode: #"multiple"`**, the user can choose several values from the dialog box. The *gadget-class* keyword lets you specify which type of collection gadget is displayed in the dialog box. This lets you, for example, display a list of check boxes or radio boxes. Finally, gadget-options let you specify a set of options to be applied to the collection gadgets in the dialog box.

You can also configure the appearance of the menu itself. The *width* and *height* keywords let you set the size of the menu. The *foreground* and *background* keywords let you set the text color and the menu color respectively. The *text-style* keyword lets you specify a font to display the menu items.

2.4.5 Documentation for choose-from-menu

The following replaces the entry for **`choose-from-menu`** in Chapter 5 of the *DUIM Reference* manual.

choose-from-menu *Generic function*

`choose-from-menu` ***items***
  `#key` ***frame owner title value default-item label-key value-key***
    width height foreground background text-style multiple-sets?
`=>` ***value success?***

Prompt the user to choose from a collection of *items*, using a pop-up menu.This generic function is similar to **`choose-from-dialog`**. The value given to *items* is an instance of **`type-union(<sequence>, <menu>)`**. This generic function returns the value chosen by the user, and a boolean value; `#t` if a value was chosen, `#f` if nothing was chosen.

At its most basic, **`choose-from-menu`** can be passed a simple sequence of items, as follows:

```
choose-from-menu(#(1, 2, 3));
```

However, any of a large number of keywords can be supplied to specify more clearly the menu that is created. A range of typical options can be chosen: The *frame* keyword specifies a frame whose top level sheet becomes the owner of the menu. Alternatively, you can specify this top level sheet explicitly using *owner*. The *title* keyword lets you choose a title for the dialog. By default, each of these values is **`#f`**.

In addition, **`choose-from-menu`** offers options similar to collection gadgets, that can act upon the items specified. The *default-item* keyword lets you specify an item that is returned by default if no value is chosen explicitly (thereby ensuring that *success?* will always be **`#t`**). You can also specify a *value-key* or *label-key* for the items in the menu.

Finally, you can configure the appearance of the menu itself. The *width* and *height* keywords let you set the size of the menu. The *foreground* and *background* keywords let you set the text color and the menu color respectively. The *text-style* keyword lets you specify a font to display the menu items.

2.4.6 Documentation for clipboard support

Documentation for support of the Windows clipboard was not available for Harlequin Dylan 1.1.

In order to manipulate the Windows clipboard from within DUIM, the clipboard needs to be locked, so that its contents can be manipulated. DUIM uses the functions **`open-clipboard`** and **`close-clipboard`** to create and free clipboard locks. The **`open-clipboard`** function creates an instance of the class **`<clipboard>`** which is used to hold the contents of the clipboard for the duration of the lock. For general use of the clipboard, use the macro **`with-clipboard`**, rather than calling **`open-clipboard`** and **`close-clipboard`** explicitly. This lets you manipulate the clipboard easily, sending the results of any code evaluated to the clipboard.

Once a clipboard lock has been created, you can use **`add-clipboard-data`** and **`add-clipboard-data-as`** to add data to the clipboard. Use **`get-clipboard-data-as`** to query the contents of the clipboard, and use **`clear-clipboard`** to empty the locked clipboard. Finally, use **`clipboard-data-available?`** to see if the clipboard contains data of a particular type.

You can put arbitrary Dylan objects onto the clipboard, and retrieve them within the same process. This gives you the ability to cut and paste more interesting pieces of an application within the application's own domain than would normally be possible.

The DUIM GUI test suite contains a demonstration of how to use the clipboard in DUIM, in the file **`Examples\duim\duim-gui-test-suite\clipboard.dylan`** in the Harlequin Dylan installation directory.

The following clipboard-related interfaces are all exported from the DUIM-Sheets module.

open-clipboard *Function*

`open-clipboard` ***port sheet*** **`=>`** ***clipboard***

Creates a clipboard lock for *sheet* on *port*. Once a clipboard lock has been created, you can manipulate the clipboard contents safely. An instance of **`<clipboard>`** is returned, which is used to hold the clipboard contents.

You should not normally call **`open-clipboard`** yourself to create a clipboard lock. Use the macro **`with-clipboard`** to create and free the lock for you.

close-clipboard *Function*

`close-clipboard` ***port sheet*** **`=> ()`**

Closes the current clipboard lock for *sheet* on *port*. A clipboard lock needs to be closed safely after it the clipboard has been used, to free the clipboard for further use.

You should not normally call **`close-clipboard`** yourself to close a clipboard lock. Use the macro **`with-clipboard`** to create and free the lock for you.

<clipboard> *Open abstract class*

The class of clipboard objects. An instance of this class is created when a clipboard lock is created, and is used to hold the contents of the Windows clipboard for the duration of the lock. You do not need to worry about creating instances of **`<clipboard>`** yourself, since this is handled automatically by the macro **`with-clipboard`**.

clipboard-sheet *Generic function*

`clipboard-sheet` ***clipboard*** **`=>`** ***sheet***

Returns the sheet with the clipboard lock.

clipboard-owner *Generic function*

`clipboard-owner` ***clipboard*** **`=>`** ***owner***

Returns the sheet that owns the current clipboard data.

with-clipboard *Statement macro*

`with-clipboard (`***clipboard*** **`=`** ***sheet*****`)`** ***body*** **`end`**

This macro evaluates *body* with the clipboard grabbed, returning the results to the clipboard. The **`with-clipboard`** macro grabs a lock on the clipboard, using **`open-clipboard`**, and then executes body. Once the results of evaluating *body* have been sent to the clipboard, the clipboard lock is freed using **`close-clipboard`**. The *clipboard* argument is a Dylan variable-name$_{\text{bnf}}$ used locally in the call to **`with-clipboard`**. The *sheet* argument is a Dylan variable-name$_{\text{bnf}}$ that evaluates to the sheet associated with *clipboard*.

This macro is the easiest way of manipulating the clipboard from DUIM, since it removes the need to create and destroy a clipboard lock yourself.

You can add more than one format of your data to the clipboard within the scope of the **`with-clipboard`** macro. So, for example, you could place an arbitrary object onto the clipboard, for use within your own application, and a string representation for other tools applications to see.

add-clipboard-data *Generic function*

`add-clipboard-data` ***clipboard data*** **`=>`** ***success?***

This generic function adds *data* to *clipboard*. It returns **`#t`** if *data* was successfully added to the clipboard.

add-clipboard-data-as *Generic function*

`add-clipboard-data` ***type clipboard data*** **`=>`** ***success?***

This generic function adds *data* to *clipboard*, first coercing it to *type*. The argument *type* is an instance of **`type-union(<symbol>, <type>)`**. It returns **`#t`** if *data* was successfully added to the clipboard.

clear-clipboard *Generic function*

`clear-clipboard` ***clipboard*** **`=> ()`**

This generic function clears the contents of the clipboard. The *clipboard* argument is an instance of **`<clipboard>`**, and represents the locked clipboard.

clipboard-data-available? *Generic function*

`clipboard-data-available?` ***type clipboard*** **`=>`** ***available?***

This generic function returns **`#f`** if and only if there is any data of type *type* on the clipboard. The argument *type* is an instance of **`type-union(<symbol>, <type>)`**.

get-clipboard-data-as *Generic function*

`get-clipboard-data-as` ***type clipboard*** **`=>`** ***data***

This generic function returns *data* of *type* from the clipboard. The argument *type* is an instance of **`type-union(<symbol>, <type>)`**.

2.4.7 Corrected example for choose-directory

The following example replaces the example given for the entry for choose-directory in Chapter 5.

```
define frame <directory-dialog-frame> (<simple-frame>)
  pane dir-file-button (frame)
    make(<menu-button>,
         label: "Choose directory ...",
         documentation:
          "Example of standard 'Choose Dir' dialog",
         activate-callback:
          method (button)
          let dir = choose-directory (owner: frame);
            if (dir) frame-status-message(frame)
             := format-to-string
                  ("Chose directory %s", dir);
            end
          end);
  pane dir-layout (frame)
    vertically ()
      frame.dir-file-button;
    end;
  layout (frame) frame.dir-layout;
  keyword title: = "Choose directory example";
end frame <directory-dialog-frame>;
```

2.4.8 Documentation for choose-text-style

Documentation for the **`choose-text-style`** generic function is missing from Chapter 5.

choose-text-style *Generic function*

`choose-text-style #key` ***frame owner title*** **`=>`** ***font***

Displays the built-in font dialog for the target platform, thereby letting the user choose a font.

The *frame* argument is an instance of type **`<frame>`**. By default, this is the value returned by **`current-frame()`**.

The *owner* argument is an instance of type **`<sheet>`**.

The *frame* is a frame in which the dialog is displayed, and the *owner* is the sheet that is the owner of the dialog.

If you wish, you can specify a *title* for the dialog; this is an instance of **`<string>`** and is displayed in the title bar of the frame containing the dialog. If you do not specify *title*, then DUIM uses the default title for that type of dialog on the target platform.

2.4.9 Correction to do-with-pointer-grabbed and with-pointer-grabbed

The reference entries in Chapter 5 for these two interfaces claim that the mouse pointer is immobilized during calls to these interfaces. This is not the case. In fact, inside a call to these interfaces, all pointer events are forwarded to the associated sheet, even if the mouse pointer leaves the sheet region of sheet. With this in mind, the following information replaces the relevant entries in Chapter 5.

do-with-pointer-grabbed *Generic function*

`do-with-pointer-grabbed` ***port sheet continuation*** **`#key => #rest`** ***values***

Runs the code specified in *continuation*, forwarding all pointer events to *sheet*, even if the pointer leaves the sheet-region of *sheet*. The argument continuation is an instance of **`<function>`**.

The function **`do-with-pointer-grabbed`** is called by **`with-pointer-grabbed`**, and *continuation* is actually the result of creating a stand-alone method from the body of code passed to **`with-pointer-grabbed`**.

with-pointer-grabbed *Statement macro*

`with-pointer-grabbed ({`*sheet*`} #rest {`*options*`}*) {`*body*`} end`

Executes a body of code, forwarding all pointer events to *sheet*, even if the pointer leaves the sheet-region of *sheet*. The *sheet* specified should be an instance of type **`<sheet>`**.

The macro calls methods for **`do-with-pointer-grabbed`**. The code specified by *body* is used to create a stand-alone method that is used as the code that is run by **`do-with-pointer-grabbed`**.

2.4.10 Replacement documentation for handle-event

The description for the **`handle-event`** generic function is incomplete. The following information replaces the entry in Chapter 5.

handle-event *Generic function*

`handle-event` ***sheet event*** **`=> ()`**

The argument *sheet* is an instance of **`<sheet>`**, and the argument *event* is an instance of **`<event>`**.

The **`handle-event`** generic function gets called by DUIM in an application thread in order to handle a queued event. By calling available methods, it implements any defined policies of *sheet* with respect to *event*. For example, to highlight a sheet in response to an event that informs the sheet when the pointer has entered the region it occupies, there should be a method to carry out the policy that specializes the appropriate sheet and event classes.

DUIM itself implements no semantically meaningful **`handle-event`** methods; It is the responsibility of any application to implement all of its own **`handle-event`** methods. It is also the responsibility of the application to decide the protocol and relationship between all of these methods.

Take care when adding **`next-method()`** calls in any **`handle-event`** methods that you write. Because DUIM itself supplies no built-in methods, you must ensure that you have supplied a valid method yourself. For each event class you are handling, you should decide whether a call to **`next-method`** is actually required.

2.4.11 Replacement documentation for handle-repaint

The description for the **`handle-repaint`** generic function is incomplete. The following information replaces the entry in Chapter 5.

handle-repaint *Generic function*

`handle-repaint` ***sheet medium region*** **`=> ()`**

The argument *sheet* is an instance of **`<sheet>`**, the argument *medium* is an instance of **`<medium>`**, and the argument *region* is an instance of **`<region>`**.

The **`handle-repaint`** generic function gets called by DUIM in an application thread in order to handle repainting a given part of the screen. By calling available methods, it repaints the *region* of the *sheet* on *medium*.

DUIM itself implements no semantically meaningful **`handle-repaint`** methods; It is the responsibility of any application to implement all of its own **`handle-repaint`** methods. It is also the responsibility of the application to decide the protocol and relationship between all of these methods.

Take care when adding **`next-method()`** calls in any **`handle-repaint`** methods that you write. Because DUIM itself supplies no built-in methods, you must ensure that you have supplied a valid method yourself. For each sheet class you are handling, you should decide whether a call to **`next-method`** is actually required.

2.4.12 Superclasses of pointer events

The following are corrections to reference entries provided in Chapter 5.

The **`<pointer-drag-event>`** class is described as being a subclass of **`<object>`**. In fact, it is a subclass of **`<pointer-motion-event>`** and **`<pointer-button-event>`**.

The **`<pointer-enter-event>`** class is described as being a subclass of **`<object>`**. In fact, it is a subclass of **`<pointer-boundary-event>`**.

The **`<pointer-exit-event>`** class is described as being a subclass of **`<pointer-event>`**. In fact, it is a subclass of **`<pointer-boundary-event>`**.

See also Section 2.4.2 on page 14 of this document for more information.

2.5 DUIM-Graphics library

The reference to CLIM at the end of Section 6.3 “Drawing is approximate”, on page 331 of the *DUIM Reference* manual should, in fact, refer to DUIM.

2.6 DUIM-Layouts library

2.6.1 Init-keywords available to <layout>

The manual entry for the **`<layout>`** class refers to a **`resizeable:`** init-keyword. This name of this init-keyword is in fact **`resizable:`**.

2.6.2 New init-keyword for <stack-layout>

The **`mapped-page:`** init-keyword has been added to the **`<stack-layout>`** class in Harlequin Dylan 1.1. This takes an instance of **`<sheet>`** as its value. The mapped-page: init-keyword allows you to assign a page to be mapped onto the screen when a stack layout is first created. If it is not specified, then the first page in the stack layout is mapped.

The following are descriptions of the associated getter and setter.

stack-layout-mapped-page *Generic function*

`stack-layout-mapped-page` ***stack-layout*** **`=>`** ***page***

Returns the currently mapped *page* for the specified *stack-layout*. The return value is an instance of **`<sheet>`**.

stack-layout-mapped-page-setter *Generic function*

`stack-layout-mapped-page` ***page stack-layout*** **`=>`** ***page***

Sets the mapped page for the specified *stack-layout* to *page*. The *page* argument is an instance of **`<sheet>`**.

2.6.3 Init-keyword values available to <table-layout> and <row-layout>

The manual entries for the **`<table-layout>`** and **`<row-layout>`** classes refer to the **`#"baseline"`** value of the **`y-alignment:`** init-keyword. In fact, this is not an allowed value for the **`y-alignment:`** init-keyword.

2.7 DUIM-Gadgets library

2.7.1 Available types of <border>

The manual entry for the **`<border>`** class contains a figure that shows some of the different types of border that can be created. Please note that the border labeled as “ridged” should really be labeled “ridge”, to be consistent with the **`#"ridge"`** symbol that is used to create that type of border.

2.7.2 Possible return values for gadget-y-alignment

The manual entry for **`gadget-y-alignment`** states that **`#"baseline"`** is a possible return value for this generic function. In fact, this is not the case.

2.7.3 Usage example for <spin-box>

The example given in the manual entry for **`<spin-box>`** is incorrect. The following is a correct example:

```
contain(make(<spin-box>, items: range(from: 1, to: 10)));
```

2.7.4 Documentation for splitters

Support for splitters has been added to DUIM in Harlequin Dylan 1.1. The following documentation describes the interfaces available for creating and controlling splitters in your applications.

<splitter> *Abstract instantiable class*

The class of splitter gadgets. Splitters are subclasses of both **`<gadget>`** and **`<layout>`**. Splitters (sometimes referred to as split bars in Microsoft documentation) are gadgets that allow you to split a pane into two resizable portions. For example, you could create a splitter that would allow more than one view of a single document. In a word processor, this may be used to let the user edit disparate pages on screen at the same time.

A splitter consists of two components: a button that is used to create the splitter component itself (referred to as a split box), and the splitter component (referred to as the split bar). The split box is typically placed adjacent to the scroll bar. When the user clicks on the split box, a movable line is displayed in the associated pane which, when clicked, creates the split bar.

The **`split-box-callback:`** init-keyword is an instance of type **`false-or(<function>)`**, and specifies the callback that is invoked when the split box is clicked.

The **`split-bar-moved-callback:`** init-keyword is an instance of type **`false-or<function>)`**, and specifies a callback that is invoked when the user moves the split bar.

The **`horizontal-split-box?:`** init-keyword is an instance of type **`<boolean>`**, and if true a horizontal split bar is created.

The **`vertical-split-box?:`** init-keyword is an instance of type **`<boolean>`**, and if true a vertical split bar is created.

splitter-split-bar-moved-callback *Generic function*

`splitter-split-bar-moved-callback` ***splitter*** **`=>`** ***function***

Returns the function invoked when the split bar of *splitter* is moved.

The *splitter* argument is an instance of type **`<splitter>`**. The *function* argument is an instance of type **`<function>`**.

splitter-split-bar-moved-callback-setter *Generic function*

`splitter-split-bar-moved-callback-setter` ***function splitter*** **`=>`** ***function***

Sets the callback invoked when the split bar of *splitter* is moved.

The *splitter* argument is an instance of type **`<splitter>`**. The *function* argument is an instance of type **`<function>`**.

splitter-split-box-callback *Generic function*

`splitter-split-box-callback` ***splitter*** **`=>`** ***function***

Returns the callback invoked when the split box of *splitter* is clicked.

The *splitter* argument is an instance of type **`<splitter>`**. The *function* argument is an instance of type **`<function>`**.

splitter-split-box-callback-setter *Generic function*

`splitter-split-box-callback-setter` ***function splitter*** **`=>`** ***function***

Sets the callback invoked when the split box of *splitter* is clicked.

The *splitter* argument is an instance of type **`<splitter>`**. The *function* argument is an instance of type **`<function>`**.

gadget-ratios *Generic function*

`gadget-ratios` ***splitter*** **`=>`** ***ratios***

Returns the ratios of the windows in *splitter*. This generic function lets you query the position of a splitter.

The *splitter* argument is an instance of type **`<splitter>`**. The *ratios* argument is an instance of type **`false-or(<sequence>)`**.

gadget-ratios-setter *Generic function*

`gadget-ratios-setter` ***ratios splitter*** **`=>`** ***ratios***

Sets the ratios of the windows in *splitter*. This generic function lets you set the position of a splitter.

The *splitter* argument is an instance of type **`<splitter>`**. The *ratios* argument is an instance of type **`false-or(<sequence>)`**. Set *ratios* to **`#f`** if you do not care what ratios are used.

2.7.5 New keyword for tab-control-pages-setter

A **`page:`** keyword has been added to the **`tab-control-pages-setter`** generic function for Harlequin Dylan 1.1. The following documentation replaces the entry in Chapter 8 of the *DUIM Reference* manual.

tab-control-pages-setter *Generic function*

`tab-control-pages-setter` ***pages tab-control*** **`#key`** ***page*** **`=>`** ***pages***

Sets the tab pages available to *tab-control*, optionally setting *page* to the default page to be displayed. The pages argument is an instance of **`limited(<sequence>, of: <page>)`**. The *page* argument is an instance of **`<page>`** and, moreover, must be one of the pages contained in *pages*.

The **`tab-control-pages-setter`** function is used as follows:

```
tab-control-pages(my-tab-control, page: my-page) := my-pages
```

2.8 DUIM-Frames library

2.8.1 New Commands library

All commands-related interfaces are now defined directly in the Commands library. However, these same interfaces are imported to and re-exported from DUIM-Frames, so they can be used in almost the same way as for Harlequin Dylan 1.0. You should continue to look for commands-related documentation in Chapter 9 of the *DUIM Reference* manual.

A consequence of the introduction of the Commands library is that a slight change in syntax is required in the definition of commands in command tables. In Harlequin Dylan 1.0, both two approaches could be taken when specifying a command in a table. For example, a menu item could be specified by either of the following:

```
menu-item "My Command" = make(<command>, function: my-command),
```

```
menu-item "My Command" = my-command,
```

In Harlequin Dylan 1.1, only the last of these may be used. This may require you to change some of your code.

2.8.2 Signature for call-in-frame

The manual entry for **`call-in-frame`** erroneously refers to **`apply-in-frame`** in the Signature section. This should, of course, refer to **`call-in-frame`**.